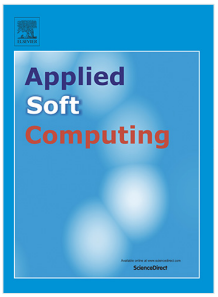
Journal Pre-proof



A distributed density estimation algorithm and its application to naive Bayes classification

Ahmad Khajenezhad, Mohammad Ali Bashiri, Hamid Beigy

PII: S1568-4946(20)30775-4
DOI: https://doi.org/10.1016/j.asoc.2020.106837
Reference: ASOC 106837

To appear in: *Applied Soft Computing Journal*

Received date : 24 November 2019
Revised date : 14 October 2020
Accepted date : 20 October 2020

Please cite this article as: A. Khajenezhad, M.A. Bashiri and H. Beigy, A distributed density estimation algorithm and its application to naive Bayes classification, *Applied Soft Computing Journal* (2020), doi: https://doi.org/10.1016/j.asoc.2020.106837.

A Distributed Density Estimation Algorithm And Its Application To Naive Bayes Classification

Ahmad Khajenezhad[a], Mohammad Ali Bashiri[a], Hamid Beigy[a,*]

[a]*Sharif Intelligent Systems Laboratory,*
Department of Computer Engineering,
Sharif University of Technology,
Tehran, Iran

Abstract

We consider the problem of learning a density function from observations of an unknown underlying model in a distributed setting, where the observations are partitioned into different sites. Applying commonly used density estimation methods such as Gaussian Mixture Model (GMM) or Kernel Density Estimation (KDE) to distributed data leads to an extensive amount of communication. A familiar approach to address this issue is to sample a small subset of data and collect them into a central node to run the density estimation algorithms on them. In this paper, we follow an alternative to the sub-sampling approach by proposing the *nested Log-Poly* model. This model provides an accurate density estimation from a small sized statistic of the entire data. In distributed settings, it transfers the small sized statistics from the client nodes to a central node. The estimation process is then run in the central node. The proposed model can be used in different learning tasks such as classification in supervised learning and clustering in unsupervised learning. However, the properties of nested Log-Poly make it a suitable model for one-dimensional density estimations in the distributed settings. This makes Log-Poly a good choice for naive Bayes classifier, where one-dimensional density estimation is required for every feature conditioned on the class label. We provide a theoretical analysis of the efficiency of our model in estimating a wide range of probability density functions. Our experiments show that nested Log-Poly outperforms the state of the art density estimators on several synthetic datasets. We compare the accuracy and the communication load of naive Bayes classifier using nested Log-Poly and other related density estimators on several real datasets. The experimental outcomes depict that nested Log-Poly has less communication load, while maintaining a competitive classification accuracy compared to similar methods that use the entire data. Moreover, we present a comprehensive comparison between nested Log-Poly and validated KDE with sub-sampling, in terms of number of communicated variables and the number of bytes transferred between the clients and the central node. Nested Log-Poly provides comparable accuracy with the validated KDE with sub-sampling, while communicating fewer variables. However, our method needs to compute and transmit the variables with a high precision in order to accurately capture the details of the underlying distributions.

Keywords: Distributed learning, density estimation, probabilistic models, naive Bayes classifier

1. Introduction

The importance of large-scale learning and inference is growing with continuing advances in big data. Traditional machine learning algorithms have been largely concerned with developing techniques for small or modestly-sized datasets. These algorithms fail to effectively scale up in the presence of large-scale data. This issue is becoming increasingly common in big data era[1]. In the wide range of scalable algorithms, the distributed inference is becoming a key tool, as large data often require to be kept in different sites. In the distributed settings, data repositories might be stored at different locations with low communication speed between them. An efficient distributed inference algorithm should be capable of performing the statistical inference with minimal interaction between these sites (nodes) [2].

Among the problems of interest in distributed inference is density estimation. In this problem, the observations of an (unknown) underlying probability distribution are stored in different sites and the goal is to accurately estimate the underlying probability distribution of the observed data with minimum required communication. Density estimation is used in a variety of learning problems such as clustering and classification. In this paper, we focus on distributed one-dimensional density estimation. As a real-world application, the distributed one-dimensional density estimation can be used for naive Bayes classification in the Internet of Things (IoT) systems. In IoT systems, the observations are gathered by distributed sensors, where the sensor nodes are usually assumed

*Corresponding author
Email addresses: khajenezhad@ce.sharif.edu (Ahmad Khajenezhad), mbashiri@ce.sharif.edu (Mohammad Ali Bashiri), beigy@sharif.edu (Hamid Beigy)

to be power efficient. In such systems, low power design is a well-studied challenge [3, 4, 5]. Particularly in some cases where the sensor nodes communicate over a Wi-Fi network, they need to restrict their communication since communicating over Wi-Fi networks is a power-consuming process.

Kernel Density Estimation (KDE) is a popular non-parametric method that can accurately estimate low dimensional density functions. However, in distributed naive Bayes classification, KDE needs to gather all the attributes of the data (or a sufficient number of samples for each attribute) into one site. This causes a considerable communication load, especially in high dimensional data.

Gaussian Mixture Model (GMM) is another widely used density estimator that estimates the underlying probability distribution with a finite set of parameters. The common method to obtain the parameters of a GMM is the iterative Expectation-Maximization (EM) method. In the case that the samples are not congregated in a single node, each node needs to communicate with other nodes in each EM iteration. This leads to a large amount of communication for an accurate estimation of the parameters.

We propose a family of density functions that enable us to accurately estimate the underlying probability distribution of distributed data while only communicating a small number of variables. Using the exponential families, the proposed method transfers a small-sized sufficient statistic from each client to a central node to provide an accurate estimation of the density function. We call this set of density functions the *Log-Poly* family. Although *Log-Poly* can be generally extended to multidimensional data, its properties make it an excellent choice for one-dimensional density estimation in distributed settings. Additionally, to compensate for the shortcomings of parametric models, we consider a nested hierarchy of Log-Poly sub-models of different degrees (*nested Log-Poly*) and use a sub-model selection among them. Favorably, the communication needed by this nested hierarchy is equal to the communication needed by its largest sub-model. We mathematically show that nested Log-Poly can properly estimate a wide range of density functions. By performing experiments, we show that our method can statistically compete with the state of the art density estimators, with a small communication load.

The rest of this paper is organized as follows: Section 2 introduces the background and some notations. The proposed method is presented in Section 3. While presenting the method, we mention some advantages and challenges of it as well. Section 4 states some important properties of the proposed method and presents the theoretical analysis. Section 5 discusses the related works. Empirical results and experimental analyses are presented in Section 6 and finally, conclusions and future directions will be discussed in Section 7.

2. Preliminaries

In this section, we give a brief overview of the background of the classic and distributed naive Bayes classifiers.

2.1. Density Estimation

Density estimation is a basic statistical problem which is widely used in machine learning applications. Considering the observations as a set $D = \{x^{(1)}, \ldots, x^{(n)}\}$ of n i.i.d. samples from an unknown distribution P, the goal of density estimation is to estimate a model $\hat{P}$ with minimum distance to P. Maximum Likelihood Estimation (MLE) is a common method for learning parametric models.

2.2. Naive Bayes Classifier

Suppose there are m possible classes $\mathcal{C} = \{C_1, \ldots, C_m\}$ and let $\mathbf{x} = \langle x_1, \ldots, x_T \rangle$ be an instance to be classified, where $x_1, \ldots, x_T$ represent the T attributes of the given instance. The Bayes optimal classifier assigns instance $\mathbf{x} = \langle x_1, \ldots, x_T \rangle$ to class C_k with the following probability:

$$P(C_k|x_1, x_2, ..., x_T) = \frac{P(x_1, x_2, ..., x_T|C_k)P(C_k)}{P(x_1, x_2, ..., x_T)} \tag{1}$$

Let us assume that each feature is independent of the other features given the class label (naive independence assumption). Noting that $P(x_1, x_2, \ldots, x_T)$ is constant for an instance (and is equal for all classes), the conditional probability in naive Bayes classifier can be expressed as:

$$P(C_k|x_1, x_2, ..., x_T) \propto \prod_{i=1}^{T} P(x_i|C_k)P(C_k) \tag{2}$$

where $P(x_i|C_k)$ is the distribution of the i^{th} attribute for a given class C_k. For training this classifier, the estimation of each $P(x_i|C_k)$ distribution (for each value of i and k) is needed. Therefore, naive Bayes classifier needs to solve a set of one-dimensional density estimations. A common assumption is that within each class, the values of each attribute come from a normal distribution. One can represent such a distribution in terms of its mean and standard deviation and can efficiently estimate its mean and standard deviation using the Maximum Likelihood Estimation (MLE) method. This model parameterization is computationally efficient and also can be easily used in distributed cases. Nevertheless, the assumption that the attributes obey a Gaussian distribution may not hold for some domains. In [6], it is shown that using more flexible density estimators can improve naive Bayes classifiers in real applications. However, the estimator proposed in [6], the Kernel Density Estimator, suffers from a large communication load in the distributed settings.

2.3. Geographically Distributed Data

In distributed settings, data are kept in separate sites where usually the communication cost between them is expensive. These settings have received much interest in recent years since large-scale datasets and distributed computation platforms are becoming increasingly more common [7]. In this paper, we assume that data are horizontally partitioned which means each site has all attributes of a subset of data. To be more formal, we assume that there exists a set D of n T-dimensional observations stored in S sites (we call them S client nodes). All the T attributes of each observation are stored in the same node. Also, there is a central node (a computational unit) that is connected to all the clients. Each client just communicates with the central node and the central node is responsible to estimate the density function. The i^{th} client, s_i, has n_i data samples denoted by D_i ($|D_i| = n_i$, $\sum_i n_i = n$).

