Support Vector Machines and Kernel Functions for Text Processing

Celso A.A. Kaestner [1]

Abstract: This work presents kernel functions that can be used in conjunction with the Support Vector Machine – SVM – learning algorithm to solve the automatic text classification task. Initially the Vector Space Model for text processing is presented. According to this model text is seen as a set of vectors in a high dimensional space; then extensions and alternative models are derived, and some preprocessing procedures are discussed. The SVM learning algorithm, largely employed for text classification, is outlined: its decision procedure is obtained as a solution of an optimization problem. The "kernel trick", that allows the algorithm to be applied in non-linearly separable cases, is presented, as well as some kernel functions that are currently used in text applications. Finally some text classification experiments employing the SVM classifier are conducted, in order to illustrate some text preprocessing techniques and the presented kernel functions.

1 Introduction

Text classification, also known as text categorization or topic spotting, is the activity of labeling natural language texts with thematic categories from a predefined set [26]. In recent years, due to the large availability of texts in digital media, the task has gained strong importance and the necessity of being executed by automatic procedures. Some applications that can be formalized as automatic document classification are: (a) spam filtering – a process which tries to discern spam email messages from legitimate emails; (b) email routing – the task of routing an email sent to a general address to a specific address or mailbox depending on its topic; (c) language identification – automatically determine the language of a text; (d) genre classification – automatically determine the genre of a text; (e) readability assessment – automatically determine the degree of readability of a text, either to find suitable materials for different age groups or reader types or as part of a larger text simplification system; and (f) the classical task of document indexing and filing – the task of filing documents according to their content or other specific information.

Most of the work in automatic text classification employ automatic procedures generated by applying Machine Learning – ML – techniques [21]. This work aims to present

[1] Informatics Department
Federal University of Technology – Paraná
Avenida Sete de Setembro, 3165 – Rebouas
80.230–901, Curitiba – Paraná – Brazil
{celsokaestner@utfpr.edu.br}

several kernel functions that are currently employed for text classification. Such functions must be used together with the SVM learning algorithm in order to allow the classifier to be applied in cases where the separation between classes is non-linear.

The following structure is used in this text: Section 2 defines the text classification problem and presents the Vector Space Model, largely employed in Information Retrieval tasks, as well as some derived models; Section 3 presents in some detail the SVM learning algorithm and the underlying optimization problem; it also discusses the 'kernel trick", which is used in order to allow the application of the SVM classifier to non-linearly separable data; Section 4 presents some kernels that are currently used in text-processing applications; Section 5 deals with a illustrative experiment: selected kernels in conjunction with the SVM learning algorithm are applied to a restricted text database; and finally, Section 6 presents some conclusions and discuss possible future works.

2 Text Classification and the Vector Space Model

Text classification deals with the assignment of classes to natural language texts. Formally, text classification can be defined as the assignment $\mathcal{A}$ – normally made by some human expert – of a Boolean value ($True, False$) to each pair $(d_i, c_i) \in \mathcal{D} \times \mathcal{C}$, where $\mathcal{D}$ is a domain of documents and $\mathcal{C} = \{c_1, \ldots, c_{\#\mathcal{C}}\}$ is a finite set of predefined classes. The value $True$ is assigned to (d_i, c_i) indicates the decision to assign the class c_i to document d_i, while the value of $False$ indicates the opposite decision. Here only the restricted case where a document d_i is assigned to one class – called hard classification – is considered. In this case an oracle classifier is a function $\Phi : \mathcal{D} \rightarrow \mathcal{C}$ that correctly computes the class of each document, that is, $\Phi(d_i) = c_j$ if and only if $\mathcal{A}(d_i, c_j) = True$.

In practical applications Φ is unknown, so it is necessary to find a suitable approximation $\widetilde{\Phi}$ so that the values of Φ and $\widetilde{\Phi}$ "coincide as much as possible" [26]. The function $\widetilde{\Phi}$ is in general a decision procedure; according to the ML approach, its construction relies on the availability of a previously classified database. Hence a labeled database is a collection of documents where the set of $True$-assigned pairs $\mathcal{D}_L = \{(d_1, c_1), \ldots, (d_N, c_N)\}$ is known. Here N stands for the size of $\mathcal{D}_L$ (number of documents), and $c_k \in \mathcal{C}$ for all $k \in \{1, \ldots N\}$. $\mathcal{D}_L$ is usually split into two databases: (a) the training database $\mathcal{D}_{TR}$ used to construct $\widetilde{\Phi}$ inductively by observing the characteristics of the documents; and (b) the test database $\mathcal{D}_{TS}$ used to evaluate the effectiveness of the obtained classifier: results obtained by $\widetilde{\Phi}(d_k)$ are compared with the labels c_k for all $d_k \in \mathcal{D}_{TS}$ in order to compute evaluation measures.

It is important to emphasize that in general it is not adequate for an automatic procedure to coincide completely with Φ. In fact, as $\widetilde{\Phi}$ must be used to classify *new* instances – which are not in $\mathcal{D}_L$ – a decision structure that coincides exactly with Φ in $\mathcal{D}_{TR}$ probably will overfit the data and have poor predictive performance. This is a fundamental difficulty in ML applications: the set of all possible data is in general too large to be included in the set of observed examples. Hence the learning algorithm must generalize from the given examples in order to produce a useful output from new data input, and therefore an adequate compromise between generalization and overfitting must be achieved [21].

2.1 The Vector Space Model and Extensions

The first issue in text classification is to find an adequate representation of the documents. The largely employed model in Information Retrieval is the Vector Space Model – VSM – also known as a "bag-of-words" model, which was originally proposed in [25]. According to this model each document $d_i \in \mathcal{D}$ is composed by a set of index terms, or simply *terms*. In the overall collection $\mathcal{D}$ the set of terms is represented as $\mathcal{T} = \{t_1, \ldots t_T\}$, where T is the number of different terms in the collection. If a linear order over $\mathcal{T}$ is considered, each document d_i naturally corresponds to a vector in a T-dimensional space, or $\vec{d_i} = [w_{i1}, w_{i2} \ldots w_{iT}]$, where w_{ij} is the weight of the term t_j in the document d_i. In order to avoid a heavy notation, d_i will be used to refer to a document or to its correspondent vector representation.

There are several ways to define the weight w_{ij} of a term t_j in a document d_i [1]. The simplest one is the Boolean case: $w_{ij} = 1$ if t_j appears on d_i and 0 otherwise. Another possibility is to use $w_{ij} = f_{ij}$, the frequency of the term t_j in d_i, called term frequency or tf scheme. One of the most employed weighting schemes is called $tfidf$ – term frequency inverse document frequency. In this scheme w_{ij} is computed by:

$$w_{ij} = f_{ij} \cdot \log \frac{N}{df_j} \tag{1}$$

where f_{ij} is the frequency of the term t_j in the document d_i, df_j is the number of documents in which the term t_j appears, and N is the total number of documents in the collection. The element df_j is employed to compensate elevate weights given to terms that appear in a great number of documents, and are therefore non-discriminant [1]. According to this model a document collection $\mathcal{D}$ corresponds to a (huge) $N \times T$ matrix $D = [w_{ij}]$.

