MASTER'S DEGREE THESIS

Master of Science in Advanced Mathematics and Mathematical Engineering

Title: Cholesky-Factorized Sparse Kernel in Support Vector Machines

Author: Alhasan Abdellatif

Advisors: Jordi Castro, Lluís A. Belanche

Departments: Department of Statistics and Operations Research and Department of Computer Science

Academic year: 2018-2019

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat de Matemàtiques i Estadística

Universitat Politècnica de Catalunya
Facultat de Matemàtiques i Estadística

Master in Advanced Mathematics and Mathematical Engineering

Master's thesis

Cholesky-Factorized Sparse Kernel in Support Vector Machines

Alhasan Abdellatif

Supervised by Jordi Castro and Lluís A. Belanche

June, 2019

I would like to express my gratitude to my supervisors Jordi Castro and Lluís A. Belanche for the continuous support, guidance and advice they gave me during working on the project and writing the thesis. The enthusiasm and dedication they show have pushed me forward and will always be an inspiration for me. I am forever grateful to my family who have always been there for me. They have encouraged me to pursue my passion and seek my own destiny. This journey would not have been possible without their support and love.

Abstract

Support Vector Machine (SVM) is one of the most powerful machine learning algorithms due to its convex optimization formulation and handling non-linear classification. However, one of its main drawbacks is the long time it takes to train large data sets. This limitation is often aroused when applying non-linear kernels (e.g. RBF Kernel) which are usually required to obtain better separation for linearly inseparable data sets. In this thesis, we study an approach that aims to speed-up the training time by combining both the better performance of RBF kernels and fast training by a linear solver, LIBLINEAR. The approach uses an RBF kernel with a sparse matrix which is factorized using Cholesky decomposition. The method is tested on large artificial and real data sets and compared to the standard RBF and linear kernels where both the accuracy and training time are reported. For most data sets, the result shows a huge training time reduction, over 90%, whilst maintaining the accuracy.

Keywords

Support Vector Machines, RBF Kernel, Sparse Kernel, Large data sets, Machine Learning

Contents

1. Introduction

Classification in machine learning is a problem of assigning certain labels to patterns using a pre-labeled training set. Support Vector Machine (SVM) is a supervised learning model used to perform tasks such as classification. It can be considered one of the most powerful machine learning algorithms due to several properties including that it is formulated as a convex optimization problem so a global minimum is guaranteed, it can adapt to non-linear problems using the kernel trick which uses non-linear kernels to map points into a feature space, and it is memory efficient as it uses a subset of the data points called support vectors. In addition, it is one of the leading models in text and document categorization.

However one of the main limitations of using SVM is that it does not scale up well to large data sets as its time complexity is $O(m^3)$ [5], where m is the number of the training points. Hence, solving the associated optimization problem with very large data sets can be computationally expensive. Several solvers have been developed to solve the optimization problem associated with SVM including LIBSVM library which has a time complexity between $O(n \times m^2)$ and $O(n \times m^3)$ [4], where n is the dimension of the training points, and LIBLINEAR library [20] which is practically much faster than LIBSVM [6] but can work only with a linear kernel which is a disadvantage, since it is often required to better separate the data in high-dimensional feature space using non-linear kernels (e.g. RBF Kernel).

Many contributions have been made to speed up SVM classification for large data sets. One idea is to perform a preprocessing step on the large data set to obtain a smaller but yet well-representing subset. The smaller subset is then supplied directly to an SVM solver which takes much shorter time compared to training on the whole large data set. For example, in [21], they performed two steps to reduce the training set: a data cleaning based on a bootstrap sampling and algorithm to extract informative patterns. In [13], a data compression technique based on Learning Vector Quantization (LVQ) neural network is tested on large training sets and in [10] an algorithm for instance selection is developed especially for multi-class problems and is based on clustering. However, one drawback of these approaches is that removing some data points from the training set can often result in an underfitting model that performs poorly on the original data set.

Another idea is to replace the RBF kernel by an approximate one, which is much faster to compute, and hence training the data set can take less time. The experiments in [8] show that the approximate kernel can be more than 30 times faster than the exact RBF kernel. In [9], they substitute the RBF kernel by a kernel with a sparse matrix into the dual formulation of the optimization problem, in which they achieve a time reduction of nearly 47% while the accuracy is preserved. While replacing the RBF kernel by an approximate or a sparse RBF kernel can offer a time reduction, these approaches used non-linear solvers (e.g. LIBSVM) which are relatively slow.

In this thesis, we study an approach that speeds-up SVM training combining both the better separation by non-linear kernels and the faster training by a linear solver, LIBLINEAR. The approach uses a compactly-supported RBF kernel which have been studied and used in [9] and [3]. Its associated sparse gram matrix is factorized, using *Cholesky decomposition*, into a lower triangular matrix which acts as the mapped training points in the feature space. The matrix is then fed into LIBLINEAR, to solve the primal optimization problem associated with SVM. The the dual variables are then computed to classify new test points. To verify our approach, it is tested against many real data sets and an artificial data set of large sizes. The training time and test accuracy are reported and compared to the ones obtained using LIBSVM with RBF Kernel and LIBLINEAR with a linear kernel.

The remainder of the thesis is organized as follows: an overview of SVM classification is provided in section 2.1 and some known SVM solvers are discussed in section 2.2. In section 2.3, the compactly-supported RBF kernel is discussed and some of its properties are highlighted in section 3. In section 4 some related literature results are shown. Our method is then explained discussing both training and the testing procedures in section 5. The results of the approach are presented and compared to other methods in section 6. Finally, a conclusion is drawn pointing out both the benefits and limitations of the approach in section 7.

2. Background

In this section, we point out some basics behind SVM including the hard and soft margin problems, the use of the kernel trick, the primal and dual formulations and multi-class classification. We then review some popular SVM solvers and their main applications. The sparse kernel used in the approach is introduced and finally we discuss *Cholesky factorization* and the *approximate minimum degree* permutation algorithm. In the discussion below, consider m as the number of training points and n as the number of features.

2.1 Support Vector Machines

Support Vector Machine (SVM) is a machine learning algorithm that was first introduced by Vapnik in 1963 [23] and further developed in 1995 [5]. It tries to find a hyperplane that best separates the data points into two different classes. The hyperplane is then used to classify new points to either of the two classes. One method to obtain such hyperplane is to maximize the separation between the two classes, this separation is called a *margin*. Any hyperplane can be written as

$$w^T x - b = 0 \tag{1}$$

where $w \in \mathbb{R}^n$ is the normal vector to the hyperplane, $x \in \mathbb{R}^n$ any point satisfying the hyperplane and

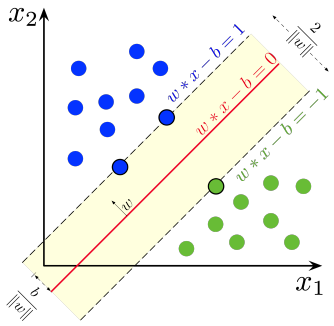

Figure 1: Margin between two classes in SVM, adapted from [12].

$b \in \mathbb{R}$ is the intercept.

Consider binary-classes data points $x_i \in \mathbb{R}^n$ and their labels $y_i \in \{1, -1\}$ $\forall i = 1 \dots m$, the method tries to obtain the best parameters w and b such that the associated hyperplane maximizes the margin as

depicted in Figure 1.

Lemma 2.1.1 *The margin between two parallel hyperplanes* $w^T x - b = a_1$ *and* $w^T x - b = a_2$ *is* $\frac{|a_1 - a_2|}{\|w\|}$.