2.4. Distributed Naive Bayes

For training a naive Bayes classifier, we need to estimate $P(x_i|C_k)$ for each feature x_i and class C_k, based on the data with labels C_k that are located in all different sites. In other words, the goal is to efficiently estimate $T \times m$ unknown one-dimensional density functions from observations located in S different sites. By "efficiently", here, we mean to use minimum communication and to be statistically accurate enough. According to Equation (2), we also need the prior probability of each class ($P(C_k)$). This can be easily obtained if we ask each client node to send the total number of samples from each class it stores to the Central node. This process will generate a very small communication load.

3. Proposed Model

We will now introduce the proposed method for designing the distributed naive Bayes which relies on estimating the one-dimensional distributions when data are placed in different sites. From this point onward, we only focus on finding the distribution $P(x_i|C_k)$, which we denote by $f(x)$ for simplicity.

Our proposed model for density estimation is a set of parametric sub-models. We select the best sub-model for estimating the true density function using a validation set. The main contribution of this work is the functional form of the sub-models that we have utilized. These sub-models are indexed by natural numbers ($d \in \{1, 2, 3, \ldots, d_M\}$), where the d^{th} sub-model has the following form:

$$f_d(x;\theta) = \begin{cases} \frac{1}{Z(\theta)} e^{(\theta_1 x + \theta_2 x^2 + \ldots + \theta_d x^d)} & \text{if } x \in [L, R] \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

This is a special form of *Exponential Families*. We call it *Log-Poly* because its logarithm is a polynomial function of x. In Equation (3), θ is the parameter set of the sub-model and Z is a normalizing factor which depends only on θ. In more details, $Z(\theta)$ equals to:

$$Z(\theta) = \int_L^R e^{(\theta_1 x + \theta_2 x^2 + \ldots + \theta_d x^d)} dx. \tag{4}$$

where L and R are some pre-known lower and upper bounds for x. In the absence of prior knowledge about L and R, they can be set to two numbers less than $\min_{x \in D}(x)$ and larger than $\max_{x \in D}(x)$, respectively. Here, D is the set of all (n) samples. All the existing observations lie in the interval $[L, R]$. Thus, in cases where n (the number of samples) is large enough, we can infer that with a high probability, the integral of the real density function outside the range $[L, R]$ is small enough and can be considered as zero. Hence, restricting our sub-models to the set of distributions that are equal to zero outside the range $[L, R]$ will not cause any issues. The d^{th} sub-model has d parameters which indicates its degree of complexity. Since the $(d-1)^{\text{th}}$ sub-model is a subset of the d^{th} sub-model, we call our model the *Nested Log-Poly* model.

The Log-Poly form of the sub-models has several benefits that enable us to achieve a promising accuracy in estimating densities with low communication overhead in comparison with existing methods. First, we express the procedure of density estimation using just the d^{th} sub-model (using maximum likelihood estimation) in Sections 3.1 and 3.2. This procedure needs at most a single communication round (it needs no communication if a more complex sub-model has been run before), and the amount of communication in that one round (if required) is independent of the sample size (n); it depends only on d. Then, in Section 3.3, we will discuss the procedure of sub-model selection and its challenges.

3.1. Density Estimation Using a Single Sub-model

Each single sub-model given in Equation (3) is an exponential family. In general, an exponential family of distributions is a set of density functions of the following form:

$$f(x;\theta) = \frac{1}{Z(\theta)} g(x) \exp(\theta^T \phi(x)) \quad \forall \theta \in \Omega \tag{5}$$

where

$$Z(\theta) = \int_{-\infty}^{\infty} g(x) \exp(\theta^T \phi(x)) dx. \tag{6}$$

$Z(\theta)$ is called the *partition function* and Ω is the set of all feasible values of θ. It is clear that f_d (the Log-Poly of degree d) is an exponential family for which

$$\phi(x) = [x, x^2, ., x^d], \tag{7}$$

$g(x)$ is an indicator function

$$g(x) = I(L \leq x \leq R) \tag{8}$$

and $\Omega = \mathbb{R}^d$. Many of the most common distributions including normal, exponential, and gamma distributions are some instances of exponential families.

When fitting the d^{th} sub-model to the data, our goal is to find the best θ which maximizes the likelihood of the d^{th} sub-model; $f_d(D;\theta)$. The likelihood function has the following form:

$$L_d(\theta|D) = f_d(D;\theta) = \frac{1}{Z(\theta)^n}\Pi_{x\in D} g(x)\exp(\theta^T \sum_{x\in D}\phi(x)). \quad (9)$$

According to Equation (9), the likelihood is a function of $\sum_{x\in D}\phi(x)$ (the sufficient statistic). For a computational agent to find $\hat{\theta} = \arg\max_\theta L_d(\theta|D)$, it is adequate to know this sufficient statistic from the data which is equal to the summation of the sufficient statistics of all the clients, since $\sum_{x\in D}\phi(x) = \sum_i \sum_{x\in D_i}\phi(x)$. Thus, each node computes the sufficient statistic of its local data ($\sum_{x\in D_i}\phi(x)$) and sends them to the central node. The central node computes the sufficient statistic of the whole data (which is just the summation of all clients' sufficient statistics) and finds the point that maximizes the likelihood function. This optimization problem can be solved using Newton's method. We will describe the optimization procedure in detail in Section 3.2. In addition, in order to assign proper values to L and R, the central node also needs to know the minimum and the maximum values of the data. Since $\min_{x\in D} x = \min_i \min_{x\in D_i} x$ and $\max_{x\in D} x = \max_i \max_{x\in D_i} x$, each node is also required to send the minimum and maximum values of its local data, adding a communication load of only two numbers. So, the whole process will be a procedure with just one round of communication where each client should send just $d+2$ numbers to the center.

Discussion: A useful property of the Log-Poly form of the sub-models is that if $d_2 > d_1$, then the sufficient statistic needed by the $d_2{}^{\text{th}}$ sub-model contains the whole sufficient statistic needed by the $d_1{}^{\text{th}}$ sub-model. Therefore, if we have run the procedure of maximum likelihood estimation for the $d_2{}^{\text{th}}$ sub-model beforehand, we need no communication to run the procedure for the $d_1{}^{\text{th}}$ sub-model. On the other hand, if we run the procedure for the $d_1{}^{\text{th}}$ sub-model priorly, running the procedure for the $d_2{}^{\text{th}}$ sub-model requires each client to send only $d_2 - d_1$ numbers to the central node.

Another useful property of the exponential family is the concavity of its log-likelihood as a function of θ. Also, the gradient and the Hessian of the log-likelihood function can be computed for each value of θ by calculating moments of the density function (more details will be explained in Section 3.2). So, numerical optimization methods such as gradient ascent and Newton's method are commonly used for maximum likelihood estimation in exponential families. In contrary to several other models such as GMM, there is no risk of being trapped in local optimum points in the procedure of maximum likelihood estimation. However, the lack of precision in computational operations is a challenge of numerical methods. The precision of numerical approximations becomes more challenging as d grows.

3.2. Details of the Optimization Algorithm

We first show that the logarithm of the density function of an exponential family (Equation (9)) is a concave function of θ. For a likelihood function of the form presented in Equation (9), we have

$$\frac{\partial \log L_d(\theta|D)}{\partial\theta} = -n\frac{\partial \log Z(\theta)}{\partial\theta} + \sum_{x\in D}\phi(x). \quad (10)$$

The term $\sum_{x\in D}\phi(x)$ does not depend on θ and is a constant value. The derivative of the logarithm of the partition function is as follows:

$$\begin{aligned}\frac{\partial \log Z(\theta)}{\partial\theta} &= \frac{\partial}{\partial\theta}\log\int g(x)\exp(\theta^T\phi(x))dx\\ &= \frac{1}{\int g(x)\exp(\theta^T\phi(x))dx}\int g(x)\exp(\theta^T\phi(x))\phi(x)dx\\ &= \int \frac{1}{Z(\theta)} g(x)\exp(\theta^T\phi(x))\phi(x)dx\\ &= \int f(x;\theta)\phi(x)dx = E[\phi(x)]. \end{aligned} \quad (11)$$

With similar computations it can be shown that

$$\begin{aligned}\frac{\partial^2 \log Z(\theta)}{\partial\theta^2} &= cov[\phi(x)]\\ &= E[\phi(x)\phi(x)^T] - E[\phi(x)]E[\phi(x)]^T. \end{aligned} \quad (12)$$

Expectations in Equations (11) and (12) are taken with respect to $x \sim f(x;\theta)$. Since every covariance matrix is a positive semi-definite matrix, Equation (12) proves that $\log Z(\theta)$ is a convex function of θ, and consequently the log-likelihood function of an exponential family is a concave function of θ.

