Evaluating Kernel Performance in Multi-Category Hypertext Classification

{Ilya Grigorik, Francis Lau}
ivgrigor@uwaterloo.ca, fkwlau@uwaterloo.ca

Introduction

Support Vector Machines (SVMs) are learning systems that utilize a hypothesis space of separating functions in a high dimensional feature space. With rigorous mathematical foundations from optimization theory and statistical learning theory, this approach first introduced by Vapnik has been shown to outperform many other systems in a variety of applications. One of the successful uses of SVM algorithms is the task of text categorization into fixed number of predefined categories based on their content. Commonly utilized representation of text documents from the field of information retrieval (IR) provides a natural mapping for construction of Mercer kernels utilized in SVM algorithms; when dealing with hypertext and plaintext documents which do not have a natural vector representation, explicit kernel structures have to be constructed, a procedure for which a number of construction algorithms have been proposed and investigated: TFIDF (Bag of Words), String, Syllable and Composite kernels. In this paper we investigate the efficacy of these methods for multi-category classification of hypertext documents. We discuss the theory and computational construction algorithms for each of the aforementioned kernel structures and the possibility of creating and utilizing composite kernel structures to simulate information boosting. We also use the SVMPython package[1] on a set of 1000 categorized documents to empirically evaluate the performance of the discussed kernels.

Composite Kernels

The idea proposed by Joachims et al. (2001) that the performance of a kernel increases when it is the combination of two or more independent kernels has made the task of finding good kernels a lot easier. Rather than trying to capture all the required features into one kernel simultaneously, one can now use Joachims et al.'s results to design and combine smaller kernels. Not only is the task of designing smaller kernels easier, the idea of composite kernels also allows the reuse of existing kernels that are known to have good performance.

To construct a composite kernel, the first step is to verify that all kernels are valid. Joachims et al. (2001) states that as long as a kernel satisfies all the kernel closure properties, it can be classified as valid and thus can be used as a building block of more complex kernels. After the kernels that are designated to be combined into the composite kernel are all classified as valid, two kernels can be combined using the following simple technique (Joachims et al., 2001):

Let $\phi^i : X \to \mathbb{R}^{m_i}$ be the feature mapping corresponding to the kernel $\mathcal{K}_i$, $i = 1, 2$. Now consider the feature vector

$$\phi(x) = [\sqrt{\lambda}\phi^1(x), \sqrt{(1-\lambda)}\phi^2(x)].$$

The corresponding inner product satisfies

$$\begin{aligned}\langle\phi(x), \phi(z)\rangle &= \lambda\langle\phi^1(x), \phi^1(z)\rangle + (1-\lambda)\langle\phi^2(x), \phi^2(z)\rangle \\ &= \lambda\mathcal{K}_1(x,z) + (1-\lambda)\mathcal{K}_2(x,z) \\ &= \mathcal{K}(x,z)\end{aligned}$$

The resulting composite kernel will be better than the two kernels considered separately because it has a smaller soft margin[2] (Joachims et al, 2001). This leads to the conclusion that the composite kernel optimizes the upper bound.

We will now use the results from Joachims et al. (2001) as a basis to study the String Kernel and the Syllable Kernel, both of which utilize the concepts of composite kernels to construct enhanced performance kernels.

Text Classification using String Kernels

In this paper a new kernel construction theme is proposed, where the feature space is generated by all subsequences of length *k* as derived from the document *T* to be classified. As argued by the authors, the simple Bag of Words approach loses all word order information – by discarding stop words, only the words' frequency is used and useful structural information is lost in the process. Instead, they consider documents as symbol sequences and construct the feature space as a set of all (non-contiguous) substrings of k-symbols. In turn, the feature vector similarity is then determined by the number of common substrings each document contains. This approach does not utilize any domain knowledge but is nonetheless capable of capturing topic information, making it a powerful technique for all text-classification applications.