Proof. Consider a point x_1 lies on the first hyperplane and a point x_2 lies on the second, then x_2 can be written as $x_2 = x_1 + \alpha w$. Then we have

$$\begin{aligned} w^T x_2 &= w^T x_1 + \alpha w^T w \\ a_2 + b &= a_1 + b + \alpha \|w\|^2 \\ \alpha &= \frac{a_2 - a_1}{\|w\|^2}. \end{aligned}$$

The margin between the hyperplanes is $\|\alpha w\|$ which is

$$\|\alpha w\| = |\alpha| \|w\| = \frac{a_2 - a_1}{\|w\|^2} \|w\| = \frac{|a_1 - a_2|}{\|w\|}.$$

□

From Lemma 2.1.1, it can be deduced that the margin between the two hyperplanes $w^T x - b = 1$ and $w^T x - b = -1$ is $\frac{2}{\|w\|}$. Since maximizing $\frac{2}{\|w\|}$ is equivalent to minimizing $\frac{1}{2}\|w\|^2$, the optimization problem associated with SVM can be formulated as

$$\min_{w,b} \quad \frac{1}{2}\|w\|^2 \tag{2}$$

$$\text{Subject to} \quad y_i(w^T x_i - b) \geq 1 \; \forall i = 1 \ldots m \tag{3}$$

The above optimization problem is called *hard margin* SVM and only works with data sets that are linearly separable, i.e., there exists a hyperplane that can separate all labeled points into the two classes with no point lies in the incorrect side of the margin. Another formulation was developed called *soft margin* that allows each data point x_i to have a corresponding error $\xi_i \geq 0$ such that points with $\xi_i > 0$ lie in the incorrect side. The difference between the hard and soft margins is shown in Figure 2. The soft margin

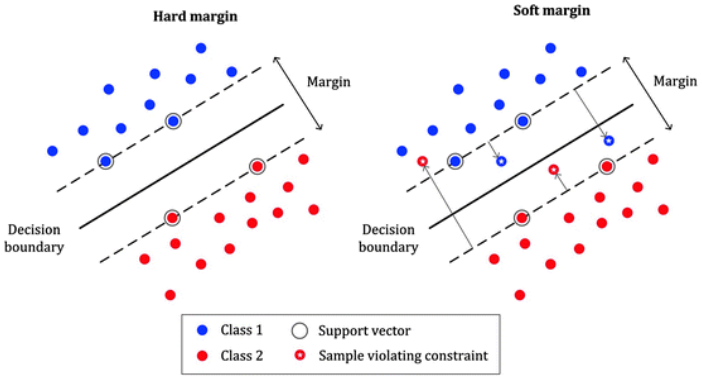

Figure 2: Hard and soft margin in SVM

optimization problem can be formulated as following

$$\min_{w,b,\vec{\xi}} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}\xi_i \tag{4}$$

$$\text{Subject to} \quad y_i(w^T x_i - b) \geq 1 + \xi_i \tag{5}$$

$$\xi_i \geq 0 \;\; \forall i = 1 \ldots m \tag{6}$$

where $C > 0$ is a hyper-parameter that represents the trade-off between the training error and margin maximization. In order to classify a new data point, $z \rightarrow \{1, -1\}$, we use the following rule

$$z \rightarrow \text{sgn}(w^T z - b), \tag{7}$$

where $sgn(x)$ is the sign function. The above methods work with data points in the input space, another method was developed in [5] uses what so-called *kernel trick* which maps the data points from the input space to a high dimensional feature space allowing better separation for linearly inseparable data points. The method seeks to find a mapping function

$$\phi : \mathbb{R}^n \rightarrow \mathbb{R}^N, \tag{8}$$

where N is the dimension of the feature space, such that the new feature space provides a better separation, however it is not clear how we define the mapping ϕ.

It turns out that the kernel function defined by

$$k(x_i, x_j) = \phi(x_i)^T \phi(x_j) \tag{9}$$

appears directly in the dual formulation of the soft margin optimization problem with no need to compute the mapping $\phi(x)$ if we know $k(x_i, x_j)$ beforehand. The kernel function $k(x_i, x_j)$ is associated with a gram matrix K where $K_{ij} = \phi(x_i)^T \phi(x_j)$ which can be written as

$$K = BB^T, \tag{10}$$

where B is an $(m \times N)$ matrix and each row represents a data point mapped in the feature space. When ϕ is the identity function, k is often called a linear kernel in which case $k(x_i, x_j) = x_i^T x_j$ and B will be the original $m \times n$ data points matrix. Several non-linear kernels have been developed such as the Radial Basis Function (RBF) kernel

$$k_{rbf}(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right) \tag{11}$$

and polynomial kernel

$$k_{pol}(x, y) = (x^T y + c)^d, \tag{12}$$

where c, d and σ are parameters to be tuned. In our method, we focus only on the RBF kernel.
In order to derive the dual formulation, we use Wolfe Duality.

Theorem. 2.1.2 *(Wolfe Duality) If $f(x), h(x)$ and $g(x)$ are convex and differentiable functions in the primal problem*

$$\min_{x} \quad f(x) \tag{13}$$

$$\textit{Subject to} \quad h(x) = 0 \tag{14}$$

$$g(x) \leq 0, \tag{15}$$

then a sufficient and necessary condition for the optimality of the dual function $q(\lambda, \mu) = \min_x L(x, \lambda, \mu)$ *is*

$$\nabla_x L(x, \lambda, \mu) = 0, \tag{16}$$

where L is the Lagrangian function, λ *and* μ *are the dual variables associated with the constraints (14) and (15), respectively.*

The dual problem of of (13) can then be written as

$$\begin{aligned} \max_{x,\lambda,\mu} \quad & L(x, \lambda, \mu) = f(x) + \lambda^T h(x) + \mu^T g(x) && (17)\\ \text{Subject to} \quad & \nabla_x L(x, \lambda, \mu) = 0 && (18)\\ & \mu \geq 0. && (19) \end{aligned}$$

Since the objective function (4) and the constraints (5) and (6) are convex and differentiable functions, the Wolfe duality can be used. The Lagrangian function of the soft margin problem (4), using $\phi(x)$ instead of x, is

$$L(w, b, \xi, \lambda, \mu) = \frac{1}{2} w^T w + C\, e^T \xi + \lambda^T(-D(Bw - be) - \xi + e) - \mu^T \xi, \tag{20}$$

where $\lambda \in \mathbb{R}^m$ and $\mu \in \mathbb{R}^m$ are the dual variables of the constraints (5) and (6), respectively, D is an $m \times m$ diagonal matrix with the labels of the data points and $e \in \mathbb{R}^m$ is a vector of ones. Then the dual problem can be written as

$$\begin{aligned} \max_{w,b,\xi,\lambda,\mu} \quad & \frac{1}{2} w^T w + C\, e^T \xi + \lambda^T(-D(Bw - be) - \xi + e) - \mu^T \xi && (21)\\ \text{Subject to} \quad & w - (\lambda^T DB)^T = 0 && (22)\\ & \lambda^T De = 0 && (23)\\ & C\, e - \lambda - \mu = 0 && (24)\\ & \lambda \geq 0, \mu \geq 0. && (25) \end{aligned}$$

Substituting equation (22) in (21) and using equations (23) and (24), we can arrive at the following formulation

$$\begin{aligned} \max_{\lambda} \quad & \lambda^T e - \frac{1}{2} \lambda^T DBB^T D\lambda && (26)\\ \text{Subject to} \quad & \lambda^T De = 0 && (27)\\ & 0 \leq \lambda \leq C. && (28) \end{aligned}$$

Hence, knowing the kernel matrix $K = BB^T$ is enough to solve the SVM dual problem. After solving the last optimization problem and obtaining λ, equation (22) can be used to calculate the weights

$$\begin{aligned} w &= \sum_{i=1}^{m} \lambda_i y_i \phi(x_i) && (29)\\ &= B^T D\lambda. && (30) \end{aligned}$$

In order to calculate the intercept b, we need to find a point x_k which is a support vector, such point will have $\xi_k = 0$ and $0 < \lambda_k < C$. Using $y_k(w^T\phi(x_k) - b) = 1$, we get

$$b = w^T\phi(x_k) - \frac{1}{y_k} \tag{31}$$

$$= \sum_{i=1}^{m} \lambda_i y_i k(x_i, x_k) - y_k. \tag{32}$$

To classify a new point z using the dual formulation we use

$$z \to \text{sgn}(w^T\phi(z) - b) = \text{sgn}(\sum_{i=1}^{m} \lambda_i y_i k(x_i, z) - b). \tag{33}$$

To perform multi-classification, two common methods are one-vs-rest and one-vs-one. In our work, we focus on one-vs-rest method which trains k classifiers, where k is the number of classes in the data set. For each classifier, one class is labeled by 1 and the rest by -1 and we assign to a new point z the label that maximizes the decision function

$$z \to \underset{j=1\ldots k}{\text{argmax}}(\sum_{i=1}^{m} \lambda_i^j y_i k(x_i, z) - b^j). \tag{34}$$

Before the sparse kernel is introduced, some properties of kernels are listed below.

Theorem. 2.1.3 *Let $x_i \in \mathbb{R}^n, \forall i = 1, \ldots, m$, then a symmetric function $k(x, y)$ is a kernel if and only if the matrix*

$$K = (k(x_i, x_j))_{i,j=1,\ldots,m}$$

is positive semidefinite.