According to concavity of log-likelihood of the exponential families, the general way for obtaining the maximum likelihood in an exponential family is through running steepest ascent methods on the log-likelihood function. Here, we use Newton's method with backtracking line search [8] for determining the step size. According to Equation (7), for the form of our sub-models, derivatives of the partition function are equal to:

$$\frac{\partial \log Z(\theta)}{\partial\theta} = E[(x, x^2, \ldots, x^d)^T] \quad (13)$$

and

$$\frac{\partial^2 \log Z(\theta)}{\partial\theta^2} = A - E[(x,\ldots,x^d)^T]E[(x,\ldots,x^d)], \quad (14)$$

where A is a $d\times d$ matrix with elements $a_{ij} = E[x^{i+j}]$. According to Equations (13) and (14), for running Newton's optimization method we need to compute $E[x^i]$ for all $1 \le i \le 2d$ in each step, with respect to the value of θ in that step:

$$E[x^i] = \int_L^R \frac{x^i}{Z(\theta)} exp\left(\sum_{j=1}^{d}\theta_j x^j\right) dx. \quad (15)$$

In Equation (15), if we know the value of $Z(\theta)$, expectations can be approximated by Riemann sums. Thus, the remaining problem is computing $Z(\theta)$. We approximate $Z(\theta)$ using a Riemann sum as well. But the challenge in approximating

$$Z(\theta) = \int_L^R \bar{f}_d(x;\theta)dx = \int_L^R exp\left(\sum_{j=1}^{d} \theta_j x^j\right) dx \tag{16}$$

is that the value of $\bar{f}_d(x;\theta)$ may be too large or too small for some values of x. Therefore, it might overflow or underflow the computer variables. To address this issue, we do calculations in the logarithmic space to compute $\log Z(\theta)$. In addition, to avoid numerical overflows and underflows and to increase the accuracy of Riemann sums, we use the following computational technique:

$$\log\big(\exp(a)+\exp(b)\big) = \max(a,b) + \log\Big(\exp\big(a-\max(a,b)\big)+\exp\big(b-\max(a,b)\big)\Big). \tag{17}$$

When the order of magnitude of one of a and b is much greater than the other one, then the lack of precision caused by the possible underflow will be negligible; Otherwise, our computation will be precise.

Also, to avoid large numbers in our computations, by a simple shift and scaling, we map the data to the interval of $[0.05, 0.95]$. We consequently set $L = 0$ and $R = 1$ in our experiments. Finally, it is worth mentioning that we used *mpmath* python package with a decimal precision of 50 digits to perform high precision calculations in our implementation.

3.3. Sub-model Selection

In Sections 3.1 and 3.2, we demonstrated how to perform maximum likelihood estimation for different sub-models. Now, we explain how we select the right sub-model. As we have mentioned, nested Log-Poly is comprised of a set of sub-models of parametric families. To choose which sub-model provides the closest distribution to the true distribution from which the data are sampled, we need to compare the true distribution with each sub-model and choose the best one. For this cause, we use a validation set. To evaluate each sub-model, we first estimate its parameters using the training data. Then, we compute the likelihood of the validation data by that sub-model using the tuned parameters. According to the law of large numbers, it is clear that in the case of a large validation set drawn from the true distribution, the likelihood of this set by a sub-model is a measure of the Kulback-Leibler (KL) divergence between the distribution represented by that sub-model and the true distribution. Thus, we select the sub-model with the highest likelihood of the validation set as the best sub-model.

In practice, we consider a value d_M as the maximum allowed value for d, and select the best sub-model among the sub-models with degrees in $\{1, 2, \ldots, d_M\}$.

However, in theory, we can assume a hierarchy of infinite sub-models with degrees $\{1, 2, \ldots\}$ (in practice, this assumption needs computations with a precision of infinite digits). For model selection, in this case, we can use a look-ahead window of size w. We can start from $d = 1$ and repeatedly increase d by one until we reach the d^{*th} sub-model that has the highest likelihood value on the validation set among all the sub-models of degrees $d^*, \ldots, d^* + w$. It is worth noting that by increasing d, the sub-models get more powerful and consequently, the likelihood of the training data increases. By increasing the value of d, the likelihood of the validation data will also increase until the sub-models start to overfit to the training data. At this point, the likelihood of the validation set will start to decrease.

4. Properties of the Nested Log-Poly Model

In this section, we first restate all the benefits and contributions of the proposed method which we discussed sporadically in previous sections. Then, in Section 4.1, we present a theoretical analysis of the capability of the proposed model for approximating density functions. Using the Log-Poly form of sub-models has several benefits that make it efficient for distributed density estimation.

Low communication cost: For each sub-model, we need a single communication round and the amount of communication in that one round is independent of the sample size (n) and it depends only on the degree of the polynomial (d). We can compare this communication cost with a KDE and a GMM with k components. We have used Log-Poly's with degrees of at most 20 in our experiments reported in Section 6 (except for one of the real datasets for which we also used Log-Poly of degree 100). Hence, the nested Log-Poly models need communication of 20 numbers from each client to the central node to estimate a density function. KDE needs to transfer all the training data (or at least several thousands of samples) to the central node. Thus, it needs much more communication than the Log-Poly method. On the other hand, the common method to fit the parameters of GMM is Expectation-Maximization (EM) in which the update formulations of the mean (μ_k), variance (Σ_k), and the prior probability of each component in each EM iteration are as follows [9]:

$$\mu_k \leftarrow \frac{1}{n^{(k)}} \sum_{i=1}^{n} \gamma(z_{ik}) x^{(i)} \tag{18}$$

$$\Sigma_k \leftarrow \frac{1}{n^{(k)}} \sum_{i=1}^{n} \gamma(z_{ik})(x^{(i)} - \mu_k)^2 \tag{19}$$

$$\pi_k \leftarrow \frac{n^{(k)}}{n} \tag{20}$$

where

$$n^{(k)} = \sum_{i=1}^{n} \gamma(z_{ik}) \tag{21}$$

and $\gamma(z_{ik})$ is the probability that $x^{(i)}$ comes from the k^{th} Gaussian component. It is clear that in the distributed setting, each client node should compute the following three values for all instances residing in that client; $\sum_i \gamma(z_{ik})$, $\sum_i \gamma(z_{ik})x_i$, and $\sum_i \gamma(z_{ik})x_i^2$, for each component k, and then send them to the central node in each EM iteration. Also, the central node should compute the new values of μ_k, Σ_k, and π_k for each component k and send them back to all the clients. Therefore, each client node should send $3k$ numbers and receive $3k$ numbers in each EM iteration. Hence, running the EM algorithm for I iterations needs a total communication of $I \times 6k$ numbers between the central node and each client node. Table 1 shows the required number of communication rounds and the communication load of each round for different density estimators.

Nested hierarchy: Since the Log-Poly sub-models are nested and the sufficient statistics of them are subsets of each other, if $d_2 > d_1$, the sufficient statistic of data for f_{d_2} will contain the whole sufficient statistic of data for f_{d_1}.

Concavity: Another useful property of the exponential family is the concavity of its log-likelihood as a function of θ. Also, the gradient and Hessian of its log-likelihood function can be computed for each value of θ by calculating moments of the density function. Thus, numerical optimization methods such as gradient ascent and Newton's method can be used for maximum likelihood estimation in these models (however, they need high precision computations). In contrary to several other models such as GMM, there is no risk of being trapped in local optimum points in the procedure of maximum likelihood estimation.

Finally, it can be theoretically shown that if d grows large enough, Log-Poly is capable to fit a wide range of density functions. In the following subsection, we prove a theorem that shows how Log-Poly can fit different density functions. However, the upper bound we find for the KL divergence of Log-Poly and the true distribution in this analysis is not necessarily a tight one.

Table 1: Number of communication rounds and communication size of the client s_j with n_j samples for different density estimation methods, in the distributed setting where a central node is responsible to estimate the density function.

Method	number of communication rounds	Communication size per round
KDE	1	n_j
GMM with k components	$2\times$ number of EM iterations	$3k$
Log-Poly of degree d	1	d

4.1. Theoretical Analysis

In this section, we study how accurate a density function under specific conditions can be estimated using a Log-Poly model. For this purpose, we first prove a lemma about estimating the logarithm of continuous functions using polynomials.

Lemma 1. *Let f be a continuous function defined on $[0,1]$, such that $\max_{x\in[0,1]} |\log f(x)| \le M$. Let also $h : [0,1] \mapsto \mathbb{R}$ be defined as*

$$h(\tau) = \max_{\substack{x,y\in[0,1]\\ |x-y|\le\tau}} (|\log f(y) - \log f(x)|). \tag{22}$$

Then for all $d \in \mathbb{N}$, for all $x \in [0,1]$, and for all $t > 0$, there exists a polynomial P of degree d such that

$$|P(x) - \log f(x)| < h\left(\frac{t}{4d}\right) + \frac{2M}{t^2}. \tag{23}$$

Proof. The proof is directly induced from a constructive proof of the Stone-Weierstrass theorem using Bernstein polynomials [10]. Suppose a random variable K as the sum of d Bernoulli trials with parameter x (x can be any ordinary value in $[0,1]$). The mean and the variance of K/d are equal to x and $x(1-x)/d$, respectively. Thus, noting that $x(1-x)/d \le 1/4d$, using Chebyshev's inequality, we conclude that

$$Pr\left(\left|\frac{K}{d} - x\right| \ge \frac{t}{4d}\right) \le \frac{1}{t^2}. \tag{24}$$

This implies that

$$Pr\left(\left|\log f\left(\frac{K}{d}\right) - \log f(x)\right| \ge h\left(\frac{t}{4d}\right)\right) \le \frac{1}{t^2}. \tag{25}$$

Using inequalities (25) and $\max_{x\in[0,1]} |\log f(x)| \le M$, we obtain (note that K is a random variable, but x and d are constant numbers):

$$\begin{aligned}
&E\left[\left|\log f\left(\frac{K}{d}\right) - \log f(x)\right|\right] \\
&= \sum_k |\log f(\tfrac{k}{d}) - \log f(x)| Pr(K=k) \\
&\le \sum_{k:\ \left|\log f(\frac{k}{d}) - \log f(x)\right| < h(\frac{t}{4d})} h(t/4d) Pr(K=k) \\
&+ \sum_{k:\ \left|\log f(\frac{k}{d}) - \log f(x)\right| \ge h(\frac{t}{4d})} 2M Pr(K=k) \\
&\le h(t/4d) + \frac{2M}{t^2}.
\end{aligned} \tag{26}$$

According to the inequality $|E[a] - E[b]| \le E[|a-b|]$, we infer that

$$\begin{aligned}
&E[\log f(\tfrac{K}{d})] - \log f(x) = E[\log f(\tfrac{K}{d})] - E[\log f(x)] \\
&\le E[|\log f(\tfrac{K}{d}) - \log f(x)|] \le h(t/4d) + \tfrac{2M}{t^2}.
\end{aligned} \tag{27}$$

Finally we should notice that

$$E[\log f(\frac{K}{d})] = \sum_{k=0}^{d} \log f(\frac{k}{d}) \left(c(d,k) x^k (1-x)^{(d-k)}\right)$$

is a polynomial of degree d with the desired property. □

Using Lemma 1, now we can prove a theorem about estimating density functions using Log-Poly sub-models. The following theorem shows that the Log-Poly can approximate any continuous probability density function.

