KTH Royal Institute of Technology in Stockholm

DD2434: Advanced Machine Learning

Group Project

A study and reimplementation of the original article:

String subsequence kernel (SSK) for text classification and its approximation

---

Authors:
Thony Price
Erik Söderbeg
Gustav Pettersson
Eduardo Horst Morales
Christoffer Jansson

Supervisor:
Prof. Pawel Herman

January 21, 2018

Abstract

This report builds upon an earlier paper: String subsequence kernel (SSK) for text classification and its approximation [1]. We have made a close study of the original report and by our own implementation reproduced some of the results which consist of text classification on the dataset Reuters-21578 [4] using different kernels for text classification. In particular, the report focuses on describing the SSK and an approximation of the SSK kernel presented in the original paper but also presents the fundamentals for two other text classification kernels, n-grams, and word kernel.

The report contains results of testing each kernel as implemented by the authors. The SSK and approximated SSK was found significantly computational heavy and therefore were not run on the same amount of data as the original report or the other kernels making it hard to compare its performance against those presented in the original article. The n-grams and word kernel were run successfully and match results from the original article.

1 Introduction

1.1 Introduction of the SSK

SSK stands for string subsequence kernel and measures the similarity between two text strings. The SSK kernel takes the dot product between two feature vectors and since the result is the dot product, it is also a measure of similarity between them. The essence of the SSK is producing the feature vectors of the two strings which are compared. The feature extraction involves a couple of steps:

1. Filter the strings of all strange characters, punctuation, comma and so on. Also, filter the text from stop words, words which are non-descriptive: 'as', 'and', 'so' and many more. Lastly, all large characters should be converted to small characters.

2. An alphabet Σ must be chosen. The alphabet should at least contain all the letters of the string and the empty character (space). Normally if the text is written in English the alphabet consists of the 26 characters in the English alphabet plus the empty character.

3. A length parameter n and decay factor λ is chosen.

4. All subsequences s_i of length n from the alphabet represents an entry of the feature vector. The s_i are used to calculate the value of the entry it represents.

5. The value for each one of these entries is computed for both stings. The value is computed by, for each subsequence s_i, try to find that subsequence in the string. The subsequence must not be continuous, but the value depends on the compactness and frequency of the subsequence. The exact value for the entry of s_i is calculated by taking the length of every subsequence (continuous or non-continuous) of s_i in the string and for every one of them take λ to the power of the length (number of characters in string between first and last letter in the string for s_i). Then all of the values for the subsequence s_i in the string are summed to get the final value for the entry of s_i in the feature vector. The feature extraction can be hard to get and the following example will hopefully give a better understanding of how it works.

Example:

Suppose we have two strings $s =$ *"Good morning!"* and $t =$ *"Good evening!"*. To calculate the SSK-distance between these two strings one first have to filter the strings.

1. $S_{new} =$ "good morning" and $T_{new} =$ "good evening"

Secondly an alphabet Σ must be chosen for the two strings, for demonstrative purposes we will be working with an alphabet only consisting of the characters from the two strings.

2. Σ = ['d', 'e', 'g', 'i', 'n', 'm', 'o', 'r', 'v', ' ']

Thirdly all possible sequences of length n, in this example $n = 2$, should be formed. Where each subsequence will represent an index for the feature vector.

3. The subsequences of length 2 are: ['de', 'dg', 'di', 'dn', 'dm', 'do', 'dr', 'dv', 'd ', 'ed', 'eg', 'ei', 'en', 'em', 'eo', 'er', 'ev', 'e ', 'gi',...] the number of entries in the vector will be the number of characters in the used alphabet powered to n (2 for the example).

In the fourth step the value for each index in the feature vector should be calculated. The value for the fourth index in the feature vector (represented by "dn") for string s is calculated the following way.

4. In s there is only one subsequence for "dn" (good morning) the length of this subsequence is 5 and that will give the value of the index to be λ^5. If the string has contained more subsequences of "dn" the value of these would be summed together.

