Technical University Of Crete

School of Electrical and Computer Engineering

A Functional Geometric Approach to Distributed Support Vector Machine (SVM) Classification

Author:
Sofia Kampioti

Supervisor:
As. Prof. Vasilios Samoladas

Abstract

We live in the information age, and with every passing year, our environment becomes more and more heavily defined by data, leading to a major need for better decision-making models. The breakthroughs in data analytics have already seen through machine learning. Support vector machines (SVM) are a popular, adaptive, multipurpose machine learning algorithm with the ability to capture complex relationships between data points without having to perform difficult transformations. We study the problem of prohibitive communication costs that a centralized architecture implies if most of the data is generated or received on different remote machines. The past few years notable efforts have been made to achieve parallelism on the training procedure of machine learning models. We propose the use of Functional Geometric Monitoring (FGM) communication protocol which is used to monitor high-volume, rapid distributed streams to decrease the communication cost on a distributed SVM architecture. Our main goal is both to achieve centralized-like prediction loss and to minimize communication costs. In our proposal, the sklearn library, for centralized machine learning, is used in a distributed manner resulting in a notable speedup for the training procedure.

Acknowledgements

"Foremost, I would like to express my sincere gratitude to my supervisor Prof. Vasilis Samoladas for the continuous support of my Diploma thesis study and research, for his patience, motivation, enthusiasm, and immense knowledge.

Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Minos Garofalakis and Antonios Deligiannakis for their time.

I would also like to thanks my fellow students, Ilias Balampanis, Edward Epure, Eftichia Seisaki, for there wonderful collaboration and support during my thesis process

Finally I would like to thanks my family and friends since this project would have been impossible without them "

Contents

1 Introduction

In an age of ever-increasing information collection and the need to evaluate it, building systems that utilize the yet untapped and available resources is driving the development of more sophisticated distributed computing systems. Driven by this urgent need, and the fact that the demand for processing training data has outpaced the increase in computation power of computing machinery, distributing the machine learning workload across multiple machines, gain a lot of scientific interest. Unfortunately, a major role in distributed systems performance plays the communication cost, since communication is often the bottleneck of applications and so it directly relates to energy consumption, network bandwidth usage, and overall running time.

Support Vector Machines (SVM) have a strong theoretical foundation and a wide variety of applications. On the other hand, the underlying optimization problems can be highly demanding in terms of run-time and memory consumption. Distributed scenarios emerge when data are captured in many places and their transport and storage to a unique location is undesirable. A rather straightforward procedure for achieving distribution in SVMs is a sort of distributed chunking technique where the result of the training procedure are exchanged with the other nodes.

However, the amount of information that needs to be transmitted might rapidly make this approach unfeasible in real-world conditions. For this reason, this work focuses on reducing the communication cost and run time duration by implementing Functional Geometric Monitoring (FGM), a protocol that provides substantial benefits in terms of performance and scalability in monitoring problems [12].

This work aims to efficiently test distributed SVM in an online manner, with reduced communication, in a real-time system. We managed to use sklearn library, a library for centralized machine learning integration, for distributed online training and achieve rather encouraging results in terms of speedup and centralized like accuracy.

1.1 Related Work

In the past few years, distributed training SVM models have gained a lot of interest. Driven by this concept, many different approaches came up, focusing each time in a different way to accomplice distribution. Particularly, [10] have proposed distributed technique for training SVMs in sensor net-

works,[19] propose a communication avoiding SVM (CA-SVM) for shared memory architecture by combining several approaches like the cascade SVM, DC-SVM, where [11] casts SVM problem as a set of coupled decentralized convex optimization subproblems with consensus constraints imposed on the desired classifier parameters. This work contribution, though, focuses on reducing the communication cost of a distributed online SVM training process.

1.2 Contribution

This study aims to utilize the advancements in the field of distributed stream monitoring for the problem of distributed machine learning classification and more precisely, Support Vector Machines. The main goal is to practically implement a functional geometric approach to this machine learning algorithm in a real distributed environment using python Dask. This work efficiently combines sklearn and Dask to integrate the SVM algorithm in a distributed online manner. Note that sklearn is a library for centralized machine learning, but this work proves that sklearn can be also used in a distributed system and perform equivalently.

1.3 Outline

The rest of this section is focus on describing related work and the contribution of this work to the distributed SVM concept.

Section 2 describe the theoretical bases, on which the implementation is based on. The main theoretical concepts are machine learning basics focusing on classification, Support Vector Machines (SVM) and the Functional Geometric Monitoring protocol (FGM).

Section 3 focuses on the implementation and mainly how distributed SVM is combined with the FGM protocol to achieve speedup and reduction of the communication cost.

Later on, section 4 explains the result of this combination and how this improves communication and archives speedup towards centralized structure.

2 Theoretical Background

This section is dedicated to introducing the main theoretical concepts and research in which this work is based on. Support Vector Machines (SVM) [3],

and Functional Geometric Monitoring [12] are, as previously mentioned, the main subjects of this project. So, this section describes the theory of both algorithms in a way that will help to understand the connection between the research and the implementation of this work. Note that the theoretical research, is not constrained into studying only the related work described in 1.1 but is also based to books [1],[9],[14] and Wikipedia [7] research.

2.1 Machine Learning Basics

Machine-learning algorithms are responsible for the vast majority of artificial intelligence advancements and applications. It is a branch of artificial intelligence based on the idea that systems can learn from data, identify patterns, and make decisions with minimal human intervention. The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future, based on the examples that we provide. The discipline of machine learning employs various approaches to help computers learn to accomplish tasks where no fully satisfactory algorithm is available. Machine learning approaches are divided into three broad categories, depending on the nature of the "signal" or "feedback" available to the learning system.

2.1.1 Types of learning algorithms

The types of machine learning algorithms differ in their approach, the type of data they input and output, and the type of task or problem that they are intended to solve. The following figure indicates the hierarchy of learning types.

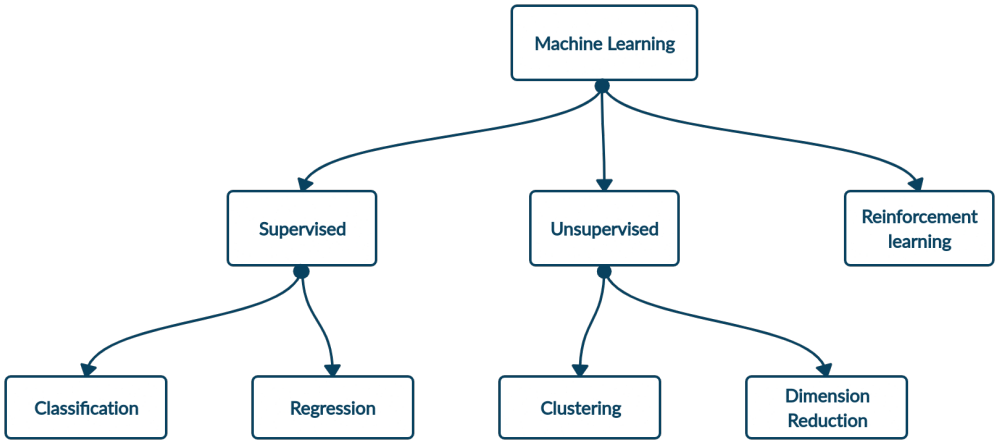

Figure 1: Machine Learning hierarchy

Supervised Learning is a task of learning a function that maps an input to an output based on example input-output pairs. It infers a function from labeled training data consisting of a set of training examples. In supervised learning, each example is a pair consisting of an input object and a desired output value.This is usually written as a set of data (x_i , t_i), where the inputs are x_i , the targets are t_i , and i runs from 1 to the number of input dimensions (see notation). A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples. An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances.

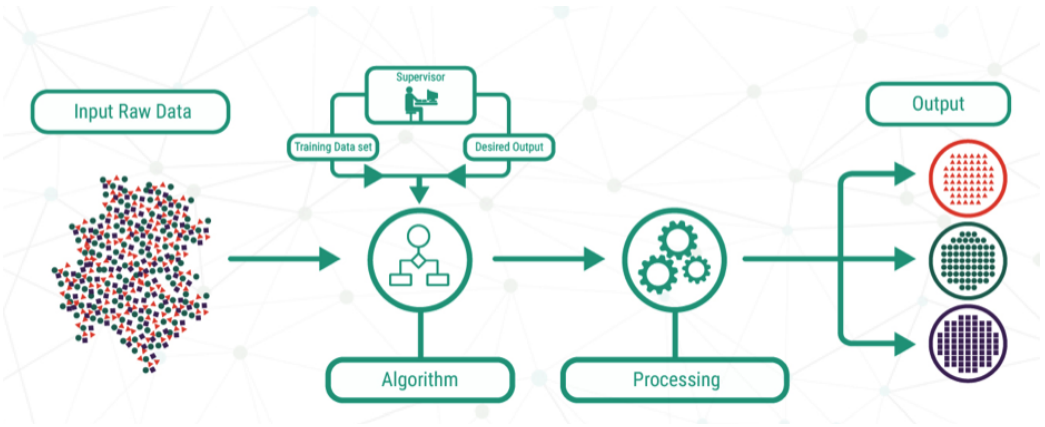

Figure 2: Supervised Learning flow

Types of supervised learning algorithms include classification and regression. Classification algorithms are used when the outputs are restricted to a limited set of values, and regression algorithms are used when the outputs may have any numerical value within a range. Classification and regression are described in more detail later in this section.

