DD2434 Advanced Machine Learning Project

Text Classification using String Kernels

Group Fang-Lin-Song-Zhang P

Authors:

Heng Fang	9708240289	hfang@kth.se
Shuyuan Zhang	9608088614	shuyuanz@kth.se
Weihong Sung	9503068356	weihongs@kth.se
Yuhao Lin	9612152430	yuhaol@kth.se

Academic year 2018-2019

20th January 2019

Abstract

In this report, the task of text categorization is done with regard to Reuters data set. We re-implement the String Subsequence Kernel(SSK, approximated SSK) and compare its performance with word-kernel(WK) and n-gram kernel(NGK). Considering these kernel-based methods ignore the semantic similarity between words, we then adopt and implement semantic kernel. Due to the limitation of computational capability, we only choose part of the data set to train and test the performance of our classifier which is based on Support vector Machine(SVM).

1 Introduction

The aim of text classification is to assign different text samples into corresponding categories according to their contents. Usually, this process involves explicit feature extraction, which may be very complicated and time-consuming.
An alternative approach is to use kernel methods to partly avoid these problems with the help of kernel functions. In this project, we will mainly focus on such kernel methods by looking into the paper "Text Classification using String Kernels"[1] and other relevant articles.
Specifically, we will first introduce the methods in the paper, then re-implement the core ideas(SSK, approximation of SSK) and evaluate them using standard machine learning criteria. Key implications of approximations will be introduced with detail. Comparison between the proposed kernel(SSK) and other traditional methods(NGK ,WK) will also be made. Furthermore, we go a little beyond things mentioned in the paper. We will implement another kernel called semantic kernel and evaluate its performance on the same data set.
Finally, by combining and evaluating results of different kernels, we can get a better understanding of kernel methods in the world of text classification.

2 State of the art in string kernels for text classification

Traditional learning algorithms like naive Bayes classifier, decision trees and k-nearest neighbors usually operate on input data by mapping documents into feature vectors. However, it is not a easy task to describe the explicit feature vectors in the problem of text classification. In this case, it is really important to propose an efficient feature vector extractor, whose complexity directly determines the complexity of the entire problem[1].
Since Lodhi proposing string kernels in 2002, a lot of papers have been employing string kernels as the efficient alternative to extract feature vector. We are familiar with kernel method as it provides a way of nonlinear classification without high computational cost in Support Vector Machine (SVM). In the kernel-based SVM, the kernel functions perform scalar products in a virtual high-dimensional space. Similarly, string kernels represent a way of using information at the character level by measuring the similarity of strings[2].
Recently, methods based on string kernels have demonstrated remarkable performance in the field of text classification, ranging from text categorization by topic to authorship attribution. In [1], Lodhi proposed a dynamic-programming-technique based string subsequence kernel (SSK), and obtained very good results for document categorization. Besides, Marius and Cristian treated texts as strings and used string kernels with kernel-based learning methods to complete the Traditional Authorship Attribution Task and the Sexual Predator Identification Task[3].
Surprisingly, string kernels provide applications not only in the field of text classification, but also in cross-domain settings without domain adaptation. In [4], Shervin and Aoife proposed a string kernel function to measure feature independence and analyzed its effectiveness on a standard Native Language Identification. As a result, their approach improved the accuracy by 28.3%. What's more, Radu and Andrei combined several kernels using multiple kernel learning for Arabic Dialect Identification and German Dialect Identification and got rank 1 in the ADI shared Task [5].
From these papers we can conclude that methods working at the word level or above very often restrict their feature space according to theoretical or empirical principles. However, using string kernels makes the corresponding learning method completely language independent, because the texts will be treated as sequences of symbols (strings) [6]. In Section 3, we will re-implement the proposed kernel for text, especially SSK and Approximate SSK.

3 Re-implementation of the proposed kernel for text

3.1 String subsequence kernel

3.1.1 Methods

In this section, we are going to re-implement string subsequence kernel(SSK) and combine it with SVM to classify the text provided. Since direct computation of the SSK involves $O(|\Sigma|^n)$ time and space complexity, we choose to use the recursive computation method to calculate SSK. For two strings s and t, we need to initialize two sets of matrix: $K'(K, M, N)$ and $K''(K, M, N)$ where K is the substring length, M is the length of s+1, N is the length of t+1. After setting the initial states, we should iteratively calculate K'' and K', at each position, we calculate K'' first:

$$\begin{cases} K_i''(sx, tu) = \lambda K''(sx, t), & x \neq u \\ K_i''(sx, tu) = \lambda(K''(sx, t) + \lambda K_{i-1}'(s, t)), & x = u \end{cases} \tag{1}$$