The formal definition of the SSK kernel presented in the report is the following (The definition is taken directly from the report)[1]:

Definition 1 (String subsequence kernel- SSK) *Let Σ be a finite alphabet. A string is a finite sequence of characters from Σ, including the empty sequence. For strings s, t, we denote by $|s|$ the length of the string $s = s_1 \dots s_{|s|}$, and by st the string obtained by concatenating the strings s and t. The string $s[i : j]$ is the substring $s_i \dots s_j$ of s. We say that u is a subsequence of s, if there exist indices $\mathbf{i} = (i_1, \dots, i_{|u|})$, with $1 \leq i_1 < \dots < i_{|u|} \leq |s|$, such that $u_j = s_{i_j}$, for $j = 1, \dots, |u|$, or $u = s[\mathbf{i}]$ for short. The length $l(\mathbf{i})$ of the subsequence in s is $i_{|u|} - i_1 + 1$. We denote by Σ^n the set of all finite strings of length n, and by Σ^* the set of all strings*

$$\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n. \tag{1}$$

We now define feature spaces $F_n = \mathbb{R}^{\Sigma^n}$. The feature mapping ϕ for a string s is given by defining the u coordinate $\phi_u(s)$ for each $u \in \Sigma^n$. We define

$$\phi_u(s) = \sum_{\mathbf{i}:u=s[\mathbf{i}]} \lambda^{l(\mathbf{i})}, \tag{2}$$

for some $\lambda \leq 1$. 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

$$\begin{aligned} K_n(s,t) &= \sum_{u \in \Sigma^n} \langle \phi_u(s) \cdot \phi_u(t) \rangle = \sum_{u \in \Sigma^n} \sum_{\mathbf{i}:u=s[\mathbf{i}]} \lambda^{l(\mathbf{i})} \sum_{\mathbf{j}:u=t[\mathbf{j}]} \lambda^{l(\mathbf{j})} \\ &= \sum_{u \in \Sigma^n} \sum_{\mathbf{i}:u=s[\mathbf{i}]} \sum_{\mathbf{j}:u=t[\mathbf{j}]} \lambda^{l(\mathbf{i})+l(\mathbf{j})}. \end{aligned}$$

The time complexity of this naive implementation of the SSK kernel is $\mathcal{O}(|\Sigma|^n)$. Instead of the naive implementation a Dynamic programming approach can be taken which reduces the time complexity of the kernel to $\mathcal{O}(|doc1||doc2|n)$, where $|doc|$ is the length of the document.

1.2 SSK Approximation

In the report, it is shown that the SKK kernel, in general, gives good results, but a big flaw with the kernel is the time complexity presented above, $\mathcal{O}(|doc1||doc2|n)$. Therefore an approximation method is needed in order to be able to use the kernel on larger text documents.

The approximation method for the SSK presented in the report is based on a special case of an empirical kernel map. Which says that:

$$K(x,z) = \frac{1}{C}\sum_{s_i \in S} K(x,s_i)K(z,s_i)$$

The above definition is true if the number of elements of the set $S = s_i$ is the same as the dimensionality of the space F, where F is the feature space mapping:
$\emptyset$: X $\rightarrow$ F, and $K(s_i, s_j) = 0$ for $s_i \neq s_j$ (the vectors $\emptyset(s_i)$ are all orthogonal).

If the cardinality of $\tilde{S} \subseteq S$ is less than the dimensionality of F or the vectors $\emptyset(s_i)$ are not fully orthogonal the above kernel formulation can still be used to approximate the true kernel and the following approximation can be constructed.

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

In the approximation presented in the report, a set $\tilde{S} \subseteq S$ was selected where the number of elements of the set was less than the dimensionality of F. However, all the vectors $\emptyset(s_i)$ of $\tilde{S}$ were selected to be orthogonal.

The selection of s_i was done by enumerating all possible continuous substrings of length n (n is the same n as used in the SSK kernel) in the documents. The top continuous substrings were selected as the different s_i which made up the new set $\tilde{S}$. All the vectors $\emptyset(s_i)$ and $\emptyset(s_j)$ (for $i \neq j$) are orthogonal since there is only one possible substring of length n of the string s_i which is s_i and the same for s_j, so the kernel between the two will always be zero.
If the set $\tilde{S}$ is the same set as S then the original empirical kernel map is true and the value of the approximation of the SSK will be the same as the original kernel value for the SSK. The time complexity for the approximation is reduced to $\mathcal{O}(|doc1|n|\tilde{S}| + |doc2|n|\tilde{S}|)$ ($|\tilde{S}|$ is the number of elements in the set $\tilde{S}$), provided that the features have been found.

1.3 Other text classification kernels

Along with the SSK kernel proposed in the original article, there are two more state-of-the-art kernels for text classification described. These are used to enable comparison of the SSK to other available methods, these are:

1.3.1 N-grams

N-grams are contiguous sequences of N characters made from some longer sequence. For example, "Dog" would become "D" "Do" "og" "g" with 2-grams. Including the first and last characters with "empty" characters is done to ensure that they are all weighted equally. Otherwise, characters in the middle would be weighted more heavily since they occur in more subsequences.

