IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 28 November 2023, accepted 28 December 2023, date of publication 22 January 2024, date of current version 1 February 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3356572

RESEARCH ARTICLE

A Graph Classification Method Based on Support Vector Machines and Locality-Sensitive Hashing

MARÍA D. GONZALEZ-LIMA[1], CARENNE C. LUDEÑA[2], AND GIBRAN G. OTAZO-SANCHEZ[3]
[1]Department of Mathematics and Statistics, University of the North, Barranquilla 080003, Colombia
[2]Matrix CPM Solutions, Bogotá 110231, Colombia
[3]AI Factory, BBVA, México City 06600, México

Corresponding author: María D. Gonzalez-Lima (limad@uninorte.edu.co)

ABSTRACT Graphs classification is a relevant problem that arises in many disciplines. Using graphs directly instead of vectorization allows exploiting the intrinsic representations of the data. Support Vector Machines (SVM) is a supervised learning method based on the use of graph kernel functions used for this task. One of the problems of SVM, as the number of samples increases, is the cost of storing and solving the optimization problem related to SVM. In this work, we propose a method capable of finding a small representative subset of the whole graph data set such that an approximate solution of the SVM optimization problem can be obtained in a fraction of the time, and without significantly degrading the classification prediction error. The method is based on the use of Locality-Sensitive Hashing for projecting over the Hilbert spaces defined by appropriate graph kernels that measure similarity between the graphs. A description of the algorithm, as well as numerical results using two graph kernels (Simple Product and Random Walk) on simulated and real life data sets are presented. The numerical experiments compare the training times and the classification error between the SVM obtained with our smart sampling approach, and the SVM obtained over the complete data set or over a random sub-sample. The results offer evidence of the advantages of our proposal for solving large scale graph classification problems when using SVM.

INDEX TERMS Graph classification, locality sensitive hashing, support vector machines.

I. INTRODUCTION

Many kinds of objects can be naturally represented as graphs, for example biological sequences, chemical compounds, RNA secondary structures or fraud and money laundering. The wide use of graphs to model data structures, the existence of large data basis (e.g. NCBI's PubChem) and the demands from new applications could explain the effort devoted, in the recent decades, to develop efficient methods for processing graph data. In particular, the classification problem, which consists in building models to predict the class label of a given object by learning from training data, has become a graph mining problem of great interest among researchers and practitioners because of its ample field of applications. For a general presentation on graph classification methods, we refer to [28].

Support Vector Machines (SVM) is a popular supervised machine learning technique for classifying objects. It is characterized by the use of kernel functions to embed objects in a feature space, such that parameter estimation depends only on the chosen kernel and not on the actual embedding. This, in addition to the proven robustness of the method along with the elegant theory behind it, explain the appeal and popularity of SVM. However, a bottleneck for its use, is the dependence on the choice of adequate kernel functions as well as the high computational cost when considering large data sets. These are particularly relevant when trying to use SVM for graph classification, since adequate similarity measures are needed. Recent developments have tried to close the gap between the use of kernel machines and graph classification by proposing similarity functions for graphs, so particular kernels can be built [22].

Even for small sized graph data sets, their nature makes the SVM problems to be solved very costly, so efficient methods are of interest. In this work, we deal with the problem

The associate editor coordinating the review of this manuscript and approving it for publication was Jad Nasreddine.

of classifying graphs by using Support Vector Machines (SVM). We propose a method based on Locality-Sensitive Hashing (following the ideas presented in [12]) in order to choose a subset of the graphs to represent the original data set without loosing too much significant information. For this, we use the rationale of finding an adequate subset of representative points via projections of the transformed graphs. More precisely, using the *kernel-trick*, for any chosen kernel there exists a transformation such that this kernel represents the inner product in an appropriate Hilbert space. Graph projections will be based on this kernel (without having to actually consider the transformed graphs) and they will allow us to create bins so that with high probability points belonging to the same bin are close, and points which are far apart will not be in the same bin. Based on these bins, it is not necessary to consider the whole original set, thus reducing the effective size of the data set.

The paper is organized as follows. In Section II we present some related works. In Section III we briefly introduce SVM methods. Section IV is devoted to presenting basic facts and notation on graphs. There, the Simple Product and Random Walk kernel, the kernels used in our numerical studies, are introduced. The proposed algorithm is presented in Section V. Section VI resumes the numerical experiments and their results. Finally, some conclusions and remarks are presented in the last section.

II. RELATED WORK

Because the computational time of using SVM becomes very demanding when training large data sets, there have been many attempts in the literature to address this bottleneck. A group of them are oriented to find an approximated solution of the SVM optimization problem by solving a sequence of smaller problems. Relevant algorithms in this respect are SMO [23] and the extension, LIBSVM [19]. The latter is at the foundation of several codes presented in well known and used packages, as for example sickit-learn of Python.

There are also other strategies that aim at building an approximate optimization problem using data reduction. These approaches are very diverse in nature, including methodologies that use geometry, clusterization, distance-based approaches, and random sampling. The best method to use for data reduction is still an active research topic, as can be seen from the recent reviews [3] and [21]. Some issues to address are the efficient treatment of imbalanced data sets and the effect of noise and the kernel functions [3].

Among the approaches that use sub-sampling for data reduction, is the random sampling algorithm (RSA) [1] which is a technique based on assigning a probability to each sample to be chosen. Once a random number of samples has been selected, an SVM is trained, and it updates the initial probabilities, increasing the ones with samples that have been miss classified. The reduced SVM method (RSVM) [18] is another method that randomly selects a subset of the data to represent the original training set. The kernel matrix is replaced by a rectangular kernel matrix formed by this selection. The formulation of the SVM problem differs from Problem (3) since the kernel matrix found is included in the restrictions of the optimization problem to be solved, instead of the objective function. The Knee-cut SVM algorithm in [24] uses the kernel as a measure of the distance between the classes and eliminates samples that are far from the boundary of the classes. A different approach is presented in [14] where the reduction of the data includes the assignments of large weights to important samples and the reduction of the features, by using graph and self-paced learning. The methods in [5] and [6] use nearest neighbors with sub-sampling in order to select a subset of significant instances. They start by training an SVM formed by a very small sub-samples of the data set. The support vectors obtained with this initial sample are enriched by nearest neighbors found within the complete data set, and a new SVM problem is trained. This process is repeated until the classification error becomes almost constant. In [5] the authors also provide a theoretical framework for the analysis of random sub-sampling when using SVM.

