Machine Learning Advanced course
String Subsequence Kernel

Erik Rosvall, Viktor Karlsson, Lucas Grönlund & Marcus Alsterman

January 21, 2018

1 Introduction

In the area of machine learning, classification of unseen data is of great interest. For this to be possible feature vectors of the input data are needed which is trivial for some types of data, but not so much for others. Text documents is an example of one such kind. Transforming the data into a *feature space*, by quantifying some aspect of the document, and then compare similarity between data points through their inner product, called the *kernel method*. Using the '*kernel trick*' for the *Support Vector Machine* (SVM) classifier the explicit transform into this feature space can be avoided which in many cases is what enables the method to be used since the feature spaces can be of very high dimensionality, in some cases even infinite.

The paper [1] presents a new method of performing the feature extraction from text documents which Lohdi et al. names *String Subsequence Kernel* (SSK). The SSK is compared with two other kernel feature extraction methods, namely *Word Kernel* (WK) and *N-Gram Kernel* (NGK). Kernels are not the only successful type of method in the text classification field, neural networks (Convolutional Neural Networkds (CNN) and Recurrent Convolutional Neural Networks (RCNN)) as well as a *fastText* have shown great results. These are regarded as state of the art algorithms and will be discussed in comparison to the kernels methods presented above.

Throughout [1] the dataset used for classification was the *Reuters dataset* consisting of news articles from Reuters news agency. *Modified Apte*, also known as ModeApte, was used to split this data set into 7769 training documents and 3019 test documents, divided into 90 classes. These classes are however not mutually exclusive which is common for text data, a document can for example be tagged with both 'earn' and 'acquisition' which is an extension making the classification more involved.

The aim of this report is to replicate the results in [1] regarding the SSK, where the emphasis is put on the approximative implementation of their kernel method. We have therefore used Python packages as `Sklearn` and `Numpy` for supplying SVM algorithms and other conveniences, but written code for all three kernels ourselves. Further, we have put no effort into replicating the results regarding the combination of kernels Lohdi et al. presentes since for one it did not improve the classification performance in any significant way. Finally, we also propose and test a possible computational complexity reduction of the SSK since this is one of the major downside with the algorithm.

2 Method

2.1 Support Vector Machine

The SVM is a binary classifier separating two classes of data with a maximum margin decision boundary. Since this is not of central interest in this article or [1], we suppose the details of this algorithm to be known. The algorithm can be formulated in a way such that the training features only occur in inner products with other training feature vectors, which often is called the *dual formulation*. When the problem is formulated in this way the kernel trick can be used, which lets us avoid having to explicitly transform our input data into the kernel space.

Since the Reuters dataset have text documents assigned to multiple classes, a problem often called *multi-label-* or *vector output classification*, the SVM cannot be used directly since it is a binary classifier. To deal with this issue we implemented a *one-vs-rest* approach. Thus, we train one classifier for each class with data from that class regarded as positive samples and the rest of the dataset as negative samples. This algorithm was imported from the `sklean` package in python.

2.2 String subsequence kernel

[1] proposes a feature extraction method for text documents, extending the n-gram approach to features also being non-contiguous substrings. The feature vectors for each document is thus created through first generating all such subsequences, from all documents in the dataset, of length k. Each such substring is regarded as a dimension of the feature space.

The transformation of a document to this feature space is then performed "simply" trough finding all occurrences of these subsequences and weighing them according to how compactly the sequence is embedded in the text. This is handled through the introduction of a *decay factor* $\lambda \in (0, 1)$, which lets us put smaller emphasis on less compactly embedded subsequences than ones more compactly embedded in the document. An example of this, presented in [1], is if we consider the subsequence **c-a-r** which is more compactly found in the document '**car**d' than in '**c**ust**ar**d', giving the former a value of λ^3 and the latter

λ^6, where the exponent directly corresponds to the length of the sequence for which the subsequence is contained.