Proof. Proving the forward direction, let $k(x, y)$ be a kernel and using equation (9), then for any non-zero vector $v \in \mathbb{R}^m$, we have

$$\begin{aligned} v^T K v &= \sum_{i,j=1,\ldots,m}^{m} v_i v_j K_{ij} \\ &= \sum_{i,j=1,\ldots,m}^{m} v_i v_j \langle \phi(x_i), \phi(x_j) \rangle \\ &= \langle \sum_{i=1}^{m} v_i \phi(x_i), \sum_{j=1}^{m} v_j \phi(x_j) \rangle \\ &= \| \sum_{i=1}^{m} v_i \phi(x_i) \|^2 \geq 0 \end{aligned}$$

which proves that K is positive semidefnite. To prove the reverse direction, It is enough to prove the following result on positive semidefinite matrices: *If K is positive semidefinite matrix, then it can be written as $K = BB^T$ for some real matrix B.*
Then using equation (10), it is straight forward to see that $k(x, y)$ is a kernel function.

This result can be proved by observing that K is a symmetric diagonalizable matrix and thus can be written as $K = Q\Lambda Q^T$, where Q is an orthogonal matrix and Λ is a diagonal matrix. Let $B = Q\sqrt{\Lambda}$, then

$$BB^T = Q\sqrt{\Lambda}\sqrt{\Lambda}Q^T = Q\Lambda Q^T = K$$

and this completes the proof. □

Lemma 2.1.4 *The multiplication of two kernels is also a kernel.*

Proof. Let A and B be kernels and let $(C)_{ij} = (A)_{ij}(B)_{ij}$, this is known as Hadamard product and is denoted by $C = A \circ B$. Using eigendecomposition, we can write

$$A = \sum \alpha_i a_i a_i^T$$
$$B = \sum \beta_j b_j b_j^T.$$

Then we have

$$\begin{aligned} A \circ B &= \sum_{ij} \alpha_i \beta_j (a_i a_i^T) \circ (b_j b_j^T) \\ &= \sum_{ij} \alpha_i \beta_j (a_i \circ b_j)(a_i \circ b_j)^T. \end{aligned}$$

Since $(a_i \circ b_j)(a_i \circ b_j)^T$ is positive semidefinite and $\alpha_i \beta_j \geq 0$ for any i, j, then $A \circ B$ is positive semidefinite and hence C is a kernel using Theorem 2.1.3. □

2.2 Support Vector Machines Solvers

Many libraries have been developed to solve the optimization problems associated with support vector machines, such as SVMLIGHT, LIBSVM and LIBLINEAR. We consider here LIBSVM and LIBLINEAR libraries. LIBSVM [4] was first developed in 2000 and can work with both a linear kernel and non-linear kernels (e.g RBF kernel). It has a time complexity between $O(n \times m^2)$ and $O(n \times m^3)$, which can be very slow in case of very large datasets. It uses a decomposition method named Sequential Minimal Optimization (SMO) which was developed in 1998 at Microsoft Research Lab [17]. The method breaks the quadratic optimization problem into small sub-problems which are solved analytically.

On the other hand, LIBLINEAR [20] works only with linear classification problems. It can be very efficient when the number of features is large and it is often much faster than LIBSVM. For example, in [6], they tested both libraries on a data set where the numbers of instances and features are $20,242$ and $47,236$, respectively. LIBLINEAR gave almost the same accuracy as LIBSVM with a training time of about 3 seconds whereas LIBSVM took nearly 346 seconds. LIBLINEAR solves an unconstrained optimization problem of the form:

$$\min_w \ \frac{1}{2} w^T w + C \sum_{i=1}^{m} max(0, 1 - y_i w^T x_i), \tag{35}$$

where $C > 0$ is a penalty parameter.

2.3 Compactly-Supported RBF Kernel

SVM solvers do not store the gram matrix associated with the RBF kernel, instead each entry is directly computed and used in solving the optimization problem. This is because such a matrix is dense and the available storage will not meet its space requirement. Since sparse matrices offer much smaller space requirements and reduction of time complexity, a family of kernels with sparse gram matrices have been studied and used in [9], [3] and [14]. These kernels can be constructed by multiplying the RBF kernel by some compactly-supported radial basis function. We particularly consider the function

$$k_c(x, y) = (1 - \alpha\|x - y\|)_+^l, \tag{36}$$

where $a_+ = max\{a, 0\}$, $x, y \in \mathbb{R}^n$ and $\alpha > 0$. R. Askey in [19] proved that k_c is positive definite if $l \geq \lfloor n/2 \rfloor + 1$, where $\lfloor . \rfloor$ is the floor function. Satisfying the condition on l, k_c can then be regarded as a kernel function using Theorem 2.1.3.

Applying Lemma 2.1.4, an RBF kernel with a sparse gram matrix can be constructed by multiplying the kernel k_c by the RBF kernel:

$$k(x, y) = k_c(x, y)k_{rbf}(x, y) \tag{37}$$

$$= (1 - \alpha\|x - y\|)_+^l \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right). \tag{38}$$

The parameter α controls the kernel sparsity, the percentage of zero elements in the kernel matrix, increasing α makes the kernel matrix more sparse. The parameter l affects the smoothness of the kernel function.

2.4 Cholesky Decomposition

The Cholesky decomposition [11] of a real positive definite matrix A is of the form

$$A = LL^T,$$

where L is a lower-triangular matrix, called Cholesky factor. The Cholesky factor is unique if A is positive definite matrix, whereas it needs not be unique if A is positive semidefinite.

Cholesky decomposition finds many applications in numerical solutions, for example it can be more efficient than LU factorization in solving a system of linear equations [18]. For large matrix A, computing the Cholesky factor L can be computationally expensive, however if the matrix is sparse, Cholesky factorization can be very fast. Several libraries have been developed to perform Cholesky factorization including CHOLMOD library [22], which can also perform a sparse Cholesky factorization.

Applying Cholesky decomposition to a sparse matrix does not always guarantee that the Cholesky factor will be as sparse as possible. Several algorithms have been studied to perform a permutation of the rows and columns of the matrix A before applying the Cholesky factorization to increase the sparsity of the Cholesky factor. One common algorithm is the *approximate minimum degree* [2], which tries to find the best permutation P so that L is as sparse as possible. In Figure 3, the Cholesky factorization is done on an arrowhead matrix before and after applying the approximate minimum degree algorithm in Matlab (AMD function), it shows that the number of nonzero elements is much smaller in the Cholesky factor when using approximate minimum degree algorithm.

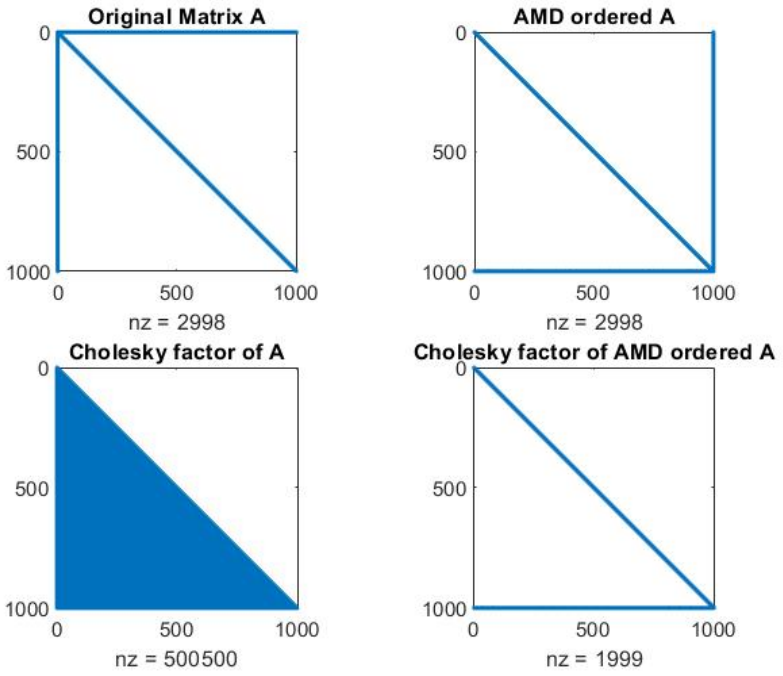

Figure 3: Comparison between Cholesky factors with and without applying AMD function in Matlab.

3. Sparse Kernel Properties

In this section, some properties of the sparse kernel are discussed. Since the choice of the parameters is crucial in training SVM, the limit values of the α parameter are discussed below .

Lemma 3.1 *Let K be the gram matrix associated with the kernel $k(x_i, x_j)$ defined in equation (38), then*

$$\lim_{\alpha \to 0} K = K_{rbf}$$
$$\lim_{\alpha \to \infty} K = I_m$$

where I_m is an $m \times m$ identity matrix and K_{rbf} is the matrix of the RBF kernel.

Proof. Using the expression of $k_c(x_i, x_j)$, it is clear that for any i, j

$$\lim_{\alpha \to 0} (1 - \alpha \|x_i - x_j\|)_+^l = 1,$$

which from equation (37) results in $\lim_{\alpha \to 0} K = K_{rbf}$. On the other hand

$$\lim_{\alpha \to \infty} k_c(x_i, x_j) = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases}$$

since $(K_{rbf})_{ii} = 1 \, \forall i = 1 \dots m$ then $\lim_{\alpha \to \infty} K = I_m$. □

From Lemma 3.1, we can observe that as we increase α, the sparse kernel matrix K gets sparser. As α becomes very large the kernel matrix becomes totally distorted as the identity matrix. The sparse kernel follows nearly the same performance as the RBF kernel when α is very small.

Theorem 3.2 : (Gershgorin Circle) *Let A be a matrix with entries a_{ij} and let $R_i = \sum_{j \neq i} |a_{ij}|$. Let $D(a_{ii}, R_i)$ be a closed disk centered at a_{ii} with radius R_i. Then every eigenvalues of A lies within at least one of the Gershgorin disks $D(a_{ii}, R_i)$*

Lemma 3.3 *The radius of the Gershgorin Disk of matrix of the sparse kernel k defined by (37) is less than that associated with the RBF Kernel.*

Proof. Let d_{ij} and z_{ij} be the entries of the matrices of k_{rbf} and k_c, respectively. Since the entries of both matrices are non-negative, $d_{ij} > 0$ and $z_{ij} \geq 0$, we can drop out the notation of absolute values defined of radii in Theorem 3.2. We can show that

$$\sum_{j \neq i} K_{ij} = \sum_{j \neq i} d_{ij} z_{ij} < \sum_{j \neq i} d_{ij}.$$

if we can prove that,

$$z_{ij} < 1 \text{ for } j \neq i.$$

Since $\alpha > 0$ and l is a positive integer, it is clear that $z_{ij} = \max\{0, (1 - \alpha \|x_i - x_j\|)^l\} < 1$ for $j \neq i$. □

4. Literature Review

In section we review some literature papers that have used the compactly supported kernel in support vector machines. A closely related work was developed in [9], where they used the same sparse kernel in equation (38). They pointed out some properties of the sparse kernel and describe techniques to tune the sparsity parameter α. One technique is to perform a trade-off between similarity $A(\alpha)$ and sparsity $S(\alpha)$, they define $A(\alpha)$ and $S(\alpha)$ as follows:

$$A(\alpha) = \frac{\langle K, K_{rbf} \rangle_F}{\sqrt{\langle K, K \rangle_F \langle K_{rbf}, K_{rbf} \rangle_F}}, \tag{39}$$

where $\langle K_1, K_2 \rangle_F = \sum_{i,j=1}^{m} K_1(x_i, x_j) K_2(x_i, x_j)$ is the Frobenius inner product between two matrices and

$$S(\alpha) = \frac{\text{number of zero entries in K}}{m^2}. \tag{40}$$

The trade-off can be done by maximizing a linear combination between similarity and sparsity,

$$\max_{\alpha} A(\alpha) + \beta S(\alpha). \tag{41}$$

where $\beta > 0$ is a tuning parameter. Another procedure is to pre-evaluate the maximum number of nonzero elements that can be stored on the machine and from that we can choose a lower bound of the sparsity. For example, if only 25% of the nonzero entries can be stored, then we may set $S(\alpha) \geq 0.75$ and $1/\alpha$ to

be the first quartile of the pairwise distances $\|x_i - x_j\|, i, j = 1, \dots m$.

One application they performed is replacing the matrix of the RBF kernel in the dual problem of SVM, $K = BB^T$ in equation (26), by the sparse matrix of the compactly supported RBF kernel. They tested their method on a small data set (100 instances) and reported the accuracy which was very close to the one obtained using the RBF kernel. They also used the sparse kernel in training an LS-SVM classifier, which instead of solving a quadratic optimization problem solves a linear system, and tested that on a relatively larger data set of (5000) where they reported both accuracy and time, in the best case the sparse kernel saved the computation time by 47% while maintaining good accuracy.

In [3], they used the same sparse kernel for regression problems using LS-SVM. They set the parameter $\alpha = 1/3\sigma$, where σ is the same parameter in the RBF kernel. They solved the linear system associated with LS-SVM by using Cholesky factorization. They tested both the compactly-supported kernel and the RBF kernel on a sinc-toy problem with 1334 training points. The compactly-supported kernel obtained a similar accuracy with a speed-up of about 50% compared to the RBF kernel.

While the work done in [9] was able to reduce the training time, they solved the dual optimization problem using non-linear solvers such as LIBSVM which is often slow. Although they did not show results for very large data sets, it is likely that the speed-up by the sparse kernel will not compensate the painful slowness in LIBSVM for very large data sets. In our method, we were able to reach a speed up of over 90% for data sets of more than 100k and 200k instances. In the next section, we dicuss the details of our method.

5. Method

As discussed in section 2.2, LIBLINEAR is much faster than LIBSVM but only works with a linear kernel and hence favours data sets where the linear kernel is enough to best separate the points. On the other hand, when n is much smaller than m, non-linear kernels are favoured to get better accuracy in which case LIBSVM is better.

Our approach combines both the better performance of non-linear kernels and the faster training by LIBLINEAR. It seeks to find an $m \times N$ matrix B that represents the data points in N-dimensional feature space which best separates the data such that $K = BB^T$, where K is a non-linear kernel. Having obtained B, we can directly feed it to LIBLINEAR, as input matrix, to solve the primal optimization problem. It is not clear however what is the optimal dimension N of the space in which we can separate the data points.

Definition 5.1 *(Shattered set) Let A be a set and C be a class of sets, we say that C shatters A if for each subset a of A, there is some element c of C such that*

$$a = c \cap A.$$

In the context of classification, shattering means that for all possible assignments of the labels to the set points A , any classifier model $c \in C$ is able to correctly classify them all.

Definition 5.2 *Vapnik–Chervonenkis (VC) dimension of a class of classifier is the maximum number of points the classifiers can shatter.*
The VC dimension measures the complexity of the set of classifiers. For example, a straight line in $\mathbb{R}^2$ has

a VC dimension of 3, meaning that there is a combination of 4 points in $\mathbb{R}^2$ such that no straight line can shatter them, as shown in Figure 4.

Theorem 5.3 *The VC-dimension of the set of affine classifiers* $\{f : f(x) = sgn(w^T x + b), w \in \mathbb{R}^d, b \in \mathbb{R}\}$ *is* $d + 1$.

A proof of Theorem 5.3 can be found in Appendix A. From Theorem 5.3, we can say that any m data points can be shattered in $\mathbb{R}^m$ using hyperplanes.

In our approach, we choose the dimension $N = m$, in which case the matrix B becomes an $m \times m$ square

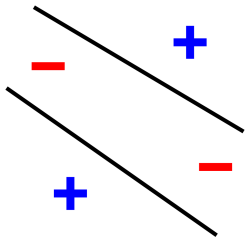

Figure 4: A line can not shatter this combination of 4 points, adapted from [16]

matrix, although there might be spaces with dimension less than m that can shatter the m points as well. The question becomes how the kernel matrix K can be decomposed into BB^T. One way is to use *Cholesky decomposition* discussed in section 2.4 in that case B becomes a lower triangular matrix, for the sake of simplicity we denote B by L.

Since kernel matrices are in general dense, Cholesky decomposition can be computationally expensive, in addition to the huge space requirements needed to store large matrices associated with large data sets. For example, a data set of $m = 200,000$ instances requires at least $160GB$, using 8 bytes for every entry, to store the kernel matrix before applying the factorization. In order to overcome the space and time problems, the sparse RBF kernel discussed in section 2.3 is used, where it acts as an approximation for the RBF kernel. The usage of the sparse kernel matrix significantly reduces the storage needed for both the kernel matrix and Cholesky factor. For example, if the sparsity is over 95% for the same 200,000 instances, we need less than $4GB$ to store the kernel matrix. In addition, sparse Cholesky factorization algorithm is much faster than the one with a dense matrix.

Although L is a triangular matrix, it is still needed to be sparse because of the space requirement. For this reason, before applying the decomposition, a permutation (P) of the rows and columns of the sparse kernel matrix K was performed by *approximate minimum degree* algorithm discussed in section 2.4. It is important to note that the permutation for the kernel matrix K is equivalent to the permutation of the rows of L, i.e., it is a reordering of the data points in the matrix L. Hence, before solving the primal optimization problem we need to reorder the labels, $y_i \, \forall i = 1 \dots m$, with the same permutation P.

Solving the primal problem using LIBLINEAR yields the weights w and the intercept b, but now $w \in \mathbb{R}^m$ since each data point (each row of the matrix L) is in $\mathbb{R}^m$. To classify a new test point $z \in \mathbb{R}^n$, we use equation (7) :

$$z \rightarrow \text{sgn}(w^T \phi(z) - b),$$

but since we don't know the explicit expression of ϕ, we can solve for the dual variables λ using equation

(22) which can be rearranged as

$$(L^T D_p)\lambda = w, \tag{42}$$

where D_p is a permuted version of the labels diagonal matrix D. The reason that D has to be permuted is that both L and w are not in the original order but in the permuted order. The linear system of $L^T D_p$ can be very large but yet sparse and triangular which makes it very fast to solve. Then we can use the classification rule (33)

$$z \to \text{sgn}(\sum_{i=1}^{m} \lambda_i^P y_i k(x_i, z) - b), \tag{43}$$

where $k(x_i, z)$ is the sparse kernel, y_i and x_i are in their original order and λ^P is a P-permuted version of λ.

The approach can be viewed as training using the primal problem and testing using the dual variables and is summarized in Algorithm 1.

Algorithm 1 Binary Classification using Cholesky-Factorized Compactly Supported Kernel

Input : Data set $x_i \in \mathbb{R}^n, y_i \in \{1, -1\}$, Test point $z \in \mathbb{R}^n$
Output : Class of z

- Choose the parameters α, σ, l and C.
- Compute the sparse matrix of the compactly supported kernel K where $K_{ij} = (1 - \alpha\|x_i - x_j\|)_+^l \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$.
- Apply the approximate minimum degree algorithm to the matrix K to find the permutation matrix P.
- Apply the permutation P to both K and y to get K_P and y_P
- Apply the Cholesky factorization to the permuted matrix K_p to obtain the Cholesky factor L.
- Solve the primal optimization problem using LIBLINEAR with matrix L and labels y_P and trade-off parameter C, to get the weights w and intercept b.
- Solve the system $(L^T D_P)\lambda = w$ for λ, where D_P is a P-permuted labels diagonal matrix.
- Apply $z \to \text{sgn}(\sum_{i=1}^{m} \lambda_i^P y_i k(x_i, z) - b)$, where λ^P is the P-permuted version of λ.

In order for the Cholesky factor L to be unique, we need to make sure that the sparse kernel matrix is positive definite, for this reason a diagonal matrix with small entries (0.0001) is added to the sparse kernel matrix. Since the diagonal elements of the kernel matrix is 1, adding such diagonal matrix will not change the main characteristics of the kernel matrix.

When choosing the parameter α, we followed a similar idea proposed in [3] where they set the parameter $\alpha = 1/3\sigma$, where σ is the same parameter in the RBF kernel. The sparse RBF kernel can then be written as

$$k(x, y) = (1 - \frac{\|x - y\|}{3\sigma})_+^l \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right). \tag{44}$$

The value of 3σ can be regarded as the cut-off of the pairwise distances such that distances that are greater than 3σ has a corresponding *zero* in the RBF kernel which means that points that are very far from each other has a *zero* similarity. The value of σ now has the reverse effect of α as discussed in section 3, decreasing σ will make the kernel matrix sparser. We tried different values of σ for each data set, hopefully to find the best value that makes the kernel matrix sparse enough and yet gives high test accuracies, details about these values are shown in the next section 6.

The parameter l is set to be the smallest integer satisfying $l \geq \lfloor n/2 \rfloor + 1$ in order to satisfy the condition set by Askey in [19] for the positive definiteness of the kernel. In the experiments we performed, we choose the value of C to be 1. We note that C can be fine tuned for every data set, though the accuracies obtained using RBF and linear kernels suggested that the value of 1 is good enough.

To perform multi-classification, we used the one-vs-rest method discussed in section 2.1. The corresponding modification in Algorithm 1 would be in the last three steps as we obtain k-values of w and b, where k is the number of classes. The system

$$(L^T D_P)\lambda_j = w_j, \forall j = 1, \dots, k \tag{45}$$

is solved k times and finally we compute the decision function $\sum_{i=1}^{m} \lambda_i^{P,j} y_i k(x_i, z) - b^j$ for each class $j = 1 \dots k$ and pick the class achieving the maximum decision value.

The implementation of the algorithm is done in Python using scikit-learn [24] libraries which have implementation of LIBSVM and LIBLINEAR. The computation of the sparse kernel uses a Cython wrapper so that the comparison could be fair as both LIBSVM and LIBLINEAR in scikit-learn uses Cython wrappers as well. For the sparse Cholesky factorization, CHOLMOD library [22] is used, it also provides the implementation of the approximate minimum degree algorithm. The code of the algorithm is shown in Appendix B and can be accessed on the online repository [1]. The next section shows the results of the approach of the sparse RBF kernel on some data sets and the comparison to RBF and linear kernels.

6. Results and Discussion

To validate our approach, we performed a comparison with the standard RBF kernel using LIBSVM and with the linear kernel using LIBLINEAR. Several data sets are used in the comparison and each one was splitted into training and testing sets. The different methods are trained on the training sets and validated on the test sets. The training time (in seconds) and the testing accuracy are reported and compared for each method.

Since our approach uses a non-linear kernel and a linear solver, we wanted to make sure that it still performs well even when a linear kernel performs poorly. For this reason, we created an artificial data set that is linearly inseparable such that the RBF kernel performs much better on accuracy than the linear kernel. The artificial data set has 3 dimensions and 3 classes and is shown in Figure 5. Other data sets are obtained from online resources [15] and summarized in Table 1.

For each data set, we tried different values of σ to obtain the best accuracy and to make sure that both the kernel and Cholesky factor matrices are as sparse as possible. Since we are limited with the amount of RAM on the machine, we start with small values of σ and then gradually increase σ and observe the corresponding test accuracy. We choose the best value of σ that gives the highest accuracy on the test set and still provides a high sparsity such that the matrices can be stored on the machines. The best value of σ, is then used to train the same data set with an RBF kernel using LIBSVM.

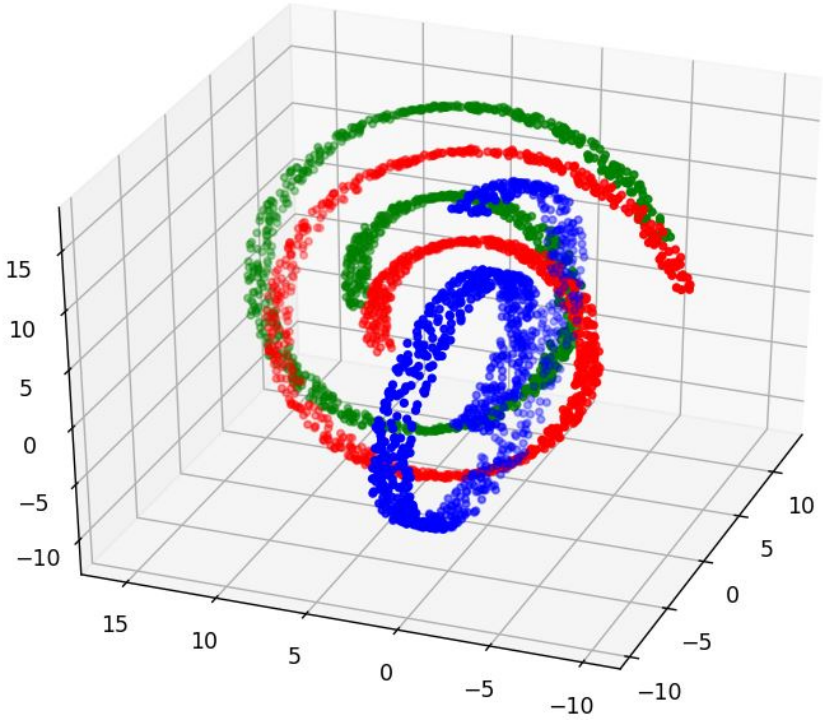

Figure 5: Swiss-roll artificial data set.

Data sets	Train points	Test points	Features	Classes
Artificial_1	225000	75000	3	3
Artificial_2	100500	49500	3	3
Skin Segmentation	108783	71913	3	2
Covtype	97319	47934	54	2
CodRNA	59535	80890	8	2
Shuttle	43500	14500	9	7
a6a	11220	21341	123	2
usps	7291	2007	256	10

Table 1: Summary of data sets used in the experiments

The comparison of our approach to LIBSVM with RBF kernel and to LIBLINEAR is presented in Table 2, where we also report the sparsity of the kernel matrix and time reduction percentage obtained when using the sparse RBF kernel. The platform used for the experiments was Intel Core i7-3520M dual core 2.9 GHz with 8MB RAM and all the codes were compiled by Python 3.6.7 under a Linux operating system (Ubuntu 18.04.2).

Concerning the first three data sets in Table 2, the linear kernel performed poorly, in terms of accuracy, compared to the RBF kernel and sparse RBF kernel. Moreover, the sparse kernel approach outperforms both the RBF kernel and linear kernel, where the accuracy is very close to that of the RBF kernel but with a huge training time reduction (between 90 and 99%) . For example, the artificial data set with over

Data sets	σ	Sparse Kernel		RBF Kernel (LibSVM)		Linear Kernel (LibLinear)		Sparsity	Time Reduction
-	-	Accuracy	Time(s)	Accuracy	Time(s)	Accuracy	Time(s)	-	-
Artificial_1	0.01	0.995	182.5	0.995	12413	0.45	17.2	0.9998	0.985
Artificial_2	0.01	0.983	51	0.995	2600	0.45	7.6	0.9998	0.98
Skin Segmentation	0.5	0.965	52.3	0.988	566.3	0.85	13.8	0.9960	0.91
CodRNA	0.3	0.934	29.7	0.935	343	0.94	9.4	0.9997	0.91
Shuttle	0.002	0.90	25	0.996	101.36	0.922	2.3	0.9992	0.75
a6a	0.3	0.76	9.43	0.77	50.86	0.84	0.23	0.9998	0.81
Covtype ($l = 29$)	0.075	0.55	548	0.92	4640	0.755	3.6	0.998	0.88
Covtype ($l = 3$)	0.075	0.90	543.4	0.92	4640.8	0.755	3.6	0.998	0.88
usps ($l = 129$)	3	0.19	50.5	0.933	13	0.92	4.63	0.8300	-
usps ($l = 3$)	3	0.72	64.5	0.933	12.8	0.92	4.63	0.8300	-

Table 2: Comparison of the experimental results using the sparse kernel, RBF kernel and linear kernel

200k instances took around 12,400 seconds using the RBF kernel, whereas the sparse RBF kernel took only 182.5 seconds which is around 68 times faster. When training on the skin data set, our approach was 10 times faster with only 2% reduction in accuracy compared to the RBF kernel. In the CodRNA data set, the accuracies of the three methods are almost the same, while our approach is much faster than the LIBSVM with RBF kernel with 91% time reduction. In the Shuttle data set, the accuracy was dropped by 10% when using the sparse kernel, which can be substantial, but provides faster performance.

In situations where the linear kernel is sufficient (e.g. a6a), LIBLINEAR gave the highest accuracy and the fastest performance as well. In the experiments with the usps and Covtype data set, the accuracy of the sparse kernel was dropped significantly, the reason is the condition set on the parameter l. When the number of features is large, the value of l will also be large as $l \geq \lfloor n/2 \rfloor + 1$ and since the values of $k_c(x, y) = (1 - \alpha\|x - y\|)^l$ defined in equation (36) is less than one then the values of the sparse RBF kernel become negligibly small compared to the value of intercept b and thus the decision function, used in equation (43) as

$$g(z) = sgn(\sum_{i=1}^{m} \lambda_i^P y_i k(x_i, z) - b),$$

will give most of the time a sign that is determined by the dominant value b. One way to fix this problem is to reduce the value of l, although this will violate the condition set by Askey [19] to make sure that matrix is positive definite. However, we note that this condition is a sufficient, not a necessary one. In Covtype and usps data sets, we rerun the experiments with $l = 3$, which gave a very good accuracy in Covtype data set, close to that of the RBF kernel with faster training, but failed to give a similar performance in usps data set.

It is worth mentioning that, the second step in Algorithm 1 (computing the sparse RBF kernel) took most of the training time. Other steps such as Cholesky factorization and solving the linear system are very fast, when the matrices are sparse. The testing time was not reported as we were only concerned with the training performance. It is not also necessary to store the kernel matrix between the training and testing sets, we can compute each entry of the matrix and use it directly when classifying a testing point.

In some data sets, the RBF kernel requires large values of σ to attain a high accuracy in which case

our approach with the sparse RBF kernel might not be a good choice. The large value of σ will produce a dense kernel matrix which in case of very large data sets can not be stored on the machines. In addition, training with the sparse RBF kernel can even be much slower. For instance, we show in Figure 6 different experimental results performed on a small data set, used in [6], named svmguide1 with 3,089 training instances. The accuracies of both our approach and RBF kernel are plotted for different values of σ and we also plot the sparsity of the kernel matrix. We observe that the accuracy increases with the increase of σ while the sparsity decays. If we decide that $\sigma = 16$ is a good choice which provides an accuracy of 94%, then we will have a sparsity of 0.88 which will not be very good for large data sets.

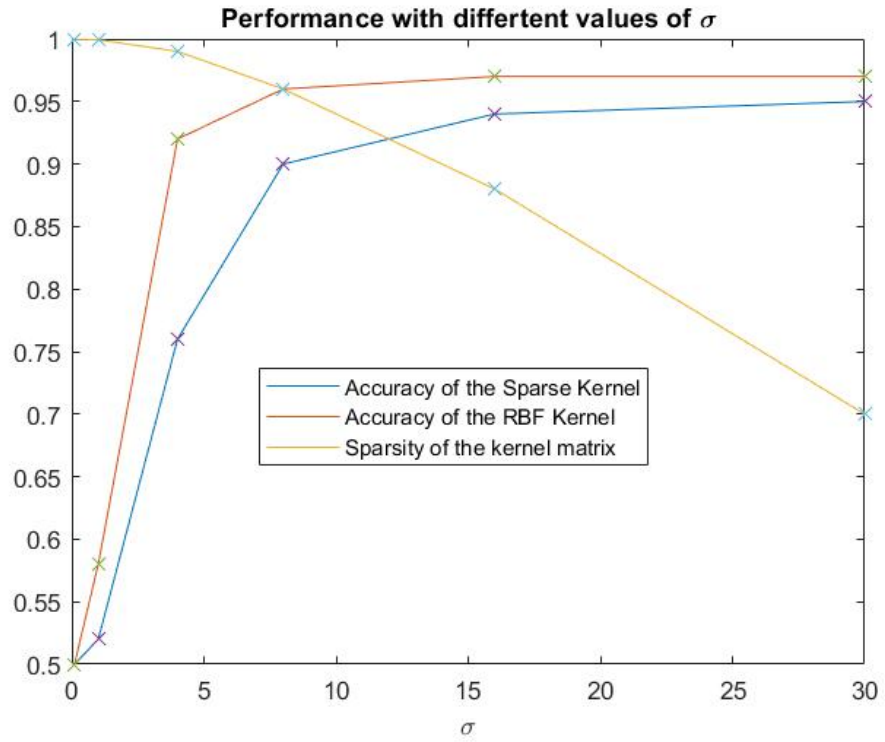

Figure 6: Performance of the sparse RBF kernel and RBF kernel on svmguide1 data set for different values of σ .

7. Conclusion

In this thesis, we studied a novel approach of training support vector machines that has shown a significant speed-up over traditional training methods. As discussed in section 2, support vector machines is a classification method used to assign labels to patterns. It builds a model by using a set of pre-labeled training set and solving an optimization problem to find a hyperplane that separates the data points into two classes. One advantage of using SVM is that it can use a *kernel trick* that maps the data points into high-dimensional features space where it is easier to separate the points.

Working with the non-linear kernels can be computationally expensive and usually solving the SVM optimization problem with them is painfully slow for large data sets. A kernel with sparse matrix was discussed in section 2.3 and we showed some of its properties in section 3. The kernel is constructed by multiplying a compactly- supported radial basis function by the RBF kernel. This multiplication leads to a kernel by using results from Theorem 2.1.3 and Lemma 2.1.4 provided that a condition is met on the parameter l in the kernel definition. Some literature papers have used the same sparse kernel, particularly [9] and [3], in support vector machine problems. One has replaced the RBF kernel by the sparse kernel directly in the optimization problem. Although, such substitution led to a speed-up in the training time, they used non-linear solvers which are still slow in case of large data set. Further results are shown in section 4.

In section 5 we discussed the details of our algorithm. The algorithm tries to combine the better performance of RBF kernel and the faster training of LIBLINEAR [20]. It computes the sparse matrix of RBF kernel discussed in section 2.3 and instead of solving the dual problem, we a perform a *Cholesky factorization* of the sparse matrix of the constructed RBF kernel. To make sure that the Cholesky factor is sparse as well, to meet space requirements, the *approximate minimum degree algorithm* [2] is used. The factor matrix, which now represents the data points mapped into the feature space, is then used to solve the primal optimization problem of SVM using LIBLINEAR solver. The novel approach showed very promising results over the standard training with RBF kernel. Experiments were conducted on several data sets and the time reduction obtained was sometimes over 90% while the accuracy was nearly the same as the one obtained using the RBF kernel. In cases where the linear kernel did not perform well, our approach was still able to perform similar to the RBF kernel but with much faster training procedure.

The main limitation of this approach is the need to store the sparse kernel matrix. This limitation can lead to out-of-memory errors if the parameter σ is chosen inappropriately and makes the approach not a good choice if σ has to be large to obtain a high accuracy. Another issue happens when the number of features is large, this will make the parameter l large as well because of Askey's condition. The large value of l may lead to poor accuracy as discussed in section 6. Reducing the value of l practically enhanced the accuracy but theoretically we are not sure if the function defined in equation (38) will be a kernel in general.

One possible continuation of this work is to solve its limiting situations. One could try to find an alternative relation between the sparsity parameter α and the parameter σ that does not lead to memory problems if σ is large. One could also try to fix situations where the number of features is large either by providing theoretical bounds on the parameter l that guarantee that the function k_c is a kernel or by finding a practical framework that performs well when the number of features is high and still meets the current theories. Another future work is to apply similar ideas on other applications that use the kernel trick such as SVM in regression, kernel PCA and spectral clustering.

References

[1] A. Abdellatif (2019). Master thesis code. GitHub repository, http://www.github.com/AlhasanAbdellatif123

[2] A. George and W. H. Liu (1989).The evolution of the Minimum Degree Ordering Algorithm. *SIAM Review.* 31 (1): 1–19.

[3] B. Hamers, J. A. K. Suykens and B. D. Moor (2002). Compactly Supported RBF Kernels for Sparsifying the Gram Matrix in LS-SVM Regression Models. *International Conference on Artificial Neural Networks* 2415, 720-726.

[4] C.-C. Chang and C.-J. Lin (2011). LIBSVM A library for support vector machines, *ACM Transactions on Intelligent Systems and Technology* 2, 27:1-27:27. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm

[5] C. Cortes and V. Vapnik (1995). Support-vector networks. *Machine learning* 20(3): 273-297

[6] C.-W. Hsu, C.-C. Chang, and C.-J. Lin (2003). A Practical Guide to Support Vector Classification. Department of Computer Science, National Taiwan University.

[7] F. Lauer(2014). Lecture notes. Retrieved from http://mlweb.loria.fr/book/en/VCdimhyperplane.html

[8] H. Cao, T. Naito and Y. Ninomiya (2008). Approximate RBF Kernel SVM and Its Applications in Pedestrian Classification. *The 1st International Workshop on Machine Learning for Vision-based Motion Analysis* .

[9] H. H. Zhang and M. Genton (2004). Compactly Supported Radial Basis Function Kernels.

[10] J. Chen, C. Zhan X. Xue and C. -L. Liud (2013). Fast instance selection for speeding up support vector machines. *Knowledge-Based Systems* 45, 1-7.

[11] J. C. Nash. (1990)"The Choleski Decomposition." Ch. 7 in *Compact Numerical Methods for Computers: Linear Algebra and Function Minimisation, 2nd ed.* Bristol, England: Adam Hilger, pp. 84-93.

[12] Larhmam - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=73710028

[13] M. Blachnik (2014). Reducing Time Complexity of SVM Model by LVQ Data Compression. *Artificial Intelligence and Soft Computing* 687-695.

[14] M. G. Genton (2002). Classes of kernels for machine learning: a statistics perspective. *The Journal of Machine Learning Research* 2, 299-312.

[15] M. Lichman (2013) UCI machine learning repository. URL http://archive.ics.uci.edu/ml

[16] MithrandirMage - Own work based on: VC4.png by BAxelrod., CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=19998946

[17] Platt, John (1998). Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines, *Advances in Kernel Methods-Support Vector Learning* 208.

[18] P. William, S. Teukolsky, W. Vetterling; B Flannery (1992). *Numerical Recipes in C: The Art of Scientific Computing* (second ed.). Cambridge University England EPress. p. 994. ISBN 0-521-43108-5.

[19] R. ASKEY (1973). Radial characteristic functions. Mathematical Research Center, University of Wisconsin, Madison. Technical summary report *n* 1262.

[20] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. (2008). LIBLINEAR: A Library for Large Linear Classification, *Journal of Machine Learning Research* 9, 1871-1874. Software available at `http://www.csie.ntu.edu.tw/~cjlin/liblinear`

[21] S. Wang, Z. Li, C. Liu, et al. (2014). Training data reduction to speed up SVM training. *Applied Intelligence* 41, 405–420.

[22] T. A. Davis and W. W. Hager (2009). Dynamic supernodes in sparse Cholesky update/downdate and triangular solves, *ACM Trans. Math. Software*, Vol 35, No. 4.

[23] V. Vapnik and A. Lerner (1963) . Pattern recognition using generalized portrait method. *Automation and Remote Control*, 24, 774–780.

[24] F. Pedregosa, G. Varoquaux, et al. (2011) Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12, 2825–2830.

A. Proof of Theorem 5.3

The proof is adapted from [7].

Theorem 5.3 *The VC-dimension of the set of affine classifiers* $\{f : f(x) = sgn(w^T x + b), w \in \mathbb{R}^d, b \in \mathbb{R}\}$ *is* $d + 1$.

In order to prove Theorem 5.3, we first prove a result on linear classifiers.

Theorem A.1: *The VC dimension of the set of linear classifiers* $\{f : f(x) = sgn(w^T x), w \in \mathbb{R}^d\}$ *is* d.

Proof. Consider a set of points $S = \{x_1, \dots, x_d\}$, that are canonical basis in $\mathbb{R}^d$ and let $w = \sum_i^d y_i x_i$, then we have

$$f(x) = sgn(\sum_i^d y_i x_i^T x).$$

Since we have that $x_i^T x_j = 0$ if $i \neq j$ and 1 otherwise, then $f(x_j) = sgn(y_j) = y_j$. Hence, this set of points can be shattered by the linear classifier and VC-dim$(f) \geq d$.
Assume that there are d+1 points $S = \{x_1, \dots, x_{d+1}\}$ that can be shattered by f. Then for any of the 2^{d+1} combinations of possible labels, there is a parameter w_k such that $f(x) = sgn(w_k^T x)$ produces the correct labeling.
Consider the matrix $H = XW$, where $X = [x_1, \dots, x_{d+1}]$ and $W = [w_1, \dots, w_{2^{d+1}}]$. let a be a non-zero vector, then there exists k such that $sgn(Xw_k) = sgn(a)$ and as a result $a^T X w_k$ is a sum of positive numbers and since there is no such vector a where $a^T H = 0^T$, the rows of H are linearly independent and the $rank(H) = d + 1$. But since $H = XW$, we have

$$rank(H) \leq \min\{rank(X), rank(W)\} \leq d$$

which is a contradiction and hence VC-dim$(f) \leq d$. Using the lower bound above we conclude that VC-dim$(f) = d$ □

Proving Theorem 5.3 where now $f(x) = sgn(w^T x + b)$.

Proof. Consider a set of points $S = \{x_1, \dots, x_d\}$ that are canonical basis in $\mathbb{R}^d$ and extend it with vector $x_{d+1} = 0$. Set $w = \sum_i^d (y_i - y_{d+1}) x_i$ and $b = y_{d+1}$. Then

$$f(x_j) = sgn(\sum_i^d (y_i - y_{d+1}) x_i^T x_j + y_{d+1}) = \begin{cases} y_j & j \neq d \\ y_{d+1} & j = d+1 \end{cases}$$

then there is a set of $d + 1$ points shattered by f and hence VC-dim$(f) \geq d + 1$.

Using the fact that an affine classifier in $\mathbb{R}^d$ is a linear classifier operating on a subspace of $\mathbb{R}^{d+1}$ and since the VC-dimension of a linear classifier in $\mathbb{R}^{d+1}$ is $d+1$, then there is no set of $d+2$ points shattered by an affine classifier in $\mathbb{R}^d$ and hence VC-dim$(f) \leq d + 1$, using the upper bound above we conclude that VC-dim$(f) = d + 1$. □

B. Implementation of the Algorithm in Python

The code below is just to show how the algorithm is implemented and full access to the code can be found on the online repository [1]. The inputs to the algorithm are
- The files names of the training and testing sets, lines 17 and 18.
- Parameters (σ, l, C), lines 27 to 33.

Remarks:

- The code is written as cells on a Jupyter notebook, so it should be run cell by cell as below, cells are separated by lines.
- The code first runs the approach with the sparse kernel and then with LIBSVM and LIBLINEAR.
- The main routine in the algorithm is *sp_rbf*(), defined in line 49. Its input arguments are the training set and the parameters γ and μ which are computed from σ in lines 35 and 36 and the parameter l. It returns the nonzero entries of the kernel matrix along with their associated rows and columns.

```
1  # ------- Importing libraries ---------
2  import numpy as np
3  import math
4  import time
5  from sklearn.svm import SVC, LinearSVC
6  from sklearn.model_selection import train_test_split
7  from sklearn.preprocessing import StandardScaler
8  from sklearn.metrics import accuracy_score, f1_score, confusion_matrix
9  from scipy.sparse import csc_matrix, lil_matrix, csr_matrix, bsr_matrix, diags, coo_matrix,
       dok_matrix
10 from scipy.sparse.linalg import spsolve_triangular, spsolve
11 from sksparse.cholmod import cholesky
12 from sklearn.datasets import load_svmlight_file
13 from sklearn.datasets.samples_generator import make_swiss_roll
14 %load_ext Cython
15
16 #--------- reading data set ---------
17 x_tr, y_tr = load_svmlight_file("a6a.txt")
18 x_tes, y_tes = load_svmlight_file("a6at.txt")
19 x_tes = x_tes.toarray()
20 x_tr = x_tr.toarray()
21
22 #---------specifying parameters ---------
23 m = x_tr.shape[0]
24 n = x_tr.shape[1]
25 m_test = len(y_tes)
26
27 sigma = 0.3
28 C=1
29 # compute l
30 r = math.floor(n/2)+1
31 if(r%2==0):
32     r = r+1
33 l = r
34
```

```
gamma = 1/(2*(sigma**2))
mu= 1/(3*sigma)

# classes
classes = np.unique(y_tr)
n_classes = len(classes)
print( m ,m_test,n)

#------- cython wrapper to compute the kernel for the training test -------

%%cython
from libc.math cimport exp
cdef extern from "math.h":
    double sqrt(double m)
def sp_rbf(double[:,:] X, double mu,double gamma,double l ):
    cdef list data = []
    cdef list row =[]
    cdef list col =[]
    cdef int i,j,d
    cdef double r = 0
    cdef double yj
    cdef int n1 = X.shape[1]
    cdef int m1 = X.shape[0]
    for i in range(m1):
        for j in range(i,m1):
            r =0
            for d in range(n1):
                    r += (X[i,d] - X[j, d]) ** 2
            yj = 1-mu*sqrt(r)
            if yj>0:
                yj = (yj**l)*exp(-gamma*r) #=data # col = j # row i
                data.append(yj)
                col.append(j)
                row.append(i)
    return data,row,col

#------- TRAINING USING THE SPARSE RBF KERNEL ---------

# Computing the sparse RBF kernel
s = time.time()
data,row,col = sp_rbf(x_tr,mu,gamma,l)
K2 = csc_matrix((data, (row, col)), shape=(m, m))
del data,row,col
K2 = K2+K2.transpose()-diags([1],shape = (m,m))
e = time.time()
print("Time of computing the sparse kernel" ,e-s)
print("sparsity of the kernel" ,1-K2.count_nonzero()/m**2)
total_time = e-s

# Cholesky factorization
f = cholesky(K2, beta=0.0001, mode='auto', ordering_method= 'amd', use_long=None)
Ls1 = f.L()
P = f.P()
print("Sparsity of Matrix L", 1-Ls1.count_nonzero()/m**2)

# check that the labels are (1,-1) for binary classes data set
class2 = False
```

```
92  if(n_classes==2):
93      class2 = True
94      if(classes[0] == 1):
95          pos = y_tr==1
96          y_tr[pos] = -1
97          neg = y_tr==2
98          y_tr[neg] = 1
99          pos = y_tes==1
100         y_tes[pos] = -1
101         neg = y_tes==2
102         y_tes[neg] = 1
103
104 # LibLinear Classifier
105 clf = LinearSVC( C = 1,dual = False)
106 clf.fit(Ls1,y_tr[P])
107 w = clf.coef_.T
108 b = clf.intercept_
109
110 # Solving for lamdas
111 if(class2 == True):
112     lamda = spsolve_triangular(Ls1.transpose().dot(diags(y_tr[P])),
113     w,lower = False)
114 else:
115     lamda = np.zeros((m,n_classes))
116     for i in range(n_classes):
117         pos = y_tr==classes[i]
118         y_tr_i = -np.ones(m)
119         y_tr_i[pos] = 1
120         lamda[:,i] = spsolve_triangular(Ls1.transpose().dot(diags(y_tr_i[P])),
121         w[:,i],lower = False)
122 e = time.time()
123 total_time = e-s
124 print("Total Training time ",total_time )
125
126 # ------- cython wrapper to compute the kernel for the test set ------
127
128 %%cython
129 from libc.math cimport exp
130 cdef extern from "math.h":
131     double sqrt(double m)
132 def sp_rbf_test(double[:,:] X_train,double[:,:] X_test,
133                 double mu,double gamma,double l ):
134     cdef list data = []
135     cdef list row =[]
136     cdef list col =[]
137     cdef int i,j,d
138     cdef double r = 0
139     cdef double yj
140     cdef int n1 = X_train.shape[1]
141     cdef int m_tr = X_train.shape[0]
142     cdef int m_tes = X_test.shape[0]
143     for i in range(m_tr):
144         for j in range(m_tes):
145             r =0
146             for d in range(n1):
147                     r += (X_train[i,d] - X_test[j, d]) ** 2
148             yj = 1-mu*sqrt(r)
```

```
149                 if yj>0:
150                     yj = (yj**l)*exp(-gamma*r) # =data # col = j # row i
151                     data.append(yj)
152                     col.append(j)
153                     row.append(i)
154         return data,row,col
155 
156 # ------- Testing using the Sparse kernel ----------
157 
158 data,row,col = sp_rbf_test(x_tr,x_tes,mu,gamma,l)
159 K2_t = csc_matrix((data, (row, col)), shape=(m, m_test))
160 del data,row,col
161 
162 if(class2 == True):
163     y_pred = np.sign(K2_t.T.dot(diags(y_tr).dot(lamda[P]))
164     + b*np.ones((m_test,1)))
165 else:
166     dec_fun = np.zeros((m_test,n_classes))
167     lamda = lamda[P]
168     for i in range(n_classes):
169         pos = y_tr==classes[i]
170         y_tr_i = -np.ones(m)
171         y_tr_i[pos] = 1
172         dec_fun[:,i] = K2_t.T.dot(diags(y_tr_i)).dot(lamda[:,i])+
173         b[i]*np.ones((m_test))
174     l = np.argmax(dec_fun,axis = 1)
175     y_pred = classes[l]
176 acc = accuracy_score(y_pred,y_tes)
177 print("Accuracy of Sparse kernel",acc)
178 
179 # ----- Training and Testing using the RBF kernel (LibSVM)----------
180 
181 start = time.time()
182 clf_2 = SVC(C = 1,kernel = 'rbf',gamma = gamma)
183 clf_2.fit(x_tr,y_tr)
184 end = time.time()
185 print("Time of RBF Kernel",end - start)
186 rbf_acc = clf_2.score(x_tes,y_tes)
187 print("Accuracy of RBF kernel",rbf_acc)
188 print("Total Training time ",total_time )
189 print("Accuracy of Sparse kernel",acc)
190 
191 # ------ Training and Testing using the linear kernel (LIBLINEAR) ------
192 
193 start = time.time()
194 clf_3 = LinearSVC(C = 1)
195 clf_3.fit(x_tr,y_tr)
196 end = time.time()
197 print("Time of Linear Kernel",end - start)
198 lin_acc = clf_3.score(x_tes,y_tes)
199 print("Accuracy of Linear kernel",lin_acc)
```