The idea behind n-grams is that if two documents share many subsequences then it is likely that their topics are similar.[3] This proved to be a fine idea since n-grams have shown to give surprisingly good results for such a rather simple method. Since n-grams are just a way of transforming a text into a vector, one can use several different ways to measure the similarity between texts. For example, one can use the cosine similarity by counting occurrences of each n-gram in the texts or one could simply take the percentage of n-grams they have in common or one could come up with some entirely other way to compare them. Additionally, since n-grams only consider the symbols making up the document rather than the actual language, it can be applied to other sequences like DNA sequences or protein sequences.

1.3.2 Word kernel

The word kernel was originally presented in a report by Thorsten Joachims [2] for text classification. The approach is simple, the feature vector is shaped by word frequency in the texts that are to be classified.

Each distinct word, w_i, corresponds to a feature with the number of times the word w_i occurs in the document as its value. The WK implementation presented in [1] weights the values slightly different as $log(1 + tf) * log(n/df)$. Here tf represents term frequency while df is used for document frequency and n is the total number of documents. The last factor takes into account how much information a word holds. If a words occurs in almost all documents $df \to n$ and at the same time $log(\frac{n}{df}) \to 0$ and renders the term "useless" for comparison. The feature vectors are normalized in order to remove bias by document length.

The WK inherently produces very large feature vectors where the length corresponds to the number of unique words in all of the documents. In similarity with the previously discussed kernels, WK benefits from some preprocessing of the data like removal of stop words and semantic notation like 'and',' 'or', '!', '?', etc. One could also only consider Words as features only if they occur in the training data least 3 times. Lastly, Stemming or lemmatization, this would, for example, result in 'friends' and 'friendly' both be replaced with 'friend', and achieve a more fair count of words that are essentially the same.

1.4 Our objectives

Our goal is to implement the three kernels described above, SSK, NGK, and WK. We aim to test them on the same dataset that is used in the original article. Our findings are presented in the *Results* section.

2 Methods

2.1 SVMs

Support Vector Machine(SVM) is a supervised learning technique. SVMs are a good approach when we want to classify data but we do not specifically know how to relate the data to the classes. SVMs advantages are its optimization view of the margins. They construct a boundary to separate classes and try to find the best separator, which looks to get the largest distance between the margin and its nearest classes samples. Another great feature about SVMs is their ability to convert or translate data into high dimensional spaces. This is called the "kernel trick". The reason behind this translation is that there will be times when the data will not be linearly separable and thus we would require a higher dimensional separator. Both [1] and [2] agree on the strength of approaching text classification with SVM:s is their built-in independence of the dimensionality of feature space. This comes well in hand because text classification seemingly brings high dimensionality.

2.2 Implementation

All above kernels were reimplemented from scratch using python with its included packages along with the following external packages:

- Numpy, to enable more efficient vector and matrix operations.
- Nltk (Natural language toolkit) to enable more efficient tokenization and removal of stop words.
- Cvxopt, to work with an efficient solver in our SVM.

The Reuters dataset [4] was available in the Nltk package and from that, we were able to extract a training and test dataset distributed the same way as the *one* dataset used in the original article. Namely the one on the smallest dataset they performed tests on. We used each kernel to train and classify that data on the same category as in the original report to provide a side by side comparison.

3 Results

The measurements used to evaluate the results are accuracy, precision and recall. The F1 score used in the article[1] is a combination of precision, and recall.

$$accuracy = \frac{(true\,positive + true\,negative)}{(true\,positive + true\,negative + false\,positive + false\,negative)}$$
$$precision = \frac{(true\,positive)}{(true\,positive + false\,positive)}$$
$$recall = \frac{(true\,positive)}{(true\,positive + false\,negative)}$$
$$F1 = 2 * \frac{precision * recall}{precision + recall}$$

We used a subset of the Reuters dataset for training and testing. The n-gram and word kernel had the proportions between the categories of the articles as follows: 152 earn, 38 corn, 114 acq and 76 crude docs where used to train the SVM. Docs for testing; 40 earn, 10 corn, 25 acq and 15 crude. This was the same split that was used in the article[1]. For the SSK we used 4 earn, 4 corn, 4 acq and 4 crude docs for training and 2 earn, 2 corn, 2 acq and 2 crude for testing. with a lambda of 0.5. The approximated SSK was trained on 20 of each type earn, corn, acq, crude and tested on 5 docs from each category.

