DD2434 Machine Learning, Advanced Course

Project Report

Group name: Handarbeni-Ioannidou-Padmanaban-Mochammad P

Muhammad Irfan Handarbeni
handa@kth.se
940729 - 1518

Polyxeni Parthena Ioannidou
ppio@kth.se
920403 - 6207

Deepika Anantha Padmanaban
deap@kth.se
950303 - 4408

Fadhil Mochammad
fadhilm@kth.se
950910 -3637

20.01.2019

1. ANALYSIS OF THE PURPOSE

The purpose of this project is to read, understand and re-implement the kernel methods in text classification. The paper that we followed is "Text Classification using String Kernels" by Lodhi, Saunders, Shawe-Taylor, Cristianini and Watkins. In this project we focus on classifying text documents using kernels and support vector machines. We use kernels to extract features from data unambiguously. More specifically, Kernel functions return the inner product between the mapped data points in a higher dimensional space, where the data could become more easily separated or better structured.

We consider three different types of kernels as mentioned in the original paper, namely, String Subsequence Kernel(SSK), n-Grams Kernel(NGK) and Word-Kernel(WK). We tried re-implementing each of the kernels and used them for training our SVMs and later used the model for prediction. We also observe the effect of varying the hyper-parameters 'n' and 'λ' for the SSK and 'n' for NGK. We compare the results obtained from our experiments with the original results in the paper and discuss the main implications of our results. We also, try providing an extension to the framework of SSK, in order reduce the computation time, called Lambda Pruning. Finally, we conclude with the mention of a few state-of-the-art method in text classification and their advantages over these traditional methods.

2. TEST CONFIGURATIONS

- The experiments presented in this project were performed on a subset of the Reuters dataset. We haven't used the same dataset as has been used by the authors, rather we use a very simple dataset of 100 training documents and 20 test documents, in order to simplify our calculations.
- We consider the same four categories for classification, as has been used in the original paper, namely, 'corn', 'crude', 'earn' and 'acq'. 'earn' and 'acq' are most frequent categories of the Reuters dataset. The direct correspondence between the respective words and the categories 'crude' and 'corn' make them potential candidates.
- For running the test on SSK, we considered a training set of 20 and 8 test documents each. Since, the algorithm is recursive and hence, computationally heavy and memory occupying, we ran the tests on Google Cloud Instances with 24 vCPUs and 154GB memory. Each of the test run ran for 2 - 8 hours in the cloud instance depending on the configuration used.
- For evaluations, we considered the quantitative measures, such as F1, recall and precision for all the categories for each kernel configuration.

3. PRE-PROCESSING

- **General**:

  We obtained the Reuter dataset from http://www.research.att.com/lewis. The obtained dataset was pre-processed using the NLTK library, where we removed all non-informative words such as stop words and punctuation and words were replaced by their stems. This is done in order to capture the similarity between families of derivationally related words. This level of pre-processing was common for all the kernel methods.

- **WK**:

  The word kernel is a linear kernel that measures the similarity between documents that are indexed by word. The WK uses a complete word rather than splitting a full-text article into n-gram words. The entries of the feature vectors were weighted by using a variant of tf-idf (term frequency-inverse document frequency), log(1+tf) x log(n/df), weighting scheme,where, tf represents term frequency while df is used for document frequency and n is the total number of documents. The documents were normalized so that each document had equal length.

- **NGK**:

  Similar to WK, each document in the collection is transformed into a feature vector. However, in NGK, each entry of the feature vector represents the number of times the corresponding substring occurs in the document. In NGK, we also weighted the entries of feature vectors by using a variant of tf-idf similar to what we did on WK.

