Augmented Sequence Spectrum Kernels for Semi-Structured Document Categorization

Olivier de Vel
Information Networks Division
Defence Science and Technology Organisation
P.O. Box 1500
Edinburgh SA 5111, Australia

Abstract

We report an investigation into the learning of semi-structured document categorization using an augmented string spectrum kernel. We automatically discover low-level, short-range byte data structure patterns in the document by extracting an augmented string spectrum feature map (called the *broadband sequence spectrum kernel*) and use a modified suffix trie data structure (called the *coloured generalized suffix tree* or CGST) to efficiently store and manipulate the spectrum map. Using the CGST we are able to compute the stream's broadband spectrum Gram matrix and use the SVM classifier algorithm to categorize the document data streams. Experiments with various document data types have provided good classification performance results. Results indicate that the broadband kernel is superior to the standard fixed-length kernel for semi-structured documents.

1 Introduction

As a consequence of the dramatic growth in the volume of information generated and stored on computer networks, there is an increased demand for effectively organizing and exploring this information for supporting human decision making. There are many applications of information organization and exploration. For example, text categorization assigns natural language texts (e.g., news articles) to pre-defined categories [Yang and Liu, 1999], text mining looks for patterns such as structured information or meta-data (addresses, names etc.) in natural language texts, authorship attribution categorizes text based on their author(s) [de Vel, 2000], stream mining looks for patterns in discretized data such as stock prices, click-streams, system call traces etc. [Eskin *et al.*, 2002].

Our application area is computer forensics, which attempts to reconstruct the sequence of events leading to some computer or computer-assisted crime. Computer forensics is becoming increasingly important as computers and/or embedded computer systems (e.g., connected PDAs) are now ubiquitous in almost all crimes. The sources of evidence available to a computer forensics investigator are wide and varied. These include low-level data streams (e.g., hidden blocks of data on a hard disk, network packets), activity streams (e.g., contents of event logs), files, structured data (e.g., databases), and so on. Many data sets in computer forensics are large-to-extreme in size (criminal investigations that involve terabytes of data are not unusual these days) that are highly skewed with the class/category of interest occurring relatively rarely. To make matters worse, in some cases, only fragments of these data are available due to the (possibly) intentional deletion of data by the perpetrator (some data remnants may still be found, for example, in the unallocated file space, i.e. non-file system space on the storage device). Once these data have been extracted, they have to be identified/categorized, patterns extracted, events correlated, link analysis and link discovery undertaken etc. after which the investigator is hopefully able to piece together the case. In this paper we are particularly motivated by the categorization of semi-structured document data streams which may or may not include natural language text. Semi-structured document data will generally have some low-level, generally short-range structure that categorizes the data. Low-level structures include characteristic group sequences, byte block identifiers, mark-up tokens etc. Example semi-structured data are files such as Microsoft Word, Excel and Acrobat PDF documents, images (e.g., JPEGs, GIFs), executables (e.g., DLLs) etc. Some semi-structured document data streams may even be compound streams: embedded documents are examples of these (e.g., zipped documents, a MS PowerPoint document embedded within a MS Word document). The ability to accurately and rapidly categorize such semi-structured documents is of particular importance in computer forensics as criminal investigations typically involve thousands of document streams characterized by a large number of different document types. Most documents stored as files are characterized by a file extension and/or a unique file type signature or *magic number* (a few pre-defined bytes inserted at the beginning of the file). However, some of these documents may have been intentionally tampered with to avoid detection. For example, suspect image files may be renamed with different file extensions and magic numbers or, may be embedded/hidden within other files.

Document categorization can be undertaken using various techniques. Firstly, the simplest method is to use domain experts to identify new documents and allocate them to well-defined categories. In view of the size of computer forensics data sets, this method can be time-consuming and expensive and, perhaps most limiting, is that the method provides no continuous measure of the degree of confidence with which the allocation was made. Secondly, the domain expert can establish a set of fixed rules which can be used to classify new documents. Unfortunately, in many cases, the rule-set can be large and unwieldy, typically difficult to update, and unable to adapt to changes in document content and/or structure. For example, different versions of the same document generating software can produce different rule sets. Finally, categorization can be undertaken automatically by inductively learning the classifiers from training example documents. This approach will, hopefully, generalize well to new, unseen documents and has the advantage that it should able to adapt to a measure of drift in the characteristics of documents.

In this paper we investigate a method for inductively learning categorization classifiers for semi-structured documents. We automatically discover the relevant low-level, short-range byte data structure patterns in the documents by extracting an augmented string spectrum feature map called the broadband sequence spectrum map. We then use a modified suffix trie data structure to efficiently store and manipulate the spectrum map, compute the stream's broadband spectrum Gram matrix and use the Support Vector Machine (SVM) classifier algorithm. The original contribution of this paper is three-fold namely, the classification of short-range semi-structured document data streams, the mapping of the features in these data streams using a broadband k-spectrum kernel representation, and the use of a modified suffix tree (called the coloured generalized suffix tree) to efficiently store the spectrum map and compute the Gram matrix. We first describe data streams in Section 2. We outline the broadband spectrum kernel, Gram matrix computation and Support Vector Machines learning algorithm in Section 3. An efficient data structure for creating the spectrum feature map and for computing the Gram matrix is described in Section 4. We present the experimental methodology and database of documents used in the experiments in Section 5. Validation of the method is then undertaken by presenting results of categorization performance in Section 6. Finally, we conclude with some general observations and present future directions for the work in Section 7.