where u and x represent just one character. Next we will use the result to calculate K'. If we want to compute $K_n(s, t)$, we need to compute all $K_i'(s, t)$ from $i \in [1, n-1]$:

$$K_i'(sx, t) = \lambda K_i'(s, t) + K_i''(sx, t) \tag{2}$$

Now we have calculated all K', we can follow the recursive formula to get the $K_n(s, t)$ value we need.
The last thing is that we normalise the SSK so that we can effectively remove the bias generated by the length of documents, which is:

$$\hat{K}(s, t) = \frac{K(s, t)}{\sqrt{K(s, s)K(t, t)}} \tag{3}$$

Through this efficient computation method, we successfully reduce the complexity from $O(|\Sigma|^n)$ to $O(n|s||t|)$, and we are able to calculate the SSK of the our Reuter dataset in 2-12 hours depending on the different training sets we choose and the different substring length k.

3.1.2 Efficient Implementation

We try to improve the efficiency of calculating the SSK by saving the K' and K'' matrices. When we are using the recursive form to calculate the SSK between two strings s and k under the same λ value and different k values, it's very inefficient because of its repeated calculation of these two matrices. For example, after we successfully calculate the SSK with $k = 4$, and we want to calculate SSK with $k = 5$, the only difference is we need to calculate K_4' and K_4'', so we don't need to re-calculate $K_{1,2,3}'$ and $K_{1,2,3}''$ if we have saved these matrices before. This method only applies when we are going to vary k under the same λ value.
We also use parallel computing when calculating the alignment scores in section 3.2. It's extremely slow when we calculate $\langle K_1, K_2 \rangle_F$. Since it's independent for computing each elements of $\langle K_1, K_2 \rangle_F$, we use the parallel computing function *Parallel* by setting n_jobs = 2 from the python library *joblib* to do the calculation.

3.1.3 Experimental Results

We implemented SSK method by choosing 380 training documents and 90 testing documents from Reuter dataset with four different categories: "earn", "acq", "crude" and "corn". Part of the results

of SSK with different data sets, length and λ are shown in Table 1 and 2. In Table 1, the value of weight decay parameter λ was set to 0.5, and we compared the results between original results in [1] and our re-implementation results. In Table 2, the value of sub-sequence length n was set to 3, and we obtained our re-implementation results.

Category	Length	Original Results			Re-implementation		
		F1	Precision	Recall	F1	Precision	Recall
earn	3	0.925	0.981	0.878	0.958	1.000	0.920
	4	0.932	0.992	0.888	0.961	1.000	0.925
	5	0.936	0.992	0.888	0.974	1.000	0.950
acq	3	0.785	0.863	0.724	0.889	0.828	0.960
	4	0.822	0.898	0.760	0.889	0.828	0.960
	5	0.867	0.914	0.828	0.873	0.800	0.960
crude	3	0.881	0.931	0.853	0.909	0.833	1.000
	4	0.905	0.980	0.853	0.968	0.938	1.000
	5	0.936	0.979	0.900	0.933	0.933	0.933
corn	3	0.665	0.940	0.540	0.750	1.000	0.600
	4	0.783	0.924	0.690	0.889	1.000	0.800
	5	0.779	0.886	0.700	0.824	1.000	0.700

Table 1: Partial Results: Classification performance of different SSK subsequence length n

3.2 Approximate SSK

3.2.1 Methods

Although we can improve the efficiency of SSK using recursive algorithm, it still takes a lot of time. Therefore, an approximate version of SSK was implemented so that we can further improve the efficiency of our code. Mainly, there are 3 steps to implement the approximate SSK. First, based on the original article, we should extract features from the whole dataset, that is, we enumerate all the possible combination of contiguous strings of length k and then sort these strings by their frequencies in the dataset.

Second, we select first n strings (most frequently, we also try most infrequently and randomly) and calculate the ssk between each data sample(totally N samples) and these strings by using equation (3). Then, we will get a N $\times$ n matrix.

Finally, by using the equation(4) we can efficiently compute our gram matrix.

$$K(x, z) \approx \sum_{s_i \in \hat{S}} K(x, s_i) K(z, s_i) \tag{4}$$

Now we have two gram matrices: one contains the true SSK values and the other contains the approximate values. We can compare the similarity of these two matrices, and the definition of alignment score is shown in equation (5). The alignment score measures the similarity of two kernel matrix K_1 and K_2 where $\langle K_1, K_2 \rangle_F = \sum_{i,j=1}^{m} K_1(x_i, x_j) K_2(x_i, x_j)$.

$$\hat{A}(S, K_1, K_2) = \frac{\langle K_1, K_2 \rangle_F}{\langle K_1, K_1 \rangle_F \langle K_2, K_2 \rangle_F} \tag{5}$$

