String Subsequence Kernels for Text Classification

Hjaltason, Bjartur
first1.last1@xxxxx.com

Carlsson, Hugo
first2.last2@xxxxx.com

Brynjarsson, Gujn Ragnar
grbr@kth.se

Sigurgeirsson, Atli r
first2.last2@xxxxx.com

Abstract

This paper is about the string subsequence kernel and how it works. The feature space is generated by subsequences of strings of predetermined length k occurring in the training documents. The kernel then compares two strings in our case documents based on the number of occurrences of the same substrings between the two. We then went on two use support vectors created from the Kernel matrix to assume which category a test documents belong to. Then we compared those results to other known document classifications methods such as n-grams.

Pimp this up with new infomration about the new kernel and wk comparison etc.

1 Introduction

In machine learning, a lot of the standard learning methods we use require input data living in some feature vector space. We will then do some work in that vector space that can again be transcribed into some knowledge. These vector spaces can easily blow up in terms of dimensionality which makes any sort of regression or classification efforts harder or even impossible. However a lot of the times we can not readily describe our input data by explicit feature vectors or it can be terribly expensive and complex. This is the case with text document analysis as well as images and graphs to name a few.

Instead of constructing these complex feature spaces we can use Kernel Methods. Kernel methods are based on a defined function, called the Kernel, that calculates the inner product (in some sense) between 2 data points in a higher dimensional space. This kernel thus gives us some idea of how related 2 data points are. In the case of text analysis, these methods are extremely convenient since the Kernel methods do not depend on the dimensionality of the feature space which can be gigantic for large documents. In their article Text Classification using String Kernels the authors proposed a certain Kernel for text

sequences (which can be thought of as documents). The idea behind the chosen Kernel is to compare the documents by means of their (not necessarily contiguous) substrings. The substrings are weighted by their length so that shorter substrings are highly weighed. In this case the length to consider is not the number of letters in the substring but rather the difference in their position in the document. To do that the authors introduce a decay factor. The kernel is therefore characterized by the decay factor as well as the number of letters to consider in a substring.

Definition 1. Subsequence Kernel

Let s and t be string from the finite alphabet Σ, where length of $s = s_1...s_{|s|}$ is $|s|$. The sub-string $s_i...s_j$ is denoted by $s[i:j]$. The target string u is a subsequence of s i.e $u = s[i]$ if there exist indices $i = (i_1, ..., i_{|u|})$, which of course entails $1 \leq i_1 < ... < i_{|u|} \leq |s|$ i.e. the sub-string can't have indices outside of the "main" string. $u_j = s_{i_j}$ for each character $u_j \in u$ which will be noted in the following fashion on-wards $u = s[i]$. The length $l(i)$ of the subsequence in s is given by $i_{|u|} - i_1 + 1$. which stands for the number of characters the subsequence $s[i]$ spans in the string s.

The feature mapping for a string s in the paper is defined by giving the u coordinate for each $u \in \Sigma$ as

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

Which depicts the number of times the subsequence occurs in the string s, weighted by how continuous it is. The kernel function then represents the inner product of the feature vectors for strings s and t as the sum of the products of the above u coordinate for all the substrings in u.

$$K_n(s,t) = \sum_{u \in \sum^n} \langle \phi_u(s) \cdot \phi_u(t) \rangle = \sum_{u \in \sum^n} \sum_{i:u=s[i]} \lambda^{l(i)} \sum_{j:u=t[j]} \lambda^{l(j)} \tag{1}$$

$$= \sum_{u \in \sum^n} \sum_{i:u=s[i]} \sum_{j:u=t[j]} \lambda^{l(i)+l(j)} \tag{2}$$

So, n is the length of common substrings between s and t. The λ is then the weight decay factor which controls how much the Kernel value is penalized for the substring not being continuous inside the string.

$$K_i^{'}(s,t) = \sum_{u \in \sum^n} \sum_{i:u=s[i]} \sum_{j:u=t[j]} \lambda^{|s|+|t|-i_1-j_1+2}, i = 1, ..., n-1$$

Many different extensions to this kernel have been proposed. One of them was proposed by (Seewald, A. and Kleedorfer, F., 2007). The proposed change was named lambda pruning and the resulting kernel is known as SSK-LP. Lambda-pruning is a simple approximation technique that is supposed to take far less memory than the original implementation as well as usually being several orders of magnitude faster. The pruning works by introducing a constant

θ named the Maximum Lambda Exponent. By then constricting the combined length of the two subsequences being considered to being below the θ we can reduce the recursion by quite a lot and thus lower memory and computation time. What this does is it reduces strings of great lengths to 0 instead of increasingly small values since the value is calculated as the decay factor to the power of the combined length which tends to zero.

Another method of doing analysis of text documents are Syntactic Tree Kernels (Collins and Duffy, 2001) where trees encode grammatical derivations. The kernels work by counting the number of subtrees shared by the input trees, the more subtrees shared the closer the documents are related. Both of these methods, the SSK and the STK compute similarities using the number of common features whether it is substrings or subtrees. This relies on having a relatively high number of common features between documents to work properly and therefore has limitations on smaller documents.

One Kernel that has been proposed to work on smaller documents is the Language Independent Semantic Kernel (LIS) (K. Kim et al., 2014). The aim of the researchers was to effectively calculate similarities between short text-documents using both the syntactic and semantic features of the documents without having to rely on grammatical tags or lexical databases, making it language independent (K. Kim et al., 2014). One of the big advantages of not having to rely on lexical databases is that they are not at all readily available for most languages. The LIS Kernel works in three steps: pattern extraction, semantic analysis and finally similarity computation. The authors discuss a few methods for pattern extraction but prefer one that is reliant on having a syntactic parse tree representation of the document with the other methods as fallback options if such a tree is not available. After pattern extraction they propose methods for semantic analysis on word-, document- and category bases which leads to definitions of three kernels for patterns. Finally they propose a kernel for similarity calculations which is a weighted linear combination of the three previously mentioned kernels. The basis behind this kernel is the thought that two patterns with similar semantics are likely to often co-occur in the same document and also appear with many common words in documents (K. Kim et al., 2014).

In this short report we will try to implement the original SSK proposed by (Lodhi et al. 2002) and compare our results to that of the paper. We will also look into some possible changes to the kernel and their effects??????

2 Implementation

We started out by writing the code that extracted the features from documents. Originally we did it as the paper does, looking at all substrings (both contiguous as well as non-contiguous) however we quickly ran into the problem of processing power and time management where just the feature extraction was taking way too long. Therefore we decided to only consider contiguous strings of predetermined length k. Furthermore we chose to consider only the top m features

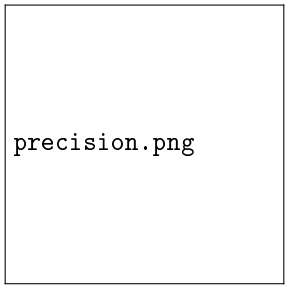
of each document. Which combined make up all of our features, as this makes calculations quicker in the following steps. Approximation of the method used in the paper we used, worked as follows:

First we start off by getting the top m features from all of the training documents for the two different categories we are comparing and add them all to a list to use as our list of features. Then we continue to create our kernel matrix in the following manner: For documents s and t we start by counting the occurrence of each feature f in our total feature list m in both of the documents. Where we will call l the count of feature f_i in document s and j the count of feature f_i in document t. We now, for all of the features in our feature list m do the same i.e extract the count of the feature in the two documents and multiply l with our selected value of λ in the power of our selected value of k i.e the max substring length and multiply that with j times λ in the same power as before i.e k. So we end up with the multiplication of the count of all features the two documents have in common as our kernel value for each two documents.

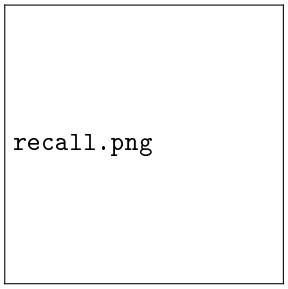
Now when we have our Kernel matrix, we go ahead and use an optimizer to find our support vectors. Then use those to classify the test documents two which class they belong too.

WE NEEED TO ADD DEFINITION ON OUR NEW KERNEL APPROXIMATION METHOD HERE!!!

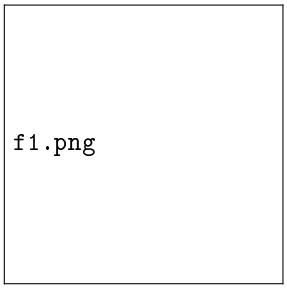
3 Discussion



(a) precision



(b) recall



(c) F1-score

We can clearly see from the results that there is a clear distinction between the precision for the *corn* dataset and the other three. *Earn, acquisition* and *crude* all show the same trend of being fairly stable for feature lengths going from 3 up to 8 but a precision drop with higher number of features although different in how steep the drop is. *Corn* however has lower precision for lower feature lengths. There is also a visible drop for $k = 7$, and 8 but an increase again for $k = 10$ which is a behaviour not seen in the other 3 datasets. We made the casual observation that many of the *corn* articles included statistical data which resulted in features that may not represent the category very well. We think this could count towards the lower precision of *corn*. The lowered precision with higher k is likely a result of the features becoming more closely linked to each article instead of capturing the features that would be common across articles in the category. The lower precision for *corn* is in line with the findings of the original paper.

If we compare our results to the original paper we notice that they seem to have greater precision for higher feature lengths. We postulate that this is caused by our approximation of having non-contiguous strings of a fixed length. Similar trends are seen for the recall and F1-score for comparable trends. Tables 1 and 2 show our results in the same format as the original paper for easy comparison.

Table 1: Results: earn vs. acquisition

Category	Kernel	Length	F1		Precision		Recall	
			Mean	SD	Mean	SD	Mean	SD
earn	SSK	3	0.90405	0.02397	0.94597	0.04332	0.87004	0.05667
		4	0.90799	0.01600	0.94762	0.02653	0.87366	0.04245
		5	0.90165	0.02073	0.96852	0.02042	0.84502	0.04159
		6	0.87217	0.04481	0.97814	0.01928	0.79140	0.07930
		7	0.89851	0.02010	0.91364	0.05940	0.88874	0.04098
		8	0.89075	0.01843	0.92755	0.04464	0.85968	0.03805
		10	0.82441	0.03614	0.81371	0.08370	0.84512	0.05224
		12	0.73048	0.04071	0.71154	0.06470	0.75502	0.03888
		14	0.58535	0.05061	0.60869	0.05608	0.57297	0.08807
	NGK	3	0.90637	0.02535	0.95793	0.02820	0.86352	0.05824
		4	0.90647	0.02291	0.97972	0.01598	0.84504	0.04485
		5	0.89605	0.02896	0.96777	0.02671	0.83732	0.05833
		6	0.91625	0.03664	0.97339	0.02456	0.86944	0.07020
		7	0.91242	0.01912	0.97566	0.01548	0.85794	0.03637
		8	0.91221	0.02805	0.95035	0.04482	0.88181	0.06038
		10	0.88302	0.01946	0.92052	0.04909	0.85345	0.05263
		12	0.83103	0.02762	0.86535	0.07404	0.80728	0.05416
		14	0.77092	0.03130	0.77528	0.05927	0.77085	0.04017
	WK							
acq	SSK	3	0.91046	0.02070	0.87979	0.04676	0.94747	0.04479
		4	0.91816	0.00981	0.88838	0.03264	0.95184	0.02644
		5	0.91525	0.01649	0.86557	0.03325	0.97235	0.01918
		6	0.89435	0.03106	0.82438	0.05829	0.98089	0.01730
		7	0.90018	0.02351	0.89305	0.03381	0.91254	0.06556
		8	0.89781	0.01653	0.86963	0.02891	0.93057	0.04496
		10	0.81607	0.05392	0.84270	0.03670	0.80030	0.10589
		12	0.71051	0.07920	0.73849	0.04837	0.69027	0.10842
		14	0.60243	0.04743	0.59200	0.05293	0.62257	0.08642
	NGK	3	0.91408	0.02005	0.87498	0.04819	0.95997	0.02858
		4	0.91943	0.01675	0.86527	0.03464	0.98215	0.01460
		5	0.91064	0.02029	0.85894	0.04445	0.97151	0.02396
		6	0.92599	0.02743	0.88414	0.05623	0.97535	0.02339
		7	0.92102	0.01498	0.87117	0.02942	0.97785	0.01408
		8	0.91844	0.02303	0.89093	0.05003	0.95206	0.04423
		10	0.88925	0.02179	0.86429	0.04111	0.92124	0.05988
		12	0.83732	0.03390	0.81733	0.04153	0.86680	0.08437
		14	0.77553	0.04113	0.77698	0.03390	0.77814	0.07216
	WK							

Table 2: Result: crude vs. corn

Category	Kernel	Length	F1		Precision		Recall	
			Mean	SD	Mean	SD	Mean	SD
crude	SSK	3	0.93576	0.02307	0.91343	0.04159	0.96073	0.02300
		4	0.95210	0.02294	0.93392	0.03421	0.97150	0.01631
		5	0.95239	0.02140	0.95015	0.04094	0.95590	0.01923
		6	0.96088	0.01581	0.96424	0.02482	0.95828	0.02409
		7	0.94053	0.02270	0.93368	0.03689	0.94938	0.03763
		8	0.92236	0.01663	0.92438	0.04652	0.92397	0.04004
		10	0.89394	0.01464	0.89111	0.04394	0.90116	0.04795
		12	0.84195	0.04415	0.85821	0.05514	0.83740	0.09129
		14	0.74116	0.02806	0.73714	0.04601	0.74914	0.04866
	NGK	3	0.94554	0.01302	0.94303	0.02951	0.94908	0.01986
		4	0.95722	0.01279	0.94453	0.02966	0.97126	0.01842
		5	0.95337	0.02403	0.93904	0.04887	0.96997	0.01607
		6	0.96534	0.01483	0.95330	0.02828	0.97836	0.01490
		7	0.95346	0.01520	0.94699	0.01903	0.96160	0.03907
		8	0.93217	0.02113	0.91840	0.03681	0.94933	0.04514
		10	0.93405	0.01219	0.92733	0.02566	0.94188	0.02378
		12	0.90452	0.02072	0.88707	0.04781	0.92614	0.03758
		14	0.88380	0.01580	0.89579	0.03223	0.87454	0.04084
	WK							
corn	SSK	3	0.88875	0.04017	0.93194	0.03618	0.85324	0.06980
		4	0.92102	0.03827	0.95278	0.02729	0.89255	0.05514
		5	0.92445	0.03874	0.93094	0.02996	0.92098	0.06787
		6	0.93741	0.02593	0.93525	0.03298	0.94124	0.04219
		7	0.90511	0.03773	0.91967	0.05865	0.89552	0.05950
		8	0.87576	0.02810	0.88147	0.05577	0.87884	0.07533
		10	0.83082	0.03095	0.84711	0.06095	0.82517	0.07988
		12	0.76422	0.04638	0.76784	0.07628	0.77694	0.09955
		14	0.58607	0.06143	0.59907	0.05923	0.58181	0.09132
	NGK	3	0.90720	0.02473	0.91451	0.03303	0.90311	0.05232
		4	0.92952	0.02149	0.95182	0.03247	0.91073	0.04623
		5	0.92189	0.04099	0.94997	0.02516	0.90027	0.08034
		6	0.94116	0.02401	0.96359	0.02345	0.92141	0.04563
		7	0.92174	0.02283	0.93759	0.06019	0.91053	0.03328
		8	0.88954	0.03242	0.92040	0.06217	0.86724	0.06185
		10	0.89357	0.02043	0.90609	0.03763	0.88381	0.04003
		12	0.84504	0.04207	0.88293	0.05296	0.81833	0.08411
		14	0.81129	0.01762	0.80124	0.05216	0.82809	0.05582
	WK							

Table 3: Results: earn vs. acquisition

Category	Kernel	λ	F1		Precision		Recall	
			Mean	SD	Mean	SD	Mean	SD
earn	SSK	3	0.86712	0.18792	0.32556	0.29397	0.35440	0.40688
		4	0.97552	0.01733	0.51546	0.11696	0.36052	0.11717
		5	0.96407	0.03316	0.52239	0.06941	0.36351	0.07134
		6	0.88474	0.06463	0.59773	0.06055	0.45895	0.08220
		7	0.88549	0.03703	0.44900	0.08408	0.30763	0.08289
		8	0.58117	0.08269	0.25021	0.09221	0.17433	0.08353
	NGK							
	WK							
acq	SSK	3	0.91046	0.02070	0.87979	0.04676	0.94747	0.04479
		4	0.91816	0.00981	0.88838	0.03264	0.95184	0.02644
		5	0.91525	0.01649	0.86557	0.03325	0.97235	0.01918
		6	0.89435	0.03106	0.82438	0.05829	0.98089	0.01730
		7	0.90018	0.02351	0.89305	0.03381	0.91254	0.06556
		8	0.89781	0.01653	0.86963	0.02891	0.93057	0.04496
		10	0.81607	0.05392	0.84270	0.03670	0.80030	0.10589
		12	0.71051	0.07920	0.73849	0.04837	0.69027	0.10842
		14	0.60243	0.04743	0.59200	0.05293	0.62257	0.08642
	NGK	3	0.91408	0.02005	0.87498	0.04819	0.95997	0.02858
		4	0.91943	0.01675	0.86527	0.03464	0.98215	0.01460
		5	0.91064	0.02029	0.85894	0.04445	0.97151	0.02396
		6	0.92599	0.02743	0.88414	0.05623	0.97535	0.02339
		7	0.92102	0.01498	0.87117	0.02942	0.97785	0.01408
		8	0.91844	0.02303	0.89093	0.05003	0.95206	0.04423
		10	0.88925	0.02179	0.86429	0.04111	0.92124	0.05988
		12	0.83732	0.03390	0.81733	0.04153	0.86680	0.08437
		14	0.77553	0.04113	0.77698	0.03390	0.77814	0.07216
	WK							

Table 4: Comparing feature lengths when all features are considered

Category	Length	F1	Precision	Recall
earn	3	0.78947	0.71429	0.88235
	4	0.78756	0.80000	0.77551
	5	0.67358	0.67708	0.67010
	6	0.65969	0.67021	0.64948
	7	0.57143	0.70588	0.48000
	8	0.37662	0.56863	0.28155
acq	3	0.72093	0.83784	0.63265
	4	0.80193	0.79048	0.81373
	5	0.69565	0.69231	0.69903
	6	0.68900	0.67925	0.69903
	7	0.68966	0.60606	0.80000
	8	0.60976	0.50336	0.77320