In the VSM a similarity measure between two documents d_1 and d_2 can be easily computed using the inner product $\langle d_1, d_2 \rangle$ of the corresponding vectors. The same procedure can be used to find the documents related to a given user query, the basic Information Retrieval task. In this case, if the query q is formed by an assignment of weights to the terms in $\mathcal{T}$ given by $q = [w_{q1}, \ldots w_{qT}]$, an ordered list of the relevant documents to the query can be obtained sorting the document list according to the values of the similarity measure between q and each d_i. The inner product is usually replaced by the cosine similarity measure, defined in Equation 2, in order to produce values ranging in the $[-1, 1]$ interval and to normalize according to the document length. This is the basic metric employed in the current Information Retrieval systems and web search engines.

$$cos(q, d) = \frac{\langle q, d \rangle}{(||q|| \cdot ||d||)} \tag{2}$$

Two measures are largely employed to evaluate retrieval results: precision and recall. If $\mathcal{D}_{cor}$ is the set of correct documents that are relevant to a task, as defined by the assignment $\mathcal{A}$, and $\mathcal{D}_{rec}$ are the set of documents recovered by an automatic procedure $\tilde{\Phi}$, then:

$$\text{Precision } = \frac{\#(\mathcal{D}_{rec} \cap \mathcal{D}_{cor})}{\#(\mathcal{D}_{rec}|)} \text{ and Recall } = \frac{\#(\mathcal{D}_{rec} \cap \mathcal{D}_{cor})}{\#(\mathcal{D}_{cor})} \tag{3}$$

The harmonic mean between Precision and Recall, called F-Measure, is also widely employed as a single evaluation metric in the area [1]. These measures are also employed in the evaluation of text classification tasks.

The second issue to deal with is how to extract adequate terms of a document collection. In a typical approach, the documents will suffer several preprocessing steps, in order to avoid redundancies, to aggregate elements with similar semantics and to reduce the dimensionality of the vector space.

The most employed steps are [1]:

1. tokenization: the original text is separated according to a set of delimiters, usually formed by the blank space, end-of-line and tabbing [28]; the characters enclosed by these delimiters will become the text units to be treated in the subsequent steps;
2. case standardization: all the characters are converted to the same case;
3. tag filtering: if the document is an Internet page or a similar hyper document, the corresponding tags are eliminated;
4. stop word removal: "stop words" are common natural language entities that do not carry strong semantics; so, they are considered irrelevant for information retrieval and for text classification purposes. In this category are included articles, prepositions, etc.; a list of stop words for each language is usually available[2].
5. stemming (or lemmatization): text elements with small variations in their lexicon but with the same semantics must be associated in order to produce the same dimension in the VSM; some examples are singular and plural variations of the same word and person and time variations of a verb. Linguistic experts produce specific procedures that convert a word into its "stem" (radix), which is the element employed to represent the corresponding dimension[3].

[2] See for example `http://www.ranks.nl/resources/stopwords.html`

[3] Some stemming algorithms are Porter's `http://tartarus.org/martin/PorterStemmer/` and Lovins' `http://snowball.tartarus.org/algorithms/lovins/stemmer.htm`

6. other term associations: it is possible to associate terms of identical semantics using external dictionaries such as the WordNet[4], or by computing successive terms that always occur together in the text like, for example, “triple heart bypass”, which is considered a triple compound term.

In the the original VSM terms are treated as a set of orthogonal vectors, so they are considered as semantically non-related to the other terms. The association of the terms that correspond to the same dimension is made only in the preprocessing steps – specially by stemming, or by a corrective procedure that takes into account existing correlations.

This is the main reason why Wong et al. [32, 33] propose the Generalized Vector Space Model – GVSM. As previously explained, in the original VSM the similarity between a query q and the set of documents $\{d_1, \dots d_N\}$ can be computed, using matrix notation, by $q.D'$, where D is the $N \times T$ [document x term] matrix and D' is the transpose of D. In the GVSM model this computation is replaced by $q.G.D'$, where G is the term correlation matrix given by $G = [\langle t_i, t_j \rangle]$, that is, a matrix in which each element is the inner product (the similarity) between the terms t_i and t_j.

In Wong et al. [33] the “term-to-term” similarity is deeply analyzed. Their work indicates that:

1. the words can be analyzed from the linguistic (semantic) point-of-view, to obtain synonyms, antonyms, hyponyms, hypernyms, meronyms, etc.; these relations nowadays are included in large linguistic repositories such as the WordNet;
2. statistical term co-occurrence computation in a corpus is a practical way to compute term similarity;
3. a more abstract element than the terms, called a “concept”, can be used to represent document dimensions;
4. a concept is characterized by a set of documents, or, more specifically, corresponds to the maximal subset of documents in the corpus that contains the concept;
5. two concepts are uncorrelated if the intersection between the corresponding sets of relevant terms of each document is empty; the greater the overlap between the sets of documents related to two concepts, the more related theses concepts are.

Perhaps the most important idea that can be extracted from this analysis is the relation between concepts and vectors: (a) concepts are represented by vectors; (b) the determination of a vector basis in the term-space involves the identification of the “fundamental concepts” in the collection, whose corresponding representative vectors are orthogonal; (c) documents and queries are linear combinations of the fundamental concepts.

[4] `http://wordnet.princeton.edu/`

Tsatsaronis et al. [30] propose a GVSM model where the semantic information is derived from word thesauri like the WordNet. The authors claim that the use of the GVSM is limited due to contradictory results obtained when the model obtained by statistical procedures is applied. The challenging problem is to correctly incorporate the semantic information according to a rigorous process and using a theoretically sound framework. They present a GVSM model that exploits WordNet's semantic information, based on a new measure of semantic relatedness between terms. Experimental results show that their approach augment text retrieval performance.

In Latent Semantic Indexing – LSI [9] the proposal is to model the relationships between terms and documents. LSI considers that a text is composed by some "latent" or "hidden" concepts; these elements are obtained from the observed [document x term] matrix D, which is used to estimate the underlying model. According to the authors the process is expected to deal with synonymy and polysemy. Synonymy indicates that there are many ways to refer to the same object, whereas polysemy indicates that one can refer to different concepts using the sign – the same lexical element in the case of texts. The model also implicitly considers that terms are not necessarily independent. In practical applications the D matrix is submitted to the Singular Value Decomposition – SVD – process to derive the particular latent semantic structure model of the text. In fact, if D is the [document x term] matrix, the decomposition is $D = U'SV$, where U and V are formed by the so-called left and right singular vectors and S is the diagonal matrix of singular values. The matrices are related to M "latent" concepts, where M is the rank of D. The SVD decomposition[5] is unique up to certain row, column and sign permutations, that can be avoided if we adopt the convention that the diagonal elements of S are all positive and placed in decreasing ordered according to its lines. Besides, the number of considered concepts can be reduced using only the L most representative ones. To do that it is sufficient to use the computation $D_r = U_r'S_LV_r$, where S_L is the sub-matrix composed by the first L rows/columns of S – the ones associated to the greatest eigenvalues, and U_r, V_r are the corresponding reductions of the matrices U and V to $(N \times L)$ and $(L \times T)$ dimensions, respectively. LSI has been successfully employed for term comparison, document comparison and also for the basic information retrieval task, to select documents from a collection that are associated to a given user query.

Dominich and Kiezer [10] also explore the GVSM. According to them Information Retrieval may be conceived as an application of mathematical measure theory. The authors argue that the classical VSM is characterized by a discrepancy between its formal framework and implementable form: its mathematical foundations are sound, but in general the meanings of the elements are not preserved. So, they propose a solution based on the mathematical measure theory, using a particular fuzzy set theory: the retrieval function is conceived as

[5] SVD is closely related to the standard spectral decomposition of a square symmetric matrix, say Y, into $Y = V'LV$, where V is orthonormal and L is diagonal; in fact, matrices U and V in $D = U'SV$ are composed by the eigenvectors of DD' and $D'D$ respectively, and S^2 is the matrix of the eigenvalues.

the cardinality of the intersection of two fuzzy sets. They give a formal unified background to consider the models VSM, GVSM and LSI. They argue that the inner product is not a necessary ingredient of the VSM, introducing the “Principle of Object Invariance” to handle the situation. Moreover, their view makes it possible to formulate new retrieval methods, such as in linear space in a general basis, entropy-based, and probability-based.