2 Data Streams

A data stream can be defined as an ordered sequence of tokens $x_1, x_1, \ldots x_l$, that can be read in a fixed order, where the $x_j \in T$ and T is a token alphabet. The token space of all possible elements of the data stream depends on the context of the data stream. For example, the token space for a text document can be the set of words, word stems (referred to as "n-grams" by many authors) or characters ("n-graphs"), the tuple of modify-access-create (MAC) file attribute timestamps for the file representing the text document, the byte string values in a binary document, the set of computer system calls etc.

For the case of semi-structured document data considered in this paper, the size of the token space of the data stream is $|T| = 256$ corresponding to the maximum number of byte types. The sequence structure is characterized by short-range group structures such as mark-up tokens, text words and block delimiters/separators. We hypothesize that each document type will have unique group structures (short-range byte sequences) that characterizes the document class.

3 The Broadband k-Spectrum Kernel and Support Vector Machine Classifier

The fundamental concepts of Support Vector Machines were developed by Vapnik [Vapnik, 1995]. Given a set of labeled (positive and negative) training vectors $(\mathbf{x}_i, y_i), i = 1, \ldots, N$, where $\mathbf{x}_i \in \chi = \mathbb{R}^n$ for some domain χ and $y_i \in \{-1, +1\}$, a hyperplane classifier learns a linear decision boundary or hyperplane with a maximal margin of separation that discriminates between the two classes. To construct the optimal decision function, one solves a convex optimization problem by means of a tractable quadratic programming solution using Lagrange multipliers. That is we find the set of values of α_i (called the support vectors) that maximizes the dual objective function

$$L(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

subject to

$$\alpha_i \geq 0 \quad \{i = 1, 2, \ldots, N\} \qquad \text{and} \qquad \sum_{i=1}^{N} \alpha_i y_i = 0$$

The resulting decision function can be shown to be

$$f(\mathbf{x}) = \operatorname{sign}\left(\sum_{i=1}^{N} \alpha_i y_i (\mathbf{x} \cdot \mathbf{x}_i) + b \right)$$

where b is an offset. See, for example [Hastie *et al.*, 2001], on the derivation of the decision function.

The hyperplane classifier we have seen so far finds the optimal decision boundary in the input space. Mappings into higher dimensional spaces often provide us with greater discriminating power (e.g., the multi-layer Perceptron inherently maps inputs into a high dimensional space ζ and attempts to fit a hyperplane to minimize the training error). The problem with high dimensionality is that the mapping can greatly increase the computation time. For example, a dth polynomial mapping can involve

$$\frac{(d + N - 1)!}{d!(N - 1)!}$$

different monomials (dimensions) which, for the case of a vector of words of length 256 and $d = 4$, yields more than 10^8 dimensions. That is, to extract these product (i.e. monomial) features would be computationally very onerous. There is, however, a way around this problem if the algorithm can be expressed solely in terms of an inner product. This allows us to work in the high dimensional feature space without having to explicitly map into it. The Support Vector Machine classifier is an extension of this concept.

We undertake the mapping $\Phi : \chi \rightarrow \zeta$ using a set of basis functions $\phi_m(\mathbf{x})$, $m = 1, 2, \ldots, M$, repeat the optimization procedure and attempt to learn a decision boundary in the higher (possibly infinite), transformed space.

Representing the transformed input features as $\phi(\mathbf{x}_i) = (\phi_1(\mathbf{x}_i), \phi_2(\mathbf{x}_i), \ldots, \phi_M(\mathbf{x}_i))$ where $i = 1, 2, \ldots, N$, the dual objective function becomes

$$L(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j (\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j))$$

and the decision function is now

$$f(\mathbf{x}) = \mathrm{sign}\left(\sum_{i=1}^{N} \alpha_i y_i (\phi(\mathbf{x}) \cdot \phi(\mathbf{x}_i)) + b \right)$$

We note that the decision function involves the inner product of the basis function transformation $\phi(\mathbf{x})$. Indeed, we do not have to explicitly compute the transformation as we only need to compute the inner product (a similarity measure) of the transformation that is, the *kernel function*, $k(\mathbf{x}_i, \mathbf{x}_j)$ where

$$k(\mathbf{x}_i, \mathbf{x}_j) = \sum_{m=1}^{M} \phi_m(\mathbf{x}_i) \phi_m(\mathbf{x}_j) = (\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j))$$

For the training set $(\mathbf{x}_i, y_i), i = 1, \ldots, N$ we can represent the kernel function in matrix form, known as the *kernel* or *Gram* matrix:

$$GM = \Big(k(\mathbf{x}_i, \mathbf{x}_j) \Big)_{i,j=1}^{N}$$

The $N \times N$ Gram matrix therefore defines the mapping from the input space into the feature space, captures a measure of the similarity between between all pairs of input data items, and acts as the interface between the input data and the SVM learning algorithm.