A simple random sub-sampling of the data is customary among the practitioners of SVM when training a large data set. The authors of [13] present a very interesting comparison among different techniques for approximating SVM in combination with simple random selection of the data. They conclude that LIBSVM is the best option when using simple random subsampling. Therefore, we include random sub-sampling as a benchmark in our numerical experiments.

Our sub-sampling strategy is novel in the sense it combines the virtues of simple random sub-sampling, including its embarrassingly-parallel property with clusterization-oriented strategies, as we aim at avoiding similar data points in the final sub-sample, without having to calculate overall distances.

Also, our sub-sample strategy is constructed based directly on the graph classification problem at hand. For graph classification, most methods are based on some kind of vectorization or representation technique. Recent papers such as [7] and [20] use convolutional or recurrent neural networks, to deal with this task. There, the graphs are transformed using the Laplacian. For a thorough revision of graph classification techniques, we refer to [26].

Unlike these approaches, our method does not use representations of the graphs but rather uses graph kernels to embed the graphs in a Hilbert space following the general SVM setup. In this framework our procedure for selecting significant sub-samples is based only on the associated inner product.

We can summarize our main contributions as:

- The method is embarrassingly-parallel as it does not depend on any iterative procedure.
- The method does not require a representation or vectorization of the graph data set as it depends solely on the chosen kernel.

- Sub-sample selection is smart as it avoids selecting similar data-points, thus obtaining the benefits of cluster-based methods without their complexity.
- The SVM problem solved using our proposed sub-sampling strategy is efficient in the sense less support vectors are required to define the model albeit no significant loss in accuracy.

III. INTRODUCTION TO SVM

SVM for binary classification (the one considered in this paper) is based on the following. Given points $\{X_i \in \mathbb{R}^d, \quad i = 1, \ldots, n\}$ belonging to two classes (identified with the corresponding tags $y_i = 1$ or $y_i = -1$), they are linearly separable if there exists an hyperplane that divides them into the two different classes. The dimension d denotes the attributes of the data, and the input (or observation) space is the set formed by the data. Among all separating hyperplanes, SVM seeks to find the one that maximizes the separation margin between classes, constrained to respecting the classification of each point of the data. This problem can be modeled, after a normalization, as the optimization problem

$$\begin{aligned} &\underset{w,b}{\text{minimize}} \ \frac{1}{2}\|w\|_2^2 \\ &\textit{subject to } y_i(w^t X_i + b) \geq 1 \ \ \forall\, i = 1, \ldots, n, \end{aligned} \tag{1}$$

Here, $\|.\|_2$ denotes the euclidean norm.

Because the data set is usually linearly non-separable (that is, there does not exist a solution of problem (1)) two variants are introduced in the previous problem. On the one hand, a perturbation variable ξ is included in order to relax the constraints, so that a margin of error in the classification is accepted. On the other, since the data might be separable by a nonlinear decision surface, such a surface is computed by mapping the input variables on to a higher dimensional feature space, and by solving a linear classification problem in that space. In other words, let us denote $\mathcal{H}$ the feature space which satisfies to be a Reproducing Kernel Hilbert set (RKHS). We denote the inner product in $\mathcal{H}$ with $< .,. >$. Then, $x \in \mathbb{R}^d$ is mapped into $\phi(x) \in \mathcal{H}$, where $\phi(.)$ is the transformation induced by the use of the kernel function K. This is, $K(z, a) =< \phi(z), \phi(a) >$ for every $z, a \in \mathbb{R}^d$. Thus, the input vectors X_i are substituted with the new "feature vectors" $\phi(X_i)$, belonging to the "feature space" $\mathcal{H}$. In this way, for the linearly non-separable case, the optimization problem is written as

$$\begin{aligned} &\underset{w,b,\xi}{\text{minimize}} \ \frac{1}{2}\|w\|_{\mathcal{H}}^2 + C\sum_{i=1}^{n}\xi_i \\ &\textit{subject to } y_i(< w, \phi(X_i) > + b) \geq 1 - \xi_i \ \ \forall\, i = 1, \ldots, n, \\ &\qquad \xi_i \geq 0 \ \ \forall\, i = 1, \ldots, n. \end{aligned} \tag{2}$$

where C is a positive constant that penalizes the errors at the constraints, and $\|w\|_{\mathcal{H}}^2 =< w, w >$. For more details we refer to the book by Cristianini and Shawe-Taylor [8].

In order to solve (2), standard duality theory may be used since the problem is convex and quadratic. Using this theory, (2) can be solved by solving its dual. Some advantages of using the dual problem is that an explicit description of ϕ above is not required and can be replaced by a function that preserves the properties of the inner product in the higher dimensional space $\mathcal{H}$, and this is satisfied by the kernel function K. Then, the dimension of the feature space for the classification can be increased without increasing the dimension of the optimization problem to be solved, and this is particularly relevant when dealing with an infinite dimensional space $\mathcal{H}$. Moreover, for graph data sets, the use of SVM for their classification only requires the existence of an appropriate kernel function.