Unsupervised Learning is a conceptually different problem to supervised learning. Unsupervised learning is a type of machine learning that looks for previously undetected patterns in a data set with no pre-existing labels and with a minimum of human supervision. In contrast to supervised learning that usually makes use of human-labeled data, unsupervised learning, tries to identify similarities between the inputs so that inputs that have something in common are categorized together. The statistical approach to unsupervised learning is known as density estimation.

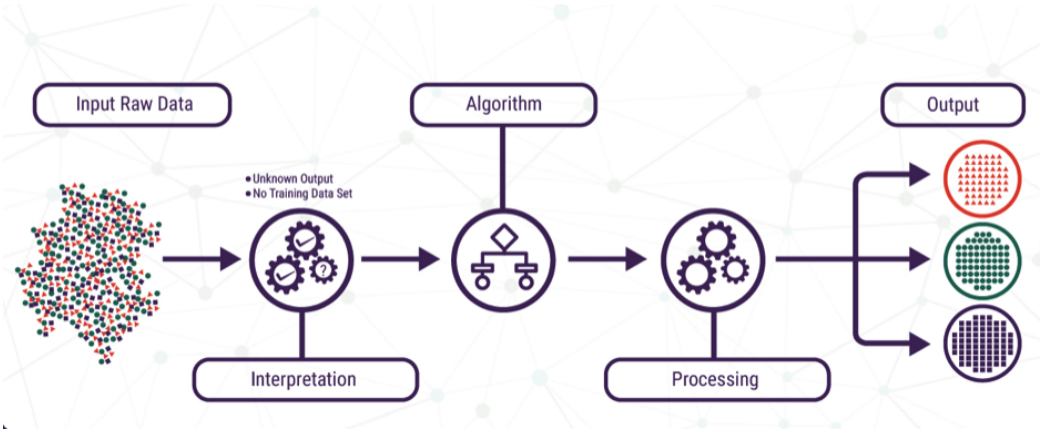

Figure 3: Unsupervised Learning flow

Within unsupervised machine learning, there are several different approaches such as clustering and dimension reduction. Clustering is the assignment of objects to homogeneous groups while making sure that objects in different groups are not similar , when dimension reduction reduces the number of features under consideration, where each feature is a dimension that partly represents the objects.

Reinforcement learning fills the gap between supervised learning and unsupervised learning. Reinforcement learning is usually described in terms of the interaction between some agent and their environment. The agent is the learner, and the environment is where it is learning, and what it is learning about. Reinforcement learning maps states or situations to actions to maximize a reward function.

Figure 4: Reinforcement Learning flow

In particular, the algorithm knows the state (the current input), and the available actions, and it aims to choose the action that maximizes the reward. The reward is given as feedback to the algorithm to guide future actions considering that the methods that seem to work should be tried over and over again, until they are perfected or better solutions are found, and those that do not work must be discarded.

2.1.2 Classification

Fundamentally, *classification* is about predicting a label, and *regression* is about predicting a quantity. Regression algorithms try to find the best fit line, which can predict the output more accurately in cases like weather prediction, house price prediction, and similar examples with continuous nature or real value outputs. On the other hand, classification algorithms try to find the decision boundary, which can divide the dataset into different classes such as identification of spam emails, speech recognition, identification of cancer cells, and any machine learning problem with discrete value as an output.

As initially mentioned, this work is dedicated to managing discrete data and especially binary nature problems such as identification of spam emails, hence regression algorithms will not form a part of it. Consequently, there follows a more thorough description of classification algorithms in order to understand the main concept and some basic notation, required for the following sections.

Classification problem consists of taking input vectors and deciding which of K classes they belong to, based on training from exemplars of each class. The most important point about the classification problem is that it is discrete — each example belongs to precisely one class, and the set of classes covers the whole possible output space. If only two classes are involved the classification is called *binary*, otherwise, it is *multiclass classification*. Since classification problems refer to different data sets in each individual problem, there are a variety of classification algorithms such as support vector machines, linear classifiers, quadratic classifiers, and more. Nevertheless, each and everyone shares the same fundamental concept that classification requires and which is hereinafter explained.

There is a basic *model* that distinguishes classification algorithms from other supervised learning algorithms. Firstly, the goal in classification is to take an input vector x and to assign it to one of K discrete classes $C_k$where $k = 1, ..., K$. The classes are taken to be disjoint, at least in the most common scenario, so that each input is assigned to one and only one class. The input space is thereby divided into decision regions whose boundaries are called *decision boundaries* or *decision surfaces*. In this work, we consider linear models for classification, by which we mean that the decision surfaces are linear functions of the input vector x and hence are defined by $(D-1)$-dimensional hyperplanes within the D-dimensional input space. Note that from now on, any referred data set is linearly separable, meaning classes can be separated exactly by linear decision surfaces.

In the case of two-class problems, is the binary representation in which there is a single target variable $t \in 0, 1$ such that $t = 1$ represents class $C_1$and $t = 0$ represents class C_2. Value of t can be interpreted as the probability that the class is C_1, with the values of probability taking only the extreme values of 0 and 1. In the simplest case, where the model is linear in the input variables, an appropriate function $y(x)$ is constructed whose values for new inputs x constitute the predictions for the corresponding values of t and therefore takes the form:

$$y(x) = w^T x + w_0, \text{where } y \in \mathbb{R}$$

Here w is called a weight vector, and w_0 (or sometimes b) is a bias. Practically, an input vector x is assigned to class C_1 if $y(x) \geq 0$ and to class C_2

otherwise. The corresponding decision boundary is therefore defined by the relation $y(x) = 0$. So, if x is a point on the decision surface, then y(x) = 0, and so the normal distance from the origin to the decision surface is given by:

$$\frac{w^T x}{||w||} = -\frac{w_0}{||w||}$$

Furthermore, note that the value of y(x) gives a *signed measure* of the perpendicular distance r of the point x from the decision surface, as shown at the following figure.

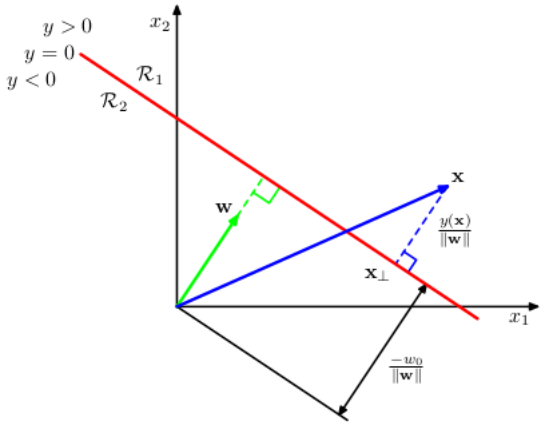

Figure 5: Illustration of the geometry of a linear discriminant function in two dimensions

Finally, such models have useful analytical and computational properties but that their practical applicability is limited by the curse of dimensionality. So, in order to apply such models to large-scale problems, it is necessary to adapt the basis functions to the data. Support vector machines (SVMs), discussed in the next section, address this by first defining basis functions that are centred on the training data points and then selecting a subset of these during training.

2.2 Support Vector Machines (SVM)

Support vector machine (SVM) is the most popular classifier based on a linear discriminant function and hence it is a linear classifier. The main

property that distinguishes SVM from other classification algorithms is that the determination of the model parameters corresponds to a convex optimization problem, and so any local solution is also a global optimum. SVMs are the so-called "non-parametric" models, meaning their "learning" (selection, identification, estimation, training) is a crucial issue since the parameters are not predefined and their number depends on the training data used. Generally, in SVM, a data point is viewed as a list of p numbers and the goal is to separate such points with a $(p-1)$-dimensional hyperplane. Since various hyperplanes can linear separate the given points, the final hyperplane is determined as the one that represents the largest separation between the classes. The rest of the section will provide some detail at the support vector machine, including the concept of maximum margin and Stochastic Gradient Descent.

2.2.1 Maximum Margin

As previously mentioned, binary classifiers construct a linear model of the form:

$$y(x) = w^T x + b \tag{2.1}$$

where b is the bias (previously mentioned and as w_0). The training data set comprises N input vectors $x_1, ..., x_N$, with corresponding target values $t_1, ..., t_N$ where $t_n \in -1, 1$, and new data points x are classified according to the sign of $y(x)$.

Given the assumption that the training data set is *linearly separable* in feature space, there exist at least one pair of parameters w and b such that a function of the form (2.1) satisfies the inequality $y(x_n) > 0$ for points having $t_n = +1$ and $y(x_n) < 0$ for points having $t_n = -1$, so that $t_n y(x_n) > 0$ for all training data points. The equality y(x) = 0 defines the decision boundaries that help classify the data points also known as *hyperplanes*.