Text Classification using String Kernels

In this paper a new kernel construction theme is proposed, where the feature space is generated by all subsequences of length *k* as derived from the document *T* to be

classified. As argued by the authors, the simple Bag of Words approach loses all word order information – by discarding stop words, only the words' frequency is used and useful structural information is lost in the process. Instead, they consider documents as symbol sequences and construct the feature space as a set of all (non-contiguous) substrings of k-symbols. In turn, the feature vector similarity is then determined by the number of common substrings each document contains. This approach does not utilize any domain knowledge but is nonetheless capable of capturing topic information, making it a powerful technique for all text-classification applications.

An important part of the String Kernel construction lies in the notion that substrings do not need to be contiguous, and the degree of contiguity of one such substring in a document determines how much weight it will have in the comparison. In order to deal with non-contiguous substrings, the authors propose the introduction of a decay factor *$\lambda \in (0,1)$* that is used to weight the presence of certain features in the text. Thus, combining the notions of *k-length* substrings and the decay factor λ, for *$k = 2$* and words **'cat'** and **'car'** we obtain the following matrix:

	c-a	**c-t**	**a-t**	**c-r**	**a-r**
φ (cat)	λ^2	λ^3	λ^2		
φ(car)	λ^2			λ^3	λ^2

Hence, the unnormalised kernel is K(car, cat) = λ^4, and the normalized version is:

K(car, car) = K(cat, cat) = $2\lambda^4+\lambda^6$ *and therefore* *K(car,cat) =* $\frac{\lambda^4}{2\lambda^4+\lambda^6} = \frac{1}{2+\lambda^2}$

Of course, a document to be classified usually contains more than one word, but same procedure can be extended to create a mapping into the feature space for any number of words. Thus, to generate a string kernel:

Let $\sum$ be a finite alphabet. We denote by $\sum^n$ the set of all finite strings of length n, and by $\sum^*$the set of all strings, therefore: $\sum^* = \cup_{\mathrm{n}=0}^{\infty} \sum^n$

And the feature mapping φ for a string s is given by defining the u coordinate $\varphi_u(s)$ for each $u \in \sum^n$. We define: $\varphi_u(s) = \sum_{i:u=s[i]} \lambda^{l(i)}$

However, experimental analysis of this approach has shown that while it is a successful technique for text-classification, it suffers from practical limitations for big datasets and for values of *$k \geq 4$*. Therefore, to scale this approach the authors propose several dynamic programming techniques to try to efficiently approximate the Gram matrices.

The authors then designed three different composite kernels using the results from Joachims et al. to perform an empirical study. Consistent with the results from Joachims et al. (2001), the performance of the system improved when a composite kernel consists of two independent kernels that contain different information. For combination of similar kernels, the system showed no performance improvements, just as Joachims et al. (2001) predicted.

Syllables and other String Kernel Extensions

Syllable Kernel constructions proposed by Shawe Taylor, Craig Saunders and Hauke Tschach aim to reduce the computational complexity of the String kernel construction by utilizing Speech Segmentation techniques widely employed in the field of Natural Language Processing. By analyzing the document as a string of syllables, this approach allows the capture of more fine-grained data relationships as compared to the Bag of Words kernel while avoiding the computational complexity of considering individual characters of the String kernel. Unlike the String kernel, the Syllable kernel captures domain specific information of the phonetic structure of the underlying language of the document. Since the number and relevance of different syllables varies with every language, this approach is not as generic as the String kernel but it allows higher level analysis of data relationships and offers a less computationally complex kernel generation scheme. The authors also propose two extensions to the String kernel to improve the performance of the Syllable kernel. The first suggestion introduces the idea of using different decay values λ for each character to assign different weights to each syllable. Therefore, under this framework the weight of any syllable can be defined in the following fashion:

