International Journal of
Science and Engineering Investigations vol. 7, issue 79, August 2018
ISSN: 2251-8843

# A Common Approach for Learning Classifiers from Distributed Data

Karlen Mkrtchyan
Ph.D. Student, Institute for Informatics and Automation Problems (IIAP), Armenia
(mkkarlen92@gmail.com)

***Abstract*-** In this paper we will discuss the problem of learning from horizontally distributed data, we will try to add theoretical background for some class of learning algorithms. We will apply obtained results on perceptron algorithm, and show that some class of learning algorithms could be learning in the exact same way as in the case of centralized data.

***Keywords*-** *Machine Learning, Sufficient Statistics, Distributed Computation, Distributed Machine Learning, Perceptron*

## I. INTRODUCTION

Learning from distributed data is not mere scientific problem; moreover, this problem comes from real production use cases. Learning in WSN (Wireless sensor network) [1] is one example where learning in distributed system is forced by environment. More generally IoT [2] infrastructure itself is meant to be distributed, and not only learning, but also all the computations should be implemented in distributed fashion, to cover the needs of infrastructure and in some cases the needs of end users. Learning from distributed system is instance of distributed computation; consequently it does inherit common problems known in distributed computation. Some architecture designs requires privacy preservation [3], [4] fault tolerance [5], encrypted data transfer [6], low communication costs [7] and etc. These are just few of the requirements, which we will try examine in the scope of distributed learning, however, some of them are not in the scope of our discussion, and they are properly addressed in other distributed computing problems [8], [9], [10].

The paper is structured as follows. In section 2 we address the problem of distributed computing more formally. In section 3 we provide theoretical results on learning from distributed data. In Section 4, as an example we apply the results on Perceptron algorithm [11].

## II. THE PROBLEM OF LEARNING FROM DISTRIBUTED DATA

Let's suppose we are given data sources $D_1, D_2, \dots D_k \subset R^n$, drawn from some unknown distribution, which are labeled examples reprinting some predefined event which is observed. Let's assume $L$ is a learning algorithm, and for the input $\bigcup_{i=1}^{k} D_i$ it outputs hypothesis $h$ that predicts $y$ label of future unlabeled examples $x = (x_1, x_2, \dots x_n)$. If take $\bigcup_{i=1}^{k} D_i$ as one data source available in one machine, then it is conventional learning case or centralized learning problem. Now let's assume $D_1, D_2, \dots D_k$ are in different machines. Being in different computation centers brings with it restrictions on the system, more specifically, restrictions on how data centers communicate and what can be broadcasted from one data center to another. We will define the set of restrictions with $Z$. $Z$ is addressed as set of restrictions or set of constraints in literature.

To sum up, we can now define the problem of learning from distributed data as follows.

**Definition 2.1**: We are given data sources $D_1, D_2, \dots D_k$, a set of restrictions Z, a hypothesis set H, and performance evaluation criterion P. Learner L learns from $D_1, D_2, \dots D_k$ that optimizes P , does not violate conditions from Z and outputs hypothesis $h$.

From the definition it is easy to see, that, in the case, when $Z = \emptyset \; and \; k = 1$ we have the centralized learning problem. From this definition centralized learning is the particular case of distributed learning. In this paper we only will discuss the problem of learning with horizontally distributed. Although vertically portioned data also has its interest, but we will limit ourselves by only horizontal partitioning due to its much more practical use.

Our purpose is to construct an algorithm that would be exactly the same as in the centralized learning case. Formally exactness can be defined in the following way.

**Definition 2.1**: Algorithm L, which is employed in $D_1, D_2, \dots D_k$ distributed system, is said to be the same as $L_C$ learned from $\bigcup_{i=1}^{k} D_i$, if they output the same hypothesis $h$ with exactly same classification error.

To complete the definition it is important to cover some specific class of algorisms that make random choices from learning data set. For example Decision trees make random choice when splitting attributes that have the same Gini index. We assume that in both cases we do exactly the same random sampling. This restriction is needed to ensure the condition of exactness addressed above is correct independently on random choices.