Category	λ	Re-implementation		
		F1	Precision	Recall
earn	0.01	0.923	0.947	0.900
	0.05	0.892	0.860	0.925
	0.5	0.902	0.881	0.925
acq	0.01	0.793	0.697	0.920
	0.05	0.792	0.750	0.840
	0.5	0.830	0.786	0.880
crude	0.01	0.889	1.000	0.800
	0.05	0.857	0.923	0.800
	0.5	0.774	0.750	0.800
corn	0.01	0.824	1.000	0.700
	0.05	0.750	1.000	0.600
	0.5	0.571	1.000	0.400

Table 2: Partial Results: Classification performance of different SSK decay λ

3.2.2 Experimental Results

In the re-implementation of Approximate SSK, we conducted three different experiments.
For first experiment part, only 380 documents were used. The results of different feature selection strategies (most frequent, infrequent, random) are shown in Figure 1 (the parameters for kernels are: n = 5 and $\lambda = 0.5$). The figure reflects that the length of features greatly impacts the gram similarity between Approximate SSK and true SSK.
For second experiment part, we also chose the pre-processed Reuter dataset with the SVM functions imported from scikit-learn library. Table 3 shows the f1 scores for the 4 categories in the dataset. We randomly chose 5000 samples for training and 500 samples for testing, and compared with the original article, the results of "earn" and "acq" are almost the same because these two categories have most samples and their results are stable while "ship" and "corn" have less sample so that their results are not so stable.

	Original Result		Our Result	
	1000	3000	1000	3000
earn	0.97	0.97	0.95	0.97
acq	0.88	0.85	0.90	0.91
ship	0.1	0.53	0.36	0.43
corn	0.15	0.65	0.10	0.36

Table 3: F1 score of 4 Reuters categories

For third experiment part, we chose the first 1000 features of the strings, the result is shown in table 4. The results of approximated SSK are Compared with WK and NGK, where WK is a linear kernel that measures the similarity between documents that are indexed by words with tfidf weighting scheme. Similarly NGK is also a linear kernel that return a similarity score between documents that are indexed by n-grams [1].

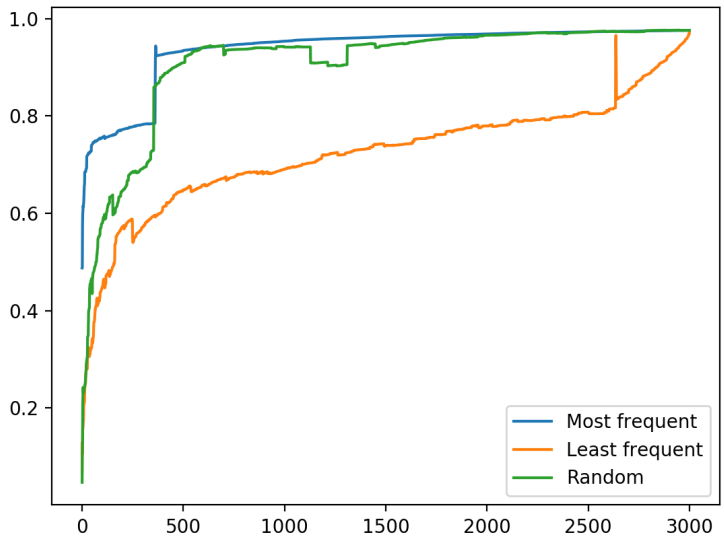

Figure 1: Alignment scores of Approximate SSK and True SSK

Category	WK	NGK			Approximated SSK		
		n			k		
		3	4	5	3	4	5
earn	0.88	0.89	0.89	0.89	0.97	0.99	0.99
acq	0.69	0.76	0.80	0.75	0.91	0.96	0.96
money-fx	0.72	0.71	0.72	0.63	0.6	0.55	0.56
grain	0.89	0.87	0.87	0.75	0.78	0.76	0.72
crude	0.80	0.77	0.87	0.71	0.87	0.86	0.71
trade	0.71	0.69	0.70	0.61	0.87	0.86	0.84
interest	0.70	0.72	0.76	0.56	0.80	0.78	0.72
ship	0.82	0.62	0.58	0.30	0.36	0.11	0.0
wheat	0.83	0.76	0.72	0.64	0.75	0.76	0.80
corn	0.75	0.84	0.89	0.75	0.36	0.40	0.56

Table 4: F1 scores for SVM with WK, NGK and SSK for top-ten Reuters categories.