2.2 Alternative Models

In the VSM model the terms are obtained from the original text documents by preprocessing, as previously explained in this section. So, as specific textual elements are used as separators – blank spaces, tabs and newline characters [28], the terms are directly related to the elements of the text that correspond to words and their variations. This is an attempt to preserve, as much as possible, the text semantics given the underlying language.

It is possible, however, to adopt alternative models for text processing. In computational linguistics an n-gram is a contiguous sequence of n items from a given sequence of text or speech, where n is a parameter [29]. In the case of text documents, an n-gram is usually constituted by ASCII characters. Normally the n-grams are obtained from the lexical tokens of the text – after the execution of the preprocessing steps 1 to 3, or 1 to 4 as described previously. For example, the lexical element “processing” gives origin to the following set of 5-grams: {`$proc,proce,roces,ocess,cessi,essin,ssing,sing$`}. The `$` mark is normally employed to indicate the beginning and the end of the lexical element. In the n-gram model, a text document is simply a list – considered ordered or not – of the corresponding n-grams of the lexical elements of the text. So, the document can be directly associated to a big n-grams string. Each n-gram can also be associated to a space dimension. For a given text the obtained n-gram vector space is usually larger than the one obtained with the classical VSM, since the number of n-grams is bigger than the number of VSM terms.

The n-gram model was successfully applied in text classification [5]. It must be observed that this approach is linguistic independent, so it can be applied to documents in any particular natural language.

Lodhi et al. [20] propose a different and more radical approach. The authors consider documents simply as ASCII symbol sequences – or strings, which are directly treated by the classification algorithm. In order to do so, they employ the SVM classifier and makes use of specific kernels. The approach does not use any previous linguistic knowledge, but it is capable of capturing topic information: the more substrings two documents have in common, the more similar they are considered. Cancedda et al. [4] also use SVM and symbol kernels. They propose a slightly different approach, that uses as basic symbols sequences of words rather than sequences of characters. The SVM classifier is discussed in detail in the next section, and kernels for text processing are discussed in Section 4.

One important note must be made in order to relate text documents with the almost new applications that appear in the Internet era. Web pages, hyper-documents, tagged documents (HTML, XML) can also be treated by the same models employed for the ordinary text documents, with minor modifications, as recently discussed in [11].

3 The SVM Classifier

This section presents the SVM, nowadays one of the most employed learning algorithms. It was initially proposed by Vapnik; a detailed description of the current version can be found in the references [2, 3, 8].

The following explanation is restricted to the two-class problem. Consider that exists a training set $\{(x_1, y_1) \dots (x_m, y_m)\}$, where the labels are $y_i = +1$ for one class and $y_i = -1$ for the other. All points are embedded in the n dimensional space $\mathbb{R}^n$. Our aim is to find a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ (within a given family of functions) that, when applied to the points x_i, evaluated positive on the first class and negative on the second class, i.e.: $f(x_i) > 0$ when $y_i = 1$ and $f(x_i) < 0$ when $y_i = -1$. When these inequalities hold, f (or its 0-level set $\{x | f(x) = 0\}$) classifies (or separates, discriminates) the two sets of points.

In SVM f is defined as an affine function $f(x) = w' \cdot x - b$. So we must have $y_i(w' \cdot x_i - b) > 0$ for all points. Since this inequality is homogeneous in w and b, it is feasible if and only if the following non-strict inequality holds, for variables w and b:

$$y_i(w' \cdot x_i - b) \geq 1 \tag{4}$$

If feasibility occurs, the points are linearly separable and there are infinite affine functions that can be used in classification task. So, one idea is to fix f in a position that is "equally" separated from the two sets. The situation is depicted in Figure 1 (a). For the first class equation $w' \cdot x_i - b = 1$ holds for points in the hyperplane H_1, which has a normal vector w and the distance to the origin $m_+ = |1 + b|/||w||$. Similarly for the second class the equation $w' \cdot x_i - b = -1$ is satisfied by points in the hyperplane H_2, with the same normal vector and distance to the origin $m_- = |-1 + b|/||w||$. The desired optimum separation hyperplane is the bisector of H_1 and H_2. The distance between H_1 and H_2 – or *margin* – is given by $|m_+ - m_-| = 2/||w||$; so, if we minimize $||w||$ we will maximize the margin. Therefore the SVM decision function can be found as the solution of the following Quadratic Optimization problem – QOP, with a convex function and linear restrictions:

$$\begin{array}{l} min \;\; (1/2)||w||^2 \\ \text{subject to: } \{ \;\; y_i(w' \cdot x_i - b) \geq 1 \end{array} \tag{5}$$

In the previous development the feasibility condition $y_i(w' \cdot x_i - b) \geq 1$ for all x_i was assumed; this is usually called *hard margin SVM*. In practical applications this is rare: usually data is not linearly separable. Even if a complex decision boundary could be obtained, a exact function that separates training data is sometimes undesirable, due to the ML “overfitting” problem: if the data has noise and outliers, a smooth decision boundary that ignores a few data points is better than one that loops around the outliers [21]. A slightly modification in the problem formulation can be done to solve this situation. Supplemental “slack variables” s_i are introduced in the original inequalities, leading to the following formulation:

$$\begin{array}{l} min\ \ (1/2)||w|| + C\sum_i s_i \\ \text{subject to:} \left\{ \begin{array}{l} y_i(w' \cdot x_i - b) - s_i \geq 1 \\ s_i \geq 0 \end{array} \right. \end{array} \tag{6}$$

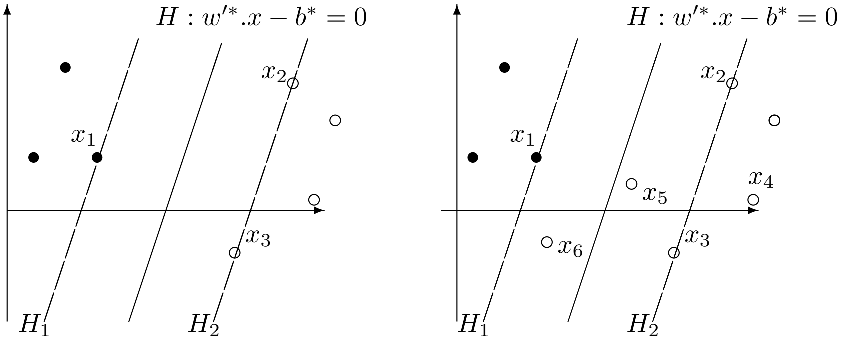
where C is a parameter that establish a trade-off between the margin size and the error in training. When C is reduced more data can lie on the wrong side of the decision hyperplane; this data is treated as outliers, producing a smoother decision boundary. This formulation is usually called *soft margin SVM*, depicted in Figure 1 (b). Note that the point x_5 is between H_2 and the linear decision function H, and x_6 is even in “the wrong side” of H; these situations are compensate by particular values of the problem variables.

Optimization problems sometimes are not directly solved. Instead it is useful to compute the corresponding *dual problem*, obtained from the Lagrangian of original problem [2]. In the case of the hard margin SVM the obtained dual problem is given in Equation 7, where the λ_i are the Lagrange multipliers which are associated to problem restrictions.

$$\begin{array}{l} max\ \ \sum_{i=1}^{m} \lambda_i - (1/2)\sum_{i=1}^{m}\sum_{j=1}^{m} \lambda_i \lambda_j y_i y_j (x_i' \cdot x_j) \\ \text{subject to:} \left\{ \begin{array}{l} \sum_{i=1}^{m} \lambda_i y_i = 0 \\ C \geq \lambda_i \geq 0 \end{array} \right. \end{array} \tag{7}$$

Remarkably, this dual problem depends only on dot products $(x_i' \cdot x_j)$ of the training points, and neither the slack variables s_i nor their associated multipliers appear in the formulation. Therefore, the decision function can be easily found as the solution of the dual problem.