Theorem 1. *For any continuous probability density function f defined on $[0,1]$ with properties declared in Lemma 1, and for any $d \in \mathbb{N}$, and any $t > 0$, there exists a Log-Poly probability density function q of degree d that satisfies*

$$KL(f||q) \leq 2\left(h\left(\frac{t}{4d}\right) + \frac{2M}{t^2}\right). \tag{28}$$

Proof. For simplicity, set $\epsilon = h(\frac{t}{4d}) + \frac{2M}{t^2}$. Due to Lemma 1, there exists a polynomial P of degree d such that $\forall x \in [0,1]$, we have

$$|P(x) - \log f(x)| < \epsilon \tag{29}$$

or equivalently

$$e^{-\epsilon} < e^{P(x)-\log f(x)} < e^{\epsilon}, \tag{30}$$

which means

$$f(x)e^{-\epsilon} < e^{P(x)} < f(x)e^{\epsilon}. \tag{31}$$

This implies that

$$e^{-\epsilon} \leq \int_0^1 \exp\{P(x)\}dx \leq e^{\epsilon}. \tag{32}$$

Therefore, since $e^{-\epsilon} < 1 < e^{\epsilon}$, there exists a $\delta \in [-\epsilon, \epsilon]$ such that

$$\int_0^1 \exp\{P(x) + \delta\}dx = 1. \tag{33}$$

It is clear that $q(x) = I(x \in [0,1]) \exp\{P(x) + \delta\}$ is the desired probability density function. □

Finally, we should note that if g is a continuous density function defined on $[0,1]$ and it is equal to zero at certain points in $[0,1]$, then $\log g$ tends to $-\infty$ at those points. For such density functions, if we first consider a density function $f(x) = \frac{g(x)+\sigma}{1+\sigma}$ (for some desired value of σ) then we can find an estimation error for f by Log-Poly density functions, using Theorem 1.

5. Related Works

The idea of using exponential families with polynomial exponents for density estimation was originally studied in [11]. This idea was also used to estimate density function in constrained memory devices [12]. Broderick et. al have proposed the use of exponential families in distributed learning for problems such as posterior estimation [13]. To the best of our knowledge, our proposed method is the first effort in using exponential families for distributed density estimation.

In the remainder of this section, we review related density estimators for one-dimensional data and how they are applied in distributed settings. We study different estimators under two categories of parametric and non-parametric density estimators. In summary, non-parametric estimators can model any complex density function with sufficient number of samples. However, they are not suitable for distributed density estimation since their parameter set is very large. On the other hand, parametric models have a fixed and small number of parameters. However, these models are either simple and statistically inefficient, or complicated with iterative training processes that have high communication load in distributed settings.

5.1. Non-Parametric Models

In non-parametric models, the number of parameters is allowed to grow with the size of the available data. This property enables the non-parametric models to accurately estimate complex density functions in presence of large samples.

Among non-parametric methods, kernel density estimators are convenient and widely used. However, to estimate a density function in distributed settings, KDE requires all training data to be collected in one site, causing an extremely high communication load when the number of data is large. In fact, the parameter set of KDE is the whole dataset. Some studies have been done on how to apply KDE in a distributed environment [14, 15]. These methods approximate KDE, using iterative message passing and sub-sampling.

Some parametric models (such as GMM) can fit any complex distribution if we let the number of their parameters (number of components for GMM) grow enough. In fact, if we consider a model as a hierarchy of infinite GMM sub-models (each sub-model with a specific number of components), this model will be a non-parametric one. However, it also requires a method for selecting the best sub-model according to the observed samples. This hierarchy of infinite GMM's has been proposed in [16, 17], where the Dirichlet process has been used as a prior for the number of components and their probabilities. However, inference in this model is performed through Gibbs sampling which is an iterative algorithm that needs many rounds of communication in distributed settings.

Our proposed method in this paper is also a hierarchy of parametric sub-models with different degrees of complexity. Although an infinite hierarchy of sub-models can be considered theoretically, in practice, we use a finite set of sub-models by determining a maximum degree of complexity for the sub-models.

5.2. Parametric Models

In parametric density estimators, a model with a finite set of parameters is assumed as the density function. The parameters of the specified model are usually estimated using Maximum Likelihood Estimation (MLE) on the whole data. Due to the small size of parameters, one might guess that parametric models are proper for distributed density estimation. Nevertheless, that is not correct because MLE does not have a closed-form formulation according to the model parameters, except for very simple parametric models such as a simple Gaussian or Gamma distribution. More complicated parametric models, such as mixture models, require distributed optimization

algorithms [18] which need iterative communications between separated nodes. This can significantly decelerate the algorithm regardless of the amount of information communicated at each iteration [19]. Although some researches, such as [20], have proposed methods with lower communication loads for some distributed optimization problems, they often have sacrificed accuracy for speed.

To get rid of the communication load of iterative optimization, some prior works have proposed running MLE on local data in each client node and combining their estimated parameters in a central node to get an approximation of the global MLE. Merugu and Ghosh [21, 22] proposed this method, where the central node finds a distribution which has the minimum distance from the average distribution of the locally estimated distributions. Although this method is communicationally efficient, it does not guarantee to provide good estimations for all underlying distributions. A theoretical analysis of the statistical performance of this method has been provided in [19]. It shows that the efficiency of the global estimation depends on the proximity of the underlying distribution families to full exponential families.

6. Experimental Analysis

In this section, we will study the empirical performance of nested Log-Poly on both synthetic and real datasets[1]. First, in Section 6.1 we briefly describe some examples that show how Log-Poly provides a closer estimate to several density functions in comparison with GMM and KDE. The goal, however, is not to claim that nested Log-Poly statistically outperforms other density estimators in all cases; Rather, it is to show that there exist some density functions where Log-Poly sub-models provide better estimates than GMM or KDE. In Section 6.2, we illustrate the effect of sub-model selection in nested Log-Poly by comparing the average likelihood of sub-models with different degrees on the train, validation, and test data. In Section 6.3, we compare the accuracy of naive Bayes classifier when nested Log-Poly and three other distributed density estimators are used on real data with high dimensional input space. Finally, in Section 6.4, we compare the communication load of nested Log-Poly against related distributed density estimation methods.

Dataset: We used synthetic data to exhibit the statistical superiority of Log-Poly against KDE and GMM in some cases. For experiments on sub-model selection (Section 6.2), naive Bayes classifiers (Section 6.3), and communication load evaluation (Section 6.4), we used *detection_of_IoT_botnet_attacks_N_BaIoT* dataset[2] [23] (we will refer to this dataset by "*the IoT dataset*"). This dataset is a collection of traffic observations on nine different IoT devices. Each observation in this dataset either belongs to the benign traffic or one of 10 network attacks. We also used *MAGIC Gamma Telescope* dataset[3] [24] with 2 classes (we will refer to this dataset as "*the MAGIC datasaet*") and *Gas sensors for home activity monitoring* dataset[4] [25] with 3 classes (we will refer to this dataset as "*the Gas-Sensors dataset*"). From the *Gas-Sensors* dataset, we only used observations from the time interval $[0.5, 1]$.

Data preparation: All the synthetic data and all dimensions of the real data were scaled to the $[0.05, 0.95]$ before applying any density estimation method on them. In fact, we assumed that the valid range of each dimension is given in prior. For distributed experiments given in Section 6.4, we partitioned the data horizontally among the processes that were representing the client nodes. For density estimators that need validation set (such as nested Log-Poly for determining the degree of the polynomial), we separated $\frac{1}{10}$ of data as the validation set.

Measurement criteria: For measuring the statistical performance of a method to estimate the underlying 1-dimensional density of a set of observations (in Section 6.1), we use the Kullback-Leibler divergence (KL divergence) between the true distribution and the estimated distribution using that method. The KL divergence of an estimated density function Q from the true density function P is defined as follows

$$D_{KL}(P||Q) = \int_{-\infty}^{+\infty} P(x) \log\left(\frac{P(x)}{Q(x)}\right) dx. \qquad (34)$$

In practice, given a sample set D of n observations drawn independently from P, Equation (34) can be approximated by:

$$D_{KL}(P||Q) \approx \frac{1}{n} \sum_{x \in D} P(x) \log\left(\frac{P(x)}{Q(x)}\right). \qquad (35)$$

We use Equation (35) to evaluate how close the output of a density estimator (Q) is to the true distribution (P). For evaluating naive Bayes classifiers (in Section 6.3), we use the accuracy measure. Furthermore, in the distributed experiments (in Section 6.4), we report the communication load of each method by counting both the number of variables and the number of bytes transferred between the center node and the client nodes.