To not introduce unnecessary confusion we will present the definition of a SSK exactly as the definition presented in [1].

2.2.1 Definition - String subsequence kernel

Let Σ be a finite alphabet. A string is a finite seqence 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 us a subsequence of s, if there exist indices $\boldsymbol{i} = (i_1, \dots, i_{|u|})$, with $1 \leq i_1 < \dots < i_{|u|} \leq |s|$, such that $u_j = s_{i_j}$, for $j = 1, \dots, |u|$, or $u = s[\boldsymbol{i}]$ for short. The length $l(\boldsymbol{i})$ 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 the 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

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

for some $\lambda \in (0, 1)$. These features measure the number of occurrences of subsequences in the string s weighting them according to their lenghts. 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

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

To prove our understanding of this definition we construct a unique example, which also illustrates how the feature vectors look. Consider two documents containing the words *fail* or *sail*.

	f-a	f-i	f-l	a-i	a-l	i-l	s-a	s-i	s-l	a-t	l-i	l-t	i-t
$\phi(fail)$	λ^2	λ^3	λ^4	λ^2	λ^3	λ^2	0	0	0	0	0	0	0
$\phi(sail)$	0	0	0	λ^2	λ^3	λ^2	λ^2	λ^3	λ^4	0	0	0	0

Table 1: Feature vectors for two simple documents.

To calculate the similarity between the documents *fail* and *sail* we simply calculate the inner product between their respective feature vectors, resulting in the kernel value $K_2(fail, sail) = 2\lambda^4 + \lambda^6$, where the subindex represent the subsequence length n. An other example which we later used to prove that our implementation of the SSK was correct was the kernel values [1] presents for the documents *'science is organized knowledge'* and *'wisdom is organized life'*. These were: $K_1 = 0.580$, $K_2 = 0.580$, $K_3 = 0.478, K_4 = 0.439, K_5 = 0.406, K_6 = 0.370$.

To explicitly calculate all the features for documents present in the Reuters dataset would require impractical amounts of computations, which is why [1] presents a recursive formulation for the kernel values. We will again present these as they were in the paper, since we followed them exactly in our implementation.

2.2.2 Definition - Recursive computation of the subsequence kernel

$$
\begin{aligned}
K'_0(s,t) &= 1, for\ all\ s,t \\
K'_i(s,t) &= 0, if\ \ \min(|s|,|t|) < i \\
K_i(s,t) &= 0, if\ \ \min(|s|,|t|) < i \\
K'_i(sx,t) &= \lambda K'_i(s,t) + \sum_{j:t_j=x} K'_{i-1}(s,t[1:j-1])\lambda^{|t|-j+2}, \qquad i = 1,\dots n-1 \\
K_n(sx,t) &= K_n(s,t) + \sum_{j:t_j=x} K'_n - 1(s,t[1:j-1])\lambda^2
\end{aligned}
$$

This recursive formulation leaves the computational complexity at $\mathcal{O}(n|s||t|^2])$ which can be improved using the efficient computation [1] presents, through first evaluating

$$K''_i(sx,t) = \sum_{j:t_j=x} K'_{i-1}(s,t[1:j-1])\lambda^{|t|-j+2}.$$

Observing that $K'_i(s,t)$ can be evaluated with the recursion:

$$K'_i(sx,t) = \lambda K'_i(s,t) + K''_i(sx,t)$$

one can then observe that $K''_i(sx,tu) = \lambda^{|u|}K''_i(sx,t)$, in the case that x does not occur in u, and otherwise

$$K''_i(sx,tx) = \lambda\left(K''_i(sx,t) + \lambda K'_{i-1}(s,t)\right).$$

With this implementation the computational complexity reduces to $\mathcal{O}(n|s||t|)$.