Following [8], the dual problem corresponding to (2) is given, in terms of the kernel function, by

$$\begin{aligned} &\underset{\lambda}{\text{mimimize}} \ -\sum_{i=1}^{n}\lambda_i + \frac{1}{2}\lambda^t Q\lambda \\ &\textit{subject to } y^t\lambda = 0, \\ &\qquad 0 \leq \lambda_i \leq C \quad \text{for } i = 1, \ldots, n \end{aligned} \tag{3}$$

where $Q \in \mathbb{R}^{n\times n}$ is a symmetric positive semi-definite matrix with positive diagonal, defined as $Q_{ij} = y_i y_j K(X_i, X_j)$. The matrix K with ij-component equal to $K(X_i, X_j)$ is called the kernel matrix. To avoid complicating notation, typically K will stand for both the generic kernel and the matrix defined by the kernel restricted to the original data set of size n.

Let λ^* be a solution of (3). Our interest is to classify a new point by means of a generalization function (or classifier) g^* using λ^*. This can be done, after using duality theory (see [8]), by computing

$$g^*(X) = \text{sign}\Big(\sum_{i=1}^{n}\lambda_i^* y_i K(X_i, X) + b^*\Big), \tag{4}$$

with $b^* = 1 - \max_{\{y_j=1, 0<\lambda_j^*<C\}}\sum_{i=1}^{n}\lambda_i^* y_i K(X_i, X_j)$. Observe that only the $\lambda_i^* > 0$ are relevant. The corresponding X_i are the so-called support vectors and their importance follows from the fact that any other data points are irrelevant for classification purposes.

Therefore the classification of a new point can be made just by selecting a (hopefully small) group of support vectors from the large original data set. The procedure of finding these vectors (which we will denote by SV) from the given data set, is usually referred to as the training process or training the machine. After training, it is customary to qualify the result by using an held-out set of examples, not seen during the training phase. This set of points is called the testing set. The estimated classification or prediction error for a given data set is the percentage of points from the test set that are incorrectly predicted.

In the following section we will introduce some basic facts and notation for graphs.

IV. BASIC FACTS AND NOTATION

In this paper the objects to be classified are graphs. They can be described, in a natural way, as a set of nodes (or vertices) that may or may not be connected by edges. Each node and edge may have corresponding labels. Formally,

Definition 1: A graph g is defined as $g = (V, E, \mu, \nu)$ where

- V denotes the finite set of nodes,
- $E \subseteq V \times V$ denotes the set of edges,
- $\mu : V \to L$ is the function that assigns labels to the nodes,
- $\nu : E \to L$ is the function that assigns labels to the edges.

Notice that L is a label alphabet set.

A great effort has been devoted to establish ways to compare graphs, i.e. graph matching methods. These methods can be separated into two types: exact and inexact ones. The latter ones cover error-tolerant graph techniques which are more suitable for using in the framework of kernel machines. Some of these approaches give rise to graph kernels. This is, functions that measure similarity among graphs satisfying the properties of being a kernel. Recent papers on graph kernels and future challenges can be found in [4] and [17].

The simplest graph kernel is the one that can be defined between two graphs with the same number of nodes by using the adjacency matrix for each one of them. This is,

Definition 2: Let $g = (V, E, \mu, \nu)$ and $g' = (V', E', \mu', \nu')$ with $|V| = |V'|$ where $|.|$ denotes the cardinality of the set.

Then, the Simple Product kernel between g and g' is defined as

$$S(g, g') = < A_g, A'_g >_F = Trace(A_g^t A_{g'}) \tag{5}$$

with $A_g \in R^{|V|\times|V|}$ the Adjacency matrix of g defined, for any pair (u, v) of nodes of graph g, as

$$A_g(u, v) = \begin{cases} 1 & \text{if } (u, v) \in E \\ 0 & \text{otherwise} \end{cases}$$

The adjacency matrices represent the structure of a graph in matrix form. The inner product $< ., . >_F$ is the usual inner product among matrices.

In the case when the number of nodes in each graph does not coincide, the Simple Product kernel may be extended by considering the adjacency matrix of size $m \times m$ with $m = \max(|V|, |V'|)$ including zeros for the artificial positions added.

Advantages of the Simple Product kernel are its simplicity and easy computation. The idea behind the definition is that two graphs with the same number of nodes should be similar if, independently on how the nodes are labeled, they are connected in a similar fashion. However, some special structure among the graphs may be missed.

In order to meaningfully extend the Simple Direct Product Kernel to be used with graphs with different number of nodes, and to take into account more of the graph structure, we also consider in this paper the Random Walk Kernel. Before introducing its definition, we define the Direct Product between graphs.

Definition 3: Given graphs $g = (V, E, \mu, \nu)$ and $g' = (V', E', \mu', \nu')$ their Direct Product is defined as the graph $g \times g' = (V_X, E_X, \mu_X, \nu_X)$ where

- $V_X = \{(u, u') \in V \times V' : \mu(u) = \mu'(u')\}$
- $E_X = \{((u, u'), (v, v') \in V_X \times V_X : (u, v) \in E, (u', v') \in E', \nu(u, v) = \nu'(u', v')\}$.
- The labels are defined as $\mu_X(u, u') = \mu(u) = \mu'(u')$ and $\nu_X((u, u'), (v, v') = \nu(u, v) = \nu'(u', v')$.

Definition 4: The Adjacency matrix A_X of $g \times g'$ is defined as

$$[A_X]_{(u,u'),(v,v')} = \begin{cases} 1 & \text{if } ((u, u'), (v, v')) \in E_X \\ 0 & \text{otherwise} \end{cases}$$

This matrix can be seen as an extension of the Adjacency matrix defined for each graph.

In the case of undirected graphs and assuming that no node can be connected with itself, it is easy to prove the following result,

Proposition 1: The adjacency matrix $A_X \in R^{|V||V'|^2}$ of $g \times g'$ is equal to $A_g \otimes A'_g$, with $\otimes$ the Kronecker product of matrices.

This direct product is connected with a graph kernel based on random walks called the Random Walk Kernel. This graph kernel is build on the idea that similarity among graphs is defined by comparing their *common* random walks. A random walk over a graph means going from a starting node and moving through the graph by randomly travelling across the edges. For a formal definition, we follow the presentation in [11].