Let $\sum$ be a finite alphabet. We denote by $\sum^n$ the set of all finite strings of length n, and by $\sum^*$the set of all strings, therefore:

$$\sum^* = \bigcup_{\mathrm{n}=0}^{\infty} \sum^n$$

Define for each symbol $c \in \sum$ its own decay factor $\lambda_c \in (0,1)$. Then the u coordinate of the feature vector for string s is now defined by:

$$\varphi_u(s) = \sum_{i:u=s[i]} \prod_{j=i_1}^{|i|} \lambda_{sj}$$

Thus, the feature **'cart'** for the document **'cartridge'** would receive the weighting: $\lambda_c\lambda_a\lambda_r\lambda_t$ and the weighted string kernel *K* of two strings *s* and *t* is evaluated as:

$$K_{\mathrm{n}}(\mathrm{s},\mathrm{t}) = \sum_{\mathrm{u} \in \Sigma^{\mathrm{n}}} \varphi_{\mathrm{u}}(\mathrm{s})\varphi_{\mathrm{u}}(\mathrm{t})$$

By allowing different values of lambda we are able to incorporate prior domain knowledge such as different weights for different length syllables, or if words can be grouped into syntactical groups (adverbs, verbs, nouns, etc.) then authors propose assigning penalties for adverbs which are usually less likely to change the meaning of the sentence.

The second suggestion proposes the idea of *soft matching* or considering not only 'equal' symbols, but also 'similar' symbols with an extra factor that ensures that soft matches have lower weights than exact matches. This suggestion is similar to evaluating noncontiguous strings in the string kernel except here we only consider predefined similar pairs, ex. {v, f} for strings **'calf'** and **'calves'**. Thus, for each pair {u, v} of similar characters we define a similarity value $A_{u,v}$ which leads to a symmetric matrix:

$$A = (A_{u,v})_{u,v} \epsilon \sum^{n} \in \mathbb{R}_{0}^{|\Sigma^{n}| x |\Sigma^{n}|}$$

Thus, we can define the new kernel K that uses this soft margin technique as:

$$K_{n}(\mathrm{s},\mathrm{t}) = \varphi_{\mathrm{u}}(\mathrm{s})^{\mathrm{T}}\mathbf{A}\varphi_{\mathrm{u}}(\mathrm{t}) = \sum_{\mathrm{u} \in \Sigma^{\mathrm{n}}} \sum_{\mathrm{v} \in \Sigma \mathrm{n}} \varphi \mathrm{u}(\mathrm{s})\mathrm{T}\varphi \mathrm{u}(\mathrm{t})\mathbf{A}\mathrm{u},\mathrm{v}$$

Furthermore, the authors also propose creating a composite kernel by combining several different length-kernels. In their experiment a new syllable kernel was created by combining syllables with lengths 1-4 to produce a new kernel which was weighted in the following way: $K(-,-) = 0.2xK_1 + 0.3xK_2 + 0.4xK_3 + 0.4xK_4$. This method proved to have slightly worse performance on small samples, but for larger datasets it outperformed all the other methods. Thus, through experimentation the authors have demonstrated that phonetic Syllable kernels work in principle and provide a more general form of String kernel which can be applied to different application domains with reduced computational costs and the ability to embed language specific domain knowledge into the kernel.

Empirical Study of Different Kernels

Experiment Setup:

To run our experiment we obtained a list of 1000 unique links classified into nine distinct categories[3]. Each document, which corresponds to a tutorial link from the website, is labeled with respect to one of the following categories:

1. Adjusting Colors
2. Animation and Image Ready
3. Basics and Tools
4. Interfaces and Buttons
5. Miscellaneous
6. Photo and Scanning
7. Special Effects
8. Text Effects
9. Textures

The original dataset contained approximately 1500 unique documents. We pruned 500 documents that contained unavailable links and links not leading to HTML documents. (e.g. pdf, jpg, gif). Next, we randomly picked a sample of 130 links for classification and used the remaining 870 links to train our multiclass SVM algorithm. To generate the $\varphi(x)$ mappings we used the following procedures:

1. Each document was downloaded by an automated script and processed as follows:
    a. All JavaScript, CSS (Cascading Style Sheet), DOM declarations and HTML comments were removed from the HTML source of the document.
    b. Structural HTML tags were removed from the document.
    c. The remaining user-visible (content as rendered by the browser and as seen by the visitor) content was stored for further analysis.
2. Kernels were constructed in the following fashion:
    a. *Syllable Kernel:*
        i. Using the Moby syllable dictionary[4] we identified the set of all distinct syllables (each syllable was given a unique ID number) and computed their weights using the Scrabble weighting scheme[5].
        ii. The feature vector was generated on the basis of the obtained syllable IDs and weights; for each document, every word was split into syllables and the document feature vector was constructed.

b. *Bag of Words (TFIDF):*
   i. Word stemming algorithm was applied to all documents in order to lower the number of overall features.[6] (Based on our experimental results, this proved to be largely redundant because the bottleneck turned out to be the misspelled words – a problem not present in academic and formal documents).
   ii. Word frequency over all documents was computed.
   iii. Feature vectors for each document were constructed, where each word had a unique feature identifier and a weight assigned according to the classic BoW (Bag of Words) formula.

c. *Bag of Words (Global Frequency):*
   i. Same as in the classic TFIDF kernel.
   ii. Global word frequency (total number of occurrences of word w_i) was computed.
   iii. Same as in the classic TFIDF kernel, except new weight values were used.

3. After computing the feature vectors of each document, each vector was normalized, as suggested in Chris Burges' tutorial[7] for *SVM-Light*.
4. Similar procedure was repeated for the remaining 130 classification links to be used later for measuring the performance of our kernels.

Computing Kernel specific feature vectors:

TFIDF Kernel: To run the bag of words approach we parsed the string contents of each page and assigned the weights according to the standard BoW (Bag of Words) formula:

$$\mathrm{weight}(\mathrm{word}) = \mathrm{TF}(\mathrm{w_i}, \mathit{document})\, x \log\left(\frac{n}{DF(w_i)}\right)$$

Where *TF(w_i, document)* is the number of times the word w_i occurs in the document, *n* is the total number of documents and *DF(w_i)* is the number of documents in which the word w_i occurs.

TFIDF-Global Kernel: We modified the BoW scheme such that in *DF(w_i)* now represents the total number of occurrences of the word w_i over all documents. Effectively this allowed us to now modify the value of *n* (total number of documents in classic TFIDF) to reflect a threshold cutoff value where any word that surpasses this limit is omitted in our final analysis. When generating our feature vectors we omitted all negatively valued features, thus removing all words that have surpassed our limit *n*.

Syllable Kernel: To set up the Syllable Kernel, we used the Moby Hyphenator[8] word list to obtain the syllables for 185,000 words. Then, we calculated the weights of the syllables with respect to the character scoring system in Scrabble. For example, the word 'boy' would be given a score of 8 because 'b' has a score of 3, 'o' has a score of 1, and 'y' has a score of 4 in Scrabble. After obtaining the weights (scores) of all the syllables, we constructed the kernel by finding the number of instances a syllable occurs in a document and giving it the appropriate weight. In our Syllable Kernel, the syllables that appeared in the documents make up the feature space.

Running of the experiment:

Using the *SVMPython* package based on the *SVM*^{multiclass} [9] quadratic optimizer we trained our SVM using the obtained feature vectors from the experiment setup. The *SVMPython* code, which comes with a *SVM*^{multiclass} sample implementation was largely unchanged, only minor data processing and setup modifications were introduced to read in our data. Once the SVM was trained we used the resulting model to evaluate the performance of our kernel by feeding the remaining 130 feature vectors and comparing the predicted category IDs with the true labels.

Experiment results:

Feature Vector Kernel	Kernel Type	Prediction Success Rate	Kernel Variables	Correct	Incorrect
TFIDF	Radial Basis Function	64.29	G = 0.5	81	45
TFIDF	Linear	57.94	N/A	73	53
TFIDF Global	**Radial Basis Function**	**67.46**	**G= 0.5**	**85**	**41**
TFIDF Global	Linear	61.90	N/A	78	48
Syllable	Radial Basis Function	54.76	G= 0.5	69	57
Syllable	Linear	49.21	N/A	62	64

Table 1.

From our experiments we can conclude that all three types of kernels perform well. Unlike the binary classification model, our experiments require classifying web-pages into one of the nine distinct categories. Given a naïve-random prediction rule, the expected accuracy rate is only 11.1 percent. However, as Table 1 shows, our SVM multiclass algorithm yields much higher prediction rates over all three types of kernels.

In general, we found that RBF (Radial Basis Function) kernels (predefined SVM^{multiclass} kernel) performed better than the default linear kernel. This increase in performance is likely due to the better fit of a non-linear model and noise smoothing properties made use of by RBF kernel. For all of our tests, we set the value of gamma for RBF kernels to 0.5.

In terms of performance, TFIDF Global kernel was the best performer, followed by the standard TFIDF kernel, and finally the Syllable kernel. We theorize that TFIDF Global kernel had the best performance because we found that only 30% of the words appeared in more than one document. (Even with word stemming, as previously described). In turn, this results in similar weighting for 70% of all features; resulting in a sparse binary representation of each document, where only the presence of a word is recorded.

We were surprised that the Syllable kernel was the worst performer, because Saunders et al. (2002) concluded that Syllable kernels should achieve a better overall performance relative to String and BoW kernels. A possible explanation for this poor result lies in our naïve Scrabble weight scheme, which does not implement any higher level phonetic analysis proposed by the authors and thus may have a poor weight distribution relative to our actual set of documents.

Conclusion

In this paper we have examined three different types of kernels for the task of multi-category classification of hypertext documents. Using recently published papers in the realm of constructing kernels for SVM-based text classification, we investigated the theory behind the String, Syllable and the Composite kernels. Specifically, we analyzed the theory behind the String kernel construction and problems associated with using it in practice. Since the computational aspect of using the String kernel makes it infeasible for practical usage, we explored Syllable kernels, which are similar to String kernels but reduce the computational complexity. Furthermore, we investigated the theory behind Composite kernels which aims to improve the general performance of kernels by combining kernels that contain different information (information boosting).

To test the theory, we empirically evaluated the performance of multi-category classification with the Syllable/String and BoW kernels. Based on our results from running our dataset of 1000 documents on *SVMPython*, we conclude that all kernels performed well. In all four instances, the kernels accurately categorized about 60% of the documents. This is a significantly better result than the expected value of a random categorization, which is only $\frac{1}{9}$ or approximately 11.1 percent.

We speculate that an improvement for a future study would entail using a larger document corpus to obtain a better generalization bound. Also, improving the preprocessing of the document, such as correction of common typographical errors should reduce the number of features and theoretically improve the accuracy of the algorithm.

References:

1. J. Thorsten, N. Christianini, H. Shawe-Taylor. *Composite Kernels for Hypertext Categorization* (2001)
2. C. Saunders, H. Tschach, J. Shawe-Taylor. *Syllables and other String Kernel Extensions* (2002)
3. H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Christianini, C. Watkins. *Text classification using String Kernels* (2002)

---

[1] SVM Struct for Python (Multiclass) is available at: http://www.cs.cornell.edu/~tomf/svmpython/

[2] We are making an implicit assumption that the kernels to be combined capture relative information about the feature space and the application domain.

[3] Dataset was obtained from: http://www.graphics-world.com/Photoshop/
Each document is equivalent to a tutorial link found on the website.

[4] http://www.speech.cs.cmu.edu/comp.speech/Section1/Lexical/moby.html

[5] http://www.thepixiepit.co.uk/scrabble/rules.html

[6] Word-stem dictionary used was obtained from: http://snowball.tartarus.org/algorithms/english/diffs.txt

[7] http://www.kernel-machines.org/papers/Burges98.ps.gz

[8] http://www.speech.cs.cmu.edu/comp.speech/Section1/Lexical/moby.html

[9] http://www.cs.cornell.edu/People/tj/svm_light/svm_struct.html