4 Extension: Semantic Kernel

The traditional word kernel classifier ignores the semantic contents of the text. In this section, we adopt semantic kernel[7] which takes the semantic similarity between words into consideration.
One core aspect here is to measure how closely two words are related. For example, if the word "society" is highly present while the words "people" and "country" are less, the application of semantic matrix will increase the values of the last two terms because "society", "people" and "country" are strongly related concepts. We use the Wordnet to measure such similarity.
By use of semantic kernel, the feature vector of text x is

$$\hat{\phi}(x) = S^T \phi(x)$$

where $\phi(x)$ is the feature vector of a text in the context of word kernel and S represents the semantic matrix.
Then the semantic kernel can be written as

$$\hat{k}(s,t) = \phi(s)^T S S^T \phi(t) = \hat{\phi}(s)^T \hat{\phi}(t)$$

Regarding S, we have

$$S = RP \tag{6}$$

where
R is a diagonal matrix giving the frequency of the word in the text.
P is Term-proximity matrix measuring semantic similarity between words.

When it comes to implementation in Python, we use Wordnet library to measure such similarity. Since calling relevant function is quite time-consuming, when training and testing, we choose 120 training samples and 48 test samples(30 training samples and 12 test samples for each among categories: earn, acq, crude, corn) to test the performance of the semantic kernel based on SVM. The results are listed below.

	precision	recall	F1 score
earn	0.56	0.83	0.67
acq	0.20	0.17	0.18
crude	0.38	0.42	0.40
corn	0.71	0.42	0.53

Table 5: Performance of 4 Reuters categories using Semantic kernel

5 Discussion

In general, the results of re-implementation are similar to the result presented in the paper.
As shown in table 1 and 2, in SSK part, slight discrepancies exist because the contents of data sets used may be a little different. We ran each experiment only once but the original paper ran each experiment multiple times. The test set was sampled randomly and thus the content of our training sets and test sets are different from which were used in the original paper.
Also, when comparing the effect of varying λ, we ran the experiment with $length = 3$ while in the original paper $length = 5$. This is one of the probable reasons why our results are slightly different from the paper.
It seems that the data is not enough for us to draw some explicit conclusions of how length and λ

affect the performance of the kernel.
In approximation of SSK, we obtained a figure of alignment scores similar to the picture presented in the paper. That is, the alignment score rises as the number of features chosen grows. Choosing the most frequent features tends to give the highest alignment score, while choosing the least frequent features tends to give the lowest alignment score. The alignment score grows in a logarithmic manner, it grows quickly when the number of features is small, but becomes slower as the number grows.
Table 3 and 4 told us something about the approximation of SSK. In table 3, the results are again similar to the original results. The discrepancy again can be explained by random choices of data sets. What's more, strategy of feature selection may also affect the result.
Finally, in table 4, we can see that when compared with WK and NGK, approximated SSK has advantages on some categories like 'earn', 'acq', 'crude', 'trade' and 'interest'.

We implemented a kernel(semantic kernel) other than kernels mentioned in the original paper, and we conducted tests on it.
As it is shown in Table 3, the performance of semantic kernel is not as ideal as we expected, especially for the category 'acq'.
One possible reason may be that insufficient training samples were used in the experiment and thus under-fitting may happen. As mentioned in the previous section, due to the time-consuming property of calling similarity-evaluation function, we only choose 30 texts for each category when training the SVM. Another reason is that the semantic kernel may not be suitable to be applied on Reuters dataset. As semantic information in this dataset may not be as much as we thought.
However, in the category 'corn', the F1 score of semantic kernel is higher than approximated SSK, this at least shows that semantic kernel has some advantages when applied to some data sets.

References

[1] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins, "Text classification using string kernels," *Journal of Machine Learning Research*, vol. 2, no. Feb, pp. 419–444, 2002.

[2] R. T. Ionescu and A. M. Butnaru, "Transductive learning with string kernels for cross-domain text classification," in *International Conference on Neural Information Processing*, pp. 484–496, Springer, 2018.

[3] M. Popescu and C. Grozea, "Kernel methods and string kernels for authorship analysis.," in *CLEF (Online Working Notes/Labs/Workshop)*, Rome, 2012.

[4] S. Malmasi and A. Cahill, "Measuring feature diversity in native language identification," in *Proceedings of the Tenth Workshop on Innovative Use of NLP for Building Educational Applications*, pp. 49–55, 2015.

[5] R. T. Ionescu and A. Butnaru, "Learning to identify arabic and german dialects using multiple kernels," in *Proceedings of the Fourth Workshop on NLP for Similar Languages, Varieties and Dialects (VarDial)*, pp. 200–209, 2017.

[6] R. T. Ionescu, M. Popescu, and A. Cahill, "String kernels for native language identification: Insights from behind the curtains," *Computational Linguistics*, vol. 42, no. 3, pp. 491–525, 2016.

[7] G. Siolas and F. d'Alché Buc, "Support vector machines based on a semantic kernel for text categorization," in *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, vol. 5, pp. 205–209, IEEE, 2000.