6.1. Log-Poly's Performance in Density Estimation

The advantage of nested Log-Poly, in comparison to other density estimators, is its low communication cost in a distributed setting. However, in what follows, we will show some cases that Log-Poly sub-models provide statistically closer estimates to the true distribution in comparison with GMM and KDE. These

[1] The source codes of our experiments are available at http://github.com/ahmadkhajehnejad/logpoly_naiveBayes.

[2] http://archive.ics.uci.edu/ml/datasets/detection_of_IoT_botnet_attacks_N_BaIoT

[3] https://archive.ics.uci.edu/ml/datasets/magic+gamma+telescope

[4] https://archive.ics.uci.edu/ml/datasets/Gas+sensors+for+home+activity+monitoring

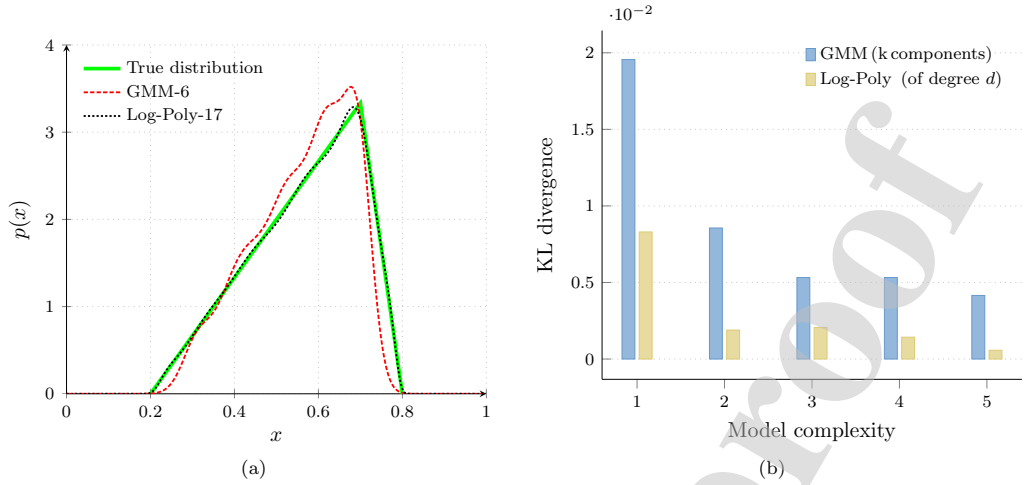


Figure 1: The results of estimating a triangular density function by Log-Poly and GMM. Figure 1a shows the true density function, a GMM with 6 components and a Log-Poly of degree 17 fit to 10^5 samples generated from the true distribution. Figure 1b shows how the KL divergence between the true distribution and the estimated distribution changes with different numbers of parameters for GMM and Log-Poly.

examples, however, don't mean that Log-Poly outperforms other density estimators in all cases. In the following subsections, we will use Equation (35) to approximate the KL divergence.

6.1.1. Log-Poly vs GMM

We compare Log-Poly with GMM in two different situations. First, we compare them when there is no limit on the number of EM iterations. Second, we compare them in a more realistic situation where there is a limit on the number of EM updates. We will clarify this limitation later in this section.

Unlimited EM iterations: The first comparison is on estimating a triangular density function shown in Figure 1a (the green curve). For a fair comparison, it is necessary to choose models with the same complexity (free parameters). Therefore, a GMM with k components must be compared to a Log-Poly of degree $3k-1$. This is because a one-dimensional GMM with k components has $3k-1$ free parameters: k mean points, k variances, and k components' probabilities where one of them can be computed knowing the others. For an illustrative comparison, we train a GMM with 6 components on 10^5 samples generated from the true distribution until convergence. Then, we similarly fit a Log-Poly of degree 17 on the same samples and compare the estimated distributions and the true distribution. Figure 1a shows the true distribution and the estimated distributions. This figure shows that Log-Poly captures the underlying true distribution of the generated samples more accurately.

For a quantitative comparison on different choices of model complexity, we train GMM and Log-Poly with different complexities and compare the KL divergences of their estimated distributions and the true distribution (Figure 1b). It can be seen that Log-Poly performs better for different choices of model complexity.

Limited EM iterations: Consider a distribution that can be represented as a mixture of k components. Having enough samples from this distribution, a GMM with k components can accurately fit this distribution. However, it needs a sufficient number of EM iterations to converge to the true distribution. On the other hand, each client node in each EM iteration needs two rounds of communication to send $3k$ numbers to the central node and receive $3k$ numbers from it (as explained in Section 4). Therefore, communication constraints in the distributed applications limit the allowed number of EM iterations. Figure 2 shows the true distribution; a mixture of six Gaussians with parameters $\pi = [\frac{9}{26}, \frac{2}{26}, \frac{3}{26}, \frac{4}{26}, \frac{1}{26}, \frac{7}{26}]$, $\mu = [1, 20, 50, 85, 130, 160]$ and $\sigma = [1, 2, 3, 1, 2, 3]$. This figure also shows the result of training a GMM with six components for 200 EM iterations on 10^5 samples from the true distribution. According to the discussions in Section 4, in a distributed application, it takes 2×200 rounds of communication and a total communication of $7200 = 200 \times 6 \times 6$ numbers for each client. However, as Figure 2 shows, GMM has not been able to capture all the six components of the true distribution in 200 iterations. On the other hand, Figure 2 shows that a Log-Poly with $d = 17$ (which needs a communication of only 17 numbers in just one communication round for each client) can approximately fit the true distribution and capture all the six components.

6.1.2. Log-Poly vs KDE

Although kernel density estimation is a non-parametric method that converges to the true distribution as the number of observations tends to infinity under mild conditions [26], nested Log-Poly outperforms KDE in some cases. Figure 3a shows a gamma distribution

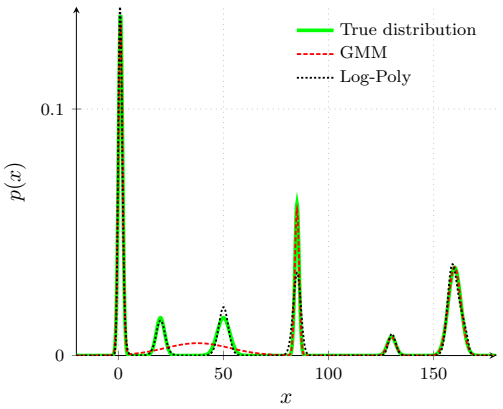


Figure 2: GMM with $k = 6$ components has failed to completely fit the true distribution (a mixture of 6 Gaussians), in 200 EM iterations. However, Log-Poly with $d = 17$ has worked nice.

with parameters $k = 2$ and $\theta = 0.5$. Nested Log-Poly provides better estimations than KDE on this distribution. Figure 3b shows the results of the two methods using 2000, 3000, and 4000 observations. In this experiment, we used 100 observations as validation data to set the kernel width of the KDE method and to select the best Log-Poly sub-model.

6.2. Sub-model Selection

In this section, we investigate the effect of sub-model selection using a validation set on our density estimation method. We used the first feature of the samples gathered from the device *Danmini Doorbell* and labeled as *gafgyt-combo attack* in the IoT dataset. We used a validation set and a test set, each of size 500. Then we trained Log-Poly sub-models with $d \in \{1, 2, \ldots, 20\}$ using two different training sets: a training set of size 500 and a training set of size 3000. Figures 4a and 4b represent how the log-likelihoods of the training, validation and test sets change as d grows. According to Figure 4a, the sub-model with degree $d = 7$ is the best one (i.e. the model with the maximum accuracy on the validation set) in that case. Figure 4b shows that when the training size increases to 3000, it is preferred to choose a more complex sub-model ($d >= 15$). Also, it confirms that having a large training set prevents the model from overfitting to the training data. This is due to the fact that the log-likelihoods on the validation set and the test set do not decrease significantly when we increase the model complexity up to $d = 20$.

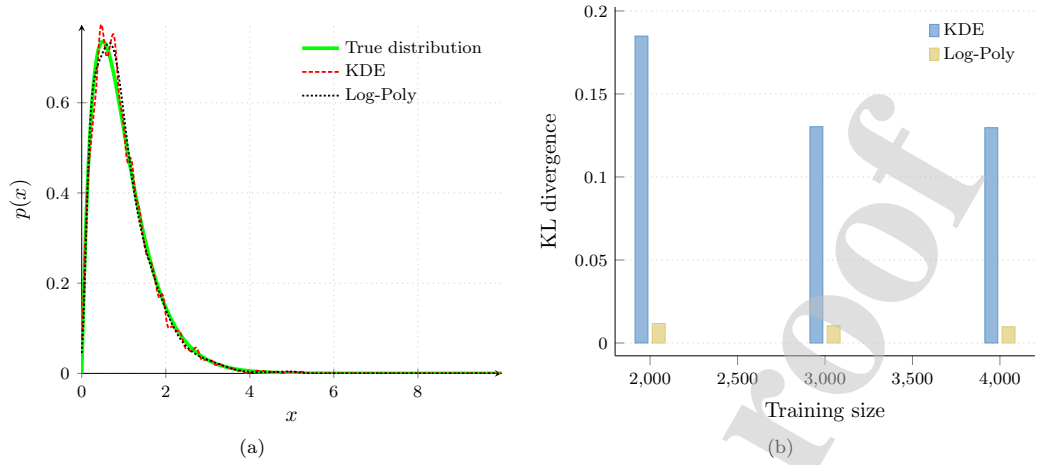
6.3. Naive Bayes with Different Density Estimators