Category	Kernel	Length	Accuracy	Precision	Recall
earn	SSK	3	0.875	1.0	0.5
earn	ASSK 20 features	3	0.8	1.0	0.2
earn	NGK	3	0.933	1.0	0.85
earn	WK	-	0.944	1.0	0.875

Table 1: the best results from each kernel

Length	F1		Precision		Recall	
	mean	std	mean	std	mean	std
3	0.965	0.034	0.976	0.034	0.958	0.058
4	0.97	0.024	0.985	0.021	0.958	0.043
5	0.968	0.024	0.983	0.024	0.955	0.042
6	0.97	0.022	0.99	0.017	0.953	0.042
7	0.964	0.024	0.99	0.017	0.94	0.047
8	0.947	0.034	0.987	0.022	0.913	0.06
10	0.939	0.034	0.99	0.019	0.895	0.063
12	0.924	0.041	0.989	0.019	0.87	0.075
14	0.904	0.051	0.994	0.012	0.833	0.086

Table 2: NGK with stop words and punctuation removed and averaged over 10 runs.

4 Discussion

What is striking in the results is the significantly smaller dataset used on the SSK kernel. The reason for this was due to the high time complexity of that kernel, any dataset larger than that would have required an unreasonable amount of time to run on the hardware available to us. With that in mind we believe it

Category	Kernel	Length	F1		Precision		Recall	
			Mean	SD	Mean	SD	Mean	SD
earn	SSK	3	0.925	0.036	0.981	0.030	0.878	0.057
		4	0.932	0.029	0.992	0.013	0.888	0.052
		5	0.936	0.036	0.992	0.013	0.888	0.067
		6	0.936	0.033	0.992	0.013	0.888	0.060
		7	0.940	0.035	0.992	0.013	0.900	0.064
		8	0.934	0.033	0.992	0.010	0.885	0.058
		10	0.927	0.032	0.997	0.009	0.868	0.054
		12	0.931	0.036	0.981	0.025	0.888	0.058
		14	0.936	0.027	0.959	0.033	0.915	0.041
	NGK	3	0.919	0.035	0.974	0.036	0.873	0.062
		4	0.943	0.030	0.992	0.013	0.900	0.055
		5	**0.944**	0.026	0.992	0.013	0.903	0.051
		6	0.943	0.030	0.992	0.013	0.900	0.055
		7	0.940	0.035	0.992	0.013	0.895	0.064
		8	0.940	0.045	0.992	0.013	0.895	0.063
		10	0.932	0.032	0.990	0.015	0.885	0.053
		12	0.917	0.033	0.975	0.024	0.868	0.053
		14	0.923	0.034	0.973	0.033	0.880	0.055
	WK		0.925	0.033	0.989	0.014	0.867	0.057

Figure 1: Results from the original report

is fair to say it performed well on such a small dataset and in that sense confirms both the statement of good performance in the original article and also the need for an approximation of the SSK.

With the approximation of the SSK, it was possible to use a slightly larger dataset. Changing the parameters cost, length and features did not give any significant changes in our tests. To see any significant effect of changing the parameters it is probable that a larger dataset needs to be used. The approximation SSK got 16 out of 20 classifications correct, so it works, but it was still too slow to run tests on large datasets.

We can see that the performance of the NGK in Figure 2 are close to those of the original report. This gives us confidence that the SVM is working properly and that our results can be trusted.

The differences that do exist, we believe come from the differences in documents selected. Since while we have the same proportions, we will not have exactly the same documents. In addition, we cannot be certain that Lodhi et al. used Cosine Similarity for measuring the similarity between the documents, if they are not, the differences could come from that.

The performance of WK was close to the measurements in the original report. The slightly higher performance may come from us only running the classification one time (because very computational heavy) and therefore gives an unfair comparison to the report which is an average over 10 runs.

One especially interesting about the WK, and also pointed out in [1], it that though this approach loses word order information only retaining the frequency of the terms in the document it performs surprisingly well compared to the NGK and SSK (considering the small dataset) that both keeps the information gained by ordering in the document.

5 References

References

[1] H. Lodi et al. Text classification using string kernels. *Journal of Machine Learning Research*, 2(2):419–444, 2002.

[2] T. Joachims. Text categorization with support vector machines. 1998.

[3] Grzegorz Kondrak. N-gram similarity and distance. In *String processing and information retrieval*, pages 115–126. Springer, 2005.

[4] D. Lewis. Reuters-21578. 1987.