Aside computational advantages in high dimensions, the concepts of the SVM classifier and kernel function are also appealing for other reasons. Firstly, the SVM is based on the idea of structural risk minimization which minimizes the generalization error (i.e. true error on unseen examples) which is bounded by the sum of the training set error and a term which depends on the Vapnik-Chervonenkis (VC) dimension of the classifier and on the number of training examples. The use of a structural risk minimization performance measure is in contrast with the empirical risk minimization approach used by conventional classifiers. Conventional classifiers attempt to minimize the training set error which does not necessarily achieve a minimum generalization error. Therefore, SVMs have theoretically a greater ability to generalize. Unlike many other learning algorithms, the number of free parameters used in the SVM depends on the margin that separates the data and does not depend on the number of input features. Thus the SVM does not require a reduction in the number of features in order to avoid the problem of overfitting. This property is clearly an advantage in the context of high-dimensional applications, such as text document categorization [Joachims, 1998]. Secondly, we can deal with discrete objects other than vectors to generate a *convolution kernel* function [Haussler, 1999]. These objects need not possess additional structure other than being a set. For example, kernels can be used to model representations that are discrete structures such as parse trees [Collins and Duffy, 2001], graphs, semantic kernels [Cristianini *et al.*, 2001] and sequences such as state sequences or system calls [Eskin *et al.*, 2002]. Highly structured objects can also be dealt with by breaking these down into smaller atomic objects, with the overall kernel being constructed from the atomic kernels. Finally, kernels can be used in conjunction with other machine learning algorithms that can be cast as inner products such as Principal Component Analysis (PCA), regression etc. (see, for example, [Scholkopf and Smola, 2002]). Some well-known kernels have been used for the purpose of natural language text document categorization. These include the word kernel using, for example, a bag-of-words representation or term-frequency-inverse-document-frequency (tfidf) weighting scheme. Unfortunately, these kernels cannot be used in the context of our application since document data streams do not necessarily have byte sequence separators (such as whitespace) or any way of specifying terms within the byte sequence.

A specific type of kernel which has been used successfully with biological sequence data is the *string kernel* or *string sub-sequence kernel* [Lodhi *et al.*, 2002]. It has been used successfully in searching for protein sequence similarity in protein classification [Leslie *et al.*, 2002a][Leslie *et al.*, 2002b]. The string kernel is defined over an input space of arbitrarily long sequences, where each sequence is composed of elements from a set of token symbols T (with size $|T| = l$). The feature space that we use to represent these sequences for a k-order string kernel is an $|T|^k$-dimensional vector space, where each coordinate corresponds to a specific k-length sub-sequence of the input token sequence. Each k-length sub-sequence is obtained by sliding a window of length k across the input sequence with unit step increment. Thus, the mapping function $\phi^{(k)}(\mathbf{x}_i)$ maps the input sequence $\mathbf{x}_i$ into a feature vector whose entries are the frequency of occurrence of the fixed length k-length sub-sequence:

$$\phi^{(k)}(\mathbf{x}_i) = (\phi_{\kappa_1}(\mathbf{x}_i), \phi_{\kappa_2}(\mathbf{x}_i), \ldots, \phi_{\kappa_l}(\mathbf{x}_i))$$

where $\kappa_1, \kappa_2, \ldots, \kappa_l \in T^k$. This is known as the *$k$-spectrum feature map*. This type of feature representation has been widely used, particularly for representing text documents where the token space is usually the set of (printable) characters. The sub-sequence is referred to as an n-graph (or n-gram by some authors), see for example [Huffman, 1995]. We augment this spectral map by defining *all* sub-sequences up to and including sub-sequences of length k. That is, we extract all suffixes within the k-length sliding window:

$$\begin{aligned}
\phi^{(k)}_{(\mathrm{BB})}(\mathbf{x}_i) \quad = \quad & (\phi_{\kappa_1^{(1)}}(\mathbf{x}_i), \phi_{\kappa_2^{(1)}}(\mathbf{x}_i), \ldots, \phi_{\kappa_l^{(1)}}(\mathbf{x}_i), \\
& \phi_{\kappa_1^{(2)}}(\mathbf{x}_i), \phi_{\kappa_2^{(2)}}(\mathbf{x}_i), \ldots, \phi_{\kappa_l^{(2)}}(\mathbf{x}_i), \\
& \ldots, \\
& \phi_{\kappa_1^{(k)}}(\mathbf{x}_i), \phi_{\kappa_2^{(k)}}(\mathbf{x}_i), \ldots, \phi_{\kappa_l^{(k)}}(\mathbf{x}_i))
\end{aligned}$$

where $\kappa_i^{(j)} \in T^j$ for $i = 1, 2, \ldots, l$. We refer this to as the *broadband k-spectrum feature map* which, as we shall see later, provides an improved classification performance compared with the standard k-spectrum feature map.