Practically, there may exist many such solutions that separate the classes exactly, so there is a need to find the one that will give the smallest generalization error. The support vector machine approaches this problem through the concept of the *margin*, which is defined to be the smallest distance between the decision boundary and any of the samples as illustrated in *Figure:*

6. Then the decision boundary is chosen to be the one for which the margin is maximized.

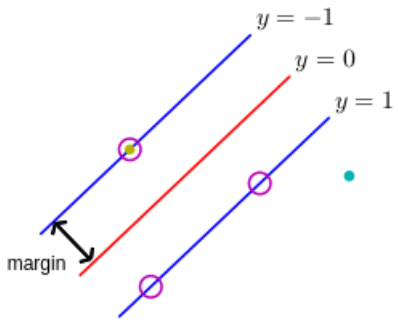

Figure 6: Illustration of margin and support vectors

Margin is defined as the perpendicular distance between the decision boundary and the closest of the data points, where the perpendicular distance of a point x from a hyperplane is given by $||y(x)||/||w||$. Furthermore, the main interest is focused on solutions for which all data points are correctly classified, so that we satisfy $t_n y(x_n) > 0$ for all n. In view of the above observation, the distance of the point x_n to the decision surface is given by:

$$\frac{t_n y(x_n)}{||w||} = \frac{t_n(w^T\phi(x_n) + b)}{||w||} \tag{2.2}$$

As mentioned before, the margin is given by the perpendicular distance to the closet point x_n from the data set and we need to maximize it. Thus, the maximum margin is given from the optimization of the parameters w and b, as follows:

$$\underset{w,b}{\text{argmax}}\{\frac{1}{||w||}\min_n[t_n(w^T\phi(x_n) + b)]\} \tag{2.3}$$

The direct solution to this optimization problem would be very complex since it involves a *quadratic function*. So, instead of solving the direct problem itself, a solution is to solve a converted, equivalent problem. After rescaling w and b , the distance from any point x_n to the decision surface is given by $t_n y(x_n)/||w||$, resulting to the *canonical representation* of the decision hyperplane above:

$$t_n(w^T\phi(x_n) + b) \geq 1, \quad n = 1, ..., N \tag{2.4}$$

The optimization problem then simply requires that we maximize $||w||^{-1}$, which is equivalent to minimizing $||w||^2$, and so we have to solve the optimization problem

$$\underset{w,b}{\operatorname{argmin}} \frac{1}{2}||w||^2 \tag{2.5}$$

subject to the constraints given by (2.4). One way to solve quadratic optimization problems is via introducing *Lagrange multipliers* $a_n \geq 0$, with one multiplier a_n for each of the constraints in (2.4). Hence, this constrained optimization problem converts into the Lagrangian function:

$$L(\boldsymbol{w}, b, \boldsymbol{a}) = \frac{1}{2}||w||^2 - \sum_{n=1}^{N} a_n\{t_n(w^T\phi(x_n) + b) - 1\} \tag{2.6}$$

where $a = (a_1, ..., a_N)^T$. Setting the derivatives of L(w, b, a) with respect to w and b equal to zero, we obtain the following two conditions:

$$w = \sum_{n=1}^{N} a_n t_n \phi(x_n) \tag{2.7}$$

$$0 = \sum_{n=1}^{N} a_n t_n \tag{2.8}$$

Eliminating w and b from $L(w, b, a)$ using these conditions then gives the *dual representation* of the maximum margin problem in which we maximize:

$$\tilde{L}(a) = \sum_{n=1}^{N}(a_n) - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(x_n, x_m) \tag{2.9}$$

with respect to the constraints:

$$a_n \geq 0 \quad n = 1, ..., N, \tag{2.10}$$

$$\sum_{n=1}^{N} a_n t_n = 0 \tag{2.11}$$

Finally, the sign of y(x) defined by (2.1), can be expressed in terms of the parameters $\{a_n\}$ as:

$$y(x) = \sum_{n=1}^{N} a_n t_n (x, x_n) + b \tag{2.12}$$

A constrained optimization of this form satisfies the Karush-Kuhn-Tucker (KKT) conditions, which in this case require that the following three properties hold:

$$a_n \geq 0 \tag{2.13}$$
$$t_n y(x_n) - 1 \geq 0 \tag{2.14}$$
$$a_n\{t_n y(x_n) - 1\} = 0 \tag{2.15}$$

Hence, any data point for which $a_n = 0$ will play no role in making predictions since it does not appear in the sum in (2.12). On the other hand, the point that doesn't satisfy the equality, they satisfy $t_n y(x_n) = 1$ so, they correspond to the points that lie on the maximum margin hyperplanes illustrated in *Figure:6*. These points are called *support vectors*, and they play a major role in making the prediction since once the model is trained, a significant proportion of the data points can be discarded and only the support vectors retained.

So far, the above observations were based on the strong assumption that the training data points are linearly separable. However, in practice, the class distributions may overlap. In this case, the exact separation of the training data is not possible or leads to poor generalization. In order to deal with this kind of training data, the support vector machine should be modified in a way that it allows misclassification. Misclassification means that data points are allowed to be on the 'wrong side' of the margin boundary, but with a penalty that increases with the distance from that boundary. For the subsequent optimization problem, it is convenient to make this penalty a linear function of this distance.

The solution is given by *slack variables*, $\xi_n = 0$ where $n = 1, ..., N$, with one slack variable for each training data point. Hence, the data points that are on or inside the correct margin boundary are defined by $\xi_n = 0$ and $\xi_n = |t_n - y(x_n)|$ for the rest of the points. So, if $\xi_n > 1$ the point is determined as misclassified , else if $\xi_n = 1$ will be on the decision boundary. After taking into consideration the above observation, the classification constraints (2.4) is replaced with:

$$t_n y(x_n) \geq 1 - \xi_n, \quad n = 1, ..., N \tag{2.16}$$

in which the slack variables are constrained to satisfy $\xi_n \geq 0$. Hence, it is now clear that $\xi_n = 0$ means the points are correctly classified , $0 < \xi_n \leq 1$

that the point lies inside the margin but on the correct sight of the decision boundary and finally those with $\xi_n > 1$ lie on the wrong side of the decision boundary, as illustrated in *Figure:7*.

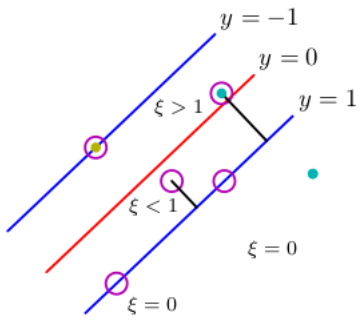

Figure 7: Illustration of the slack variables $\xi_n \geq 0$. Data points with circles around them are support vector

This is sometimes described as relaxing the hard margin constraint to give a soft margin. Now, the minimization problem (2.5) should take into consideration the penalty for the points that lie on the wrong side of the margin boundary. Consequently:

$$C \sum_{n=1}^{N} \xi_n + \frac{1}{2}||w||^2 \tag{2.17}$$

where the parameter $C > 0$ controls the *trade-off* between the slack variable penalty and the margin. The bigger C gets, the harder the margin is, so $C \to \infty$ indicates the earlier support vector machine for separable data. Finally, to minimization of (2.17) subjects to the constraints (2.16) and also $\xi_n \geq 0$. Correspondingly to (2.6) the Lagrangian function takes the following form:

$$L(\boldsymbol{w}, b, \boldsymbol{a}) = \frac{1}{2}||\boldsymbol{w}||^2 + C \sum_{n=1}^{N} \xi_n - \sum_{n=1}^{N} a_n\{t_n y(x_n) - 1 + \xi_n\} - \sum_{n=1}^{N} \mu_n \xi_n \tag{2.18}$$

where $a_n \geq 0$ and $\mu_n \geq 0$ are Lagrange multipliers. Following the same strategy as earlier, the dual Lagrangian takes the below form:

$$\tilde{L}(a) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m (x_n, x_m) \tag{2.19}$$

which is identical to the separable case, except that the constraints are somewhat different. These constraints arise from a combination of the constraints that refer to the Lagrange multipliers and parameter C. Therefore, the minimization of 2.19 is subject to:

$$0 \geq a_n \geq C \tag{2.20}$$

$$\sum_{n=1}^{N} a_n t_n = 0 \tag{2.21}$$

for $n = 1, ..., N$, where () are known as box constraints. This again represents a quadratic programming problem. As before, a subset of the data points may have $a_n = 0$, in which case they do not contribute to the predictive model (2.12) . The remaining data points constitute the support vectors. These have $a_n > 0$ and hence :

$$t_n y(x_n) = 1 - \xi_n \tag{2.22}$$

A rather common problem that support vectors faces is dealing with a non-linear data set where the data can't be separated by a straight line and, unlike the above case, even by relaxing the margins, the data set the generalization error will still be increased.

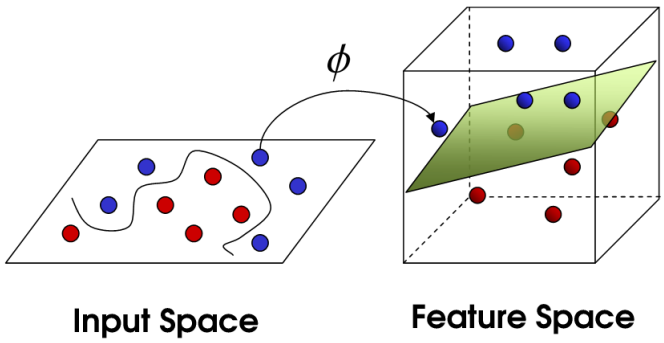

Figure 8: Example of non-linear data set

The basic idea in kernels is that when a data set is inseparable in the current dimensions, add another dimension by mapping the current feature space in one that could be separable. So, for all x_n and x_m in the input space X there exists certain functions $k(x_n, x_m)$, can be expressed as an inner product in another space V. The kernel can be written in the form of a "feature map" $X \to V$ which satisfies

$$k(x_n, x_m) = \langle \phi(x_n), \phi(x_m) \rangle_\nu \tag{2.23}$$

By following the above strategy with changed feature space the dual Lagrangian takes the below form:

$$\tilde{L}(a) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m k(x_n, x_m) \tag{2.24}$$

Concluding, there is a lot of theoretical background behind kernel function,which will not be discussed feather since it is not the subject of this work.

2.2.2 Stochastic Gradient Descent

Machine learning models typically have parameters, for SVM simple case is w and b, and a cost function to evaluate how good a particular set of parameters are. Furthermore, the SVM constraints are linear in the unknowns and any linear constraint defines a convex set. Now, a set of simultaneous linear constraints defines the intersection of convex sets, so SVM constraints can be defined as a convex set. The main property that a convex function has is that a locally optimal point is also globally optimal.

Gradient descent is an optimal algorithm used to minimize some function by iteratively moving in the direction of the steepest descent as defined by the negative of the gradient. Hence, gradient descent can be used to update the parameters of our model. Let's define W as the model parameters, then the steps to minimize a cost function $J(W)$ is:

1. **Initialize** the weights W randomly
2. **Calculate the gradients** G of cost function with respect to parameters. The value of the gradient G depends on the inputs, the current values of the model parameters, and the cost function.

3. **Update the weights** by an amount proportional to G, $W = W - \eta G$, where η is the learning rate which determines the size of the steps we take to reach a minimum.

4. Repeat until the cost $J(W)$ stops reducing.

Earlier on this section, it was made clear that in SVM the main goal is to minimizing $||w||^2$. An alternative way to represent the minimization problem is through a simple convex loss function, defined as hinge loss.

$$\ell_{hinge}(t, \hat{t}) = \max(0, 1 - \hat{t}t) \tag{2.25}$$

so the minimization problem will have the below form:

$$\min_{w,b} ||w||_2^2 + C \sum \ell_{hinge}(t_n, w \cdot x_n + b) \tag{2.26}$$

Therefore, this form of minimization problem can be solved as a constrained Optimization problem by using gradient descent and the minimization function get the following form:

$$J(W) = \frac{1}{2} w^T w + C \sum_n \max(0, 1 - t_n w^T x_n) \tag{2.27}$$

Furthermore, in real-time applications, the gradient descent can be applied in the scenario of having to take *online decisions*. Online SVM training means that the classifier changes over time, and the distribution is no longer fixed. Since we need to calculate the gradients for the whole dataset to perform one parameter update, gradient descent can be very slow.

On the other hand, *Stochastic Gradient Descent* (SGD) computes the gradient for each update using single training data points x_n (chosen at a random) or a mini-batch of the training set. The main idea is that the gradient calculated this way is a stochastic approximation to the gradient calculated using the entire training data. Each update is much faster than Gradient Descent and over many updates, the same general direction is given. So, even though a higher number of iterations are required to reach the global minimum, it is still computationally preferred over Gradient Descent. The main difference in the minimization function is that now it corresponds to a

single training data points or a certain mini-batch, meaning instead of computing $J(W)$ we now compute $J_i(W)$, for $i = 1, ..., K$, and K the number of the individual points or the number of mini-batches. Finally since the goal of this work was to train the classifier in an online fashion, Stochastic Gradient Descent was used for the training stage.

2.3 Functional Geometric Monitoring (FGM)

Functional Geometric Monitoring, is a method for distributed stream monitoring, which is applicable as Geometric Monitoring, and provides substantial benefits in terms of performance, scalability, and robustness. The strict separation of concerns between distributed systems issues and the monitoring problem, is critically important to anyone wishing to implement distributed monitoring on a general-purpose middle-ware platform.

Geometric Monitoring(GM)
With *Geometric Monitoring(GM)*, an arbitrary global monitoring task can be split into a set of constraints applied locally on each of the streams. The constraints are used to locally filter out data increments that do not affect the monitoring outcome, thus avoiding unnecessary communication. As a result, it enables monitoring of arbitrary threshold functions over distributed data streams in an efficient manner. Practically, as data arrives on the streams, each node verifies that the constraint on its stream has not been violated. The geometric analysis of the problem guarantees that as long as the constraints on all the streams are upheld, the result of the query remains unchanged, and thus no communication is required. If a constraint on one of the streams is violated, new data is gathered from the streams, the query is reevaluated, and new constraints are set on the streams.

Functional Geometric Monitoring(FGM)
Functional Geometric Monitoring, is based on the core ideas of Geometric Monitoring(GM) but instead of a binary constraint, each site is provided with a complex, non-linear function, which, applied to its local summary vector, projects it to a real number. The focus of this subsection is to present the basic principles and protocol of FGM.

Practically, lets assume that there are k distributed sites, and that at each site, a local stream is generated or collected. The sites collectively monitor

the sum of these one-dimensional projections and as long as the global sum is subzero, the monitoring bounds are guaranteed. Let $S_i(t)$, $i = 1...k$ denote the local state vectors. Every site communicates with a coordinator where users pose queries on the global stream. The coordinator maintains, for each site i, an estimated state vector E_i . When a flush occurs, the site transmits its drift vector $X_i(t) = S_i(t) - E_i$, and the coordinator updates E_i by adding X_i to it, while the site resets X_i to 0. Then, the coordinator updates the global estimate $E = \frac{1}{k}\sum_{i=1}^{k} E_i$. In geometric monitoring, the correctness criterion is described as a geometric constraint, of the form $S \in A$, where $A \subseteq R^D$ is the admissible region, that is, the set of global stream states where the constraint holds. The correctness criterion here differs and is based on the save function concept.

In FGM algorithm, the safe function is a conceptual concept.The system is in a safe state as long as $\frac{1}{k}\sum_{i=1}^{k} \boldsymbol{X_i} = \boldsymbol{S} \in A$. To guarantee safety FGM employs a real function $\phi : \mathbb{R}^D \to \mathbb{R}$ depending on A, E and k. Each site tracks its ϕ-value, $\phi(X_i)$, as X_i is updated. Hence, to guarantee that $\boldsymbol{S} \in A$, the sign of the sum $\frac{1}{k}\sum_{i=1}^{k} \boldsymbol{X_i}$, which for now one will be referred as ψ, must be less or equal to 0. So the safety is maintained as long as $\psi \leq 0$. In other words, we need to find an $A, \boldsymbol{E}, k$-safe function.

The FGM algorithm works in rounds to monitor the threshold condition

$$\sum_{i=1}^{k} \phi(X_i) \leq 0 \tag{2.28}$$

and guarantee the desired safety. Generally, the sites perform local updates and, when necessary, ship the ϕ of there local drift to the coordinator, which is responsible for monitoring the threshold condition. On the other hand, the coordinator is collecting information from the workers to compute ψ and if the safety is disturbed,$\psi > 0$, the coordinator requests the real drift vectors to recompute global estimate and restore systems safety. In this way the system stays updated with the minimum communication cost. Later on 3 we will describe the execution of the FGM algorithm, specifically for distributed, online Support Vector Machine training.

2.4 Tools

Given the nature of the problem, there is no doubt that the environment must include tools that allow distributed computing and also support complex workflows, such as machine learning algorithms. An additional factor

is the need to produce a project as simple and efficient possible. Therefore, Python became the more prevalent choice among others, for being an interpreted, high-level programming language with dynamic semantics. Also, Dask is lightweight python library that provides distribution and can be easily combined with the Scikit-Learn, which contains a lot of efficient tools for machine learning and statistical modeling.

2.4.1 Scikit-Learn

Scikit-Learn [13] is a *Python module* integrating classical machine learning algorithms in the tightly-knit world of scientific Python packages. It is designed to provide simple and efficient solutions to learning problems, which is one of the main purposes of this project. Also, provides a wide range of choices, for all the kind of machine learning problems, such as classification, regression, clustering, and more. This thesis is focused on classification problems and especially on Support Vector Machines. So, from Scikit-Learn, *SGDClassification* estimator fits the specifications of the problem since is a simple approach to discriminative learning of linear classifiers under convex loss functions, such as SVM. The previous section described in detail Stochastic Gradient Descent and hinge loss, so practically choosing hinge as loss function for this estimator, automatically results in a linear SVM classifier. Also, to test the implementation, there was a need for several datasets with different features and sizes. Scikit-Learn includes various random sample generators that can be used to build artificial datasets of controlled size and complexity such as *make classification*.

2.4.2 Dask

Dask [4] can be considered as *Hadoop/Spark* equivalent for python. It is a simple tool for data processing and is able to either be run on a local computer or be scaled up to run on a cluster. It is an easy-to-use tool since it provides advanced parallelism for analytics, enabling performance at scale with minimal rewriting. The past few years Dask has been extended with a distributed memory scheduler. This enables Dask's existing parallel algorithms to scale across 10s to 100s of nodes, and support distributed computing. In addition to the above, another essential feature, for this project, is that workers can communicate with each other to share data. This removes central bottlenecks for data transfer and offers the opportunity to apply the FGM protocol.

2.4.3 Scikit-Learn and Dask Connection

Scikit-learn uses joblib for single-machine parallelism. This supports training most estimators (anything that accepts an n_jobs parameter) using all the cores of your laptop or workstation. Alternatively, Scikit-Learn can use Dask for parallelism. This offers the ability to train those estimators using all the cores of your cluster without significantly changing code. Considering, this connection and the features each of them individually provides, there is no doubt that they get above the specifications described earlier on.

3 Implementation

This section describes the basic FGM protocol (without rebalancing) for distributively training SVM, in an online manner. First, there is a description of the final model that combines SVM-FGM protocol focused on the way that these two algorithms were combined to produce rather encouraging results and then a description of the main tools used for this implementation. The distributed SVM algorithm is based on the averaging model where the global estimate is computed as the average of the individual models computed in each node. Given this structure, FGM applies on when the nodes will send their local model to make the communication as meaningful as possible. Later follows a more thorough description which will help to understand the results in section 4.

3.1 Decentralized architecture

It must be clear by now, that the main purpose of this work is to empirically confirm the communication gains for Distributed SVM via FGM protocol. In the general case, Distributed SVM structure constructs from one coordinator and multiple, n, workers. Each worker performs online training, in order to update its local model and after each update ships the updated model to the coordinator. Then, the coordinator computes the estimated model from the average of the models received (*Figure 9*).

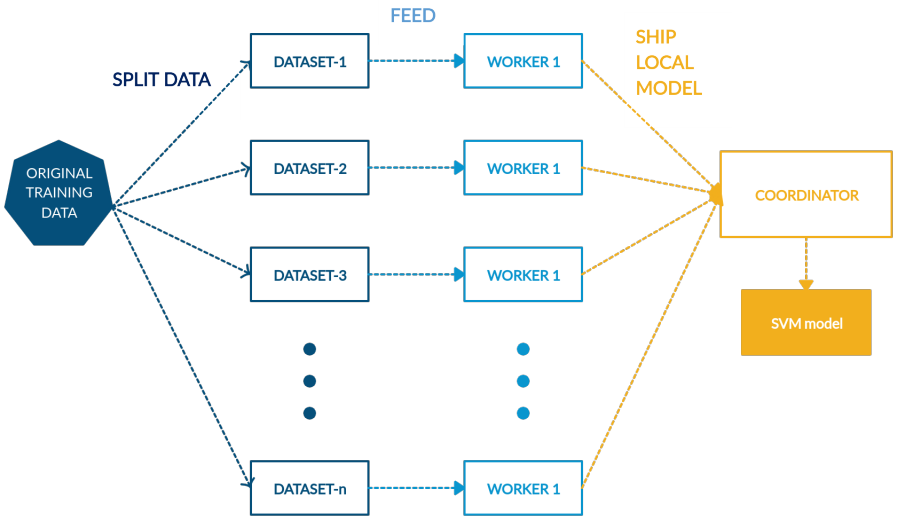

Figure 9: Distributed SVM

Hence, the goal is to prevent the enormous cost of constantly shipping the whole model and let the workers communicate when the model is truly outdated using the FGM protocol.

3.2 Safe Function

Starting the analysis of the FGM algorithm, the safe function is a conceptual concept.The configuration of the system of k sites is a (kD)-dimensional vector consisting of the concatenation of the k local drift vectors, X_i . The system is in *safe state* as long as

$$\boldsymbol{E} + \frac{\sum_{i=1}^{k} X_i}{k} = S \in A \tag{3.1}$$

To guarantee that a configuration is safe, FGM employs a real function $\phi : \mathbb{R}^D \to \mathbb{R}$, depending on $A, \boldsymbol{E}$ and k. Each site tracks its ϕ-value, $\phi(X_i)$, as X_i is updated. System safety is guaranteed by tracking the sign of the sum $\psi = \sum_{i=1}^{k} \phi(X_i)$. In particular, we need to guarantee that $\psi \leq 0$ implies $S \in A$.

There was a need to select the one safe function that fits the criteria of the SVM training. One of the simplest $(A, \boldsymbol{E}, k)$-safe functions for this kind

of monitoring is:

$$\phi(\boldsymbol{x}) = \max\{-\epsilon||\boldsymbol{E}|| - \boldsymbol{x}\frac{\boldsymbol{E}}{||\boldsymbol{E}||}, ||\boldsymbol{x} + \boldsymbol{E}|| - (1+\epsilon)||\boldsymbol{E}||\} \tag{3.2}$$

which is based on the distance of the two vectors $\boldsymbol{E}, \boldsymbol{x}$ (where $\boldsymbol{E}$ is represents the global estimate and $\boldsymbol{x}$ the drift vector) regarding a threshold ϵ.

3.3 Basic FGM Protocol for Learning

FGM Protocol
The FGM protocol works in rounds to monitor the threshold condition

$$\sum_{i=1}^{k} \phi(X_i) \leq 0 \tag{3.3}$$

to detect where the local training models of each worker has changed significantly to perform communication. *Execution of rounds*
At the beginning of a round, the coordinator knows the current state of the system $E = S$, selects a $(A, \boldsymbol{E}, k)$-safe function ϕ as defined above. Note that $\psi = \sum_{i=1}^{k} \phi(X_i)$, at each point in time. Therefore, the FGM protocol steps are:

1. Coordinator ships ϕ to every worker (or just $\boldsymbol{E}$) and local workers initialize their drift vectors to 0. Hence, initially $\psi = k\phi(\boldsymbol{E})$,where k is the number of workers, and subrounds begin.

2. A number of subrounds is initiated to be described below. When all the subrounds ends, meaning that local training models changed in a way that $\psi > e_\psi k\phi(0)$, where e_ψ is related to the desired quantization for monitoring ψ.(for this work e_ψ=0.01)

3. Finally, coordinator ends the rounds by collecting all the drift vectors and updates E.

Execution of subrounds
The purpose of the subrounds is to monitor the condition $\psi \leq 0$ coarsely, with a precision of roughly θ, performing as little communication as possible. Subrounds are executed as follows:

1. The coordinator knows the value ψ and computes the subround 's quantum $\theta = -\frac{\psi}{2k}$ and ships θ to each local worker. Furthermore, the coordinator initializes a counter $c = 0$. Now, each worker records its initial value $z_i = \phi(X_i)$, where $2k\theta = -\sum_{i=1}^{k} z_i$ and initializes a counter $c_i = 0$.

2. Each local worker i maintains its local drift vector X_i, as it updates the local model by performing partial_fit over a mini-batch of data. When X_i is updated, worker i updates its local counter as follows:

$$c_i := \max\{c_i, \lfloor \frac{\phi(X_i) - z_i}{\theta} \rfloor\} \tag{3.4}$$

   If this update increases the counter, the local worker sends a message to the coordinator, with the increases to c_i.

3. When the coordinator receives a message with a counter increment from some worker, it adds the increment to its counter c. If c exceeds k, the coordinator finishes the subround by collecting all $\phi(X_i)$ from all local workers, recomputing ψ. If $\psi \geq e_\psi k\phi(0)$, the subrounds end, else another subround begins. During the execution of a subround, if $c \leq k$ then $\sum_{i=1}^{k} \phi(X_i) < 0$. Note that, in FGM protocol there are two kinds of communications.*Downstream communication* which consists of messages from local nodes to the coordinator and *upstream communication* which consists of messages from the coordinator to local nodes.

3.4 SVM-FGM protocol

The implementation of SVM-FGM protocol is based on sklearn and Dask. Although sklearn is a library used for a centralized machine learning task, was adapted in a distributed online training scenario with the use of Dask. Note that the classic fit function that sklearn provides couldn't be used since is an online SVM algorithm and fit can't support multiple fitting on the same model. Fortunately, the SGDClassifier estimator provides partial_fit which is specifically used for this kind of task since it allows fitting the same model multiple times with different training samples. The SGDClassifier estimator implements regularized linear models with stochastic gradient descent learning including SVM and it was preferred over other sklearn estimators for

SVM for providing partial fitting. On the other hand, Dask is in charge of the distribution. Dask initializes a distributed cluster with the given number of workers and threads for each worker. Workers can directly exchange messages with the Dask scheduler through build-in communication but this is not the case for communication between workers. Since the FGM coordinator is implemented as a dask worker, there was a need for a worker to worker communication which has been provided by Dask Pub-Sub pattern. Later on, follows a thorough explanation for both partial_fit and worker-to-worker communication to finally present the SVM-FGM protocol implementation.

Partial Fit The SGDClassifier estimator implements regularized linear models with stochastic gradient descent learning and includes *partial_fit()* function to support online machine learning. This function performs one epoch of stochastic gradient descent on given samples, so there is no guarantee that the minimum of the cost function is reached. On the other hand, `partial_fit` will correct the model one step towards as new data arrive. This is especially useful when the whole dataset is too big to fit in memory at once. This method has some performance and numerical stability overhead, hence as the experiments also indicate, `partial_fit` is not crucially increasing the training time when the batch size increases significantly. The above figure (10) illustrates the time that one `partial_fit` needs for different sizes of mini-batches.

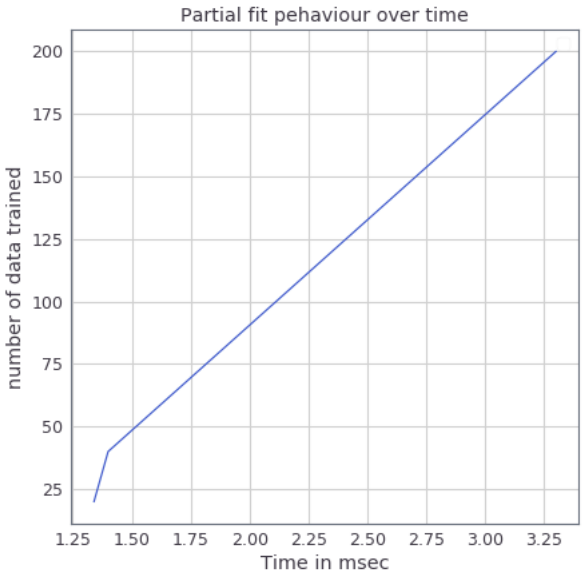

Figure 10: Partial fit behaviour for different sizes of mini-batches.Nearly 10x the batch size is only increasing the time for nearly 2 times.

Worker-to-worker communication Worker-to-worker communication is the communication directly between workers without involving the scheduler at all. So, Dask implements the Publish-Subscribe pattern, providing an additional channel of communication between ongoing tasks. This allows workers to directly communicate data between each other with a typical *Publish-Subscribe pattern*. It involves two components, Pub objects, into which we put data and Sub objects, from which we collect data. The dask workers submit 2 different kind of jobs, the coordinator function and the worker function. Note that dask sees the coordinator as a worker, so the communication between workers and coordinator can be simply performed via *Publish-Subscribe pattern*. In particular, coordinator creates a subject for each kind of messages needed for the upstream and downstream communication. For upstream communication messages we have one publisher (coordinator) and multiple subscribers (workers), where for the downstream com-

munication multiple publishers (workers) and one subscriber(coordinator). So the publisher/publishers simply pushes a message into the pub-sub buffer created for this kind of messages(each subject is a different buffer), and the relevant subscriber/subscribers receives it by getting the first element of the buffered messages. Generally, pub-sub follow the FIFO concept, so the first message pushed into the buffer is also the first one out. For now when upstream and downstream communication is referred, it indicates the Pub-Sub communication

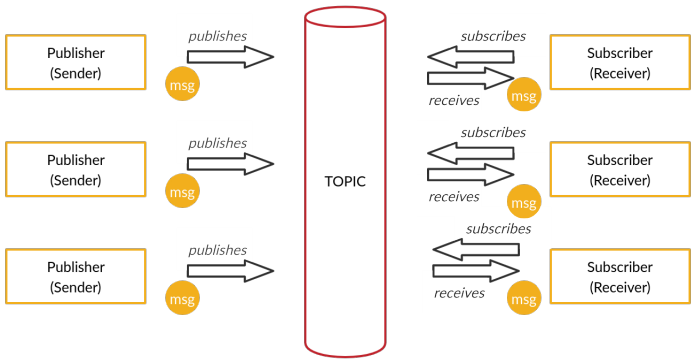

Figure 11: Illustration of Publish-Subscribe pattern

Setup

Applying the FGM protocol in the distributed SVM communication, results in a new distributed structure that includes downstream and upstream communication between the workers and the coordinator as illustrated in (*Figure* 12).

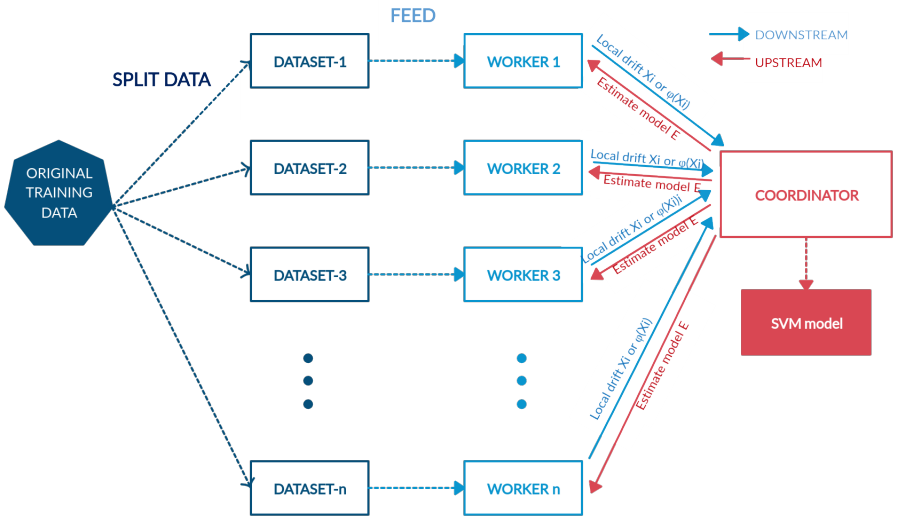

Figure 12: Distributed SVM with FGM protocol

Therefore, the system consists of multiple k workers and one coordinator that orchestrates them. At the beginning of the implementation a Dask distributed cluster is created with k+1 workers and one thread per worker. A large training dataset is split into m equal parts (in this case m=100) and each of the chunk is randomly assigned to one worker, such that all the workers come up with the same number of chunks. Furthermore, the workers read the first chunk, assigned to them, and splits it into smaller parts, mini-batches. Then, the coordinator requests that each and every worker "warms up" by training their local classifier with the first mini-batch, in order to initialize their models. Workers sends back to the coordinator their parameters and the coordinator computes the global estimate. After the "warm up" phase, the FGM protocol is applied.

Practically, each worker distributively trains his local model, given a randomly assigned chunks from the original dataset by using partial_fit. As the FGM protocol requires, while the local model changes, the worker updates its local counter as given in (3.4), and if this update increases the counter, the worker sends a message to the coordinator (*downstream*), with the increase. The coordinator receives a message with a counter increment from some worker, and it adds the increment to its counter c. If c exceeds k, the coordinator requests (*upstream*) from the workers to compute the safe func-

tion given the local drift, $\phi(X_i)$. Workers perform upstream communication to send the $\phi(X_i)$ to the coordinator. The coordinator monitors the threshold condition of the sum of the $\phi(Xi)$ and, if a violation occurs, request the local drifts in order to recompute the global estimate which then broadcasts to all the workers. The workers receives the global estimate, they initialize their model and continue with the training procedure. Overall, only if the local model changes significantly the system performs costly communication. All the above can be described by Algorithm 3.4 . Note that S_i is the local model and X_i the drift vector. Also the ml parameters are the coefficients (weights) and the intercept (bias), as defined by the SVM algorithm.

.

Algorithm 1 SVM-FGM

Initialization at the coordinator

1: Starting k workers
2: Subscribe to all the subject into which the workers will Publish
3: Warming up the global clf and end up with parameters coef, interc
4: $E \leftarrow [coef, interc]$, $c \leftarrow 0$, $\psi \leftarrow k\phi(E)$, $\theta \leftarrow \frac{\psi}{2k}$
5: **Publish** E and θ to all workers and start the first round

A. Worker i on **receiving** E and θ at the start of a new round:

1: **update** the local model: $S_i \leftarrow E$
2: $quantum \leftarrow \theta$, $c_i \leftarrow 0$, $z_i \leftarrow \phi(\boldsymbol{X_i})$

B. Worker on receiving θ at the start of a new subround:

1: $c_i \leftarrow 0$, $quantum \leftarrow \theta$, $z_i \leftarrow \phi(\boldsymbol{X_i})$

C. Worker i on **observing** data at time t:

1: **load** mini-batch $batch_i$
2: **if** $batch_i \neq None$ **then**
3: **update** the local model S_i by partial_fit with $batch_i$
4: compute drift $X_i = S_i - E$
5: $ObservedBatches \leftarrow ObservedBatches + 1$
6: **compute** $c_{i\_new} := \max\{c_i, \lfloor \frac{\phi(\boldsymbol{X_i}) - z_i}{\theta} \rfloor\}$
7: **if** $c_{i\_new} > c_i$ **then**
8: $Increment_i \leftarrow c_{i\_new} - c_i$
9: $c_i \leftarrow c_{i\_new}$
10: **Publish** $Increment_i$ to the coordinator
11: **end if**
12: **end if**

D. Coordinator on **receiving** an increment:

1: $c \leftarrow c + Increment_i$
2: **if** $c > k$ **then**
3: **request** and **collect** all $\phi(\boldsymbol{X_i})$from all workers
4: $\psi \leftarrow \sum_{i=1}^{k} \phi(\boldsymbol{X_i})$
5: **if** $\psi \leq k\phi(\boldsymbol{E})$ **then**
6: **request** and **collect** all $\boldsymbol{X_i}$ from all workers
7: $E \leftarrow E + \frac{1}{k}\sum_{i=1}^{k} \boldsymbol{X_i}$
8: **update** $k \leftarrow$*number of pending workers*
9: $c \leftarrow 0$, $\psi \leftarrow k\phi(\boldsymbol{E})$, $\theta \leftarrow \frac{\psi}{2k}$
10: **Publish** E and θ to all workers and start a new round (code A)
11: **else**
12: $c \leftarrow 0$, $\theta \leftarrow \frac{\psi}{2k}$
13: **Publish** θ to all workers to start a new subround(code B)
14: **end if**
15: **end if**

4 Experimental Results

This section presents the practical performance of SVM-FGM protocol, in a real distributed environment. The main goal is to empirically confirm the performance gains against a centralized (sequential) structure.

4.1 Datasets

The datasets were generated with the use of sklearn random sample generators. These generators can be used to build artificial datasets of controlled size and complexity. Since the main subject is to test classification over a dataset, function make_classification was used. This function generates a random n-class classification problem given parameters that specify the size and the nature of the problem and balance and the desired noise and balance.

The system was tested with 3 different datasets with the same number of samples and features but different percentage of noise. The noise is determined by using different values for flip_y parameter which gives the fraction of samples whose class is assigned randomly. Larger values introduce noise in the labels and make the classification task harder.

Dataset1 The dataset used for this experiment has the following parameters:

- number of samples: 30000
- number of features: 1000
- weights: 50-50 (balanced dataset)
- flip_y **0**

Dataset2 The dataset used for this experiment has the following parameters:

- number of samples: 30000
- number of features: 1000
- weights: 50-50 (balanced dataset)
- flip_y: **0.1**

Dataset3 The dataset used for this experiment has the following parameters:

- number of samples: 30000
- number of features: 1000
- weights: 50-50 (balanced dataset)
- flip_y: **0.2**

4.2 Results

The presentation was illustrated via jupyter notebook and pyplot library. It's important to mention that the Pub-Sub structure requires a small timeout penalty when receiving messages. The following experiments show that the time and hence the speedup is not proportional to the number of workers since when the number of workers increases so does the communication, leading to a bigger timeout penalty.

Also, one of the parameters that affect the communication, hence the performance of the distributed architecture is the threshold, e, that changes the sensibility on the changes that occurred at the local model. Higher values of e make the protocol more resistant to local model changes, so the communication occurs more rarely, where on the other hand lower values of e makes it more sensible resulting in frequent communication. Figure (13) illustrates how the communication and time changes with different threshold values.

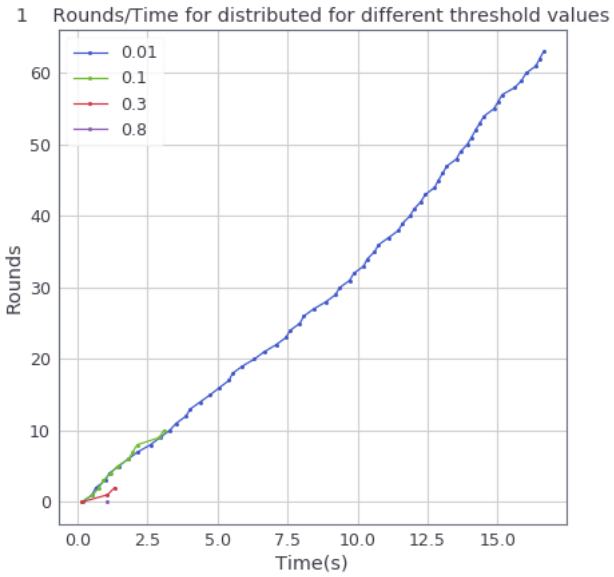

Figure 13: Rounds/Time distributed for different threshold values

It is clear that really small values of threshold results in a centralized like behavior and extremely high communication. For now on the selected threshold is a medium threshold value, e=0.3, to illustrate the average case scenario.

Below there is an illustration of the performance of the system when a different number of workers is chosen.

- Dataset 1

Figure 14: 1.An illustration of the performance of the SVM-FGM protocol for different number of workers, 2. The accuracy for different number of workers. Both for 2 passes of the same dataset without noise

- Dataset 2

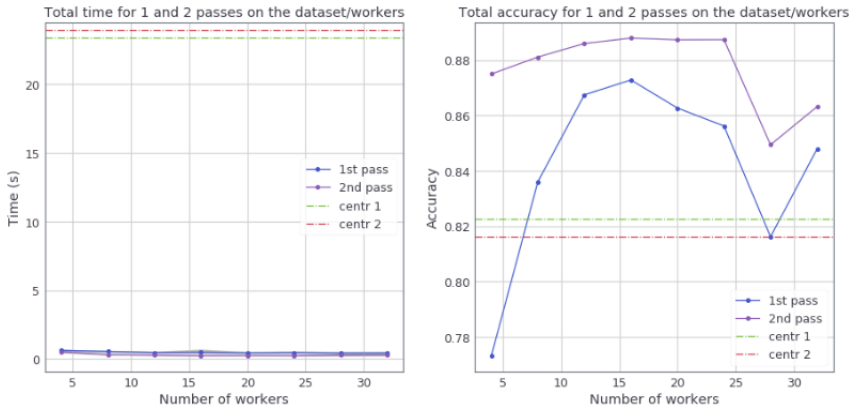

Figure 15: 1.An illustration of the performance of the SVM-FGM protocol for different number of workers, 2. The accuracy for different number of workers. Both for 2 passes of the same dataset with noise 0.1

- Dataset 3

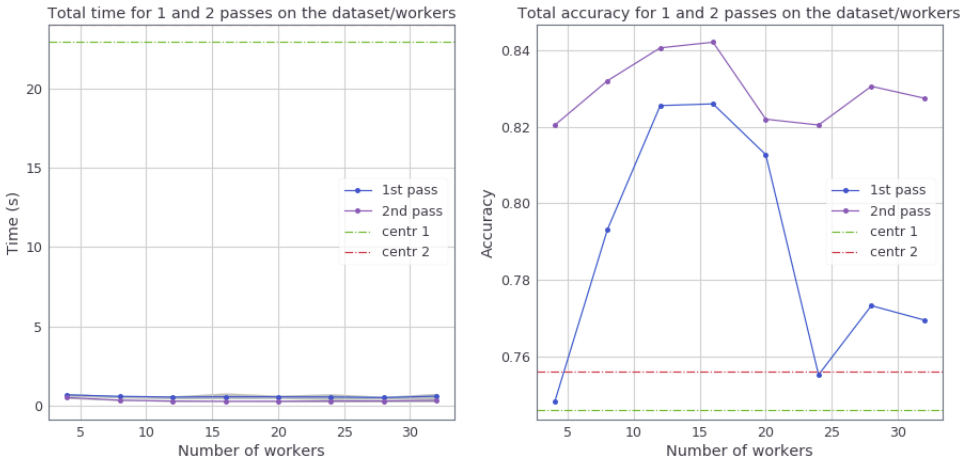

Figure 16: 1.An illustration of the performance of the SVM-FGM protocol for different number of workers, 2. The accuracy for different number of workers. Both for 2 passes of the same dataset with noise 0.2

To count the speedup of the system, we added the time for the second pass to the first one to get the total time needed for 2 passes and compared it with the total time that the centralized architecture needs. So, below is illustrated the speedup for the 3 different noise states.

- Dataset 1

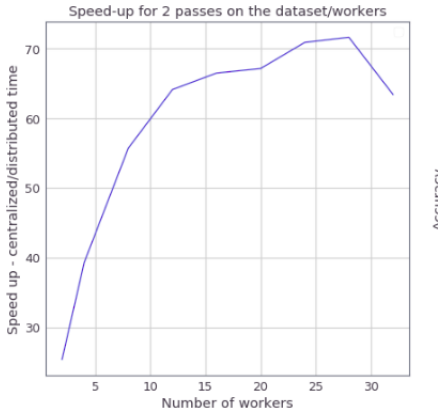

Figure 17: An illustration of the speedup for different number of workers with no noise

- Dataset 2

Figure 18: An illustration of the speedup for different number of workers with 0.1 noise

- Dataset 3

Figure 19: An illustration of the speedup for different number of workers with 0.2 noise

The second pass is used in order to reach a better, much smoother accuracy, that converges to the maximum accuracy for this dataset. Also, the total time for 2 passes doesn't affect much the speedup since the total time for 2 passes in a distributed manner results also to a superlinear speed towards 1 pass of the centralized.

Note that the number of nodes doesn't affect much the total time of the distributed system. This due to the almost standard overhead that Dask with Pub-Sub pattern adds to the system. The above figure illustrates the time that coordinator needs for different phases,

- start time: the time needed to begin the process of the round and warm up.
- wait time: the time that the coordinator processes a small amount of bits, and mainly waits for a significant change to occur
- process time: the time that the coordinator need to collect information from the workers and perform computation.

the table that illustrate the times for each number of workers :

- Dataset 1

	n_workers	rounds	subrounds	start_time	wait_time	process_time_sub	process_time_round
0	2.0	2.333333	2.333333	0.012090	0.948359	0.008034	0.006199
1	4.0	3.000000	3.000000	0.015233	0.576435	0.012186	0.010359
2	8.0	3.400000	3.466667	0.017706	0.428317	0.020663	0.019114
3	12.0	3.066667	3.066667	0.023661	0.359363	0.029820	0.021359
4	16.0	3.000000	3.000000	0.025935	0.326304	0.027234	0.023004
5	20.0	2.933333	2.933333	0.033186	0.318197	0.031415	0.025693
6	24.0	2.733333	2.733333	0.030242	0.308540	0.034148	0.033306
7	28.0	2.466667	2.466667	0.034798	0.265662	0.039637	0.035717
8	32.0	2.333333	2.333333	0.043985	0.278282	0.044344	0.032157

Figure 20: The average times for 15 runs for each number of workers.

and the corresponding plot:

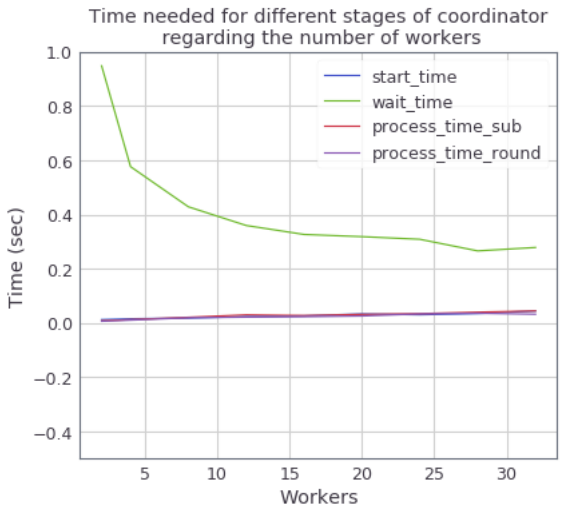

Figure 21: The average times for 15 runs for each number of workers. There are tree plots one for each phase.

- Dataset 2

	n_workers	rounds	subrounds	start_time	wait_time	process_time_sub	process_time_round
0	2.0	2.200000	2.200000	0.011299	0.889799	0.007485	0.005405
1	4.0	3.466667	3.466667	0.015948	0.611605	0.015217	0.013908
2	8.0	3.933333	4.000000	0.019539	0.447036	0.032223	0.024873
3	12.0	4.200000	4.266667	0.022364	0.416360	0.050496	0.035199
4	16.0	3.600000	3.600000	0.027102	0.362509	0.042830	0.035906
5	20.0	3.400000	3.400000	0.032440	0.341142	0.042945	0.038432
6	24.0	2.866667	2.866667	0.035386	0.306783	0.042732	0.033228
7	28.0	2.833333	3.000000	0.050992	0.290526	0.042361	0.030979
8	32.0	2.777778	2.888889	0.044798	0.325619	0.053081	0.038720

Figure 22: The average times for 15 runs for each number of workers.

The corresponding plot:

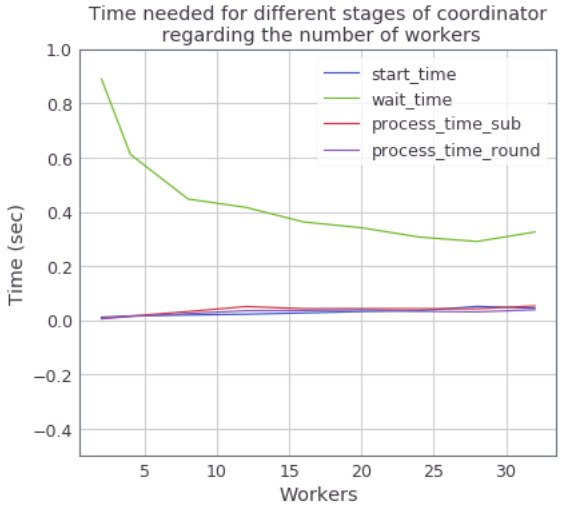

Figure 23: The average times for 15 runs for each number of workers. There are tree plots one for each phase.

- Dataset 3

	n_workers	rounds	subrounds	start_time	wait_time	process_time_sub	process_time_round
0	2.0	2.166667	2.166667	0.013019	0.908606	0.007731	0.006334
1	4.0	3.866667	4.000000	0.016037	0.634150	0.015948	0.014164
2	8.0	4.733333	4.800000	0.018884	0.502959	0.031064	0.029086
3	12.0	4.866667	4.866667	0.025879	0.453641	0.051006	0.041318
4	16.0	4.266667	4.266667	0.028476	0.407764	0.054089	0.045749
5	20.0	3.866667	3.866667	0.031561	0.400529	0.060124	0.047237
6	24.0	3.666667	3.666667	0.035979	0.373683	0.069071	0.060553
7	28.0	3.500000	3.583333	0.035229	0.350289	0.054580	0.055410
8	32.0	3.000000	3.066667	0.042921	0.351723	0.061794	0.046127

Figure 24: The average times for 15 runs for each number of workers.

The corresponding plot:

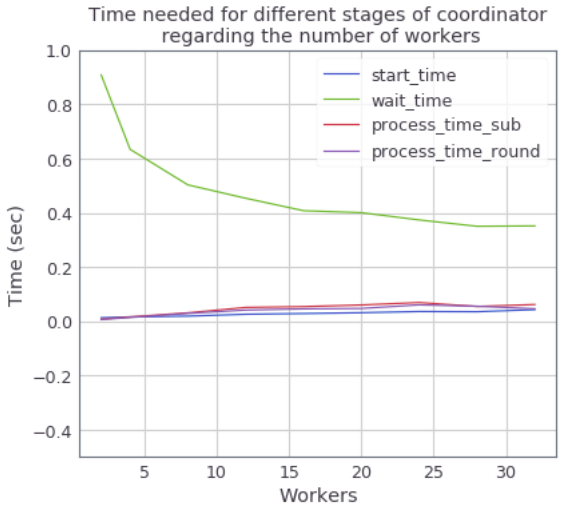

Figure 25: The average times for 15 runs for each number of workers. There are tree plots one for each phase.

As illustrated in the above plot,wait_time decreases when the number of workers increase due to the reduced number of chunks that each worker has to process. Apparently, even thought the wait_time decreases the process_time for the subrounds and the process time for rounds increase due to the work load that the coordinator needs to handle when processing the messages.

The experimental results indicated that in this work, with the use of functional geometric monitoring method we both achieved superlinear speedup and centralized like accuracy, with a minimum communication cost. Also

indicates that sklearn can perform equivalently in a distributed system when using Dask and averaging model. The average cases that performs better in most case includes the use of 0.3 threshold and workers.

Superlinear Speedup The case when the parallel execution of a persistent algorithm can obtain a superlinear speedup due to utilizing more cache memory. Since more cache memory is used in parallel execution, for some region of problem size, it can store the whole problem size, while the sequential execution cannot. In this case, the supper-linear speedup may occur cause the overlap of CPU/IO that the distributed architecture performs doesn't occur in the centralized structure, resulting to a much lower execution time than the minimum expected (Centralized time execution $(\mathrm{t}_c ent) per number of workers : t_c ent/k)$

5 Conclusions

Distributed Support Vector Machine is mainly solved by a sort of distributed chunking technique where the result of the training procedure is combined by using an averaging protocol. Although previous approaches resulted in a centralized-like efficiency, none of them takes into consideration the communication cost of a distributed structure. This work proposed an SVM-FGM averaging protocol that achieves high predictive performance, yet requires substantially less communication than any other contemporary static or distributed averaging SVM protocol. In addition to the reduced communication, experimental results showed that a significantly high speedup, in fact superlinear speedup, has been accomplished establishing that SVM-FGM protocol is indeed suitable for real-time application.

Although SVM-FGM protocol achieved high performance, it still has plenty of room for improvement. First of all, a interesting work will be to achieve an inversely proportional relation between time and number of workers, so that the system performs better for larger systems. Additionally, machine learning includes a wide range of different algorithms that correspond to different problems each time. Hence, a very interesting study could be to practically test other distributed machine learning algorithms and evaluate the results to test machine learning - FGM applicability.

References

[1] Christopher M. Bishop. *Pattern recognition and machine learning*. en. Information science and statistics. New York: Springer, 2006. ISBN: 978-0-387-31073-2.

[2] Stephen P. Boyd and Lieven Vandenberghe. *Convex optimization*. en. Cambridge, UK ; New York: Cambridge University Press, 2004. ISBN: 978-0-521-83378-3.

[3] Corinna Cortes and Vladimir Vapnik. "Support-vector networks". en. In: *Machine Learning* 20.3 (Sept. 1995), pp. 273–297. ISSN: 0885-6125, 1573-0565. DOI: 10.1007/BF00994018. URL: http://link.springer.com/10.1007/BF00994018 (visited on 07/06/2020).

[4] *Dask: Scalable analytics in Python*. URL: https://dask.org/ (visited on 06/29/2020).

[5] *Futures — Dask 2.19.0+6.g7138f470.dirty documentation*. URL: https://docs.dask.org/en/latest/futures.html (visited on 06/29/2020).

[6] *Gradient Descent — ML Glossary documentation*. URL: https://ml-cheatsheet.readthedocs.io/en/latest/gradient_descent.html (visited on 06/18/2020).

[7] *Machine learning*. en. Page Version ID: 960955284. June 2020. URL: https://en.wikipedia.org/w/index.php?title=Machine_learning&oldid=960955284 (visited on 06/06/2020).

[8] Michael E. Mavroforakis and Sergios Theodoridis. "A Geometric Approach to Support Vector Machine(SVM) Classification.pdf". In: IEEE.

[9] Mohri Mehryar, Rostamizadeh Afshin, and Talwalkar Ameet. *Foundations of Machine Learning*.

[10] A. Navia-Vázquez et al. "Distributed Support Vector Machines.pdf". In:

[11] A. Forero Pedro, Alfonso Cano, and Georgios B. Giannakis. "Consensus-Based Distributed Support Vector Machines". In: URL: http://www.jmlr.org/papers/volume11/forero10a/forero10a.pdf (visited on 09/29/2019).

[12] Vasilis Samoladas and Minos Garofalakis. "Functional Geometric Monitoring for Distributed Streams". In: *EDBT2019*. Lisbon, Portugal: submitted to.

[13] *scikit-learn: machine learning in Python — scikit-learn 0.23.1 documentation*. URL: https://scikit-learn.org/stable/index.html (visited on 06/29/2020).

[14] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. en. Cambridge: Cambridge University Press, 2014. ISBN: 978-1-107-29801-9. DOI: 10.1017/CBO9781107298019. URL: http://ebooks.cambridge.org/ref/id/CBO9781107298019 (visited on 06/29/2020).

[15] Izchak Sharfman, Assaf Schuster, and Daniel Keren. ""A geometric approach to monitoring threshold functions over distributed data streams"". In: *SIGMOD*. 2006.

[16] Izchak Sharfman, Assaf Schuster, and Daniel Keren. ""A geometric approach to monitoring threshold functions over distributed data streams"". In: *ACM Trans. Database Syst.* 32.4 (2007).

[17] *sklearn.linear_model.SGDClassifier — scikit-learn 0.23.1 documentation*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html (visited on 06/29/2020).

[18] *Supervised learning*. en. Page Version ID: 956197838. May 2020. URL: https://en.wikipedia.org/w/index.php?title=Supervised_learning&oldid=956197838 (visited on 06/06/2020).

[19] Yang You et al. "CA-SVM: Communication-Avoiding Support Vector Machines on Distributed Systems". en. In: *2015 IEEE International Parallel and Distributed Processing Symposium*. Hyderabad, India: IEEE, May 2015, pp. 847–859. ISBN: 978-1-4799-8649-1. DOI: 10.1109/IPDPS.2015.117. URL: http://ieeexplore.ieee.org/document/7161571/ (visited on 11/19/2019).

[20] Yumao Lu, V. Roychowdhury, and L. Vandenberghe. "Distributed Parallel Support Vector Machines in Strongly Connected Networks". en. In: *IEEE Transactions on Neural Networks* 19.7 (July 2008), pp. 1167–1178. ISSN: 1045-9227, 1941-0093. DOI: 10.1109/TNN.2007.2000061. URL: http://ieeexplore.ieee.org/document/4470008/ (visited on 11/19/2019).