The set of constrains $Z$ is the main difficulty added by distributed system. We have defined $Z$ but we didn't mention what kind of elements it consist of and where they come from. The set in fact, is always predefined, and it comes from practical use cases. To make it clear, about the nature of $Z$ we will give some common and practical examples of the elements of $Z$. One of most common examples is restriction on raw data transfer. That means, no data center can directly read the data of other participants, but may execute remote code, to obtain aggregated results. Some data centers have restriction on traffic and certain amount of data transfer is allowed, for example 5Mb per hour. Some constraints are designed to preserve data privacy, hence raw data transfer is not allowed. There may be restrictions on executing remote code, because some aggregation results can reveal data structure more or less.

These are very common and very general type of restrictions. In practical use cases there are much more restrictions on technical level. And in fact the restrictions are provided in the form of SDKs or public APIs which clearly define the communication interface among participants. And main problem is to meet all requirements of the distributed system, while integrating centralized learning algorithm to decentralized system.

## III. LEARNING FROM DISTRIBUTED DATA

There are many methods for learning from distributed data, and in fact, for any set of restrictions $Z$, can be found method which approximates the needed solution enough to satisfy practical needs. The most common and practically useful algorithms, such as SVM, Decision Trees, Boosting have adapted versions for distributed environments, which are practically useful.

For instance in [12] given distributed SVM algorithm called DSVM. They took as $Z$ privacy preservation, and defined the network topology where each participant may communicate with its neighbors. In essence the algorithm is incremental and sequential algorithm. At each iteration, every node solves SVM problem locally then it transfers statistics to its neighbors. They solve the problem with ADMoM [13]. It is proven there that the algorithm converges to the centralized version of SVM. Although the algorithm has sequential nature, and it does not output the exact same hypothesis as in centralized version, nevertheless it shows good practical results, suitable for many applications.

There are many algorithms for Boosting, which one of the most popular algorithms for learning classifiers. Back in 2001 A. Lazarevic, Z. Obradovic [14] purposed a modified version of boosting algorithm that could learn from distributed data. The essence of algorithm is to look at distributed data as one data center, and imitate the method used in centralized version. They define global distribution of data, by using the local distributions, and for each iteration, they broadcast weak learner to every participant, they update their weights locally and begin new iteration with new local and global distributions. There many modifications of this approach, but the cores are the same.

There are many approximations for Decision trees as well. Most used among them are Random Forests [15], which differ the methods of SVM and Boosting. In fact [13] and [14] use data aggregation method, [15] can be classified as model aggregation method.

Surely there exists more methods, even better, to solve particular problem, but we've seen that these methods don't share common approach or method for solving the problem; in fact they solve the problem by robust approach to specific use case, which is hard to generalize for other cases. Thus our aim is to fill this gap from more general perspective and provide learning strategy that could be applied to specific algorithm. Firstly, let's make some common definitions, for covering notions used later in this paper.

**Definition 3.1:** For given parameter $a$ statistic $s(D)$ is called sufficient if it gives enough information about $a$ from dataset $D$ for a task $T$.

In fact it is evident that the whole set D is sufficient statistics itself, and this the trivial of sufficient statistics. Nevertheless, more interest has $s(D)$ that is minimal, from performance, time, and privacy perspectives. To make it more clear of sufficient statistics let's review an practical example from leaning algorithms. For instance CART decision tree uses information gain, and with the help of that in each step chooses splitting attribute. For CART information gain is the sufficient statistics. For Support Vector Machines (SVM) the final solution depends on vectors that lie on marginal hyper planes. These vectors represent sufficient statistics for SVM. In fact almost all learning algorithms work this way. In the first phase they all have information extraction that is they analyze data, and according to some predefined algorithm calculate sufficient statistics, by using training examples. These steps can be observed by Neural Networks, Naïve Bayes, SVM, C4.5, ID3 and many other popular algorithms. In other words, we may say that $s(D)$ is sufficient statistics for learning hypothesis $h$, when there algorithms L, which accepts $s(D)$ and outputs $h$. This must be reviewed as general explanation because; some algorithms may use $s(D)$ with batches, and converge iteratively, which is observed for example by neural networks, on the other hand, SVM uses all data to retrieve support vectors. But in both cases the notion of sufficient statistics is the same in essence.

The stated above means in fact sufficient statistics is what we get from information extraction, and it is sufficient for learner to produce hypothesis. By using this fact, we can use sufficient statistics to scale centralized algorithms to distributed versions. Let's suppose we are given distributed system with $K$ participants, each one has its dataset $D_i$ appropriately. Our aim to construct general strategy for learning exact same hypothesis with algorithm $L$, as in centralized case. For this purpose, we will use the notions of information extraction and hypothesis generation to separate learning process. What we need in distributed environment is to divide information extraction. If we had centralized data we would extract data with calculations we needed, because we may select any data we want. In distributed environment it is not the case. The same calculation result, which is statistics in essence, should be

obtained from $D_1, D_2, \dots D_k$ by separating calculations of statistics.

**Observation 3.1**: For any learning algorithm $L$, that uses sufficient statistics for generating hypothesis $h$ in centralized model, there is transformation of L algorithm in distributed system, in a way, that the generated results are exactly the same.

Truthiness of the statement could be inferred by constructive example. Suppose we have D data in centralized model, and that $L$ algorithm uses $I$ iterations to generate $h$. As $L$ learns by using sufficient statistics, it means at each iteration it calculates $s(d_i)$, $d_i \subset D$ statistics. No let's suppose $D$ in distributed environment, that is now we have $D_1, D_2, \dots D_k$ and $\bigcup_{i=1}^{k} D_i = D$. It is evident that $d_i \subset \bigcup_{i=1}^{k} D_i$, which means $d_i$ could be selected for $D_i$ part by part, then send all data to predefined data center and calculate $s(d_i)$. Result will obviously be the same as in centralized case. So the basic case is covered, and it is possible to use this method in distributed environment. Nevertheless, it is not as simple as the example in the case when we consider $Z$, set of constrains. The constructive method we have used implies that raw data transfer is allowed, which is not the case in many cases. It means that we need method not to send raw data, but send only sufficient statistics, and somehow combine the results to obtain $s(d_i)$. In fact the notation $s(d_i)$ may be confusing in terms of $d_i$. The algorithms usually make statistical queries, and obtain the result of the query as a number or vector and etc. We used $s(d_i)$ notation to demonstare that in $i-th$ iteration the statistical query q, made by algorithm $L$, used $d_i$ data portion to compute sufficient statistics for $q_i$. As we have seen the main problem is access to data, which could be restricted in different ways. Usually data access can be two types parallel and sequential. In the case of sequential access statistics computed for $D_i$ is transferred to $D_{i+1}$ and so on in the final stage all statistics is combined to compute sufficient statistics. In the case of parallel access data centers could be accessed independent from each other. As we can see main task is to decompose one on statistical query into multiple queries and then integrate those results to obtain the same result as in centralized case. That is statistical query $q$ should be decomposed into $q_1, q_2, \dots q_k$ and combine them by some $C$ combiner function. Formally the statement is be the following.

**Lemma 3.1**: For the distributed learning algorithm $L$ the sufficient condition for learning from distributed data is the following. If for any $q$ there exists $C$ combiner function, $q_1, q_2, \dots q_k$ such that $q(D) = C(q_1(D_1), \dots q\_k(D_k))$

The proof for this lemma is evident, because as we have $C$ and $q_1, q_2, \dots q_k$, it is the same calculation as in centralized case, which means we will get the same sufficient statistics, consequently the same $h$.

**Lemma 3.2**: If the statistical query $q$ is decomposable into $q_1, q_2, \dots q_k$ independent queries, then there exists function $C$ such that $q(D) = C(q_1(D_1), \dots q\_k(D_k))$.

This lemma is also evident, because if we can decompose $q$ into independent parts that means q operates in such manner. Generally said the lemma says that the existence of $q_1, q_2, \dots q_k$ the same as existence C. Indeed, to decompose query, means that there is way to go back from $q_1, q_2, \dots q_k$ to $q$, otherwise it is not decomposition. The transformation function $q_1, q_2, \dots q_k \Rightarrow q$ is the function $C$ we need. The vice-versa holds true as well. When we have C, for which $q(D) = C(q_1(D_1), \dots q\_k(D_k))$ then we have also $q_1, q_2, \dots q_k$. The way we find $q_1, q_2, \dots q_k$ is out of the scope of this paper. It is more of problem Factorization [15],[16] problem properly addressed in literature.

## IV. Linear Classification in Distributed System with Perceptron

Firstly, let's make it clear, what is sufficient statistics for perceptron algorithm. As it is known, the perceptron algorithm does classification with weight vector $w$, which is learned iteratively. At every step algorithm updates $w$ according to current data sample $(x_i, y_i)$ randomly selected from training examples $D$. It means that at iteration $i$ we have $w_i(D)$ which is constructed on $w_{i-1}(D)$ and so on by induction, It means $w_{i+1}(D)$ depends on $w_i(D)$ and some $(x_j, y_j)$. For sufficient statistics we can write the following expression $w_{i+1}(D) \leftarrow s((x_j, y_j), w_i(D))$, and it is obvious that $w(D)$ is computed when the iteration is finished on whole data set.

Now assume we are given data sets $D_1, D_2, \dots D_k$ in distributed system. According to statement above all we need is to calculate weight updates at each iteration. As we have mentioned earlier, by addressing to random choice of data we will assume that in both cases we will sample in the same way. To make it clear of exact way we could do that, we will present the following simple algorithm. Firstly, lets order data sets $D_1, D_2, \dots D_k$ in some way, that their data indexes will be fixed with their data sizes. In centralized version we will index data as well. Let's denote $M$ the number of all data samples. This means that we can generate random number from 1 to $M$ with uniform distribution and then pick data sample according to that index. It is necessary to note that the same index cannot be chosen more than once. It is evident that chosen random number $r$ could be mapped to distributed data indexes, which evident will be the same sampling. After this we initialize weight vector $w$ with random values, let's say with zero vector. Then at each iteration $t$ we choose random number $r$, according to $r$ we will chose the participants index. Then weight vector $w$ is broadcasted to participant, then vector is updated and send back to computation center. The iteration continues until no one updates weight vector during whole iteration.

**Theorem 4.1**: The algorithm of distributed perceptron is exact.

The exactness of the algorithm in fact is consequence of the general strategy. In terms of perceptron algorithm, we calculate exactly the same weight vector, because we iterate through data in exactly the same way, thus, the final vector is the same. Below is given the pseudo code for the algorithm.

Init $w = (0,0 \dots ,0,)$

For all $D_i, i = 1,2..K$

Generate $r$ randomly

Update $w$ according to data samples selected from $D_{ij}$

Send back $w$ to computation center.

Stop when for all $i$, $w$ is not updated in one pass

**Theorem 4.2** The algorithm of distributed perceptron has exactly the same time complexity as centralized algorithm.

This is also evident because we pass though data exactly the same way, and perceptron convergence method is the same. However, if we add some restrictions on data set, we may even boost performance in distributed system. Suppose that we now that $D_1, D_2, \dots D_k$ is linearly separable, which obviously means that $D_i$ is linearly separable as well. Consequently, we may construct hyper planes in $D_i$, $i = 1,2..K$ separately. It means we can separate $D_i$ into $D_{i1}$ and $D_{i2}$ sets according to their hyper plane. Let's denote $conv(D_{i1})$ and $conv(D_{i2})$ convex hulls of $D_{i1}$ and $D_{i2}$ for all $i$. The construction of convex hull $conv(S)$ has complexity $||S||$, and it can be done by Quickhull [17]. The cost we pay for convex hulls is $|| D_{i1}|| + || D_{i2}||$ for all $i$. In distributed algorithm we change $D_i$ with $v(D_{i1}) \cup conv(D_{i2})$. The rationale for this can be inferred easily, because hyper planes depend on marginal points, and the points inside convex hull make no sense from perspective of separating hyper plane. This simple change results in significant boost in communication complexity, $because\ ||S|| >> ||conv(S)||$ for large enough data sets. Thus by this we have minimized communication complexity and, by not using inner points of data sets $D_{il}$, $l = 1,2$ we reduce data privacy violation risk. We don't transfer raw data, nevertheless, not using essential parts of data sets in computation results in even better privacy preservation.

## V. CONCLUSION

The strategy described in this paper tries to approximate a very general approach of distributed data classification. As we have seen in the method there is one key condition, that is, resulted algorithm should be exactly the same as in centralized version. This condition is not merely a demand on accuracy of classification. By having exactly the same hypothesis $h$ as in centralized version we make bridge between distributed data classification theory and conventional data classification theories. More precisely, we are eligible to use all theoretical legacies of conventional data classification algorithms, which is much wider than of distributed data classification. The abstract nature of the method does not allow us to derive implementations for exact algorithms immediately, like perceptron, and each algorithm requires solution uniquely. This is the main problem our future work will be dedicated. The core problem is to decompose statistical queries no matter of their form. By having under consideration data privacy preservation and communication complexity. The last statement in fact is restriction on set Z. The justification of this kind of restriction comes from practical use cases.

## REFERENCES


[1] C. Buratti , A. Conti, D. Dardari ,R. Verdone, An Overview on Wireless Sensor Networks Technology and Evolution, ISSN 1424-8220 2009.

[2] M. Rana, W. Xiang, E. Wang, M. Jia IoT Infrastructure and Potential Application to Smart Grid Communications, GLOBECOM IEEE,2017

[3] Y.Brun, N.Medvidovic, Preserving Privacy in Distributed Systems, 36th IPCCC IEEE, 2017

[4] B. Fung, K. Wang, R. Chen, P. S. Yu, Privacy-preserving Data Publishing: A Survey of Recent Developments, ACM Computing Surveys (CSUR), vol. 42, no. 4, pp. 14, 2010

[5] Z. Wang, N. H. Minsky Fault Tolerance in Heterogeneous Distributed Systems, 10th IEEE ICCC, 2014

[6] R. Pranesh, M. Vigneshwaran, V. Harish, G. Manikandan A New Approach for Secure Data Transmission, International Conference on Circuit, Power and Computing Technologies (ICCPCT), 2016

[7] A. Cordoba, F. Farina, J.R. Garitagoitia, J.R.G. de Mendivil, J. Villadangos A Low Communication Cost Algorithm For Distributed Deadlock Detection and Resolution, 11th Euromicro Conference on Parallel, Distributed and Network-Based Processing, 2003

[8] Jing Wang, G. Beni Distributed Computing Problems in Cellular Robotic Systems, IEEE International Workshop on Intelligent Robots and Systems, Towards a New Frontier of Applications, 1990

[9] A. Sinha, Tapas Saini, S. V. Srikanth Distributed Computing Approach to Optimize Road Traffic Simulation, International Conference on Parallel, Distributed and Grid Computing, IEEE, 2014

[10] Y. Zhang, Y. Tian, W. Kelly, C. Fidge A Distributed Computing Framework for All-to-All Comparison Problems, IECON 40th Annual Conference of the IEEE Industrial Electronics Society, 2014

[11] F. Rosenblatt The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain, Psychological Review, Vol. 65, No. 6, 1958

[12] P. A. Forero, A. Cano, Georgios B. Giannakis Consensus-Based Distributed Support Vector Machines 11(May):1663−1707, 2010

[13] A. ADMOM The Distributed Boosting Algorithm, Knowledge Discovery and Data Mining: 311-316,2001

[14] A. Lazarevic, Z. Obradovic The Distributed Boosting Algorithm, Knowledge Discovery and Data Mining: pp 311-316, 2001

[15] S. Chung, C. Seabrook A singular Value Decomposition: Analysis of Grade Distributions. Georgia Institute of Technology; VIGRE REU: 2004

[16] Fan J, Li R. Statistical Challenges with High Dimensionality: Feature Selection in Knowledge Discovery, The Mathematics Arxive: Math. ST/0602133, 2006

[17] C. B. Barber, D. P. Dobkim, H. Huhdanpaa The Quickhull for Convex Hulls, ACM Transactions on Mathematical Softwae, vol. 23, no.2, pp. 12 ,1995