Definition 5: Given a graph $g = (V, E, \mu, \nu)$, the set of walks in g with n edges is $w_n(g) = \{(u_1, e_1, u_2, e_2, \ldots, u_n, e_n, u_{n+1}) \in V \times E \times \ldots \times E \times V : e_i = (u_i, u_{i+1}), i = 1, \ldots, n\}$. For each element of this set $w = (u_1, e_1, u_2, \ldots)$, a corresponding sequence of node and edge labels is defined as $\rho(w) = (\mu(u1), \nu(e_1), \ldots)$.

Given a sequence s of node and edge labels of a graph g, let us define the vector $\Phi_s(g) = \lambda^{\frac{n}{2}} |\{w \in w_n(g) : \rho(w) = s\}|$ some $\lambda \geq 0$. This is, $\Phi_s(g)$ represents the number of walks with corresponding node and edge sequence (multiplied by a weighting factor). It can be shown that $\mathcal{H}$, the set formed by all the vectors $\Phi(g) = (\Phi_{s_1}(g), \Phi_{s_2}(g), \ldots)$ is a Hilbert space. The direct computation of $\Phi(g)$ according to this definition can be very difficult. An important result from [22] is that the inner product of each transformed pair of graphs $< \Phi(g), \Phi(g') >$ can be found by the direct product $g \times g'$ of the graphs, using the adjacency matrices.

Definition 6: The Random Walk kernel of graphs g and g' is defined as

$$R(g, g') = \sum_{i,j=1}^{|V_X|} (\sum_{n=0}^{\infty} \lambda^n A_X^n)_{ij}, \text{ some } \lambda \geq 0. \tag{6}$$

In [11] it is proved that $< \Phi(g), \Phi(g') >= R(g, g')$ for any fixed value λ. Therefore K is a kernel function. Even though this kernel may be affected by noisy data, its simplicity makes it attractive as an alternative to measure similarities between graphs under machine kernel methodologies.

Let us observe that since the Random Walk kernel computation is based on the powers of the Adjacency matrix whose components are 1 or 0, it can be efficiently calculated. And, if λ is chosen smaller than 1, the contribution of $\lambda^n A_X^n$ is small for large n. Then, in practice it is enough to sum over a finite number N_k of terms. However, even with these considerations, the running time for the computation of the Kernel using (6) is $O(k^6)$ with $k = |V||V'|$, so it can be very high for large graphs. A cheaper way to compute the Random Walk Kernel is to use the following equivalence

$$K(g, g') = e^t (I - \lambda A_X)^{-1} e.$$

Here e is the vector of all ones and I is the identity matrix, both in the corresponding dimension. This computation may be done in $O(m^3)$ with $m = |E||E'|$, reducing the computational time (see [16]).

In the following section we present the algorithm based on LSH.

V. METHODOLOGY

A. USING LSH FOR SVM

Locality-sensitive Hashing (LSH) was introduced as an efficient way to look for nearest neighbors in high dimensional spaces [15]. The idea is to hash the vectors in the data space using several hash functions so that, for each one, the probability of collision is much higher for points that are close to each other than for those which are far apart. Then, LSH can be used to search approximate nearest neighbors of a given query point by retrieving elements stored in the same bin containing this point. Formally, the definition is as follows.

Definition 7: (LSH functions) For given $R_2 > R_1 > 0$ and $1 > p1 > p2 > 0$, a family of functions belonging to the set $H = \{h : D \to \mathcal{N}\}$, where D is a metric space with metric $\tilde{d}$, P a probability measure over D and $\mathcal{N}$ is the set of integers, are LSH if for each $\tilde{q}, q \in D$ and each $h \in H$ the following are satisfied

- if $\tilde{d}(\tilde{q}, q) \leq R_1$ then $P[h(q) = h(\tilde{q})] \geq p1$,
- if $\tilde{d}(\tilde{q}, q) > R_2$ then $P[h(q) = h(\tilde{q})] \leq p2$.

In this paper we are interested in the Projection-based hash functions as presented in [9]. For any p dimensional vector v, define the maps $h_{\mathbf{a},\theta}(v) : \mathbb{R}^p \to \mathcal{N}$ indexed by a choice of an α-stable random vector $\mathbf{a}$ (see [9] for a definition) and a real number θ chosen uniformly from the range $[0, r]$ in the following way. For a fixed $\mathbf{a}$, θ the hash function $h_{\mathbf{a},\theta}$ is given by

$$h_{\mathbf{a},\theta}(v) = \left\lfloor \frac{\mathbf{a}^t v + \theta}{r} \right\rfloor. \tag{7}$$

Here, $\lfloor . \rfloor$ denotes the floor function.

In [9] it is shown that the Projection-based functions h, as previously defined, are LSH.

B. ALGORITHM LSH-SVM-GRAPH

We will use the hash functions (introduced in the previous Section) in the feature space in order to find bucket-representatives of the data set used to train the SVM problem. We will do this by projecting the data several times over random directions (see Step 1 of Algorithm 1) using a hash function based on the inner product defined by an appropriate kernel function. For each projection a number of bins defined by the hash functions are built for each sample point. By construction, this procedure is not iterative and can thus be paralleled. Finally, concatenating the bins obtained for each projection we obtain a set of buckets, each one corresponding to a cluster of the original sample points. From each bucket we select a random representative (Step 2 from Algorithm 1). These sample representatives form the selected sub-sample used for solving the approximate SVM problem (Step 3 of Algorithm 1).

A detailed description of the algorithm is presented in Algorithm 1.

Algorithm 1 Algorithm LSH-SVM-GRAPH (LSVMG)

Input:
K the graph kernel,
$\mathcal{S}$ the original labeled graphs dataset ($\mathcal{S} = \{x_1, \ldots, x_n\}$),
B number of bins,
N the number of projections.
Output:
SVM model defined by the support vectors,
Errors: train and test,
Runtimes: training and fit.
Procedure LSVMG($\mathcal{S}$, K, B, N):

Step 0 Set l_i, an empty string for each $i = 1, \ldots, n$.