For example, the structured sequence $\mathbf{x}$ = “<<../CreationDate” (as found in the header stream of Adobe PDF documents) has the broadband k-spectrum feature map features (“.”, “/”, “<”, . . ., “t”, “..”, “./”, . . ., “ti”, “../”, “./C”, . . ., “tio”, “../C”, “./Cr”, . . ., “tion”) and feature map values as the vector

$$\phi^{(k)}_{(\mathrm{BB})}(\mathbf{x}) = (2, 1, 2, \ldots, 1)$$

We can now define the *broadband k-spectrum kernel/Gram matrix* for two input sequences $\mathbf{x}_i$ and $\mathbf{x}_j$ as

$$GM^{(\mathrm{BB})}(\mathbf{x}_i, \mathbf{x}_j) = \phi^{(k)}_{(\mathrm{BB})}(\mathbf{x}_i) \cdot \phi^{(k)}_{(\mathrm{BB})}(\mathbf{x}_j)$$

We usually normalize the kernel matrix to remove any bias due to different data stream lengths:

$$GM^{(\mathrm{BB})}_{\mathrm{norm}}(\mathbf{x}_i, \mathbf{x}_j) = \frac{GM^{(\mathrm{BB})}(\mathbf{x}_i, \mathbf{x}_j)}{\sqrt{GM^{(\mathrm{BB})}(\mathbf{x}_i, \mathbf{x}_i) GM^{(\mathrm{BB})}(\mathbf{x}_j, \mathbf{x}_j)}}$$

In our experiments we do not require any normalization as length of the input data streams is the same.

Previous work on natural language text document categorization and natural language processing using string kernels has been undertaken by various authors, including Lodhi *et al* [Lodhi *et al.*, 2002], Cancedda *et al* [Cancedda *et al.*, 2002] , Teytaud and Jalam [Teytaud and Jalam, 2001], and Collins and Duffy [Collins and Duffy, 2001]. These authors have shown that string kernels in combination with machine learning algorithms such as the SVM classifier and Perceptron may be a useful tool for studying structures in text documents. For example, Lodhi *et al* compared their results with kernels based on standard text categorization methods such as the word kernel (using the tfidf weighting scheme) and bag-of-words kernel and observed better results for small-to-moderately sized data sets. In our application, we deal with document byte streams that typically have a broad byte histogram distribution and only have a small fraction of natural language text. Therefore, as mentioned above, categorization methods that make use of classical text representation techniques where a document is mapped to a high-dimensional feature vector (e.g., bag-of-words) are not applicable and, therefore, cannot be compared with.

In summary, the broadband k-spectrum kernel is a convenient way of representing the similarity among a set of data stream sequences and can conveniently be used in conjunction with learning algorithms. Two sequences that share many matching sub-sequences up to length k will thus have a large broadband k-spectrum kernel value.

4 The Generalized Suffix Tree

We have observed above that the feature space is potentially very large with a dimensionality of up to $|T|^k$, or more than 10^9 nodes for byte sequences and $k = 4$. However, in general, the space is extremely sparse since most document data stream strings do not contain all of the possible k-length sub-sequences which span the feature space. An efficient data structure for generating and storing the broadband k-spectrum feature map, and computing the associated Gram matrix is the generalized suffix tree. The suffix tree is a widely used data structure for text processing.

A suffix tree is a PATRICIA trie built over the set of all suffixes of a given sequence [Bays, 1974]. A trie is an indexing data structure used for indexing sets of key values of varying sizes. The suffix tree is an efficient data structure for storing and aligning sequences as the tree for a sequence of length $L = \mathrm{length}(\mathbf{x})$ can be constructed in $O(L)$ time and the number of tree nodes is generally a linear function of L [McCreight, 1976]. Suffix tree walking and suffix matching/aligning is also very efficient ($O(1)$). An example partial suffix tree for the sequence given in Section 3 is shown in Figure 1.

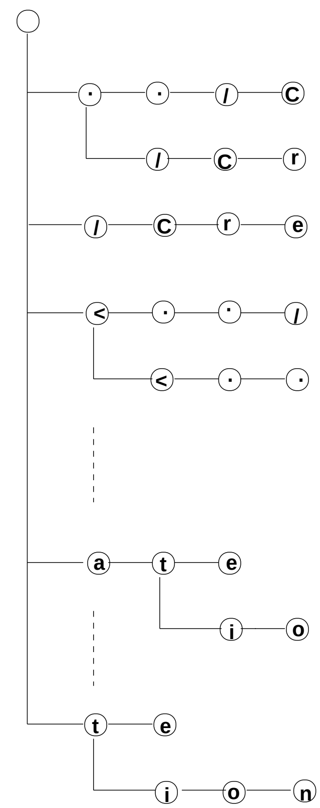

Figure 1: Example partial suffix tree with nodes in lexicographic order the for data stream sequence “<<../CreationDate” with $k = 4$.

The basic suffix tree data structure can be augmented, referred to as the *generalized suffix tree* (GST), by allowing multiple sequences to be stored in the same suffix tree. GSTs have been used as a very efficient data structure for supporting biological sequence alignment [Bieganski *et al.*, 1994]. We further augment the suffix tree representation by storing the frequency of occurrence of each suffix in the GST, giving rise to the *coloured generalized suffix tree* (CGST). Each depth-p node ($1 \leq p \leq k$) of the CGST stores N counts in a vector $(c_1, c_2, \ldots, c_N)$, where each ith vector element count c_i represents the number of times the sub-sequence of length p of $\mathbf{x}_i$ terminates at this node. The CGST thus captures all of the possible sub-sequences in the input sequence (and not just the k-length sub-sequences). The overall cost for constructing the CGST for N sequences is $O(kLN)$ where, in general, $k, N << L$.