In this section, we study the experimental results of the naive Bayes classifier on the *IoT* dataset using different density estimators including nested Log-Poly. We have used the traffic observations collected from five devices as five different experiments. Each observation is a vector of 115 real-valued features and there exist more than 800000 observations for each device. Table 2 shows the number of available observations from each device. The data from each device is divided to 11 classes : *benign traffic* and 10 different network attacks. Table 2 also shows the results on the *MAGIC* 2-classes dataset and the *Gas-Sensors* 3-classes dataset which both have 10 real-valued features. In each experiment, we used a 5-fold evaluation to approximate the mean and the standard deviation of the classification accuracy. We used naive Bayes classifiers with nested Log-Poly, a single Gaussian, GMM, KDE, and validated KDE (a KDE in which the kernel width has been tuned using a validation set) density estimators. We used one-tenth of the training data as the validation set in nested Log-Poly, GMM, and validated KDE. For nested Log-Poly model, we used sub-models of degrees $d \in \{5, 10, 15, 20\}$. For GMM, we considered a hierarchical model of GMM's with $k \in \{2, 5, 10, 20\}$ components as its sub-models and selected the best sub-model using the validation set. For each sub-model, we first initialized the parameters using a k-means with at most 100 iterations and then we ran the EM algorithm for 100 iterations. For KDE we used Gaussian kernels. Following [6], we set the kernel width of the Gaussian KDE equal to $1/\sqrt{n_{train}}$. However, to achieve a fairer comparison, we also tested validated KDE (VKDE). We used the validation set to select the best kernel width. Table 2 shows the accuracy of KDE, VKDE, GMM, single Gaussian, and nested Log-Poly for 5 different datasets. As can be seen in this table, the classification performance of naive Bayes using nested Log-Poly as its underlying distributed density estimator is highly competitive compared to other density estimators. However, for a better comparison, we should take into account the amount of communication each method needs to achieve this accuracy. In the next section, we measure the exact amount of communication for each method to obtain the corresponding accuracy.

From the large diversity in the accuracy of different methods on the *Gas-Sensors* dataset given in Table 2, we infer two facts: there are some challenging one dimensional distributions in this dataset, and naive Bayes classifier needs precise estimations of these distributions to provide more accurate results. In order to measure the ability of Log-Poly to estimate these more challenging density functions, we also tested naive Bayes classification with Log-Poly functions of degree 100 (we also needed to increase our computational precision from 50 to 200). As anticipated, it provided a better result than nested Log-Poly of degree 20. It achieved an accuracy of 85.26% (std=0.31). Figure 5 shows the results of Log-Poly with degrees 20 and 100 on the 6-th dimension of the three different classes of the *Gas-Sensors* dataset. It depicts how Log-Poly of degree 100 can capture more details of the density functions than Log-Poly of degree 20.

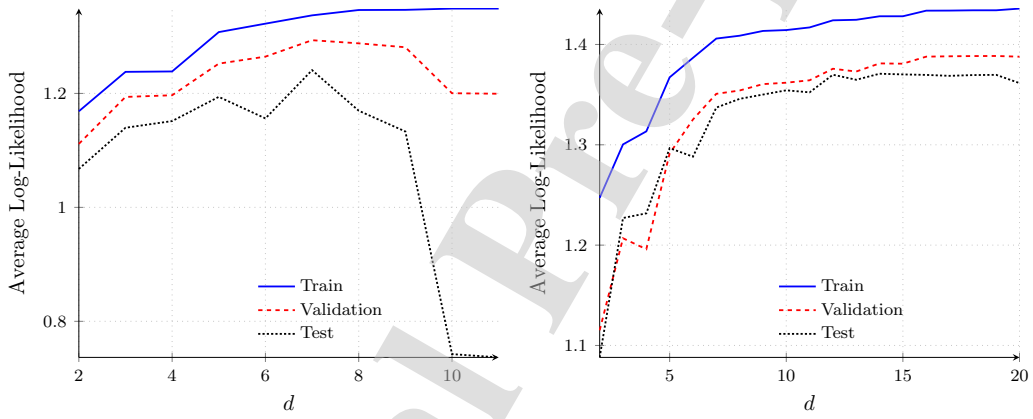
6.4. Communication Load

In this section, we compare the communication load of different mentioned density estimators (KDE, GMM, and nested Log-Poly) to train a naive Bayes



(a) (b)

Figure 3: Log-Poly density estimation vs Gaussian kernel density estimation using 2000 samples from a gamma($k = 2, \theta = 0.5$) distribution.



(a) The training set of size 500 observations (b) The training set of size 3000 observations

Figure 4: Average log-likelihood of the training, validation and test data, for the estimated Log-Poly sub-models of different degrees.

Table 2: Mean and std of the accuracy of the naive Bayes classifier using different density estimation methods.

Device Name	Data Size	KDE	VKDE	GMM	nested Log-Poly	Gaussian
Danmini Doorbell	1018297	88.39 (0.42)	87.97 (0.75)	88.93 (0.28)	87.89 (0.18)	86.75 (30)
Ecobee Thermostat	835875	85.49 (0.34)	84.94 (1.32)	86.45 (0.45)	85.53 (0.21)	71.32 (0.62)
Phil. B120N10 Baby Mon.	1098676	86.57 (0.19)	89.18 (0.42)	89.25 (0.26)	88.81 (0.8)	84.64 (3.71)
Prov. PT 737E Sec. Cam.	828259	83.98 (0.30)	85.49 (0.55)	84.93 (0.25)	85.00 (0.24)	62.29 (0.65)
Prov. PT 838 Sec. Cam.	836890	86.30 (0.22)	86.36 (0.75)	86.47 (0.97)	85.38 (0.50)	62.90 (0.63)
MAGIC	19020	75.74% (0.74)	77.39% (0.67)	77.80% (0.69)	76.31% (0.71)	72.69% (0.75)
Gas-Sensors	169153	84.52% (0.27)	91.99% (0.30)	87.14% (0.36)	79.33% (0.29)	71.23% (0.41)

classifier. For such comparison, we implemented a parallel python code to simulate the distributed setting. The central node is represented by a python process where the only way for it to access the data is through several independent python processes that represent the client nodes. The data is partitioned horizontally and randomly divided among the client processes. The central process communicates with the client nodes by transferring messages through the network. We implemented the message passings by socket programming and we will report both the number of variables and the number of bytes transferred in each experiment. For the number of variables, we simply report the number of variables that the processes send to each other, whether they are integer or real numbers. For monitoring how many bytes are exchanged between the processes, we use *tcpdump* to capture the packets and compute sum of their lengths.

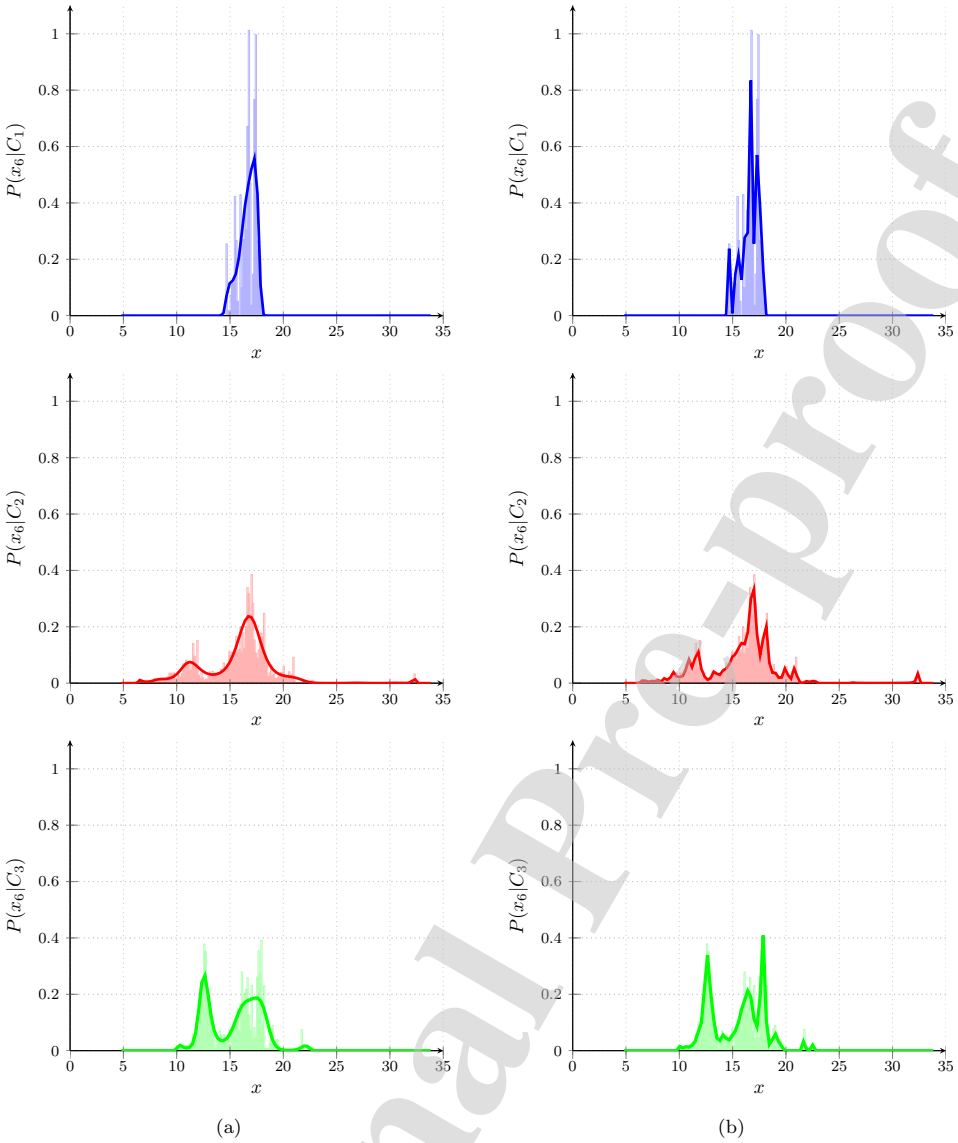
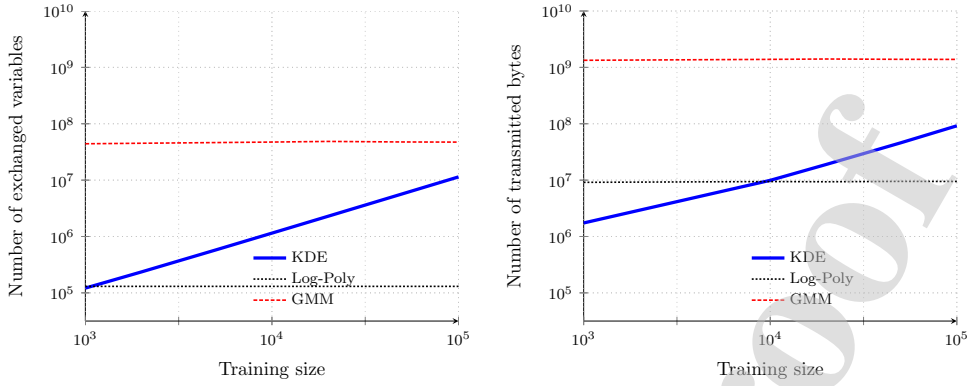