2.2.3 Approximating string subsequence kernel

To test SSK on the entire Reuters dataset an approximative approach is needed, we call this aSSK. This is enabled through the general empirical kernel map presented in [2] which states that if we have a set $S = \{s_i\}$ of vectors such that these vectors, when transformed into the kernel space, are both orthogonal and span the entire kernel space (that is $\mathcal{K}(s_i,s_j) = C\delta_{ij}$), we can write the kernel value for any other two feature vectors

$$\mathcal{K}(x,z) = \frac{1}{C}\sum_{s_i\in S}\mathcal{K}(x,s_i)\mathcal{K}(z,s_i) \tag{3}$$

If we instead of constructing a set S with full cardinality limit ourselves to only choosing a subset $\tilde{S} \subset S$ we can approximate the true kernel value with fewer computations. How 'close' this approximation $\mathcal{K}$ is to the true kernel matrix K can be measured through the alignment of the two matrices, defined as

$$A(K,\mathcal{K}) = \frac{\langle K,\mathcal{K}\rangle_F}{\sqrt{\langle\mathcal{K},\mathcal{K}\rangle_F\langle K,K\rangle_F}} \tag{4}$$

Here the subindex F indicates that the inner product is Frobenius normed defined as $\langle K_1,K_2\rangle_F = \sum_i\sum_j K_1(x_i,x_j)K_2(x_i,x_j)$.

To generate $\tilde{S}$ we followed [1], first choosing the substring length n, then counting all contiguous substrings of length n in all the documents and finally constructed $\tilde{S}$ with the x most occurring substrings. Using the alignment measure between the approximated kernel matrix and the exact one, we studied the the effect the number of features had on the approximation.

2.3 Proposed improvement of SSK

Despite both dynamic programming implementation and approximative methods for SSK, it still is has a high computational cost resulting in long training and testing times which in some cases renders it useless. We therefore propose a simple modification to the SSK in order to reduce both these times.

When searching for a subsequence in a large document, there are cases when the length of such non-contiguous sequence is very large. Since this length then appears as the exponent of the decay factor λ, the relevance of such a feature should approach zero when the length increases. We therefore introduced a *cut-off* length, limiting our search and thus removing the influence from long subsequences.

2.4 N-gram kernel

The n-gram kernel (NGK) transforms text documents to the feature space trough considering contiguous subsequences of length n, called *n-grams*. The value of each dimension in the feature space of a document is thus the number of occurrences of that n-gram in the document.

With this approach all information encoded in the word order in the document is lost. The method has nonetheless proven useful, as we will see later.

2.5 Word kernel

The word kernel (WK) transforms the documents to the feature space considering each word, a string separated by spaces and/or punctuation, as a feature. The value of each dimension in the feature space is calculated using a variant of *tf-idf*, $\log(1+tf)\cdot\log(n/df)$. Here tf is the term frequency and df is the document frequency while n is the number of documents.

2.6 Metric for performance

Three metrics were used in the comparison between the kernels performance; *precision*, *recall* and F_1-*score*. The precision for each class is the ratio between *the number of correctly classified documents belonging to this class* and *the total number of classified documents*, while recall for a class is the ratio between *the number of correctly classified documents in to that class* and *the total number of documents in that class*. The F_1-score is then calculated using these metrics, P for precision and R for recall, through

$$F_1 = 2\frac{PR}{P+R}$$

2.7 Actual experiment

With all algorithms and measures defined we can now turn to what results from Lohdi et al. we set out to replicate in more detail. First we studied how the sequence length impacted performance for both SSK and NGK while WK is from its definition not effected by this parameter. These trails were, in parallel with Lohdi et al., only performed on a subset of the data since even with the efficient implementation of the SSK, the algorithm is still computationally costly. The subset consisted of of 470 documents (380 train and 90 test) from 4 classes; acq., earn, corn and crude. The split of train (test) documents between the classes were as follows: acq. 114 (25), earn 152 (50), corn 38 (15) and crude 76 (10).