Step 1 For $k = 1, \ldots, N$,
- Generate a random graph g_k (Ex: Erdos-Renyi graph).
- Find $K(g_k, x)$ for all $x \in \mathcal{S}$, and calculate $R_{\max} = \max_{x \in \mathcal{S}} K(g_k, x) - \min_{x \in \mathcal{S}} K(g_k, x)$.
- Calculate the width $r = \frac{R_{\max}}{B}$.
- Generate $\theta \sim Uniform[0, r]$, and find $h_{g_k,\theta}(x) := \lfloor \frac{K(g_k,x)+\theta}{r} \rfloor$ for all $x \in \mathcal{S}$.

Step 2
- For for all $x \in \mathcal{S}$ create the concatenated strings $\mathrm{l}_i = h_{g_1,\theta}(x_i) \;||\; h_{g_2,\theta}(x_i) \;||\; \ldots h_{g_N,\theta}(x_i)$
- Group identical strings
- From each group randomly select a sample representative. Define $\hat{\mathcal{S}}$ as the set of representatives.

Step 3 Solve SVM problem (3) using $\hat{\mathcal{S}}$ and $\hat{y}$ their corresponding classes, instead of $\mathcal{S}$ and y. Set $\hat{B} := |\hat{\mathcal{S}}|$ the size of the obtained sub-sample.

To motivate our algorithm, let us recall that for the SVM method only the support vectors are needed to define the separation surface among classes. Therefore, data points are important only in terms of their closeness to the support vectors, which of course are not known a priory. It follows that points which are close to each other offer redundant information in relation to support vectors so that only one

TABLE 1. Obtained mean values and s.d. for $\hat{B}$, using varying n=1000, 4000 and 8000 synthetic Barabasi-Albert graphs.

Data set (size)	N	B	mean	s.d
Barabasi-Albert (1000)	5	10	11.2	0.4
Barabasi-Albert (1000)	5	100	13.2	2.56
Barabasi-Albert (1000)	5	1000	14.4	4.02
Barabasi-Albert (4000)	5	10	11.2	0.39
Barabasi-Albert (4000)	5	100	13.8	3.37
Barabasi-Albert (4000)	5	1000	16	4.64
Barabasi-Albert (8000)	5	10	11.3	0.45
Barabasi-Albert (8000)	5	100	17.2	5.43
Barabasi-Albert (8000)	5	1000	17.5	4.34

representative of each cluster formed by repeated projections is needed.

The rationale behind our approach is to use projections, instead of distance measures, in order to reduce computational costs. By projecting over random directions, the algorithm is capable of finding a smart subset of the whole data set that approximates quite accurately the solution of the SVM obtained over the complete data set.

It is important to highlight that our algorithm is searching for a solution of an SVM problem using a subset of the complete data set. The nature of this approximation is analyzed in the numerical experiments, in terms of the standard classification metrics.

The main goal with our algorithm is to reduce the computational time without significantly degrading the performance by using only a smart subset of the whole data set. In the next proposition we state the time complexity of the LSVMG procedure. The proof follows straightforwardly from the description of Algorithm 1.

Proposition 2:

1) The time complexity of the procedure LSVMG described in Algorithm 1, is $O(Nnm^2 + \hat{B}^2)$, where m is maximum number of nodes of any graph in $\mathcal{S}$.
2) The relative efficiency r_n of the proposed algorithm in relation to the original SVM problem is given by $r_n = \frac{Nnm^2+\hat{B}^2}{n^2}$

A theoretical probabilistic bound for $\hat{B}$ is beyond the scope of this paper. However, because of the underlying inner product structure it is related to the covering properties of any bounded ball in the associated Hilbert space and the LSH property.

For completeness sake, we have included experimental results for the obtained $\hat{B}$ for comprehensive simulation setups, varying the size of B,N and n. Results for 10 simulations showing obtained mean values and associated s.d. of $\hat{B}$ are shown in Table 1 for the 1000, 4000 and 8000 simulated Barabasi-Albert graphs and in Table 2 for the Reddit data set [25].

In the following section we study the performance of the proposed algorithm (LSVMG) using different graph data sets and the kernels introduced in this section. We compare with the SVM using the complete data set and the SVM over a random sample.

TABLE 2. Obtained mean values and s.d. for $\hat{B}$, using the n=8000 Reddit data set graphs.

Data set (size)	N	B	mean	s.d
Reddit (8000)	5	10	245.6	0.17
Reddit (8000)	5	100	1033	33.72
Reddit (8000)	5	1000	1088	46.04

VI. EXPERIMENTS

A. EXPERIMENTAL SETTING

1) GRAPHS USED IN OUR EXPERIMENTS

- Synthetic random graphs based on the Barabasi-Albert (BA) model for simulating social networks [2]. The idea is to start with a network of m nodes, and set c the number of edges to attach to each existing node, one at a time. The probability of a new connection for any given node is proportional to the number of links the node already has. This intends to emulate the way social networks behave in real life, assuming that the more popular nodes have a greater probability of making new friends. The BA model satisfies the scale-free network property, this is, the number of nodes and edges are related by a certain family of underlying probability distributions.
- Synthetic random graphs generated on the Erdös-Renyi (ER) model [10]. Different from the previous ones, these graphs are completely random in the sense that for a fixed constant probability p, the number of nodes is fixed and the edges are generated according to a Bernoulli distribution with parameter p.
- The Reddit Threads real data set from [25], which was collected in 2018, where nodes are Reddit users who participate in a discussion and links are defined by replies amongst them.

2) DATA SETS

After extensive experimentation, we present the results obtained for three kinds of test sets described below as being representatives of our main findings.