Figure 5: Estimation of the 6th dimension of the three classes of the *Gas-Sensors* dataset using Log-Poly's of degrees 20 (a) and 100 (b)

In all experiments, we run distributed naive Bayes classification on the samples gathered from *Danmini Doorbell* device in the IoT dataset. We study the effect of the training size and the number of the client nodes on the communication load for different density estimators. In our experiments, we select the degree of Log-Poly by sub-model selection among sub-models of degrees 5, 10, 15 and 20 (note that the sufficient statistics of all the sub-models can be fetched from the clients in one round of communication). For evaluating each sub-model, we consider a portion of data in each client as the validation data. The central node just needs to have the sufficient statistic of the validation set to compute the likelihood of each sub-model on the validation set. Similarly, for GMM, we select the number of components among mixtures of 2, 5, 10 and 20 components. After training each GMM sub-model, the central node sends the trained parameters of the sub-model to each client and receives back the likelihood of the validation set stored in that client. In addition, all GMM models are trained through 100 EM iterations in the experiments of this section.

Figure 6a shows the number of communicated variables of each method for different training sizes in logarithmic scale. We used 2 client nodes for this experiment. As figure 6a shows, the amount of communication needed by nested Log-Poly and GMM does not depend on the size of the training data while the amount of communication that KDE needs is proportional to the number of training samples. We can see in this figure that nested Log-Poly communicates a significantly fewer number of variables in comparison



(a) Number of transmitted variables

(b) Number of transmitted bytes

Figure 6: Communication load of training a naive Bayes classifier using different density estimators on different number of training samples from the *Danmini Doorbell* data. Data is partitioned among 3 client nodes.

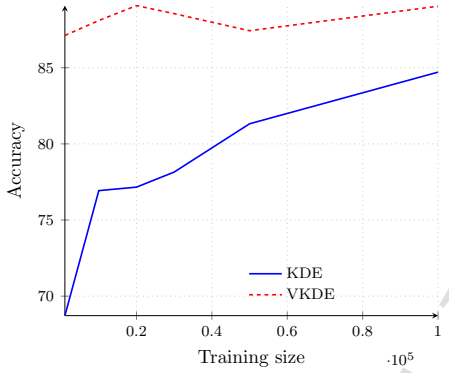


Figure 7: Accuracy of naive Bayes classifier with KDE and VKDE using different number of training sample from the *Danmini Doorbell* data.

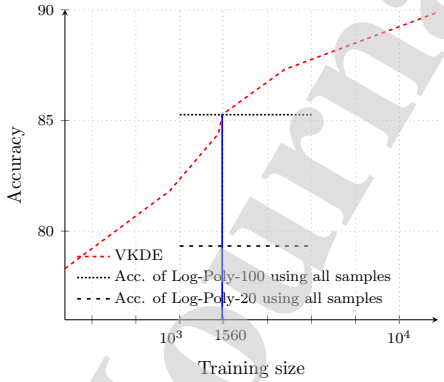


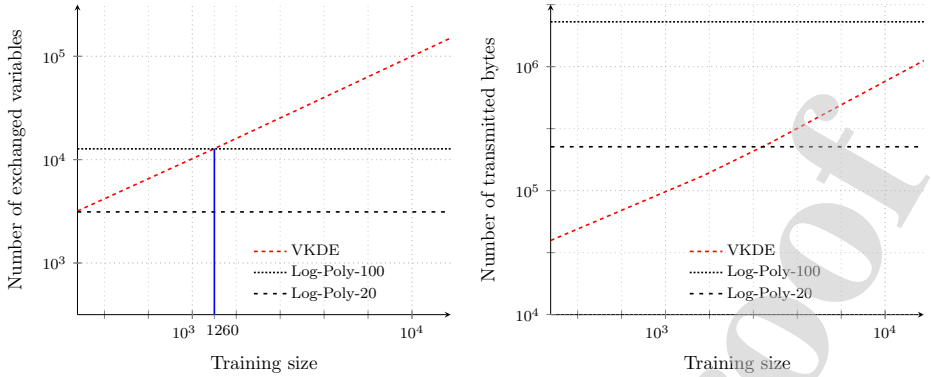
Figure 8: Accuracy of naive Bayes classifier with VKDE using different number of training samples from the *Gas-Sensors* dataset, and accuracy of naive Bayes using Log-Poly of degrees 20 and 100 on the whole samples.

with GMM for any size of training data. Note that although GMM does not need to collect all the data, it still needs to communicate a large number of variables due to its iterative manner. We see that nested Log-Poly also communicates a fewer number of variables than KDE, especially for large training sizes.

Comparing the results from Table 2 with Figure 6a, it can be inferred that nested Log-Poly proves to be highly competitive compared to other distributed methods in terms of the classification accuracy, while the communication load needed in nested Log-Poly is significantly lower than other methods. This makes nested Log-Poly an appealing choice for distributed classification tasks that use naive Bayes classifiers.

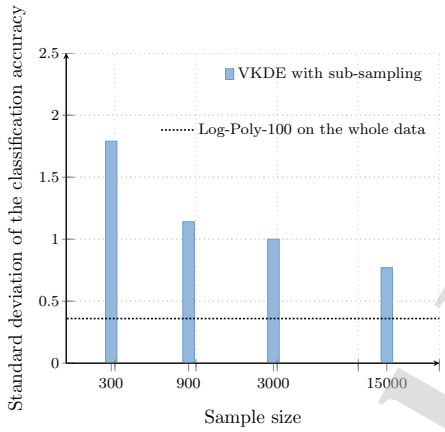
Figure 6b shows the number of bytes transmitted by each method. As anticipated, it can be seen that the communication load of each method in bytes is proportional to the number of variables transmitted by that method (Figure 6a). However, the scaling factors for different methods are not equal. By comparing Figures 6a and 6b, we can estimate how many bytes have been used for transmitting each variable on average. That number is approximately 10 bytes for KDE, 30 bytes for GMM and 70 bytes for nested Log-Poly. The reason that nested Log-Poly transmits more bytes for each variable compared to the other methods is its need for high precision computation. We have used *mpmath* python package in the implementation of Log-Poly which stores each variable as a type called *mpf*. This data type uses more bytes than python float variables which are used in implementing KDE and GMM. Also, note that GMM uses more bytes per variable in comparison with KDE. The reason is that GMM runs lots of communication rounds and each round causes some extra communication overhead, while in KDE the central node collects all its required data in one round of communication.

As described beforehand, Figures 6a and 6b show that nested Log-Poly uses less communication than KDE and VKDE on large datasets. The reason is that the communication load of KDE and VKDE depends on the number of samples. For a fair compari-



(a) Number of transmitted variables (b) Number of transmitted bytes

Figure 9: Communication load of training a naive Bayes classifier using different density estimators on different number of training samples from the *Gas-Sensors* dataset. Data is partitioned among 3 client nodes.