Secondly, we studied the SSK's performance when varying the decay factor λ, again using the data subset. $\lambda \in [0.01, 0.05, 0.1, 0.5, 0.7, 0.9]$ were the values used.

We then turned to the approximation, first studying how the number of vectors in the subset $\tilde{S}$ effected the alignment score between the approximate kernel matrix and the exact one. Then, using a suitable approximation of the kernel matrix, we compared the performance of the (approximative) SSK with both WK and NGK on the entire Reuters dataset. We varied the sequence length $n \in [3, 8]$.

2.8 Preprocessing of data

The Reuters dataset was preprocessed before training, through removing all stop words in each document. We used the python package `nltk` for the list of these non-informative words.

3 Results

The results are presented in figure 1 and tables 2, 3 and 4.

Due to very high computational costs for some of the algorithms, both in time and memory, we were not able to average over ten iterations as [1] did for every entry in the tables. **Where this was most of an issue...** We have clearly stated in the tables how many iterations were performed for that data.

NGK	n	Precision	Recall	F_1
acq	3	0.96	0.88	0.92
	4	0.90	0.89	0.89
	5	0.97	0.86	0.92
earn	3	0.97	0.93	0.95
	4	0.99	0.93	0.96
	5	0.99	0.89	0.93
corn	3	1	0.87	0.93
	4	1	0.64	0.78
	5	1	0.44	0.61
crude	3	0.90	0.90	0.90
	4	0.92	0.86	0.89
	5	1	0.73	0.84

(a) NGK performance for different subequence lengths n.

SSK	n	Precision	Recall	F_1
acq	3	0.93	0.96	0.94
	4	0.93	0.96	0.94
	5	0.97	0.93	0.95
earn	3	0.99	0.93	0.96
	4	0.99	0.95	0.97
	5	0.99	0.96	0.97
corn	3	0.97	0.87	0.91
	4	0.98	0.64	0.88
	5	0.98	0.44	0.83
crude	3	0.97	0.86	0.88
	4	0.98	0.80	0.91
	5	0.98	0.73	0.88

(b) SSK performance for different sequence lengths n. $\lambda = 0.5$ throughout.

WK	Precision	Recall	F_1
acq	0.974	0.930	0.951 (0.802)
earn	0.978	0.972	0.976 (0.925)
corn	0.992	0.867	0.923 (0.762)
crude	0.946	0.957	0.948 (0.904)

(c) WK performance results, averaged over 10 iterations. Numbers in parenthesis are reference value from [1]

SSK	λ	Precision	Recall	F_1
acq	0.05	0.960	0.944	0.952
	0.1	0.966	0.936	0.948
	0.5	0.921	0.925	0.922
earn	0.05	0.988	0.933	0.957
	0.1	0.978	0.959	0.968
	0.5	0.983	0.938	0.960
corn	0.05	1	0.880	0.937
	0.1	0.993	0.867	0.927
	0.5	0.992	0.812	0.890
crude	0.05	0.954	0.886	0.912
	0.1	0.975	0.912	0.939
	0.5	0.918	0.850	0.877

(d) SSK performance results with varying λ. Averaged over 10 iterations. $n = 5$ throughout.

Table 2: Results from SSK, NGK and WK using subset of Reuters dataset.

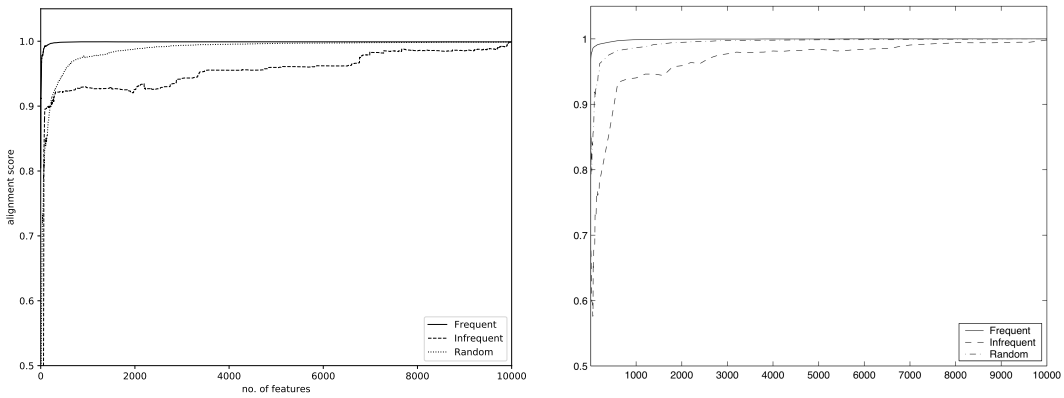
aSSK	$x = 1000$	$x = 3000$
acq	0.96 (0.88)	0.97 (0.85)
earn	0.98 (0.97)	0.98 (0.97)
ship	0.43 (0.10)	0.63 (0.53)
corn	0.84 (0.15)	0.89 (0.65)

Table 3: F1 performance of the approximative SSK using 1000 and 3000 vectors in $\tilde{S}$. $n = 5$ and $\lambda = 0.5$. Results from [1] is presented in parenthesis for comparison.

4 Discussion

4.1 Performance on subset of Reuters dataset

We can in table 5 and 6 see the same pattern between our data and that presetned in [1]; for higher values of the sequence length n we have a clear decrease in performance for both NGK and SSK. We



(a) Our results

(b) Lohdi et al.'s

Figure 1: Alignment scores for varying the number of vectors x in the subset $\tilde{S}$. Lohdi et al.'s figure is shown for comparison.

	WK	NGK $n=3$	NGK $n=4$	NGK $n=5$	aSSK $n=3$	aSSK $n=4$	aSSK $n=5$
earn	0.98	0.98	0.98	0.98	0.98	0.98	0.98
acq	0.97	0.95	0.95	0.96	0.95	0.95	0.95
money-fx	0.79	0.77	0.79	0.77	0.77	0.8	0.78
grain	0.92	0.81	0.83	0.82	0.84	0.86	0.8
crude	0.87	0.84	0.85	0.8	0.82	0.79	0.73
trade	0.8	0.73	0.77	0.77	0.72	0.79	0.77
interest	0.8	0.72	0.73	0.75	0.79	0.8	0.77
ship	0.78	0.66	0.55	0.47	0.69	0.5	0.34
wheat	0.79	0.86	0.84	0.8	0.8	0.8	0.76
corn	0.86	0.73	0.85	0.65	0.78	0.72	0.6

Table 4: The ten best performing classes from [1] shown with our F_1 results for WK, NGK and aSSK using the entire Reuters dataset. For aSSK; $\lambda = 0.5, x = 3000$

had to, as mentioned in the results, limit the number of iterations performed because of computational time, which is why we in table 2a and 2b only have averages over the three best performing values of n, 3, 4 and 5. What is worth mentioning here is that WK, in table 2c, performs consistently better than SSK, which only achieves tangential F_1-scores for two of the four classes. When then considering the computational time difference between the methods which with our implementation almost differed with two orders of magnitude in favor of WK, the practical limitation of SSK becomes apparent.

4.2 Approximating string subsequence kernel

Our implementation of the approximative SSK proved to outperform the results [1] presented, both for $x = 1000$ and $x = 3000$, see table 3. This might be due to the fact that we tuned our SVM's weighting factor C for misclassified data.

Our alignment scores shown in figure 1 also proves that already using just a few hundred vectors in $\tilde{S}$ essentially finds the same kernel matrix using aSSK as when using SSK.

4.3 Performance on entire Reuters dataset

Using the entire Reuters dataset to train WK, NGK and aSSK we find in table 4 we find similar results as when only using the subset; WK performs better than both NGK and aSSK in a fraction of the time, even though WK is implemented in Python while aSSK is implemented in C++ which generally is a much faster language.

As Lodhi et al. summarized the findings in [1], that for small text strings and small n will the SSK perform well, but quickly losses ground. This can be very logical, since when we get more data noise to signal ratio for words goes down and we can rely more on the words rather than some sequence of characters. More documents also makes the $tfidf$ transform better, since we get more data on which words contains information and which words are too frequent to offer distinguishing features. At the same time will **stuff** like NGK and the SSK hit more noisy information since more and longer text string are bound to hit more possible combinations. Another reason for why WK works well compared to SSK might be because of the nature of the dataset. Reuters journalists are proofread and held to a higher linguistic standard than say a chat log from some messenger program. The relative lower amount of spelling errors and stylistic stringency should have a positive effect on WK's performance relative SSK. Important to note when comparing our results to [1] is that we use a slightly smaller subset of Reuters than presented there. How [1] handled their data is not always clearly stated, so some assumptions have been made. In some cases this might provide a higher F_1 score than [**?**].

As with the report from Lodhi et al, we find that SSK is not particularly sensitive to values on the decay parameter λ, and as long as one avoids values close to one or zero the performance is generally very similar between classes.

Reuters, being a unbalanced dataset creates some issues for the SVM. The disparity between classes are big, sometimes thousands to one. We've tried to solve this problem with a variety of technique, but mostly trying to balance the data so that the SVM can find decision boundary that generalizes well. What we've found though is that we didn't manage to find appropriate parameters for larger n. Both NGK and SSK suffered at higher values of n and we could not replicate the performance mentioned by Lodhi et al. with many classes performing zero precision and recall. Time restrictions made tuning the parameters for the SVM to achieve acceptable results impossible for $n \in [10, 12, 14]$.

Looking at the present state of natural language processing we find that currently quite a lot of hype surrounds algorithms like Convolutional Neural Networks (CNN) and *fastText*. They work very differently, both from each other and SSK, but both perform very well. Use a deep network to classify text. Neural networks can be notoriously slow to train for large datasets, but generally performs very well across multiple classification and regression problems. *fastText* on the other hand i very fast, capable of training and classifying large datasets in seconds SOURCE. It uses language specific precomputed vectors to classify text, but has been shown to perform similarly to CNNs on some datasets. The main benefit is the drastically quicker computation time.

We looked into a method of speeding up the computations further without losing to much accuracy. Our experiments on the same four category subset of Reuters as used by Lodhi et al showed that we can achieve some reduction in time.

References

[1] Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. Text classification using string kernels. *Journal of Machine Learning Research*, 2(Feb):419–444, 2002.

[2] Bernhard Schölkopf. Support vector learning, 1997.

Appendix

SSK	n	λ	Precision	Recall	F_1
acq	5	0.01	0.96	0.92	0.94 (0.873)
		0.05	1	0.92	0.96 (0.882)
		0.1	1	1	1 (0.871)
		0.5	0.89	1	0.94 (0.867)
		0.7	0.93	0.96	0.94 (0.805)
		0.9	0.64	0.96	0.77 (0.735)
earn	5	0.01	0.95	0.95	0.95 (0.946)
		0.05	1	0.98	0.99 (0.946)
		0.1	0.98	1	0.99 (0.944)
		0.5	1	0.90	0.95 (0.936)
		0.7	1	0.90	0.95 (0.928)
		0.9	0.95	0.88	0.91 (0.914)
corn	5	0.01	1	0.80	0.89 (0.845)
		0.05	1	0.87	0.93 (0.834)
		0.1	1	0.87	0.93 (0.827)
		0.5	0.94	1	0.97 (0.779)
		0.7	1	0.80	0.89 (0.628)
		0.9	0.64	0.60	0.62 (0.348)
crude	5	0.01	1	0.70	0.82 (0.937)
		0.05	0.90	0.75	0.82 (0.945)
		0.1	1	0.91	0.95 (0.947)
		0.5	1	0.70	0.82 (0.936)
		0.7	0.90	0.90	0.90 (0.893)
		0.9	0.39	1	0.56 (0.758)