- Test set 1:
  This data set consists of $\#G$ number of graphs synthetically generated, half of them of class BA and the other half of class ER. Therefore, the classification goal should be easily reached. All the graphs are generated so that they share the same number of vertices. The probability for ER used was chosen as $p = 0.1$. The graph kernel used satisfies equation (6) but with only a finite number of terms $N_k > 0$. And, $\lambda^n = 0.0178e^{-0.115n}$ for $n = 1, \ldots, N_k = 10$.
- Test set 2:
  In these tests our main goal was to analyze the performance of the LSVMG-Algorithm for larger data sets with imbalanced data. We simulated 10, 000 synthetic

BA graphs, where 9, 500 are BA with parameters $m = 10$ and $c = 3$, and 500 are also BA with parameters $m = 15$ and $c = 2$.

- Test set 3 The Reddit Threads from [25] consist 203, 088 binary labeled balanced graphs, and the task is to predict whether a thread is discussion-based or not.

3) METHODS AND IMPLEMENTATION ISSUES

For our numerical experiments we coded the LSVMG-Algorithm (which finds a selection of the complete data set) in Python and compared with the SVM algorithm using the whole data set. For our results we name "Complete" the results obtained using the whole data set.

We also compared with the SVM algorithm using a simple random selection of the data sets. Therefore, in our numerical results we present comparison among the following methods:

- Algorithm LSVMG.
- Algorithm SVM (complete data set)
- Algorithm SVM (random selection of sub-sample)

For our experiments we use the following standard class-prediction evaluation metrics: Accuracy, F1-score, Recall, and Precision. Our goal in this numerical section is to show the performance of the proposed algorithm over different data sets comparing kernels, evaluation measures and computational times.

In all cases, we use 80% of the data set for training and 20% for model evaluation. The random graphs for the LSVMG-Algorithm are ER synthetic graphs.

B. RESULTS

1) RESULTS FOR TEST SET 1

The objective of this first test set is to evaluate the performance of the LSVMG-Algorithm in conjunction with the Random Walk kernel, and to compare with the Algorithm SVM (complete data set).

The results obtained for the Test set 1 are included in Table 3 where the first column refers to the total number of generated graphs and the second one to the number of vertices for each graph.

The notation T, P and *Error* is used for the average values of training time (in seconds), prediction time (in seconds), and classification error obtained after 5 runs for each problem. Here we are using as classification error the value of $1 -$ Accuracy. Columns 3 to 5 correspond to the results for the whole data set. Columns 6 to 9 correspond to the results for the LSVMG-Algorithm. The symbol $--$ is used for denoting non convergence of the algorithm when the running time exceeds 24 hours.

Observe that, for both approaches, the classification errors are generally low, which is to be expected for this separable data set. The training time increases as the number of vertices increases, as should be expected. In fact, when the number of vertices reaches 20, none of the algorithms is able to solve the corresponding classification problem because of the running time of the Random Walk kernel even for medium number of graphs. But, the running time in the training and prediction steps are much smaller for the LSVMG-Algorithm than for training over the whole data set. In general, the classification errors obtained by the two approaches are comparable.

TABLE 3. Performance on evaluation, using synthetics Barabasi and Erdös-Renyi graphs, with Random-Walk kernel.

# G	# V	T_C (s.)	P_C (s.)	$Error_C$	T_L (s.)	P_L (s.)	$Error_L$
100	5	6.52	0.71	0	1.09	0.36	0
100	7	10.43	0.76	0.06	2.17	0.59	0
100	10	45.38	2.62	0	10.12	1.63	0
100	12	122.98	13.95	0.03	30.72	8.24	0.03
100	15	513.92	92.47	0.10	120.86	37.00	0.10
100	20	846.41	617.85	0.44	285.79	250.06	0.43
200	5	40.72	1.09	0.03	2.97	0.55	0.03
200	7	52.89	2.52	0	7.42	1.48	0
200	10	223.39	10.41	0	40.44	5.65	0.02
200	12	612.77	32.45	0.03	98.08	20.68	0.08
200	15	2252.23	473.39	0.48	341.5	94.16	0.43
200	20	–	–	–	–	–	–
500	5	197.86	6.40	0	12.51	1.75	0
500	7	269.44	10.51	0.02	16.44	2.73	0.05
500	10	644.69	37.23	0.03	35.67	4.98	0.03
500	12	–	–	–	73.80	11.52	0.06
500	15	–	–	–	835.09	247.4	0.54

TABLE 4. Performance on evaluation, using synthetic Barabasi-Albert graphs, with simple product kernel.

Training (# graphs)	# support vectors	Recall	Precision	F1-score	Training time (sec.)
Complete (8000)	241	96%	100%	98%	468
Random (188)	22	59%	98%	65%	0.24
LSVMG (188)	44	100%	100%	100%	1

2) RESULTS FOR THE TEST SET 2

We applied the LSVMG and SVM (complete data set) for the Test set 2.

For the LSVMH Algorithm the number of bins was set as $B = 40$, an ER random graphs were chosen as projections with $p = 0.2$. The number of projections considered for our experiments was $N = 5$, for a total of $\hat{B} = 188$ selected samples.

Since the data is imbalanced, we consider the macro F1-score as a comparison metric over the test data set. In these experiments we were also interested in comparing the performance of our sub-sampling algorithm with a simple random sampling of the data set, this is Algorithm SVM (with a random selection of the data set).

For a fair comparison, we did the following. We applied the LSVMG Algorithm and used the same number of data points finally selected by the algorithm in order to randomly choose the sub-sample from the original data set and to apply SVM with these samples. In Table 4 we include the results obtained using the Simple product kernel. We call Random the results obtained with SVM over the random sample and Complete the results obtained with SVM by using the complete data set.

TABLE 5. Performance on evaluation, using synthetic Barabasi-Albert graphs, with the random-walk kernel.

Training (# graphs)	# support vectors	Recall	Precision	F1-score	Training time (sec.)
Complete (8000)	—	—	—	—	—
Random (182)	3	100%	100%	100%	7
LSVMG (182)	4	100%	100%	100%	7

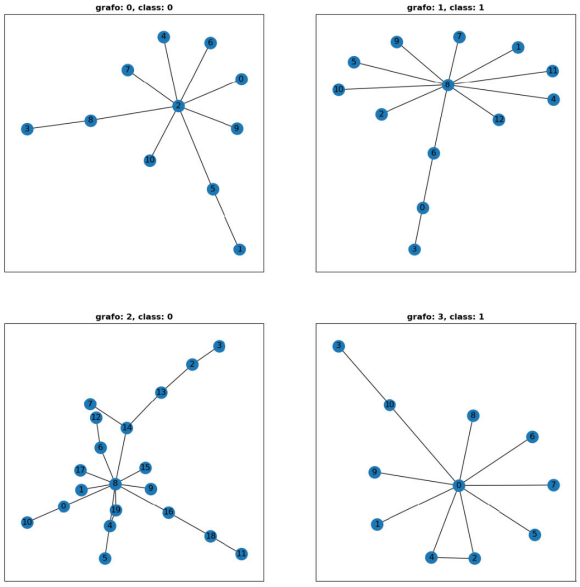

FIGURE 1. The first 4 Reddit Threads graphs in [25].

Observe that there is a significant running time difference between training with the whole data set, achieving only a slightly worse performance, where LSVMG-Algorithm shows a considerable improvement in terms of computational and spacial cost. On the other hand, we could infer that LSVMG-Algorithm is avoiding over-fitting. The number of support vectors reduces considerably when using our algorithm compared to when the whole data set is used. This also contributes to having lower prediction times without significantly decreasing the classification error.

Choosing a random sample decreases the outcome significantly as can be seen in Table 5, looking at the F1-score and Recall columns.

Using the Random Walk kernel already implemented in [27] following [16], the performance obtained with the random sample is comparable to that obtained with the LSVMG-Algorithm as it can be seen in Table 5. However, a higher computational time is needed in order to get the same performance as that of the LSVMG-Algorithm with the Simple Product kernel. Results were not obtained for the whole data set after 3 days running.

3) RESULTS FOR THE TEST SET 3

For simplicity, we have used the first 10, 000 graphs, taking 80% for training and 20% for testing.

TABLE 6. Performance on evaluation, using Reddit Threads from SNAP Datasets, with simple product kernel.

Training (# graphs)	# support vectors	Accuracy	Training time (sec.)	Prediction time (sec.)
Complete (8000)	5148	55%	1064	444.6
Random (718)	410	65%	9.45	52
LSVMG (718)	392	67%	9.76	56

TABLE 7. Performance on evaluation, using the Reddit Threads data set from SNAP Data sets, with the Random-walk kernel.

Training (# graphs)	# support vectors	Accuracy	Training time (sec.)	Prediction time (sec.)
Complete (8000)	—	—	—	—
Random (718)	343	57%	17586	92371
LSVMG (718)	325	55%	21218	90520

The number of bins for the Algorithm LSVMG was set to $B = 40$, and the ER random graphs were chosen for projections with $p = 0.8$. The number of projections performed was $N = 5$, for a total of $\hat{B} = 718$ selected samples. As in the experiment shown for Test set 2, we solved the problem with the complete data set and by using a random sample of the data of the same size as the number of samples selected using our algorithm. The results are shown in Tables 6 and 7.

As remarked for the previous test set, there is a considerable improvement in terms of training time for the LSVMG-Algorithm as compared to the complete data set. It is also worthwhile remarking that selecting a sub-sample actually improves accuracy, probably because of reducing over-fitting given by the complexity of the problem, due to the similarity between graphs of different classes, as shown in figure 1. Also, it is interesting that there is an accuracy improvement obtained by the LSVMG-Algorithm as compared to the random sampling, probably because of the more precise selection of the representative data points.

Table 7 shows the results obtained using the Random Walk kernel. As can be seen, accuracy for both sub-sampling algorithms deteriorated when using the Random Walk Kernel so this similarity measure does not seem to be adequate for the problem. As for the Simple kernel, no results were available after 3 days using the whole data set.

VII. CONCLUDING REMARKS

In this paper a novel approach for classifying graphs using Support Vector Machines was presented. The approach is based on using projection-based Locality-sensitive Hashing functions on random graphs, in order to find significant samples of the whole data set.

Because the size of the optimization problem is reduced, the solution is found in a fraction of the time that is required if the whole data set is used. And, this is performed without significantly degrading the classification errors which illustrates the relevance of the subset found by the algorithm. This relevance is also supported by the results obtained when comparing the performance of the algorithm with SVM using a simple random subsampling of the data set. Further research will include experiments with alternative graph kernels and applications to larger real life data sets.

REFERENCES

[1] J. L. Balcazar, Y. Dai, J. Tanaka, and O. Watanabe, "Fast training algorithms for support vector machines," *Theory Comput. Syst.*, vol. 42, no. 4, pp. 568–595, 2008.

[2] A.-L. Barabàsi and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, Oct. 1999.

[3] P. Birzhandi, K. T. Kim, and H. Y. Youn, "Reduction of training data for support vector machine: A survey," *Soft Comput.*, vol. 26, no. 8, pp. 3729–3742, Apr. 2022.

[4] K. Borgwardt, E. Ghisu, F. Llinares-López, L. O'Bray, and B. Rieck, "Graph kernels: State-of-the-art and future challenges," *Found. Trends Mach. Learn.*, vol. 13, nos. 5–6, pp. 531–712, 2020.

[5] R. Bárcenas, M. Gonzalez-Lima, J. Ortega, and A. Quiroz, "On subsampling procedures for support vector machines," *Mathematics*, vol. 10, no. 20, p. 3776, Oct. 2022.

[6] S. A. Camelo, M. D. González-Lima, and A. J. Quiroz, "Nearest neighbors methods for support vector machines," *Ann. Oper. Res.*, vol. 235, no. 1, pp. 85–101, Dec. 2015.

[7] P. Corcoran, "An end-to-end graph convolutional kernel support vector machine," *Appl. Netw. Sci.*, vol. 5, no. 1, pp. 5–39, Dec. 2020.

[8] N. Cristianini and J.Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge, U.K.: Cambridge Univ. Press, 2000.

[9] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proc. 20th Annu. Symp. Comput. Geometry*, Jun. 2004, pp. 253–262.

[10] P. Erdos and A. Rényi, "On random graphs I," *Pub. Math.*, vol. 6, pp. 290–297, Dec. 1959.

[11] T. Gartner, P. Flach, and S. Wrobel, "On graph kernels: Hardness results and efficient alternatives," in *Proc. 16th Annu. Conf. Learn. Theory*, B. Scholkopf and M. Warmuth, Eds., 2003, pp. 129–143.

[12] M. D. Gonzalez-Lima and C. C. Ludeña, "Using locality-sensitive hashing for SVM classification of large data sets," *Mathematics*, vol. 10, no. 11, p. 1812, May 2022.

[13] D. Horn, A. Demircioğlu, B. Bischl, T. Glasmachers, and C. Weihs, "A comparative study on large scale kernelized support vector machines," *Adv. Data Anal. Classification*, vol. 12, no. 4, pp. 867–883, Dec. 2018.

[14] R. Hu, X. Zhu, Y. Zhu, and J. Gan, "Robust SVM with adaptive graph learning," *World Wide Web*, vol. 23, no. 3, pp. 1945–1968, May 2020.

[15] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *Proc. 30th Annu. ACM Symp. Theory Comput.*, 1998, pp. 604–613.

[16] U. Kang, H. Tong, and J. Sun, "Fast random walk graph kernel," in *Proc. 12th SIAM Int. Conf. Data Mining (SDM)*. Philadelphia, PA, USA: SIAM, 2012, pp. 828–838, doi: 10.1137/1.9781611972825.71.

[17] N. M. Kriege, F. D. Johansson, and C. Morris, "A survey on graph kernels," *Appl. Netw. Sci.*, vol. 5, no. 1, pp. 1–42, Dec. 2020.

[18] Y.-J. Lee and S.-Y. Huang, "Reduced support vector machines: A statistical theory," *IEEE Trans. Neural Netw.*, vol. 18, no. 1, pp. 1–13, Jan. 2007.

[19] C.-J. Lin, "On the convergence of the decomposition method for support vector machines," *IEEE Trans. Neural Netw.*, vol. 12, no. 6, pp. 1288–1298, Nov. 2001.

[20] T. Ma, H. Wang, L. Zhang, Y. Tian, and N. Al-Nabhan, "Graph classification based on structural features of significant nodes and spatial convolutional neural networks," *Neurocomputing*, vol. 423, pp. 639–650, Jan. 2021.

[21] J. Nalepa and M. Kawulok, "Selecting training sets for support vector machines: A review," *Artif. Intell. Rev.*, vol. 52, no. 2, pp. 857–900, Aug. 2019.

[22] M. Neuhaus and H. Bunke, *Bridging the Gap Between Graph Edit Distance and Kernel Machines*. Singapore: World Scientific, 2007.

[23] J. Platt, "Sequential minimal optimization: A fast algorithm for training support vector machines," Microsoft, Redmond, WA, USA, Tech. Rep. MSR-TR-98-14, Apr. 1998.

[24] Y. Rizk, N. Mitri, and M. Awad, "An ordinal kernel trick for a computationally efficient support vector machine," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2014, pp. 3930–3937.

[25] B. Rozemberczki, O. Kiss, and R. Sarkar, "Karate club: An API oriented open-source Python framework for unsupervised learning on graphs," in *Proc. 29th ACM Int. Conf. Inf. Knowl. Manage.*, Oct. 2020, pp. 3125–3132.

[26] M. G. Seenappa, K. Potika, and P. Potikas, "Short paper: Graph classification with kernels, embeddings and convolutional neural networks," in *Proc. 1st Int. Conf. Graph Comput. (GC)*, Sep. 2019, pp. 88–93.

[27] G. Siglidis, G. Nikolentzos, S. Limnios, C. Giatsidis, K. Skianis, and M. Vazirgiannis, "Grakel: A graph kernel library in Python," *J. Mach. Learn. Res.*, vol. 21, no. 54, pp. 1–5, 2020.

[28] K. Tsuda and H. Saigo, "Graph classification," in *Managing and Mining Graph Data* (Advances in Database Systems), vol. 11. Cham, Switzerland: Springer, 2010.

MARÍA D. GONZALEZ-LIMA received the Ph.D. degree in applied and computational mathematics from Rice University, Houston, TX, USA. After the Ph.D., she returned to her home country, Venezuela, and was with Simon Bolivar University, until 2012, when she relocated in Colombia. She is currently a Professor and a Researcher with the University of the North, Colombia, Barranquilla, Colombia. Her research interests include numerical optimization, applications, and support vector machines.

CARENNE C. LUDEÑA received the Ph.D. degree in mathematical statistics from the University of Orsay (Paris XI), in 1996. She was with the Venezuelan Institute of Scientific Research, for 15 years, and then five years with the Central University of Venezuela, coordinating the master's degree in random models. She now lives in Colombia. She is currently a Lead Data Scientist with Matrix CPM Solutions, a data-solution provider company. Her research interests include process statistics, data mining, and mining of large volumes of data in variable format, in particular, graphs and text.

GIBRAN G. OTAZO-SANCHEZ received the B.S. degree in mathematics from the Central University of Venezuela, in 2012. He is currently pursuing the Ph.D. degree in engineering in the statistics program with Simón Bolivar University. He also has working experience of nine years as a Data Scientist. He is a Data Scientist Senior with the AI Factory Department, BBVA. His research interests include machine learning, graph theory, and data mining of large scale datasets.