4. SSK

A) IMPLEMENTATION

In SSK, we generate the feature space by the set of all (non-contiguous) substrings of k-symbols. The more substrings two documents have in common, the more similar they are considered. We define the parameters such as the decay factor $\lambda \in (0,1)$ and the subsequence length k for the implementation. A k-tuple will have a non-zero entry if it occurs as a subsequence anywhere (not necessarily contiguously) in the string. The weighting of the feature will be the sum over the occurrences of the k-tuple of a decaying factor of the length of the occurrence. The naive algorithm was very computation-heavy. So, we continued with the implementation of the Dynamic Programming approach. We implemented the recursive version of the algorithm. For calculating the Gram matrix elements, we use recursive functions according to the following definition from the paper by Lodhi, Saunders, Shawe-Taylor, Cristianini and Watkins:

$$K'_0(s,t) = 1, \forall s,t$$

$$K'_i(s,t) = 0, \text{if } min(|s|,|t|) < i$$

$$K_i(s,t) = 0, \text{if } min(|s|,|t|) < i$$

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

$$K''_i(sx,tu) = \lambda^{|u|} K''_i(sx,t)$$

$$K''_i(sx,tx) = \lambda^{|u|} K''_i(sx,t) \quad where the reis no k : u_k = x$$

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

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

The recursive implementation of SSK was time consuming depending on the length of the documents, with longer documents taking multiple hours to be trained. Inorder to speeden the process, we took advantage of the form of the gram matrix values. We know that the Gram matrix is symmetric, thus, we included this in the implementation to avoid duplicate calculations. Next, we also made sure the two documents are not compared more than once, by the appropriate checks. Also, while calculating the K′ and K'', the results of each value was cached, so that whenever we find the same comparison we did not have to recalculate the K values. Instead of recomputing, we simply look up and use the previously computed solution.But, this worked until the subsequence length was small and the documents were not big enough. With an increase in the length, the number of values to be cached increased exponentially, caused a memory limitation. So, we had to limit the values that were cached and traded-off between the computation time and memory occupied. Since, we used Google Cloud Instances, we could save some huge computational time with the high memory capacity of the instances.

B) RESULTS AND EXPLANATION

For varying length of substrings:

The SSK implementation described above was run over 3 iterations and scores of F1, recall and precision have been averaged and reported along with their standard deviation in the table below in figure[1].

Category	Length	F1		Recall		Precision	
		Mean	SD	Mean	SD	Mean	SD
acq	3	**0.833**	0.235	0.750	0.353	1.000	0.000
	4	0.823	0.167	0.833	0.288	0.889	0.192
	6	0.772	0.227	0.800	0.200	0.750	0.250
corn	3	0.335	0.471	0.500	0.707	0.250	0.353
	4	**0.611**	0.096	0.500	0.000	0.833	0.288
	6	0.458	0.208	0.350	0.150	0.665	0.335
crude	3	0.667	0.0000	0.750	0.353	0.750	0.353
	4	**0.778**	0.192	0.833	0.288	0.833	0.288
	6	0.622	0.178	0.700	0.300	0.584	0.084
earn	3	**1.000**	0.000	1.000	0.000	1.000	0.000
	4	0.93	0.115	1.000	0.000	0.889	0.192
	6	0.8335	0.166	0.900	1.000	0.785	0.214

Figure 1: F1, recall and precision of Approximated SSK for an average of 3 iterations for varying lengths of subsequences with λ=0.5

Decay factor	Category			
	acq	corn	crude	earn
0.3	0.5	0.8	0.667	1
0.5	0.667	0.667	0.667	1
0.7	0.667	0.667	0.667	1

Figure 2: F1 score for all the categories for varying values of decay factor.

From the results, it can be seen that SSK performs better for the smaller subsequence lengths as has been expected from the original paper. But, we also see that the F1 values do not seem to be as high as the original values in the paper, because the dataset considered for our purpose was substantially smaller than that used in the original paper by the authors so that we could afford multiple runs of the configuration.

The results show that for the categories 'acq' and 'earn', the F1 scores peak at substring length 3, this was as we expected, as we found that the dataset we used for training the model had the smallest length of documents for 'earn' and the second smallest for 'acq'. Thus, finding shorter similar subsequences is more a plausible case than finding longer subsequence. Thus, we see a gradual decline in the F1 scores for increase in substring length.

Similarly, considering the labels 'corn' and 'crude', we found that both these labels had a substantially high document length, with a minimum of 750 characters in each document. Thus, we expected a lower F1 score in these cases as we used a small dataset. Thus, justifying the lower F1 scores.

For varying decaying factor λ:

The same experiment was conducted with substring length n = 4, for different values of the decaying factor λ and the results have been tabulated in the figure[2].

The results obtained for corn is similar to the original paper, with a low decay factor showing the best performance for the SSK, while for 'acq', the F1 score is high for higher value of λ. The 'earn' category is showing up a constant F1 score, because it was found to be having very small document lengths, thus, separating it well from the other categories, thus allowing easy classification.

The result for 'crude' category seemed unexpected, because, we expected a increase in pattern considering its long document length. But, were surprised to see this. We think this may be due to the fact that the test was run only once. Probably, multiple runs using different datasets would have shown the expected trend. The result shows that the decaying factor has a direct relationship with the contiguity of substrings, with higher λ placing more weights on non-contiguous substrings. This is because SSK weights the substrings according to their proximity in the text.

Category	Length	F1		Recall		Precision	
		Mean	SD	Mean	SD	Mean	SD
acq	3	**0.771**	0.149	0.771	0.149	0.866	0.231
	4	0.756	0.074	0.800	0.163	0.733	0.056
	6	0.323	0.053	0.300	0.141	0.417	0.118
corn	3	0.393	0.110	0.393	0.110	0.570	0.158
	4	**0.467**	0.312	0.450	0.300	0.488	0.332
	6	0.290	0.056	0.200	0.000	0.464	0.050
crude	3	**0.751**	0.187	0.751	0.186	0.677	0.422
	4	0.702	0.185	0.750	0.191	0.665	0.188
	6	0.393	0.152	0.400	0.283	0.667	0.470
earn	3	**0.963**	0.064	0.963	0.064	1.000	0.000
	4	0.828	0.184	0.850	0.300	0.886	0.139
	6	0.466	0.225	0.700	0.424	0.350	0.144

Figure 3: F1, recall and precision of Approximated SSK for an average of 3 iterations for varying lengths of subsequences with λ=0.5

4. APPROXIMATED SSK

As mentioned earlier, the string kernel has time complexity of O(n|s||t|) making it computationally heavy. Thus, we tried implementing the approximation of SSK. In this implementation, for a value of substring length, we enumerate all the possible combination in the English language and picked the top 200 maximum occurring sequence in the document as features for comparison with the similar features of the other documents. The results can be found tabulated in figure[3], for different values of substring lengths and λ being a constant factor of 0.5. The table shows the mean values of 3 runs along with their standard deviations.

Since, this approximation involves ignoring certain information in the document, we expected a lower evaluation scores as compared to the original SSK and did see similar results. We also see the same trend as in the original SSK. But, as expected we saw a drastic reduction in the runtime of the experiments, thus, making this approximation a good replacement for the original SSK, with a small trade-off in the accuracy.

5. N-GRAMS KERNEL

A) IMPLEMENTATION

The n-gram kernel implemented for this project is similar to the n-gram kernel described in the original paper. Each document in the dataset is transformed into a feature vector with each feature represents a contiguous substring of the document. N-grams represent an n-length substring from the document. For instance, doc = 'an apple' then the 3-grams are an_ , n_a, _ap, app, ppl, ple. Where _ denotes a space character. The weights of the substrings are calculated using tf-idf weighing method.

The similarity of two documents in each tuple of documents is calculated based on how many identical subsequences between those two documents, using cosine similarity kernel function:

$$K_{\mathrm{NGK}}(d_1, d_2) = \frac{\phi(d_1) \cdot \phi(d_2)}{\|\phi(d_1)\| \|\phi(d_2)\|} \tag{1}$$

where (d1), and (d2) indicate the feature vector of the documents. This similarity measure is then used to generate gram matrix both for the training dataset and test dataset. With each value in the matrix indicates the similarity or kernel value of a couple of documents. The training gram matrix is feed to the SVM model for training, and the test gram matrix is feed to the trained model for classification. The implementation of the n-gram kernel is done using python. N-grams from each document are generated using TfidfVectorizer method from the scikit-learn package which able to generate all n- length substring from a given text with its tf-idf weights. For the classification part, scikit-learn SVC module is used. The model is trained using the precomputed gram matrix.

		f1-score		precision		recall	
category	n	mean	SD	mean	SD	mean	SD
	3	0.9157	0.0850	0.9048	0.1255	0.9400	0.0917
	4	**0.9325**	0.0631	0.9133	0.0872	0.9600	0.0800
	5	0.9268	0.0880	0.9048	0.1255	0.9600	0.0800
	6	0.9157	0.0850	0.9048	0.1255	0.9400	0.0917
	7	0.9092	0.0930	0.8958	0.1410	0.9400	0.0917
	8	0.8913	0.0930	0.8592	0.1329	0.9400	0.0917
	10	0.8384	0.1212	0.7792	0.1830	0.9400	0.0917
	12	0.7842	0.1522	0.7197	0.2230	0.9200	0.0980
acq	14	0.6690	0.1341	0.6353	0.2464	0.8200	0.1400
	3	0.8906	0.0849	0.9600	0.0800	0.8400	0.1200
	4	0.9156	0.0762	0.9600	0.0800	0.8800	0.0980
	5	0.9156	0.0762	0.9600	0.0800	0.8800	0.0980
	6	0.9156	0.0762	0.9600	0.0800	0.8800	0.0980
	7	**0.9267**	0.0795	0.9600	0.0800	0.9000	0.1000
	8	0.9067	0.0836	0.9400	0.0917	0.8800	0.0980
	10	0.7607	0.1595	0.8967	0.1354	0.7000	0.2236
	12	0.6842	0.1890	0.8458	0.1937	0.6200	0.2441
corn	14	0.6780	0.1923	0.9356	0.1400	0.6000	0.2530
	3	0.8524	0.0700	0.8495	0.1060	0.8800	0.1327
	4	**0.8972**	0.0974	0.9050	0.0978	0.9000	0.1342
	5	0.8944	0.1126	0.9050	0.0978	0.9000	0.1612
	6	0.8853	0.1073	0.8883	0.0943	0.9000	0.1612
	7	0.8742	0.1004	0.8883	0.0943	0.8800	0.1600
	8	0.8322	0.0784	0.8650	0.0914	0.8200	0.1400
	10	0.7643	0.0817	0.8189	0.1418	0.7600	0.1744
	12	0.6895	0.2588	0.7017	0.2980	0.7200	0.2857
crude	14	0.6327	0.2392	0.7268	0.3275	0.6600	0.3105
	3	**1.0000**	0.0000	1.0000	0.0000	1.0000	0.0000
	4	0.9909	0.0273	0.9833	0.0500	1.0000	0.0000
	5	1.0000	0.0000	1.0000	0.0000	1.0000	0.0000
	6	1.0000	0.0000	1.0000	0.0000	1.0000	0.0000
	7	0.9889	0.0333	1.0000	0.0000	0.9800	0.0600
	8	0.9889	0.0333	1.0000	0.0000	0.9800	0.0600
	10	0.9075	0.0892	0.9625	0.1125	0.8800	0.1327
	12	0.8047	0.1741	0.9625	0.1125	0.7400	0.2200
earn	14	0.6530	0.1975	0.9022	0.1591	0.5800	0.2600

Figure 4: F1, recall and precision of NGK for an average of 10 iterations for varying lengths of subsequences with 100:20 dataset

B) RESULTS AND EXPLANATION

We performed multiple tests over the Reuters dataset, where we tried to vary the subsequence length n. The experiment were repeated for 10 iterations and the results were averaged. As the original paper experiments were conducted on the documents in 4 different categories: acq, corn, crude, earn an the results can be seen in figure[4] and figure[5].

From the experiments result, it can be seen that the f1-score, precision, and recall for this implemented n-gram kernel is not as good as in the original paper. That is most likely due to the lack of training and test documents used in the experiments. By increasing the number of documents for training and test, the result becomes similar to the ones in the original paper. Also, by varying the substring length n the model, the result indicates that a small value of n gives a good result. On the contrary, if n is too large (n>8), the model failed to give a good performance. That is reasonable because documents are likely to have more identical short substrings rather than long substrings.

		f1-score		precision		recall	
category	n	mean	SD	mean	SD	mean	SD
acq	3	**0.8000**	0.3055	0.8000	0.3317	0.8500	0.3202
	4	0.7133	0.3795	0.7167	0.3948	0.7500	0.4031
	6	0.6267	0.4239	0.5833	0.4167	0.7000	0.4583
corn	3	0.8133	0.3023	0.8667	0.3055	0.8000	0.3317
	4	**0.8800**	0.1514	0.9667	0.1000	0.8500	0.2291
	6	0.8800	0.1514	0.9667	0.1000	0.8500	0.2291
crude	3	0.7038	0.3818	0.6567	0.3927	0.8000	0.4000
	4	**0.7705**	0.3032	0.7567	0.3360	0.8500	0.3202
	6	0.7371	0.2943	0.7567	0.3360	0.8000	0.3317
earn	3	**0.9133**	0.1368	0.9667	0.1000	0.9000	0.2000
	4	0.9000	0.1528	0.9500	0.1500	0.9000	0.2000
	6	0.8833	0.1833	0.9000	0.2000	0.9000	0.2000

Figure 5: F1, recall and precision of NGK for an average of 10 iterations for varying lengths of subsequences with 20:8 dataset

6. WK - WORD KERNEL

A) IMPLEMENTATION

We try to implement the word kernel by counting the frequencies of the words in both documents and computes the inner product of the resulting vectors. For example, assume that we have the documents

d_1 = {There was heavy rain} and d_2 ={There was no sun}. Then the kernel will be computed as:

$$K_{\text{WK}}(d_1, d_2) = \langle \phi(d_1) \cdot \phi(d_2) \rangle = [1,1,1,1,0,0] \cdot [1,1,0,0,1,1]^T = 2 \tag{2}$$

The ϕ value is the transformation from the text to the vector representation where the vectors are {There}, {was}, {heavy}, {rain}, {no}, and {sun} in an order. For the implementation, we used TfidfVecorizer method from the scikit-learn package to compute the gram matrix. Similar to the implementation of NGK, the similarity between 2 documents is represented in each index of the gram matrix. We also trained the SVM model using the training gram matrix and ran the classification using the test gram matrix.

B) RESULTS AND EXPLANATION

Table 1: WK - Train 20, Test 8 documents

category	F1		Precision		Recall	
	Mean	SD	Mean	SD	Mean	SD
acq	0.8133	0.3023	0.8667	0.3055	0.8	0.3316
corn	0.9133	0.1368	0.9667	0.1	0.9	0.2
crude	0.8667	0.1633	0.85	0.2291	0.95	0.15
earn	0.9667	0.1	1	0	0.95	0.15

Table 2: WK - Train 100, Test 20 documents

category	F1		Precision		Recall	
	Mean	SD	Mean	SD	Mean	SD
acq	0.9358	0.0903	0.9214	0.126	0.96	0.08
corn	0.9355	0.0691	0.98	0.06	0.9	0.1
crude	0.9216	0.0746	0.91	0.0907	0.94	0.0916
earn	0.9889	0.0333	1	0	0.98	0.06

When we trained our model with more documents we observed a higher F1 mean value, as well as the SD value was decreased in all 4 categories, which was the expected performance. The more number of documents that we use on the training, the better the result we get. We can see that when we use 100 documents on training, we have the precision and recall mean higher than what we get when using 20 document on training.

7 EXTENSION OF SSK

An extension to the approximation of SSK could be Lambda Pruning. This technique focuses on pruning some values of lambda that have very small effect on the overall result, which limits the depth of recursion. In SSK with Lambda Pruning we ignore terms that raise λ to a power greater than a threshold. The recursive definition of SSK-LP is similar to that of SSK, but the difference is the parameter m. In SSK-LP recursion we decrease the value of m by x and we multiply the result of recursion with λ^x. This limits the depth of recursion.

$$K'_{0,m}(s,t) = 1, \forall s,t$$

$$K'_{i,m}(s,t) = 0, \text{if } min(|s|,|t|) < i$$

$$K_{i,m}(s,t) = 0, \text{if } min(|s|,|t|) < i$$

$$K'_{i,m}(s,t) = 0, \text{if } m < 2i$$

$$K'_{i,m}(sx,t) = \lambda K'_{i,m-1}(s,t) + K''_{i,m}(sx,t), i = 1,...,n-1$$

$$K''_{i,m}(sx,tu) = \lambda^{|u|} K''_{i,m-|u|}(sx,t), \text{where there is no } k : u_k = x$$

$$K''_{i,m}(sx,tx) = \lambda(K''_{i,m-1}(sx,t) + \lambda K'_{i-1,m-2}(s,t))$$

$$K_{n,m}(sx,t) = K_{n,m}(s,t) + \sum_{j:t_j=x} K'_{n-1,m-2}(s,t[1:j-1])\lambda^2$$

8. STATE-OF-THE-ART TEXT CLASSIFICATION

In recent years, deep neural networks have shown remarkable success in the task of text classification. From an NLP perspective, text classification was tackled by learning a word-level representation using word embeddings and learning a text-level representation used as the feature for classification. Although successful, this kind of encoding-based methods ignores fine-grained details. The most recent idea was proposed by researchers from Shandong University and the National University of Singapore that incorporates word-level matching signals into the text classification task.

The proposed framework for text classification called **EXAM – EXplicit interAction Model**, contains three major components: a word-level encoder, interaction layer, and the aggregation layer. This addresses the problem of incorporating finer word-level matching signals by calculating matching scores between the words and classes explicitly. The word-level encoders is realized using the currently available encoding methods such as FastText or ELMo where each word is assigned a representation which is a function of the entire corpus sentences to which they belong. The interaction layer is based on projecting classes into real-valued latent representations. They use a trainable representation matrix to encode each of the classes to compute word-to-class interaction scores. The final scores are estimated simply using dot product as the interaction function between the target word and each class. Finally, as an aggregation layer, they employ a simple fully-connected two-layer MLP or CNN or LSTM. The MLP is used to compute the final classification logits taking the interaction matrix and the word-level encodings.

9. REFERENCES

[1] H. Lodhi, C. Saunders, J. Sahwe-Taylor, N. Christianini, C. Watkins: Text Classification using String Kernels, Journal of Machine Learning Research 2 (2002), p.419–444.

[2]A. K. Seewald, F.Kleedorfer: Lambda pruning: An approximation of the string subsequence kernel for practical SVM classification and redundancy clustering, Advances in Data Analysis and Classification(2007).

[3]C Du, Zh. Chen1, F. Fen, L. Zhu, T. Gan, L. Nie: Explicit Interaction Model towards Text Classification,Nov. 2018. https://arxiv.org/pdf/1811.09386.pdf