For a multi-class problem with m classes the generalization of the previous presentation is straightforward: to each one of classes a weight vector w_i and a bias b_i (for $i = 1, \ldots m$) are associated. So, the input space is split into m simply connected and convex regions. The decision function is given by $c(x) = argmax_{1 \leq i \leq m}(\langle w_i \cdot x \rangle + b_i)$. Geometrically this is equivalent to associate a hyperplane to each class, and assign to a new point x the class whose hyperplane is furthest from it [6]. Learning algorithms that compute the m hyperplanes simultaneously from the data exist and are extensions of the two-class algorithm previously outlined. Another option is to employ several two-classes classifiers and combine the results using the “one-against-all” approach [21].



(a) hard margin SVM (b) soft margin SVM

Figure 1. Support points, separating hyperplanes and optimal separation hyperplane

The SVM QOP can be solved in several ways [24]. The original proposal by Vapnik uses a method know as “chunking”: the algorithm uses the fact that the value of the quadratic form is the same if the rows and columns of the matrix that corresponds to zero Lagrange multipliers are removed. Therefore, the original problem can be broken down into a series of smaller optimization problems. Similar approach was proposed by Osuna, et al. [22]; the authors prove that a large QOP can be broken down into a series of smaller sub-problems; so, at each step the overall objective function is reduced and a set of feasible point is maintained, assuring the convergence of the method. More recently Platt et al. [24] propose the Sequential Minimal Optimization – SMO, a simple algorithm that can quickly solve the SVM QOP. SMO decomposes the overall QOP into sub-problems, using Osuna’s result to ensure convergence. It starts from the smallest possible optimization problem at every step: for the standard problem this means that it starts using only two Lagrange multipliers, because they must obey a linear equality constraint. So, at every step, SMO chooses two Lagrange multipliers to jointly optimize, finds the optimal and updates the partial solution to reflect the new optimal values [6, 24].

There are some available implementations of the SVM learning algorithm. The WEKA platform[6] includes the SMO algorithm. Other implementations are LIBSVM[7] and SVM-Light[8]. It is also possible to apply a generic optimization algorithm, such as the ones that appear in the R language and environment for statistical computing[9]. Using R it is possible to solve the SVM QOP using a primal-dual method of the standard quadratic programming

[6] `http://www.cs.waikato.ac.nz/ml/weka/`

[7] `http://www.csie.ntu.edu.tw/ cjlin/libsvm/`

[8] `http://svmlight.joachims.org/`

[9] `http://www.r-project.org/`

solver `quadprog` of package `quadprog` or the interior point method `ipop` of package `kernlab` [17].

3.1 The Kernel Trick

The "pure" SVM method is basically a linear separation classifier. To apply the algorithm in more general contexts, where complex (non-linear) decision functions are required, one possibility is to map the original data into high dimensional spaces, where data images are expected to be "well-behaved" for classification purposes. However, the bigger is the dimension of the considered space the bigger are the computational issues, and one must also consider the generalization theory problem – the curse of dimensionality [13].

In order to avoid these problems the so-called "kernel trick" is employed [7, 15, 27]. This trick solves the computational problem of working with many dimensions – it is even possible to work with infinite-dimensional spaces – and also presents other conceptual and practical advantages. The learning process using nonlinear SVM consists of two steps: (a) initially, the input vectors are transformed into high-dimensional feature vectors, where is expected that the data can be will gain meaningful linear structure and will be linearly separated; (b) secondly, the SVM learning algorithm is applied to find optimum margin hyperplane in the new feature space. This separating hyperplane is a linear function in the transformed feature space, but its inverse mapping is a nonlinear structure in the original input space.

The situation is exemplified in Figure 2 for a two-class problem embedded in the $\mathbb{R}^2$ input space. At left are shown points in the original input space, which are not linearly separable. If the mapping $\varphi : \mathbb{R}^2 \to \mathbb{R}^3$ defined by $\varphi(x_1, x_2) = (x_1^2, x_1x_2, x_2^2)$ is applied, we obtain the situation depicted at right. Now data points are linearly separable, so the SVM learning algorithm can be applied.

Let $\varphi : \mathbb{R}^n \to \mathbb{R}^p$ be a (nonlinear) mapping from the input space $\mathbb{R}^n$ to a higher-dimensional feature space (or inner-space) $\mathbb{R}^p$, usually with $p >> n$. If x is a vector in the $\mathbb{R}^n$, the hyperplane that corresponds to the decision boundary in the feature space is defined as $w' \cdot \varphi(x) - b = 0$, where w denotes a weight vector that can map the training data in the high dimensional feature space to the output space, and b is the bias. Using the mapping φ this weight becomes $w = \sum_i \lambda_i y_i \varphi(x_i)$, and the decision function for the hard-margin SVM can be computed by:

$$f(x) = \sum_{i=1}^{m} \lambda_i y_i \varphi(x_i)' \cdot \varphi(x) - b \tag{8}$$

Similarly, in the case of the dual soft-margin SVM, the problem formulation corresponds to the Equation 7 where the inner products $(x_i' \cdot x_j)$ are replaced by $\varphi(x_i)' \cdot \varphi(x_j)$. It is

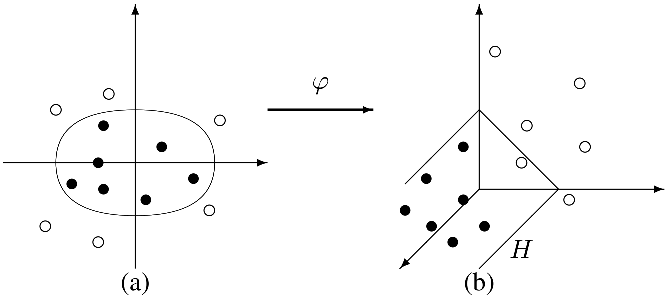


Figure 2. A geometric representation of the kernel trick: (a) original input space; (b) high dimensional feature space, where points are linearly separated by the hyperplane H.

important to observe that the feature mapping function φ references both in the optimization problem and in the decision function always appear in the scope of inner products of training data. So, in fact it is not necessary to know explicitly the mapping φ : it is sufficient to know the results of the dot product of the training data pairs in the feature space. This observation leads to the "kernel trick": the inner product of vectors in the feature space is replaced by a kernel function K that computes $K(x_i, x_j) = \langle\varphi(x_i)' \cdot \varphi(x_j)\rangle = \varphi(x_i)' \cdot \varphi(y_j)$.

For a function K to be a valid kernel it is necessary and sufficient to satisfy the so-called Mercer conditions [3, 6]. For any function g with finite norm $\int g(x)^2dx < \infty$ we must have:

$$\int K(u,v)g(u)g(v)dudv \geq 0 \tag{9}$$

The Mercer's theorem assures that the kernel function always computes an inner product between pairs of input vectors in some high-dimensional space, using only input vectors in the original space. The kernel matrix (also denoted as K) can be considered the central structure in kernel machines, since it contains all necessary information for the learning algorithm: it fuses information about the data and about the selected kernel. It is easy to see that the K matrix is a Gram matrix and therefore is symmetric and positive definite; conversely, any symmetric positive definite matrix can be regarded as a kernel matrix, that is, as representing an inner product in some (unknown) feature space. A very interesting property of the kernels is that they are closed under certain linear operations. If K and K' are kernels, them $aK + bK'$ for constants $a > 0, b > 0$ are also kernels. So, it is possible to construct more complex kernels from simple ones, according to modular compositions [6]. It is also possible to consider a kernel simply as a Gram matrix whose elements are inner-products of pairs of data, allowing the construction of kernels directly from features of the training points.

Cristianini et al. [6] also give some practical considerations regarding kernel matrices: (a) a bad kernel is represented by a matrix which is mostly diagonal: all the points orthogonal to each other, there are almost no clusters and no underlying structure; (b) if the mapping φ ranges in a space with too many irrelevant features, the kernel matrix becomes almost diagonal; (c) it is necessary to have some prior knowledge of the target application in order to choose a good kernel, since the classification performance is highly sensitive to the kernel. In the case of text processing, some appropriate options are discussed in the following section.

The most employed kernels for classification problems in general domains are: (a) the polynomial kernel of degree d: $K_d(u, v) = (u \cdot v + 1)^d$ for any positive integer d; (b) the Radial Basis Function – RBF – kernel: $K(u, v) = exp(-\gamma||u - v||^2)$, for $\gamma > 0$; and (c) the Sigmoid kernel: $K(u, v) = \tanh(\kappa u + v + c)$, where parameters c and κ defined the center and the width of the sigmoid. A discussion on kernel construction and parameter selection for several applications is available in `http://www.kernel-machines.org/`. However up to now most of the work in kernel selection is empiric. The study of adequate functions φ or of the manifold obtained by a kernel application is still an open issue. Besides SVM other algorithms are capable of operating with kernels, such as Gaussian processes, kernel discriminant analysis (an extension of Fisher's linear discriminant analysis), kernel component analysis (an extension of principal components analysis), and correlation analysis [7, 15, 27].

4 Kernels for Text Processing

The use of the SVM classifier in text applications was initially proposed by Joachims [16], which used the classical vector space model. In his work the author gives several reasons why SVM should perform well in text applications: they have high dimensional input space, but few irrelevant features; on the other hand, the document vectors are sparse, and most text categorization problems are linearly separable. Experiments were made using the "ModApte" split of the Reuters-21578 dataset[10] for the 10 most frequent classes of the "topics" classification; the obtained average values of precision/recall-breakeven point for these 10 classes (the point where precision = recall) were 0.860 and 0.864 for polynomial and RBF kernels respectively. Subsequently it was shown that the choice of the kernel has a great impact in the classification performance; since then several specific text kernels were proposed.

Haussler [14] propose a new method of constructing kernels for elements with discrete structures, like strings, trees and graphs. The obtained kernels – called convolution

[10] `http://www.daviddlewis.com/resources/testcollections/reuters21578/`

kernels – generalize the family of RBF kernels, and can also be built from different distributions, such as regular expressions, Hidden Markov Models and ANOVA (analysis of variance) decompositions. Normally features must be extracted from discrete structures. If $\mathcal{S}$ is a class of discrete structures and a finite number d of features is extracted, the process can be represented as a mapping $\mathcal{S} \to \mathbb{R}^d$; if infinitely many features are extracted, the mapping is from $\mathcal{S}$ into the Hilbert space of all square-summable sequences, denoted ℓ^2. In the last case, given two sequences $s = [s_1 s_2 \ldots]$ and $t = [t_1 t_2 \ldots]$, the inner-product in ℓ^2 defined by $K(s,t) = \langle s, t \rangle = \sum_i s_i \cdot t_i$ is a kernel. Indeed, for all $s_1 \ldots s_n \in S$, and for constants $c_1, \ldots c_n \in \mathbb{R}$, we have $\sum_{i,j=1}^{n} c_i c_j K(s_i, s_j) = \sum_{i,j=1}^{n} c_i c_j \langle s_i, s_j \rangle = \left\langle \sum_{i=1}^{n} c_i s_i, \sum_{j=1}^{n} c_j . s_j \right\rangle = \langle \sum_{i=1}^{n} c_i s_i, \sum_{i=1}^{n} c_i . s_i \rangle \geq 0$. As mentioned, an element s_i of the sequence s can be thought as the result of a feature extraction function ϕ_i. Therefore under reasonable assumptions on $\mathcal{S}$ and K every kernel can be represented as $K(s,t) = \sum_i s_i \cdot t_i = \sum_i \phi_i(s) \cdot \phi_i(t)$ for some choice of extraction functions ϕ_i. Thus, in some sense to choose a kernel is the same of choosing a series ϕ_i of extraction functions to represent each $s \in \mathcal{S}$. The main conclusion is that to deal with sequentially-structured data an adequate inner-product defined over the extracted features can be employed directly as a kernel function.

Lehmann and Shawe-Taylor [19] discuss kernels in the context of text classification using a probabilistic framework. They introduce a novel view of how documents in a set $\mathcal{D}$ might have been created: the key concept is to think in a document as a realization of some abstract or prototypical topic-dependent element or templates in a set τ, and to think about words w that appear in the documents as observed features of these documents and templates. Then kernels can be derived from likelihoods $p(w|\mathcal{D})$ and $p(w|\tau)$, and probabilities that take into account the similarity of documents. They show how the popular $tfidf$ weighting scheme can be derived as a natural consequence of their framework. They evaluate the proposal using a subpart of the Reuters corpus, the Corpus Vol.1 Newswire database. F-measure obtained values range from 0.65 to 0.95, according to several kernel and parameters options.

Lodhi et al. [20] propose a radically different approach to generate a text kernel. The proposal, called string subsequence kernel – SSK, is an inner-product in the feature space formed by all the subsequences of length k of the strings that compose the texts. Considered subsequences are not necessarily contiguous: they are weighted by an exponentially decaying factor $\lambda \in [0,1]$ in order to emphasize the symbol occurrences that are close to contiguous. When comparing two sequences, the more substrings in they have in common, the more similar they are. For example, consider the words `cat`, `car`, `bat` and `bar` [20]. If we use as parameters $k = 2$ as subsequence length and a decay factor λ, an 8-dimensional vector space is obtained, as shown in Table 1.

So, the un-normalised kernel value between `car` and `cat` is $K(\texttt{car}, \texttt{cat}) = \lambda^4$,

	c-a	c-t	a-t	b-a	b-t	c-r	a-r	b-r
$\psi(\texttt{cat})$	λ^2	λ^3	λ^2	0	0	0	0	0
$\psi(\texttt{car})$	λ^2	0	0	0	0	λ^3	λ^2	0
$\psi(\texttt{bat})$	0	0	λ^2	λ^2	λ^3	0	0	0
$\psi(\texttt{bar})$	0	0	0	λ^2	0	0	λ^2	λ^3

Table 1. An example of mapping strings into the SSK feature space

whereas the normalised value is obtained by $K(\texttt{car},\texttt{car}) = K(\texttt{cat},\texttt{cat}) = 2\lambda^4 + \lambda^6$, and therefore $K(\texttt{car},\texttt{cat}) = \lambda^4/(2\lambda^4+\lambda^6) = 1/(2+\lambda^2)$. In general a document contains several words, but the mapping for the whole document is into one feature space: the catenation of all the words and spaces (ignoring the punctuation), considered as unique sequences.

A formal definition of the SSK kernel is as follows. Let $s = [s_1, s_2 \ldots s_{|s|}]$ be a sequence over a finite alphabet Δ (i.e. $s_i \in \Delta$ for all i). Let $i = [i_1, i_2 \ldots i_k]$ with $1 \leq i_1 < i_2 \ldots < i_k \leq |s|$ be a subset of the indexes (not necessarily contiguous) in s, and let $s[i] \in \Delta^k$ denote the subsequence of s given by $[s_{i_1}, s_{i_2} \ldots s_{i_k}]$. If $l(i)$ is the length spanned by $s[i]$, that is $l(i) = i_k - i_1 + 1$, then the SSK kernel value between two strings s and t is defined as:

$$K_k(s,t) = \sum_{u \in \Delta^k} \sum_{u=i:s[i]} \sum_{j:u=t[j]} \lambda^{l(i)+l(j)} \tag{10}$$

The direct computation of the features would involve $\mathcal{O}(|\Delta|^n)$ time and space for an alphabet Δ, since this is the number of features involved. So, in order to allow the use of the technique, the authors propose an algorithm based on a dynamic programming to compute the inner products recursively as required by the SSK kernel[11].

Experimental evaluations were conducted in the “ModApte” split of the Reuters database for the top 10 most frequent classes, in order to compare with the results of the standard word feature space kernel [16]. Several variations were tested, including modifications in the parameters k and λ and kernel combinations. Also, documents were considered as composed by the original text words sequences, and also by sequences formed by n-grams obtained from the words of the text. In summary, experiments show positive results on modestly sized datasets; for the ten most frequent classes, obtained F-measures range from 0.691 to 0.982 depending on the considered class. These results are somewhat surprising, since in string kernels only consider the low-level information provided by the string sequences.

[11] For more details of the SSK formal definition and its dynamic programming implementation see [20].

Cancedda et al. [4] extend the idea of sequence kernels to process documents as sequences of words. The authors claim that their approach, called “word sequence kernel” (WSK), has the following advantages: (a) it is more efficient computationally, because less features are considered; and (b) it ties in closely with standard linguistic preprocessing techniques. The authors argue that the high performance obtained by the SSK kernel is due to the fact that the most relevant subsequences for classification correspond to noisy versions of word stems. So, sequence kernels operating at the word level might be more effective than those operating at the character level. They present the WSK kernel method as an extension of SSK method that deals with symbol-dependent and match-dependent decay factors, also using n-grams to pick parts of stems of words or even parts of stems of consecutive words. Experiments were conducted in the “ModApte” split of the Reuters database, using some preprocessing techniques, such as stop word removal, stemming and ambiguity resolution with the aid of a part-of-speech tagger. Several variations of the WSK were tested, including ones where the words obtained by preprocessing were the primitive text elements, and others where n-grams were used as primitive text elements. Also, several parameter sets (substring length k, decay factor λ) and different kernels were tested. In the results the F-measure values range from 0.828 to 0.911.

Katrenko and Adriaans [18] discuss local alignment kernels – LAK – in the context of the relation extraction from natural language. Local alignment kernels are based on the Smith-Waterman measure between strings, which was initially employed in the biological domain. According to this measure two sequences s and t are considered similar if they have many local alignments $A(s,t)$ with high scores. Given the sequences $s = [s_1, s_2 \ldots s_n]$ and $t = [t_1, t_2 \ldots t_m]$ the Smith-Waterman distance – SW – is defined as the local alignment score of their best alignment $SW(s,t)$, computed by:

$$SW(i,j) = \max : \begin{cases} 0 \\ SW(i-1,j-1) + d(s_i,t_j) \\ SW(i-1,j) - g \\ SW(i,j-1) - g \end{cases} \tag{11}$$

where $d(s_i, t_j)$ is the substitution score, which describes the rate at which the element s_i in the sequence changes to the element t_j, and g is a gap penalty. The direct application of the SW score is not a valid kernel, but a valid one can be obtained by summing up the contribution of all possible local alignments, or $K_{LA}(s,t) = \sum_{a \in A(s,t)} \exp(\beta\sigma(s,t,a))$. In the equation $\sigma(x,y,a)$ is a score of a local alignment a of sequences s and t, and A denotes the set of all possible alignments. It is also shown that in the limit the LAK approaches the SW score: $\lim_{\beta \mapsto \infty} \log(\frac{1}{\beta} K_{LA}(s,t)) = SW(s,t)$ for a parameter β. The substitution score must be computed from the similarity between elements of sequences; the authors employ the Cosine, Dice and Euclidean norms to compute such scores. Experiments were conducted in some biomedical corpora and the TREC 2006 Genomics collection[12], and obtained results suggest that the LAK kernel provides promising results,largely outperforming a baseline in some cases. Obtained F-measures range from 0.773 to 0.805 in the effective experiments.

Halawi et al. [12] recently proposed a new approach for compute word relatedness in a text collection. The authors argue that prior work on computing semantic relatedness of words focused on representing their meaning in isolation, effectively disregarding inter-word affinities. Conventional approaches – such as the ones that use the VSM model – compute semantic relatedness by representing the meaning of individual words as dimensions in a multidimensional space, and by computing the distance between the resultant word vectors. So, they propose a large-scale data mining approach for learning word-word relatedness, where known pairs of related words impose constraints on the learning process. The authors also argue that if the word relatedness of the text terms is known, applications such as information retrieval or text clustering can be done by using GVSM (see Section 2.1).

In the proposed process initially sequences of size k composed by words are extracted from the sentences of the text. Then the words – and indirectly the sequences – are mapped into a joint dimensional space of dimension l, referred as the latent factor representation[13]. If w is a word, its latent representation is $q_w \in \mathbb{R}^l$, and sequences s are mapped to the mean of their word vectors. Latent space dimension typically varies from $l = 100$ to 200, which gives a good trade-off between time and accuracy. The latent space representation strives to capture the semantics of the words, such that affinities in the latent space reflect semantic relations. To relax computations, additional biases b_w are associated to words; common words generally have high bias values, reflecting that they frequently co-occur with many other words; on the other hand words that “strongly explain” their context have low bias values.

The set of parameters Θ of the model is composed by the latent factors q_w and biases b_w. They are computed as a solution of an optimization problem, the maximization of the log-likelihood L of the training set. The likelihood of observing the word w within sequence s can be modeled by the multinomial distribution, where j ranges for all words: $p(w|s,\Theta) = \exp(r_{sw})/\sum_j \exp(r_{sj})$. So, the optimization problem is:

$$maxL(\mathcal{D}_{TR},\Theta) = \sum_{s\in\mathcal{D}_{TR}} \log p(w_s|s,\Theta) \tag{12}$$

This modeling is similar to a multinomial logistic regression [23], where the input is formed by Θ and the output are the words w that appear in the sequences s. Due to computational issues the employed likelihood computation is not exact, but obtained by sampling according to a “proposal distribution”, where each word has a probability proportional to its

[12] `http://trec.nist.gov/`

[13] This idea is similar to the consideration that texts are related to hidden topics, as explained in Section 2.1.

empirical frequency in the training dataset. The optimization problem is solved using the stochastic gradient ascent algorithm [31].

The authors report the results of applying the method for detect word relatedness using three datasets: (a) the Yahoo! Answers, available through the Yahoo! Webscope program; (b) the UKWaC database, a corpus built by crawling of the `.uk` Web domain; and (c) by using the subtitles of several movies and television series (see [12] for details). Experiments employ parsing, tokenization, stop word removal and stemming; word relatedness results – showing the obtained accuracy – are also presented.

This approach can be used for text classification when using the SVM learning algorithm. As previously explained in this algorithm the kernel matrix can be represented by the inner-products of pair of vectors in the training database. So, a term relatedness matrix can be considered as a suitable kernel matrix. There are some options when deciding what is the basic text element to consider as a word, such as: (a) the use of the original text terms obtained by text preprocessing; or (b) the use of n-grams of a parameter n. The steps previously described can be directly followed: latent factors and biases will be the problem variables in the optimization problem whose objective function to be maximized is the log-likelihood of the training set, exactly as indicated in the Halawi et al. paper.

5 Illustrative Experiments

In order to illustrate this exposition some experiments of text classification in a limited database were conducted, using tools from the Python Natural Language Tool Kit (NLTK)[14] and the classification algorithms of the Machine Learning Tool WEKA[15].

Text files were extracted from the classical "Reuters Corpus", that contains stories from Reuters news agency compiled by David Lewis and publicly available[16]. The employed version is directly available in the NLTK Toolkit; it contains 10,788 news documents totaling 1.3 million words. The documents have been classified into 90 topics, but the class assignment is not injective. To work in a reduced hard classification problem the set of documents that are assigned exclusively to the two most frequent classes of the database were selected. The resulting dataset contains 6,215 documents, assigned to `earn` (3,923 docs with 398,706 words) and to `acq` (2,292 docs with 334,554 words). The third most frequent class (`crude`) has only 374 documents. Several preprocessing options and kernels were tested. Some preprocessing procedures were applied in all the experiments: (a) all the words were transformed to lower case; (b) words whose length is less than 3 were eliminated; (c) strings that correspond to numbers (such as "1985") were eliminated. In all experiments a standard hold-out

[14] `http://nltk.org/`

[15] `http://www.cs.waikato.ac.nz/ml/weka/`

[16] See `http://www.research.att.com/lewis`.

Classifier	Accuracy (%)	F-Measure	Processing time (sec)
1-NN	86.37	0.856	0.02
Naïve-Bayes	95.31	0.953	2.98
J4.8	94.65	0.947	34.37
MLP	62.75	0.484	$8,516.64$
SMO-linear	98.25	0.982	2.96
SMO-poly2	97.77	0.978	21.98
SMO-RBF	97.02	0.970	36.96

Table 2. Machine Learning algorithm performance in a two-class text classification task

evaluation procedure was used, splitting the dataset in $\frac{2}{3}$ for training and $\frac{1}{3}$ for testing.

The initial experiment uses the VSM in its classical form. To do so, the original text was transformed according to the VSM model, using a built-in WEKA method. We use the term-frequency (tf) of a word in the document with doc length normalization as its feature value; additional preprocessing steps include the use of the Lovins stemmer and the restriction to the 300 most frequent words per class. After that a [term x document] matrix D of dimension $(427 \times 6,215)$ was obtained.

In the first experiment the SVM performance is compared with some ML learning algorithms, using their WEKA implementation and standard parameters: (a) the instance-based 1-NN (nearest neighbor); (b) the probabilistic Naïve-Bayes classifier; (c) the decision tree classifier J4.8; (d) a multi-layer perceptron (MLP) network with back-propagation algorithm for training. A detailed description of these learning algorithms can be found in [21]. For the SVM the employed implementation uses the SMO algorithm [24] with the following options: (a) the linear kernel (the inner-product is directly used); (b) the second degree polynomial kernel; and (c) the RBF kernel.

Table 2 presents the accuracy (percentage of correctly classified instances), the F-measure and the processing time to built the corresponding model for this experiment. The last value is provided only for reference[17]. As we can see in this table the SVM classifier provides the high scores at low computational costs, confirming the claim that it is a good classifier for text applications. Simpler classifiers (1-NN, Naïve-Bayes, J4.8) have the lower classification performance; the MLP spends a lot of time to built the classification model, and surprisingly presents very bad results (possibly due to some particular execution issue). So, the following experiments employ only the SVM classifier, using the SMO WEKA implementation.

[17] Experiments were conducted in a PC computer with $i5 - 2410M@2.3GHz$ processor and 6 GB of memory.

In the main experiment the SVM learning algorithm was employed in conjunction with some preprocessing options. The VSM model was extended from tf to use the $tfidf$ metric, with the same additional restrictions of the first experiment. A set of n-grams (for $n = 3, 4$) was also employed. They were obtained directly from the words that compose the text, using specific Python functions. 399 3-grams and 449 4-grams features were obtained for the same 6,215 documents. In the conversion from string text to vector space, the frequency of the n-gram was used as feature value, and also doc length normalization.

Finally the SSK kernel as presented by Lodhi et al. [20], which is also available in the WEKA platform, was employed. Due to computational issues of this algorithm an additional preprocessing step was applied: each document was considered as a string formed by the 300 most frequent words that appear on it; in order to avoid the case of an "empty string" to be associated to a document, the first word of each document was added to its string representation. The number of considered characters in the string that is associated to each document were also limited to to 50. The final preprocessed database is formed by $6,215$ strings of length 50, each one corresponding to a document associated to a class. Due to the elevated computational time an instance filter that selects 20 $\%$ of the training and test datasets was applied, so about $1,200$ instances were employed. A similar experiment was conducted using 3-grams and 4-grams in conjunction with the SSK kernel, with the same preprocessing steps and instance restrictions of the previous experiment.

In Table 3 the obtained results for the main experiment are presented (results for SVM Linear tf are repeated from previous table). The number of support vectors and kernel evaluations are also presented in each case; they are associated to the execution time.

Obtained results are consistent and stable. Some preliminary conclusions can be obtained:

- The original Vector Space Model performs very well when the "basic" SVM classifier (linear kernel) is employed; as the vectors that represent documents are very sparse, so linear separability can be easily obtained in the feature space; in fact, the F-Measure result is the better one obtained in our tests.
- Polynomial kernel (second degree) has a slightly superior performance in some cases, whereas the RBF kernel results are inferior in all cases.
- Results using tf and $tfidf$ for the several kernel options are very similar; it seems that in this collection the inverse document frequency does not influence the results; possibly this is because words appear well-distributed in the collection.
- The use of n-grams does not contribute to augment significantly the classification performance in this collection.
- The use of the SSK kernel does not improve the performance; however, as we employed limited-length string to represent the documents – obtained from additional preprocessing steps to limit computational time – one can consider that the results are

Kernel/ Preprocessing	Accuracy (%)	F-Measure	Processing time (sec)	# of support vectors	# of kernel evaluations
Linear tf	98.25	0.982	2.96	–	3,805,735
Poly2 tf	97.77	0.978	21.98	1,574	48,393,866
RBF tf	97.02	0.970	36.96	2,079	83,553,985
Linear $tfidf$	98.25	0.982	2.72	–	3,805,735
Poly2 $tfidf$	97.78	0.978	20.48	1,574	48,393,866
RBF $tfidf$	97.02	0.970	38.94	2,079	83,558,985
Linear 3-gram	97.25	0.973	13.04	–	4,674,567
Poly2 3-gram	97.68	0.977	62.68	1,088	30,275,175
RBF 3-gram	97.11	0.971	113.25	1,395	47,977,886
Linear 4-gram	97.96	0.980	10.61	–	5,288,339
Poly2 4-gram	98.11	0.981	45.58	1,026	29,159,140
RBF 4-gram	97.49	0.975	81.82	1,212	49,725,538
SSK doc string	96.88	0.969	8,873.72	844	41,312,294
SSK doc string 20%	96.05	0.960	208.35	290	1,163,734
SSK 3-gram string 20 %	95.72	0.957	3,924.41	257	1,728,382
SSK 4-gram string 20 %	96.20	0.962	4,867.17	231	1,172,674

Table 3. SVM performance with different preprocessing steps and kernels

somewhat unexpected. Possibly using a sophisticated computational environment superior results can be achieved. One important conclusion is that the execution time is a serious drawback for the effective use of string direct kernel approaches in real applications.

- The same conclusions can be made in the case of n-grams associated with SSK: we have to limit the size of the considered n-gram strings, prejudicing the algorithm performance; very high execution times were recorded.

The experiments in this work are intended to be illustrative, that is, to provide comparative basis for the described text processing options and employed kernel functions. Most of the related works employ the "ModApte" partition of the Reuters collection and documents from the ten top ranked classes; however, the work of Joachims [16] present results of similar magnitude for the classes `earn` and `acq`. A point to be emphasized in Table 3 is the number of kernel evaluations: as we can see for sophisticated kernels this number is very high; so, a kernel function with fast computation can be useful to reduce required computational time. The use of a direct kernel matrix, where results are obtained directly from pairs of training data is an alternative approach in this direction.

6 Conclusions

This work discuss how kernel functions that can be applied in conjunction with the Support Vector Machines classifier in the automatic text classification task. Several models for text representation (VSM, GVSM, n-grams, strings) are discussed; in most of them text is represented by a set of vectors in a high dimensional space.

The Support Vector Machine learning algorithm is presented as a result of an optimization procedure. Afterward the kernel trick is presented; it allows the SVM to be applied in more general situations, where linear classifiers do not apply. Some options to consider as kernel functions in the case of text processing are also presented. The use of the SVM classifier in conjunction with some text preprocessing techniques and kernel function selection is illustrated by some experiments, conducted in a limited database obtained from the two most frequent classes of the classical Reuters text corpus.

One particular point of this work can be highlighted: as explained, a recent work by Halawi et al. [12] is dedicated to the computation of word relatedness in text databases. Such goal is achieved by using an optimization procedure; in their proposal it is necessary to optimize the log-likelihood function of the model parameters, according to frequencies estimated based on training data. This approach can be used to obtain new kernel functions for text classification: word relatedness will directly constitute the elements of the kernel matrix (a symmetric positive-definite matrix), since it represents the inner-products of pairs of elements of the training data.

Acknowledgements: This work was developed in the Data Mining Lab. – Computer Science and Engineering Dept. – York University, Toronto, Canada, thanks to Prof. Aijun An. Also thanks to Prof. Andranyk Mirzaian by his Combinatorial Optimization explanations.

References

[1] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press (1999).

[2] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press (2009).

[3] C.J.C. Burges. "A Tutorial on Support Vector Machines for Pattern Recognition". *Data Mining and Knowledge Discovery*, N. 2 (1998) 121–167.

[4] N. Cancedda, E. Gaussier, C. Goutte and J-M. Renders. "Word-Sequence Kernels". *Journal of Machine Learning Research*, N. 3 (2003) 1059–1082.

[5] W.B. Cavnar and J.M. Trenkle. "N-Gram-Based Text Categorization". *Proceedings of the* 3^{rd} *Annual Symposium on Document Analysis and Information Retrieval* (1994) 161–175.

[6] N. Cristianini, J. Shawe-Taylor. *An Introduction to support vector machines and other kernel-based learning methods*. Cambridge University Press (2000).

[7] N. Cristianini. "Support Vector and Kernel Machines". *International Conference in Machine Learning (ICML) 2001 presentation*, available in `http://www.support-vector.net/icml-tutorial.pdf`, accessed in November 12th, 2012.

[8] C. Cortes and V.N. Vapnik. "Support-Vector Networks". *Machine Learning*, N. 20 (1995) 273–297.

[9] S. Deerwester, S.T. Dumais, G.W. Furnas, T.K. Landauer and R. Harshman. "Indexing by Latent Semantic Analysis". *Journal of the American Society for Information Science*, Vol. 41 (1990) 391–407.

[10] S. Dominich and T. Kiezer. "A Measure Theoretic Approach to Information Retrieval". *Journal of the American Society for Information Science and Technology*, Vol. 58, N.8 (2007) 1108–1122.

[11] G. Giannakopoulos, P. Mavridi, G. Paliouras, G. Papadakis and K. Tserpes. "Representation Models for Text Classification: a comparative analysis over three Web document types". *Proceedings of the ACM International Conference on Web Intelligence, Mining and Semantics* (WIMS'12), Craiova, Romania (2012) 1–12.

[12] G. Halawi, G. Drory, E. Gabrilovichz and Y. Koren. "Large-Scale Learning of Word Relatedness with Constraints". *Proceedings of the Knowledge Discovery in Databases'12*, Beijing, China, (2012) 1406–1414.

[13] T. Hastie, R. Tibshirani and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd. Edition (2009)

[14] D. Haussler. "Convolution Kernels on Discrete Structures". *Technical Report UCSC-CRL-99-10*, Department of Computer Science, University of California at Santa Cruz (1999).

[15] T. Hofmann, Bernhard Schölkopf and Alexander J. Smola. "Kernel Methods in Machine Learning". *The Annals of Statistics*, Vol. 36, No. 3, (2008) 1171–1220.

[16] T. Joachims. *Text Categorization with Support Vector Machines: Learning with Many Relevant Features*. Springer (1997).

[17] A. Karatzoglou, D. Meyer and K. Hornik. "Support Vector Machines in R". *Journal of Statistical Software*, Vol. 15, No. 9 (2006) 1–28.

[18] S. Katrenko and P. Adriaans. "A Local Alignment Kernel in the Context of NLP". *Proceedings of the 22nd International Conference on Computational Linguistics* (Coling 2008), Manchester, UK (2008) 417–424.

[19] A. Lehmann and J. Shawe-Taylor. “A Probabilistic Model for Text Kernels”. *Proceedings of the 23rd International Conference on Machine Learning*, Pittsburgh, PA (2006) 537–544.

[20] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini and C. Watkins. “Text Classification using String Kernels”. *Journal of Machine Learning Research*, N. 2 (2002) 419–444.

[21] T. Mitchell. *Machine Learning*. McGraw Hill (1997).

[22] E. Osuna, R. Freund and F. Girosi. “Training Support Vector Machines: An Application to Face Detection”. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, Puerto Rico, PR (1997) 130-136.

[23] C-Y.J. Peng, K.L. Lee and G.M. Ingersoll. “An Introduction to Logistic Regression Analysis and Reporting”. *The Journal of Educational Research*, Vol. 96, N. 1 (2002) 1–14.

[24] J. Platt. “Fast Training of Support Vector Machines using Sequential Minimal Optimization”. In B. Schoelkopf and C. Burges and A. Smola (Editors), *Advances in Kernel Methods - Support Vector Learning*. MIT Press (1998).

[25] G. Salton, A. Wong and C. S. Yang. “A Vector Space Model for Automatic Indexing”. *Communications of the ACM*, Vol. 18, N. 11 (1975) 613–620.

[26] F. Sebastiani. “Machine Learning in Automated Text Categorization”. *ACM Computing Surveys*, Vol. 34, N. 1 (2002) 1–47.

[27] M. Sewell. “Kernel Methods”. Available in `http://www.svms.org/kernels/kernel-me` accessed in November 12th, 2012.

[28] J. Strunk, C.N. Silla-Jr and C.A.A. Kaestner. “A Comparative Evaluation of a New Unsupervised Sentence Boundary Detection Approach on Documents in English and Portuguese”. *Computational Linguistics and Intelligent Text Processing – Lecture Notes in Computer Science*, Vol. 3878 (2006) 132–143.

[29] C.Y. Suen. “n-Gram Statistics for Natural Language Understanding and Text Processing”. In *IEEE Transactions on Pattern Analysis and Machine*, Vol. 1, N. 2 (1979) 164–172 .

[30] G. Tsatsaronis and V. Panagiotopoulou. “A Generalized Vector Space Model for Text Retrieval Based on Semantic Relatedness”. *Proceedings of the European Association for Computational Linguistics*, Athens, Greece (2009) 70–78.

[31] R. Wijnhoven, P.H.N. de With. “Fast Training of Object Detection using Stochastic Gradient Descent”. *IEEE International Conference on Pattern Recognition*, Istanbul, Turkey (2010) 424–427.

[32] S.K.M. Wong, W. Ziarko and P.C.N. Wong. “Generalized vector spaces model in information retrieval”. *Proceedings of the ACM SIGIR 85*, Montreal, CA (1985) 18–25.

[33] S.K.M. Wong, W. Ziarko, V.V. Raghavan and P.C.N. Wong. “On Modeling of Information Retrieval Concepts in Vector Spaces”. *ACM Transactions on Database Systems*, Vol. 12, N. 2 (1987) 299–321.