To calculate the broadband k-spectrum Gram matrix, the CGST is traversed and, at each node in the tree, each node vector count element pair (c_i, c_j), corresponding to input sequences $\mathbf{x}_i$ and $\mathbf{x}_j$, is multiplied and added to the Gram matrix element count $GM_{i,j}^{(\mathrm{BB})}$. The CGST need only be traversed once, and the cost of the matrix computation is $O(kLN^2)$. The CGST is thus an efficient data structure for representing and manipulating the broadband k-spectrum feature map and for computing the Gram matrix. Our implementation of the CGST is simplified (and thus not as efficient) in that we use unary trie paths rather than compressed unary trie paths and we also do not permit the cross-linking of nodes.

5 Document Corpus and Experimental Methodology

The corpus of document data streams used in the evaluation of document categorization experiments contained a total of 465 documents and five document type classes. To ensure a good mix of document types (e.g., document generator versions etc.), the document corpus was downloaded from a wide range of sources on the Internet. The document class types include Microsoft Word and Excel files, JPEG files, java source code files and Adobe PDF files. All of these document types are characterized by having sections of semi-structured data, particularly in the file front matter. A summary of the global document data stream corpus statistics used in the experiments is shown in Table 1.

All data are scaled to unit standard deviation and zero mean (i.e., $(\mathbf{x}_i - \overline{\mathbf{x}})/S_x$, where $\overline{\mathbf{x}}$ and S_x are the sample feature means and standard deviation, respectively). As Support Vector Machines only compute two-way categorization, N_{DC} two-way, one-against-all classification models were generated, where N_{DC} is the number of document type categories ($N_{DC} = 5$ for our document corpus), and each SVM categorization was applied N_{DC} times. This produced N_{DC} two-way confusion matrices.

To evaluate the categorization performance on the document corpus, we calculate the accuracy, recall (R), precision (P) and combined F_1 performance measures commonly employed in the information retrieval and text mining literature (for a discussion of these measures see, for example, [Witten and Frank, 2000]), where:

$$F_1 = \frac{2RP}{(R+P)}$$

The training-testing sampling methodology used is a 10-fold cross-validation of the entire document set.

To obtain an overall performance figure over all binary categorization tasks, a macro-averaged statistic is calculated [Yang, 1999]. Here, N_{DC} per-category confusion matrices are computed and then averaged over all categories to produce the macro-averaged statistic, $F_1^{(M)}$:

$$F_1^{(M)} = \frac{\sum_{i=1}^{N_{DC}} F_{1,DC_i}}{N_{DC}}$$

where F_{1,DC_i} is the per-category F_1 statistic for category DC_i ($i = 1, 2, , \ldots N_{DC}$):

$$F_{1,DC_i} = \frac{2R_{DC_i}P_{DC_i}}{(R_{DC_i} + P_{DC_i})}$$

However, the macro-averaged statistic penalizes document categories with few documents. To compensate for document frequency, the per-document category confusion matrices are "inversely" weighted by the document category frequency prior to summing each component. We call this the *weighted macro-averaged F_1 statistic*, $\overline{F}_1^{(M)}$.

$$\overline{F}_1^{(M)} = \frac{\sum_{i=1}^{N_{DC}} (1 - w_{DC_i})F_{1,DC_i}}{\sum_{i=1}^{N_{DC}} (1 - w_{DC_i})}$$

where w_{DC_i} is the document frequency weight:

$$w_{DC_i} = \frac{N_{DC_i}}{\sum_{i=1}^{N_{DC}} N_{DC_i}}$$

and N_{DC_i} is the number of documents in author category DC_i.

The SVM classifier used in the experiments was SVM*light* [Uni, 2002], developed by T. Joachims from the University of Dortmund. SVM*light* is an implementation of Vapnik's Support Vector Machines. It scales well to a large number of sparse instance vectors as well as efficiently handling a large number of support vectors. In our experiments we used the linear kernel function and the "LOQO" optimizer for maximizing the margin.

6 Results and Discussion

We report our results presenting the P, R, F_1, and $\overline{F}_1^{(M)}$ statistics for the Support Vector Machines (SVM) classifier. In our experiments, we have studied the effect of two variables, $MinSeqLength$ and $MaxSeqLength$, on the categorization performance. These variables are defined as:

- "minimum subsequence length" $MinSeqLength$: the minimum length (number of tokens) of the suffix within the sliding window, and
- "maximum subsequence length" $MaxSeqLength$: the maximum length (number of tokens) of the suffix within the sliding window ($k \geq MaxSeqLength \geq MinSeqLength > 0$).

$MinSeqLength$ and $MaxSeqLength$ define the token positions in the sliding window that will together determine the length of the suffixes within that window that will be mapped into feature space. See Figure 2 for a definition of these parameters.

For example, for the case of the sequence "<<../CreationDate" described in our previous example (with suffix tree shown in Figure 1) together with a sliding window of length $k = 4$, the values $MinSeqLength = 3$ and $MaxSeqLength = k = 4$ mean that only sub-sequence features "../", "../C", . . ., "tio", "tion" are mapped.

Document Type	Number of Documents	Document Size (Kbytes)	
		Minimum	Maximum
`Microsoft Word`	98	0.946	523.3
`JPEG`	77	4.8	462.3
`Microsoft Excel`	73	14.3	499.2
`Java (source)`	155	0.11	46.5
`Adobe PDF`	62	5.8	461.0

Table 1: Summary statistics of the document data stream corpus used in the experiment.

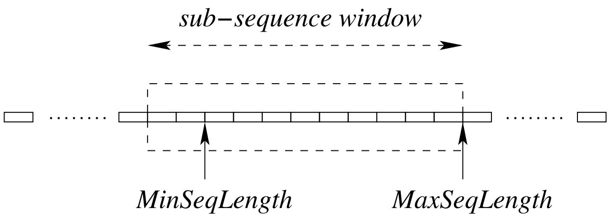

Figure 2: Schematic definition of $MinSeqLength$ and $MaxSeqLength$ parameters for a token sequence. Here, $MinSeqLength = 3$ and $MaxSeqLength = 11$.

However, prior to describing these results, we first address the problem of tuning two other parameters used in our experiments.

6.1 Parameter Tuning

There are two free parameters that can be tuned for optimal performance, namely:

- "minimum subsequence count" $MinSeqCount$: the minimum subsequence frequency, and
- "maximum string length" $MaxStringLength$: the maximum length of the input document data stream.

The value of $MinSeqCount$ determines the degree of sparseness of the feature space. Any CGST tree node in which all of the counts in the node vector is less than $MinSeqCount$ is deleted from the suffix tree. Experiments have shown that a significant compression in the size of the suffix tree can be achieved with even a small non-zero value for $MinSeqCount$.

We investigated the influence of the parameters $MinSeqCount$ and $MaxStringLength$ on the categorization performance. We observed that the per-category performance results for the R, P measures and F_1 statistic are generally poor when the input string length is either too small or too large compared with results obtained with intermediate document lengths. This holds true for all document type categories. The maximum value of $\overline{F}_1^{(M)}$ occurs when an input document data stream length equal to 256 is used. As expected, most of the discriminating features are present in the first 256 or so bytes of the document, where much of the front matter semi-structure information for the document is stored. These results are useful for the purpose of computer forensics because of the frequent existence of partial data (e.g., document files that have been deleted may only have some remnants such as the first file block in the unallocated disk space). Also, the resulting smaller length requirements mean that we can avoid the situation of dealing with large data stream lengths that require larger processing times and memory requirements for the CGST data structure.

We also observed that the categorization performance results are sensitive to the minimum sub-sequence count, for both small and large values of $MinSeqCount$. Only the performance results for JPEG documents are less sensitive to small and large values of $MinSeqCount$, possibly due to the data stream's pseudo-random nature. A choice for $MinSeqCount$ in the range of $8 < MinSeqCount < 15$ was observed to be optimal for all document types.

6.2 Minimum and Maximum Length of Suffix

Here, we present the categorization performance results for the minimum and maximum lengths of the sub-sequences (i.e. suffixes) within the sliding window and the minimum subsequence frequency, respectively. See Tables 2 and 3.

In both Tables 2 and 3, we observe good performance results, generally better than 94%. Table 2 we observe that the value of $\overline{F}_1^{(M)}$ is sensitive to the span of sub-sequences within the sliding window. We observe a maximum value for $\overline{F}_1^{(M)}$ when all sub-sequences within the sliding window (i.e., $MinSeqLength = 1$, the broadband case) are used. Surprisingly, the drop-off in performance is significant for the case of $MinSeqLength = k$ (i.e., the traditional "n-graph" representation). In this case, the performance is only satisfactory for the case of natural language text and JPEG documents. A possible reason for the former is that sub-sequences of n-graph word tokens are minimally significant in natural language texts and that, as we might expect, only larger n-graphs capture the semantics between words. Therefore, in general, we conclude that it is important to include the full spectrum of sub-sequences (rather than just the max. sub-sequence) when categorizing non-pure text data.

Table 3 shows that the effect of the sliding window length ($MaxSeqLength$) on the categorization performance results is not too significant, except when $MaxSeqLength = 1$ which corresponds to the case of the document byte histogram distribution. A satisfactory choice for the value of $MaxSeqLength$ would be in the range $4 \lesssim$

Min. Seq Length (bytes)	Document Type Category, DC_i ($i = 1, \ldots, 5$)															$\overline{F}_1^{(M)}$
	MSWord, DC_1			JPEG, DC_2			MSExcel, DC_3			Java, DC_4			PDF, DC_5			
	P_{DC_1}	R_{DC_1}	F_{1,DC_1}	P_{DC_2}	R_{DC_2}	F_{1,DC_2}	P_{DC_3}	R_{DC_3}	F_{1,DC_3}	P_{DC_4}	R_{DC_4}	F_{1,DC_4}	P_{DC_5}	R_{DC_5}	F_{1,DC_5}	
1	93.0	91.2	92.1	97.4	97.4	97.4	90.7	93.2	91.9	98.7	100.0	**99.4**	96.6	98.3	**97.4**	**95.5**
2	97.8	88.2	92.8	97.3	94.8	96.1	90.8	94.5	**92.6**	98.1	99.4	98.7	93.4	98.3	95.8	95.1
3	93.9	90.2	92.0	97.3	94.8	96.1	89.6	94.5	92.0	95.0	98.7	96.8	96.6	96.6	96.6	94.6
4	96.8	90.2	**93.4**	96.2	97.4	**96.8**	89.6	82.2	85.7	100.0	85.8	92.4	60.4	100.0	75.3	88.4
5	84.5	69.6	76.3	100.0	89.6	94.5	73.3	60.3	66.2	100.0	85.8	92.4	94.7	31.0	46.8	74.2

Table 2: Per-document-type-category P_{DC_i}, R_{DC_i}, F_{1,DC_i} and $\overline{F}_1^{(M)}$ categorization performance results (in %) for different values of the minimum suffix length of the sub-sequences, $MinSeqLength$, that span the sliding window (using a maximum sliding window length $MaxSeqLength = 5$). String kernel parameters used were $MaxStringLength = 256$ and $MinSeqCount = 10$.

Max. Seq Length (bytes)	Document Type Category, DC_i ($i = 1, \ldots, 5$)															$\overline{F}_1^{(M)}$
	MSWord, DC_1			JPEG, DC_2			MSExcel, DC_3			Java, DC_4			PDF, DC_5			
	P_{DC_1}	R_{DC_1}	F_{1,DC_1}	P_{DC_2}	R_{DC_2}	F_{1,DC_2}	P_{DC_3}	R_{DC_3}	F_{1,DC_3}	P_{DC_4}	R_{DC_4}	F_{1,DC_4}	P_{DC_5}	R_{DC_5}	F_{1,DC_5}	
1	76.4	66.7	71.2	97.3	94.8	96.1	65.2	79.5	71.6	99.4	98.7	99.0	91.8	96.6	94.1	86.1
2	87.6	90.2	88.9	97.4	96.1	96.7	88.6	95.9	92.1	100.0	99.4	**99.7**	98.3	98.3	**98.3**	95.0
3	92.9	90.2	91.5	97.4	97.4	97.4	90.9	95.9	93.3	99.4	100.0	**99.7**	98.3	98.3	**98.3**	96.0
4	97.9	91.2	**94.4**	97.4	98.7	**98.1**	92.2	97.3	**94.7**	98.7	100.0	99.4	96.6	98.3	97.4	**96.7**
5	93.0	91.2	92.1	97.4	97.4	97.4	90.7	93.2	91.9	98.7	100.0	99.4	96.6	98.3	97.4	95.5
6	95.9	91.2	93.5	98.7	97.4	98.0	91.0	97.3	94.0	98.7	99.4	99.0	96.6	98.3	97.4	96.3
8	97.9	90.2	93.9	97.3	94.8	96.1	92.2	97.3	**94.7**	98.7	99.4	99.0	93.4	98.3	95.8	95.9
10	96.8	90.2	93.4	97.4	98.7	**98.1**	91.0	97.3	94.0	98.7	99.4	99.0	96.5	94.8	95.4	95.9
12	96.8	90.2	93.4	97.4	98.7	**98.1**	91.0	97.3	94.0	98.7	100.0	99.4	96.6	98.3	97.4	96.4
15	97.9	90.2	93.9	98.7	96.1	97.4	89.9	97.3	93.4	98.7	100.0	99.4	96.6	98.3	97.4	96.2

Table 3: Per-document-type-category P_{DC_i}, R_{DC_i}, F_{1,DC_i} and $\overline{F}_1^{(M)}$ categorization performance results (in %) for different values of the sliding window length, $MaxSeqLength$. All sub-sequences that span the sliding window were used (i.e., $MinSeqLength = 1$). String kernel parameters employed were $MaxStringLength = 256$ and $MinSeqCount = 10$.

$MaxSeqLength \lesssim 7$. A similar effect can be observed with natural language text documents where a window size no larger than approx. 7 characters is sufficient to capture the semantics (see, for example, [Lodhi *et al.*, 2002]). This seems to imply that the classifier is not identifying long-range structures in the data stream. Also, the choice of a large value for $MaxSeqLength$ doesn't contribute more information to the classifier and has the additional downside of increasing the computational space-time demands.

7 Conclusions

In this paper we have investigated the learning of semi-structured document data streams. Our method automatically discovered the relevant low-level, short-range byte data structure patterns from the data streams by extracting an augmented string spectrum feature map (called the broadband sequence spectrum kernel) and using a modified suffix trie data structure (called the coloured generalized suffix tree or CGST) to efficiently store and manipulate the spectrum map. Using the CGST we are able to compute the stream's broadband spectrum kernel/Gram matrix and use the SVM classifier algorithm to categorize the data streams.

Experiments on a number of document types using different parameters (input data stream length, minimum and maximum sub-sequence length, and sub-sequence count) gave several promising results, recording $\overline{F}_1^{(M)} > 94\%$ in most experiments. Firstly, we observed that when we use the complete spectrum of sub-sequences within a data stream window the performance is better than when compared with the standard fixed k-length sequence ("n-graph"). Secondly, a satisfactory classification performance can be achieved with only a small fraction (first 256 bytes) of a document and when using only a relatively small k-length window size ($k \approx 5$). Finally, a moderate value for the sub-sequence count can greatly improve the classification performance (particularly in the range $5 \lesssim MinSeqCount \lesssim 15$) and also significantly reduce the size of the feature space.

There are several improvements to the the current approach that can be investigated. Firstly, the number of document type classes considered in our experiments at the moment is quite small. Secondly, the method appears to be only capturing short-range structure in the input document data stream. The ability to identify larger data structures (e.g., format blocks) would be an advantage. We are currently attempting to overcome these limitations.

8 Acknowledgments

The author would like to thank Jim Bell of the DSTO for implementing the initial version of the suffix tree code.

References

[Bays, 1974] J. Bays. *"The complete PATRICIA"*. PhD thesis, University of Oklahoma, 1974.

[Bieganski *et al.*, 1994] P. Bieganski, J. Riedl, and J. Carlis. "Generalized suffix trees for biological sequence data: Applications and implementation". In *Proc. 27th Annual Hawaii International Conf. on Systems Sciences (HICSS94)*, pages 35–44, 1994.

[Cancedda *et al.*, 2002] N. Cancedda, N. Cesa-Bianchi, A. Conconi, C. Gentile, C. Goutte, Y. Li, J.M. Renders, J. Shawe-Taylor, and A. Vinokourov. "Kernel methods for document filtering". In *Proc. 11th Text Retrieval Conference (TREC-11)*, 2002.

[Collins and Duffy, 2001] M. Collins and N. Duffy. "Convolution kernels for natural language". In *Proc. Neural Information Processing Systems (NIPS2001)*, 2001.

[Cristianini *et al.*, 2001] N. Cristianini, J. Shawe-Taylor, and H. Lodhi. "Latent semantic kernels". In Carla Brodley and Andrea Danyluk, editors, *Proceedings of ICML-01, 18th International Conference on Machine Learning*, pages 66–73, Williams College, US, 2001. Morgan Kaufmann Publishers, San Francisco, US.

[de Vel, 2000] O. de Vel. "Mining e-mail authorship". In *Proc. Workshop on Text Mining, ACM International Conference on Knowledge Discovery and Data Mining (KDD'2000)*, 2000.

[Eskin *et al.*, 2002] E. Eskin, A. Arnold, M. Prerau, L. Portnoy, and S. Stolfo. "A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data". In D. Barbara and S. Jajodia, editors, *Applications of Data mining in Computer Security*. Kluwer Academic Publishers, 2002.

[Hastie *et al.*, 2001] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer-Verlag, 2001.

[Haussler, 1999] D. Haussler. "Convolutional kernels on discrete structures". Technical report, Department of Computer Science, University of California at Santa Cruz, 1999. UCSC-CRL-99-10.

[Huffman, 1995] S. Huffman. "Acquaintance: Language-independent document categorization". In *Proc. 4th Text Retrieval Conference (TREC-4)*, pages 359–371, 1995.

[Joachims, 1998] T. Joachims. "Text categorization with support vector machines: Learning with many relevant features". In *Proc. European Conf. Machine Learning (ECML'98)*, pages 137–142, 1998.

[Leslie *et al.*, 2002a] C. Leslie, E. Eskin, A. Cohen, J. Weston, and W. Stafford-Noble. "Mismatch string kernels for SVM protein classification". In *Proc. Neural Information Processing Systems (NIPS2002)*, 2002.

[Leslie *et al.*, 2002b] C. Leslie, E. Eskin, and W. Stafford-Noble. "The spectrum kernel: A string kernel for SVM protein classification". In *Proc. Pacific Symposium on Biocomputing (PSB-2002)*, 2002.

[Lodhi *et al.*, 2002] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. "Text classification using string kernels". *Journal of Machine Learning Research*, 2:419–441, 2002.

[McCreight, 1976] E. McCreight. A space-economical suffix tree construction algorithm. *Journal of the Association of Computing Machinery (ACM)*, 23(2):262–272, 1976.

[Scholkopf and Smola, 2002] B. Scholkopf and A. Smola. *Learning with Kernels*. MIT Press, 2002.

[Teytaud and Jalam, 2001] O. Teytaud and R. Jalam. "Kernel-based text categorization". In *International Joint Conference on Neural Networks (IJCNN'2001)*, 2001.

[Uni, 2002] University of Dortmund. *Support Vector Machine, SVMLight*, 2002. `www-ai.cs.uni-dortmund.de/FORSCHUNG/VERFAHREN/SVM_LIGHT-/svm_light.eng.html`.

[Vapnik, 1995] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, 1995.

[Witten and Frank, 2000] I. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, San Francisco, 2000.

[Yang and Liu, 1999] Y. Yang and X. Liu. "A re-examination of text categorisation methods". In *Proc. 22nd Int. ACM SIGIR Conf. on Research and Development in Information Retrieval (SIGIR99)*, pages 67–73, 1999.

[Yang, 1999] Y. Yang. An evaluation of statistical approaches to text categorization. *Journal of Information Retrieval*, 1(1/2):67–88, 1999.