(a) Insert

NGK	n	Precision	Recall	F_1
acq	3	0.94	0.86	0.90 (0.791)
	4	0.98	0.89	0.93 (0.873)
	5	0.98	0.87	0.93 (0.882)
	6	0.97	0.79	0.87 (0.880)
	7	1	0.72	0.83 (0.870)
	8	1	0.73	0.84 (0.857)
earn	3	0.99	0.95	0.97 (0.919)
	4	0.99	0.96	0.98 (0.943)
	5	1	0.95	0.97 (0.944)
	6	0.99	0.93	0.96 (0.943)
	7	0.99	0.88	0.93 (0.940)
	8	0.99	0.88	0.93 (0.940)
corn	3	1	0.83	0.91 (0.797)
	4	1	0.64	0.78 (0.841)
	5	1	0.58	0.73 (0.847)
	6	1	0.42	0.59 (0.815)
	7	1	0.33	0.49 (0.767)
	8	0.75	0.14	0.22 (0.706)
crude	3	0.90	0.86	0.88 (0.907)
	4	0.96	0.72	0.81 (0.935)
	5	0.97	0.68	0.79 (0.937)
	6	0.97	0.60	0.74 (0.908)
	7	1	0.36	0.50 (0.904)
	8	1	0.44	0.58 (0.869)

(b) Dickbut

Table 5: Effect on F_1-score when, in (a) varying λ for SSK and in (b) varying n for NGK. Both algorithms were trained using the subset of Reuters dataset.

SSK	λ	n	Precision	Recall	F_1
acq	0.5	3	0.93	0.96	0.94 (0.785)
		4	0.93	0.96	0.94 (0.822)
		5	0.97	0.93	0.95 (0.867)
		6	0.98	0.93	0.95 (0.876)
		7	0.98	0.89	0.93 (0.864)
		8	0.99	0.80	0.93 (0.852)
		10	0.98	0.62	0.75 (0.791)
		12	0.98	0.33	0.49 (0.791)
		14	1	0.15	0.25 (0.774)
earn	0.5	3	0.99	0.93	0.96 (0.925)
		4	0.99	0.95	0.97 (0.932)
		5	0.99	0.96	0.97 (0.936)
		6	0.99	0.93	0.97 (0.936)
		7	0.99	0.91	0.96 (0.940)
		8	0.99	0.91	0.95 (0.934)
		10	1	0.86	0.92 (0.927)
		12	1	0.76	0.87 (0.931)
		14	1	0.68	0.81 (0.936)
corn	0.5	3	0.97	0.87	0.91 (0.665)
		4	0.98	0.64	0.88 (0.783)
		5	0.98	0.44	0.83 (0.779)
		6	0.98	0.42	0.78 (0.749)
		7	0.98	0.24	0.74 (0.643)
		8	1	0.43	0.59 (0.569)
		10	1	0.29	0.44 (0.582)
		12	0.86	0.16	0.26 (0.618)
		14	0.71	0.11	0.20 (0.702)
crude	0.5	3	0.97	0.86	0.88 (0.881)
		4	0.98	0.80	0.91 (0.905)
		5	0.98	0.73	0.88 (0.936)
		6	0.98	0.66	0.82 (0.901)
		7	0.98	0.60	0.73 (0.872)
		8	1	0.43	0.67 (0.828)
		10	1	0.28	0.42 (0.764)
		12	0.29	0.03	0.05 (0.709)
		14	0.14	0.01	0.02 (0.761)

Table 6: Full range of subsequence lengths used in [1], who's F_1-scores are shown in parentheses.