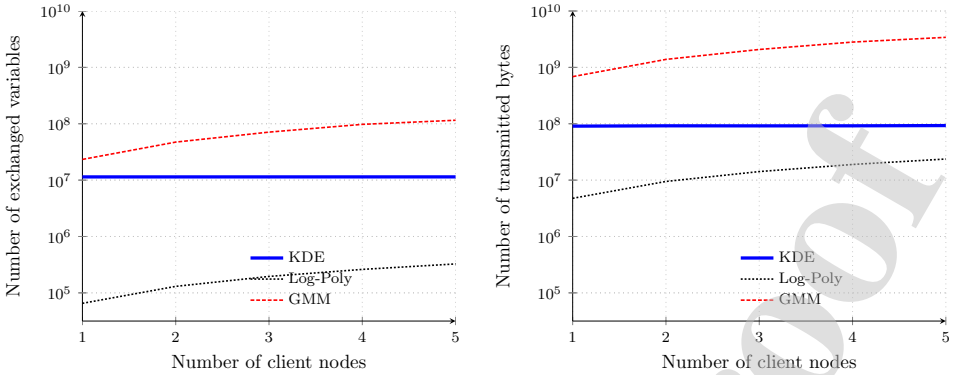
(a)

Figure 10: Variance of the accuracy of the naive Bayes classifier using VKDE with different sample sizes from the *Gas-Sensors* dataset, versus the variance of the accuracy of naive Bayes classifier using a single Log-Poly of degree 100 on the whole dataset.

son, we also need to check how accurate these methods perform on a subset of samples. Figure 7 shows the classification accuracy using KDE and VKDE on data from *Danmini Doorbell* device using different sample sizes. Notice that these two methods use exactly the same communications and their difference is just in their kernel widths. Figure 7 depicts that the accuracy of naive Bayes classification with KDE drastically reduces by sub-sampling the training data. However, by using VKDE, it can obtain the same accuracy by using a small subset of the training data. Therefore, in fact, on this dataset, VKDE can achieve the same accuracy with less communication than nested Log-Poly. It shows that the density functions of different classes of this dataset are either simple enough that can be estimated from a small set of samples, or separated from each other such that naive Bayes can perform well with a rough estimation of them from a small set of samples. To investigate a more challenging case, we used *Gas-Sensors* dataset. As we discussed in Section 6.3, the results in Table 2 demonstrate that estimating the density functions precisely is a challenge in this dataset. Figure 8 shows the accuracy of VKDE using different sample sizes on this dataset. Also, Figure 9 shows the communication load of VKDE, nested Log-Poly of degree 20 (while computing the sufficient statistics up to 50 decimal digits) and nested Log-Poly of degree 100 (while computing the sufficient statistics up to 200 decimal digits) using different number of samples from the *Gas-Sensors* dataset (when partitioned into two client nodes). Noting Figure 8 and Figure 9a simultaneously indicates that if we consider the number of transferred variables as the measure of communication load, some scenarios can be considered in which Log-Poly of degree 100 outperforms VKDE. For example, if both models are restricted to transmit at most 15000 variables, VKDE can use at most 1500 samples which results in a low accuracy than Log-Poly-100, while transmitting more variables than it. But Figure 9b shows that when the number of transferred bytes is considered as the measure of communication load, VKDE with sub-sampling outperforms Log-Poly in accuracy, while it uses less communication than Log-Poly. In fact, Log-Poly suffers from its need to high precision communication.

Moreover, it is worth mentioning that running VKDE on a small sample of the data may increase the variance of classification accuracy, in comparison with using Log-Poly or VKDE on the entire data. Figure 10a demonstrates the effect of the sample size used by VKDE on the naive Bayes classification accuracy on the *Gas-Sensors* dataset.

Although the communication loads of GMM and nested Log-Poly do not depend on the number of training samples, they are directly related to the number of clients that the data is partitioned over. This dependency is due to the fact that both methods need to get certain statistics from the data stored in each client. Moreover, GMM also needs to send the current value



(a) Number of transmitted variables

(b) Number of transmitted bytes

Figure 11: The effect of the number of client nodes on the communication load of training a naive Bayes classifier (on the *Danmini Doorbell* data), using different density estimators.

of its parameters in each iteration to all the clients. Figure 11a shows the number of variables transmitted by nested Log-Poly, GMM, and KDE for different number of client nodes on a semi-logarithmic scale. In this experiment, we used 10^6 training samples. As anticipated, the figure shows that the number of exchanged variables does not depend on the number of client nodes in KDE, but has a direct relation with the number of clients in GMM and nested Log-Poly. Figure 11b shows the same diagram for the number of bytes transmitted by each method. This property of KDE that the communication load is independent of the number of clients, could be beneficial when the number of clients is large. However, this experiment shows that for smaller ratios of training size to the number of clients, nested Log-Poly can perform better than KDE and GMM in both number of transmitted bytes and number of exchanged variables.

7. Conclusion and future directions

We have proposed *nested Log-Poly*, a communicationally efficient model for distributed density estimation in naive Bayes classification. The properties of its *Log-Poly* sub-models make it a suitable model for one-dimensional density estimations in a distributed setting. We have provided a theoretical analysis of the capability of Log-Poly to fit a wide range of probability density functions. We have also showed experimentally that nested Log-Poly competes with the state of the art estimators such as GMM and KDE in accuracy, while it needs to transfer relatively fewer variables between the client nodes and the central node. This is made possible through the use of high precision to compute, store, and transfer the statistics, which causes an extra communication load. Moreover, VKDE with sub-sampling is an alternative which provides competitive results to Log-Poly with a smaller byte transmission. A useful extension to this work would be to reduce the transmission cost of high precision communication. An efficient extension of Log-Poly to multi-dimensional density functions would be another worthy direction for the future studies.

8. Acknowledgement

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions, which significantly improved the paper.

References

[1] B. J. Kim, A classifier for big data, in: Convergence and Hybrid Information Technology, Springer, 2012, pp. 505–512.

[2] R. Bekkerman, M. Bilenko, J. Langford, Scaling up machine learning: Parallel and distributed approaches, Cambridge University Press, 2011.

[3] M. Gao, Q. Wang, M. T. Arafin, Y. Lyu, G. Qu, Approximate computing for low power and security in the internet of things, Computer 50 (6) (2017) 27–34.

[4] U. Raza, P. Kulkarni, M. Sooriyabandara, Low power wide area networks: An overview, IEEE Communications Surveys & Tutorials 19 (2) (2017) 855–873.

[5] K. Yang, D. Blaauw, D. Sylvester, Hardware designs for security in ultra-low-power iot systems: An overview and survey, IEEE Micro 37 (6) (2017) 72–89.

[6] G. H. John, P. Langley, Estimating continuous distributions in bayesian classifiers, in: Proceedings of the Eleventh conference on Uncertainty in artificial intelligence, Morgan Kaufmann Publishers Inc., 1995, pp. 338–345.

[7] D. Caragea, A. Silvescu, V. Honavar, A framework for learning from distributed data using sufficient statistics and its application to learning decision trees, International Journal of Hybrid Intelligent Systems 1 (1-2) (2004) 80–89.

[8] S. Boyd, L. Vandenberghe, Convex optimization, Cambridge university press, 2004.

[9] C. M. Bishop, Pattern recognition and machine learning, springer, 2006.

[10] L. Koralov, Y. G. Sinai, Theory of probability and random processes, Springer Science & Business Media, 2007.

[11] E. Janofsky, Exponential series approaches for nonparametric graphical models, arXiv preprint arXiv:1506.03537 (2015).

[12] N. Piatkowski, Exponential families on resource-constrained systems., Ph.D. thesis, Technical University of Dortmund, Germany (2018).

[13] T. Broderick, N. Boyd, A. Wibisono, A. C. Wilson, M. I. Jordan, Streaming variational bayes, in: Advances in Neural Information Processing Systems, 2013, pp. 1727–1735.
[14] C. Giannella, H. Dutta, S. Mukherjee, H. Kargupta, Efficient kernel density estimation over distributed data, 9th international workshop on high performance and distributed mining, SIAM international conference on data mining (2006).
[15] Y. Hu, J.-G. Lou, H. Chen, J. Li, Distributed density estimation using non-parametric statistics, in: Distributed Computing Systems, 2007. ICDCS'07. 27th International Conference on, IEEE, 2007, pp. 28–28.
[16] C. E. Rasmussen, The infinite gaussian mixture model., in: NIPS, Vol. 12, 1999, pp. 554–560.
[17] D. Görür, C. E. Rasmussen, Dirichlet process gaussian mixture models: Choice of the base distribution, Journal of Computer Science and Technology 25 (4) (2010) 653–664.
[18] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, Distributed optimization and statistical learning via the alternating direction method of multipliers, Foundations and Trends® in Machine Learning 3 (1) (2011) 1–122.
[19] Q. Liu, A. T. Ihler, Distributed estimation, information loss and exponential families, in: Advances in Neural Information Processing Systems, 2014, pp. 1098–1106.
[20] R. D. Nowak, Distributed em algorithms for density estimation and clustering in sensor networks, Signal Processing, IEEE Transactions on 51 (8) (2003) 2245–2253.
[21] S. Merugu, J. Ghosh, Privacy-preserving distributed clustering using generative models, in: Data Mining, 2003. ICDM 2003. Third IEEE International Conference on, IEEE, 2003, pp. 211–218.
[22] S. Merugu, Distributed learning using generative models, University of Texas at Austin, 2006.
[23] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, A. Shabtai, D. Breitenbacher, Y. Elovici, N-baiot—network-based detection of iot botnet attacks using deep autoencoders, IEEE Pervasive Computing 17 (3) (2018) 12–22.
[24] P. Savicky, UCI machine learning repository (2007). URL https://archive.ics.uci.edu/ml/datasets/MAGIC+Gamma+Telescope
[25] R. Huerta, T. Mosqueiro, J. Fonollosa, N. F. Rulkov, I. Rodriguez-Lujan, Online decorrelation of humidity and temperature in chemical sensors for continuous monitoring, Chemometrics and Intelligent Laboratory Systems 157 (2016) 169–176.
[26] A. C. Davison, Statistical models, Vol. 11, Cambridge University Press, 2003.

***Highlights (for review)**

Highlights

1. Introducing a new distributed learning algorithm for density estimation
2. Reducing the communication costs
3. Theoretical analysis for the efficiency of the proposed method
4. Application for large-scale classification tasks

*Credit Author Statement

CRediT author statement

Ahmad Khajenezhad: Conceptualization , Methodology, Software, Formal analysis, Writing - Review & Editing

Mohammad Ali Bashiri: Software, Writing - Review & Editing

Hamid Beigy: Conceptualization , Methodology, Writing - Review & Editing

Declaration of interests

■ The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

□The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: