Distributed Neural Networks for Missing Big Data Imputation

Alessio Petrozziello[1, 2], Ivan Jordanov[1] and Christian Sommeregger[2]
University of Portsmouth, Portsmouth, U.K.[1], Expedia Inc., London, U.K.[2]
Alessio.Petrozziello@port.ac.uk, Ivan.Jordanov@port.ac.uk, Christian.Sommeregger@hotels.com

ABSTRACT
In this paper we investigate the use of Distributed Neural Networks for the imputation of missing values in Big Data context. The presented framework for data imputation is implemented in Spark, allowing easy imputation as an additional step to the data pre-processing pipeline. The Distributed Neural Networks model is using Mini-batch Stochastic Gradient Descent, scaling well with the cluster size and minimizing the communication among the workers. The model is tested on a real-world Recommender Systems dataset, where the missing data is generally a problem for new items, as the systems ranking is usually biased towards the popular items. The model is compared with univariate (*Mean* and *Median Imputation*) and multivariate (K-Nearest Neighbours and Linear Regression) imputation techniques, and its performance is validated using prediction accuracy and speed. Furthermore, we evaluate the speedup compared to the sequential implementation of Neural Networks with Stochastic Gradient Descent.

KEY WORDS
Distributed Computation, Neural Networks, Missing Data Imputation, Big Data.

1. Introduction

A necessary step to consider before applying any machine learning technique is to deal with eventual missing data in the used dataset. Mechanisms of missing data belong to three categories [1] [2]: missing at random (MAR) the missingness may depend on observed data but not on unobserved data (in other words, the cause of missingness is considered). Missing completely at random (MCAR) — a special case of MAR, where the probability that an observation is missing is unrelated to its value or to the value of any other variable. Missing not at random (MNAR), where the missingness depends on unobserved data. The last group usually yields biased parameter estimates, while MCAR and MAR analyses yield unbiased ones (at the same time the main MCAR consequence is a loss of statistical power). Ideally, dealing with missingness requires analysis strategy that leads to least biased estimation, without losing statistical power. The problem is the contradictory nature of those criteria: using the information from the partial data in missing data samples (keeping the statistical power), while substituting the missing data samples with estimates, inevitably brings biases. Many techniques have been explored and exploited in the last few years [2] [3] and can be divided into two main categories [4] [5]: deletion methods and model-based methods. The former includes pair-wise and list-wise deletion (where the patterns with missing values are simply removed, or where, in presence of missing values, the pattern is not dropped, and its other values are still used during the analysis), the latter is divided into single imputation (not considering the correlation between the missing value and the other variables in the dataset, and imputing the data using only information of the same attribute) and multivariate imputation (where the correlation among variables in taken into account during the imputation).

When Big Data is considered, the problem of the missing data imputation is still of primary importance for the successful implementation of machine learning techniques (e.g., recommendation tasks where millions of users and thousands of items are involved). Usually, the probability of having missing data increases with the number of features in the dataset and with the number of samples, making the imputation task in big data context extremely important. However, if the missing entries are few compared to the scale of the dataset, a deletion method is applicable without losing statistical strength. On the other hand, if the number of missing values grows with the size of the dataset the imputation is necessary to preserve, or even increase, the statistical power of the data (or in general to not lose too many samples during the pre-processing stage). Unfortunately, all the imputation techniques proposed in literature [2] [3] [6] require the whole dataset to be provided for the model at imputation time, which means that adequate memory allocation is needed, making the task unfeasible for datasets composed of hundreds of features and millions of samples. Not many methods have been proposed to cope with the missing data problem in the big data field [7] due to the inherent complexity of the task (both related to time and memory constraints).

Neural Networks (NN) are state of the art machine learning (ML) approach for several different domains with recent advances in image processing [8], pattern and speech recognition [9], that involve fitting of large architectures (with thousands of weights) to large datasets (several gigabytes to few terabytes). Given the scale of these machine learning problems, training can take up to days or even weeks on a single machine using the commonly applied deterministic optimization techniques (e.g., stochastic gradient descent (SGD)) [10]. For this reason, research focussed on the distribution of machine learning

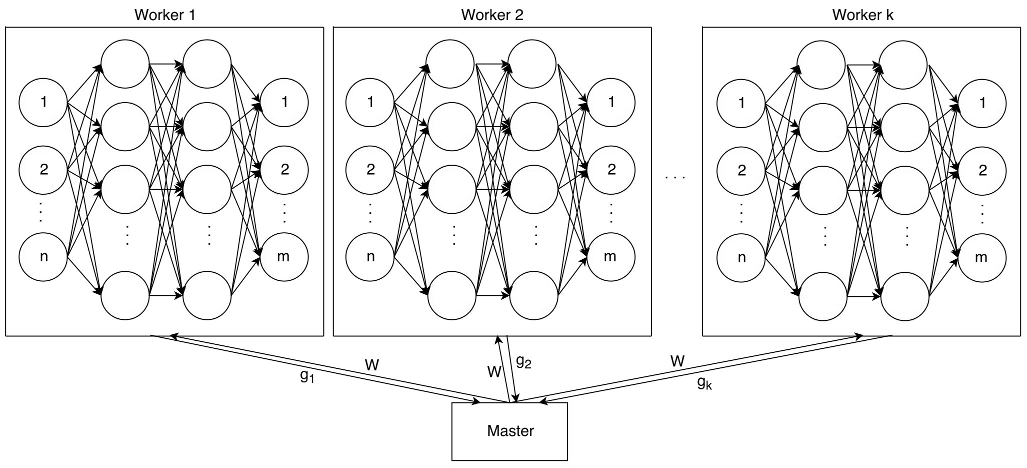


Figure 1 Distributed Neural Network architecture in Spark (Section 2.2).

algorithms across multiple machines. Different attempts have been made to speed up the training of NN using asynchronous jobs [11]. In the parameter server model [12], one master holds the latest model parameters in memory, serving the workers nodes on request. The nodes compute the gradient on a mini batch drawn from the local hard drive. The gradients are then shipped back to the server, which updates the model parameters. With the introduction of the *MapReduce* paradigm [13], different frameworks emerged to leverage resources of a cluster (e.g., *Hadoop* [14] and *Spark* [15]). These frameworks simplified the implementation of large-scale analytics and machine learning tasks (e.g., *Apache Mahout* [14]). In this paper, we propose a Distributed Neural Network Imputation (D-NNI) framework (Figure 1), leveraging the idea of mini-batch training in a distributed fashion over *Spark* to reduce training time, while making at the same time the imputation of new values possible even for larger datasets. The proposed imputation approach is tested on a real-world dataset composed of 400 thousand samples [16], and 645 features (of which 57 include missing values). The remainder of the paper is organized as follow. In Section 2 the proposed framework for Missing Data Imputation with Distributed Neural Networks is described. In Section 3 we introduce the missing data problem for properties recommendation in Online Travel Agencies, while the dataset and the imputation techniques are reported in Section 4 and Section 5 respectively. Finally, the results of the experimentation are analysed in Section 6 and the conclusion is given in Section 7.

2. Distributed Neural Networks on Spark

Apache *Spark* [15] is an open source framework for distributed computing (driver-workers paradigm) written in *Scala* [17]. It is composed of a main core package and four components: *SparkSQL, ML Spark*, *SparkStreaming* and *GraphX*. The main advantage of *Spark* is that, differently from the *Map-Reduce* paradigm it is not built on acyclic data flow model. The main data structure is the 'resilient distributed dataset' (RDD) which is distributed and stored in the fastest part of the memory (e.g., RAM), and can be easily accessed multiple times when needed for ML iterative algorithms. *ML Spark* is a distributed machine learning framework built as additional module on top of *Spark*, considered to be up to 100 times faster than the disk-based implementations of *Mahout* [14]. Many pre-processing operations, statistical procedures and ML algorithms are implemented in the package and can easily be used through the *Spark pipelines* [18].

Table 1 Neural Network Trait

```
trait NN {
   def initNetwork(nInputs:Int, nOutput:Int):Unit
   def setWeights(weights: DenseVector[Double])
   def apply(denseVector: DenseVector[Double],
        memKey:String)
   def apply(denseMatrix: DenseMatrix[Double],
        memKey:String)
   def backpropagate(loss:DenseVector[Double],
        memKey:String)
   def backpropagate(loss: DenseMatrix[Double],
        memKey:String)
   def getGradient()
   def getWeights()
   def getNumberOfWeights()
   def getLastLayerWsize()
}
```

2.1 Implementation

The implementation of the imputation stage through distributed neural networks (D-NNI) builds on *Apache Spark* [15] and the *Neuron* library [19] (a lightweight Neural Network library written in *Scala* providing all the building blocks (e.g., layers, optimizers, back-propagation, etc.), to create our own distributed version (Table 1shows the Neural Network Scala interface [19]). By building on top of *Spark*, we utilise the advantages of modern batch computational frameworks. These include the high-throughput loading and pre-processing of data and the ability to keep data in memory between operations. Furthermore, the implementation of the D-NNI as a *pipeline* stage [18] allows the imputation to be easily included in the pre-processing of any dataset. Table 2 and Table 3 show code snippets of how the imputation stage is created. In particular, Table 2 presents how the layout of the network is defined (using a sequence of layers, with the initial one representing the input, and the last one the output (which can be as large as the number of imputed features)), while Table 3 demonstrates the creation of a *Spark pipeline imputation stage*. The *NeuralNetworkImputationStage* object exposes several methods: to set the input columns to be used during the imputation (i.e., the features used during the learning phase), the target columns (the ones containing missing values), the predicted columns (the stage returns a new *dataframe* containing additional columns with the imputed values), and the additional parameters used during the imputation procedure (the NN layout as defined in Table 2, the optimizer hyper-parameters, the weights initialization procedure, the loss function to be optimized during the learning, etc).

Table 2 Example of network specification for D-NNI

```
networkLayout: Seq[layerConf] =  Seq(
   layerConf("linear", 20, 10),
   layerConf("Relu", 10, 10),
   layerConf("linear", 10, 5),
   layerConf("Relu", 5, 5),
   layerConf("linear", 5, 1),
   layerConf("sigmoid", 1, 1)
)
```

Table 3 Create a Missing Imputation Stage in a Spark pipeline, the "fit" method will train the model to predict the missing values, once the model is trained (i.e., imputationModel object), the missing values can be imputed using the "transform" method of the model.

```
val imputationModel = new NeuralNetworkImputationStage()
   .setFeatureColumnName("features")
   .setTargetColumnName("missingFeatures")
   ,setPredictColumnName("imputedFeatures")
   .setOptimizerParameters(optimizerSGD(10, 5, 0.0001)
   .setNetworkLayout(networkLayout)
   .trainWithMatrices(true)
   .setWeightInitializationType("heUniform")
   .useValidation(true)
   .setValidationPerc(0.10)
   setLossFunction("l2distance")
   .fit(data)
val newData = imputationModel.transform(data)
```

2.2 Distributed Neural Networks

In this work we use a data-parallelization scheme with synchronization and a central coordinator, labelled naïve parallelization by [20]. In every iteration, each worker node c in our cluster C computes a local gradient g_c for a batch of data b_c, then these vectors are (tree-) aggregated,

$$g = \frac{1}{C}\sum_{c} g_c(b_c) \qquad (1)$$

and sent back to the master which performs the SGD update step and broadcasts the new weights (W) to all $c \in C$ (Figure 1). In the absence of network overhead and aggregation cost this setup scales linearly with the number of worker nodes. Under more realistic conditions, the optimal number of nodes will depend on the size of b_w and the network overhead.

3. Missing Data in Online Travel Industry: A Case Study

As in all online services where user choices, items, and decisions are involved, there is a necessity of a Recommender System (RS). Online travel agencies (OTAs) provide the main service for booking hotels, apartments, and packages including many options, an adequate recommendation is what the user can mostly benefit from the system. For a RS based on users past experience (Collaborative Filtering (CF)), all users' information and interactions with the catalogue are recorded in a *user-item* matrix in order to learn behaviours and popularity trends when giving new recommendations. Since travel RS allows the user to navigate the catalogue and book hotels without being logged in the system, the use of a CF approach is not feasible, because the user-item matrix is way too sparse (and sometimes even the user information is not available at all). For this reason, a *Content-Based Filtering* (CBF) approach is considered in this work (where the recommendation is given on items similarity), focusing on one of its main problems: missing features. An important event in the market of holidays lodging is represented by the explosive popularity of *private renting* (*Vacation Rentals* (VR)) gained in the past few years [21]. and more than 400K records are expected to be added in the next year (Figure 2 shows the properties growth in the catalogue and related growth of missing data the past 15 years). The primary problem with this massive influx of new properties (1/4 of the total properties were added to the catalogue in 2017) is the lack of related historic data which results in their unfair system ranking.
New VR (e.g., apartments, condos, apart-hotels, etc.) have been introduced in the OTAs (~80K records) in the 2017, The proposed D-NNI is tested on the imputation of missing values on the OTAs problem and the results are compared with univariate imputation techniques (*Mean* and *Median Imputation* by location) and multivariate techniques implemented in Spark (e.g., K-Nearest Neighbours Imputation [22] and Linear Regression Imputation [7]). Performance results for the data imputation task and speedup are reported in Section 6.

4. Datasets

The collected data is from four sources:

- *Amenities*: includes the characteristics of all the properties (e.g., Wi-Fi, pool, TV, etc.);
- *Properties*: comprises all the relevant information in the system (e.g., property ID, name, latitude, longitude, etc.);
- *Destinations*: records of all points of interest (e.g., cities, landmarks, airports, train and metro stations, etc.);
- *Displayed Prices*: stores the historical prices shown on the online travel agency website.

Table 4 Sample of the dataset containing the features used for the properties recommendation. Missing values (values that are not available for a short history) are denoted with a '-'.

ID	nDays in Catalog	Latitude	Longitude	Am. 1	..	Am. 500	# landmarks	Dist Airport	Avg Price last 3 days	#bookings last 180 days	#bookings last 365 days	Pct bookings month 1
140759	5730	37.78	-122.40	0		0	92	19.27	1.47	511	2254	0.05
431602	1484	-2.42	-54.73	0		0	1	5.29	0.94	0	84	0
17905696	133	21.73	-79.99	0		1	1	>500	0	-	-	-
189642	5730	32.91	-97.01	1		1	1	3.36	0.65	889	4250	0.09
679498048	28	41.38	2.18	0		0	137	0.24	1.43	-	-	-
637314944	111	27.79	-82.79	1		0	0	16.03	1.39	-	-	-
636663872	112	47.80	7.67	1		1	2	24.87	2.91	30	-	-

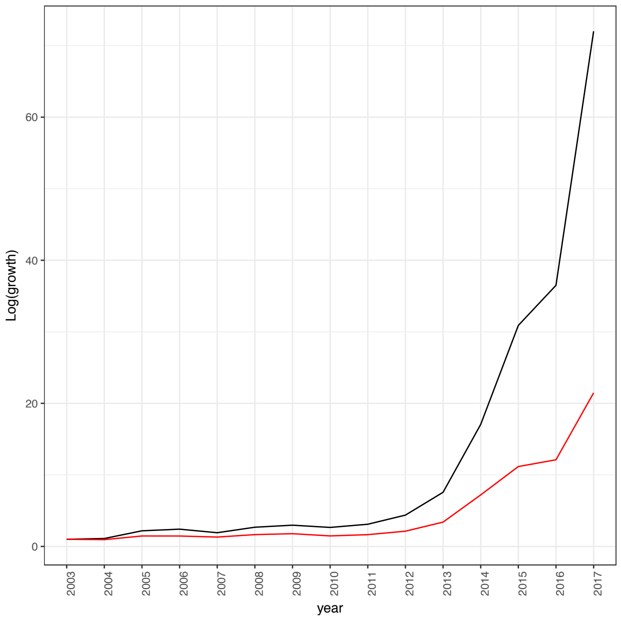


Figure 2 Log growth of the properties in the catalogue (black line) and log growth of missing data (red line)

The *Amenities* dataset is divided into four categories: *Dining* (e.g., restaurants, bar, etc.), *Room* (e.g., TV, Wi-Fi, etc.), *Property* (e.g., reception, elevator, etc.) and *Recreation* (e.g., spa, pool, etc.) amenities. The cleaning of this dataset is done in two steps: filtering out all the amenities that are not provided by the property and those requiring a fee (e.g., "No Free Parking", "No Free Water", "No Free Wi-Fi"); and secondly, for each category, removing the most frequently listed amenities (e.g., "Free Wi-Fi" appears in 190K out of 290K properties) and the least frequently listed ones (roughly getting rid of the top 2% and bottom 15% of the list). The obtained cleansed subset contains around 5 million records (16 amenities on average for each property) and 500 unique amenities. From the *Properties* dataset, only the catalogued active ones are considered (~360K) and referred as a *active* dataset. D*estinations* table contains all points of interest belonging to the following categories *landmark*, *airport*, *metro station* or *train station*. This set is cross-joined with the *active* properties in order to calculate the distances between each property and destination (*Haversine* distance [23]). The resultant set comprises 200 million records of property-destination pairs. The historical *Displayed prices* on the website are calculated using three different time spans: a 1-day; a 3-day; and a 7-day average. The four mentioned datasets (e.g., *Amenities*, *Destinations*, *Properties* and *Displayed Prices*) are joined together with the *property ID* to have all needed information for a specific property. A subset of features of this final dataset is showed in Table 4, while a summary of the features contained in each dataset is given in Table 5.

Table 5 Summary of the features used in the OTAs dataset

Feature Type	# of Features
Amenities	500
Properties	20
Destinations	11
Displayed Prices	57

4.2 Evaluation and Validation

A variety of metrics for comparing and evaluating data imputation and predictive models can be found in the literature [9]. Among them, Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) are the most popular [24]. The MAE is argued to be more accurate and informative than the RMSE [24], successively refuted by [25], where it is stated that the two measures picture different aspects of the error. One major problem of these metrics is that both RMSE and MAE are scale dependent and therefore of hard comparability across different features. For this reason, we consider the R2 coefficient of determination instead [26] (here described as function of the *Sum of Squared Explained* (SSE) and *Sum of Squared Total* (SST)):

$$SSE_k = \sum_{i=1} \left(\hat{y}_{ik} - y_{ik}\right)^2, \quad (2)$$

$$SST_k = \sum_{i} \left(y_{ik} - mean(y_{ik})\right)^2, \quad (3)$$

$$R2_k = \frac{SSE_k}{SST_k} = 1 - \frac{N * RMSE_k^2}{SST_k}, \quad (4)$$

R2 is closely related to RMSE but has an additional normalization term which maps it to the (0,1) interval.
This ensures comparability across all k elements of the label vector. Assuming that we weight prediction errors equally across all k variables, we can define our objective as finding the maximum of $\sum_k R2_k$.
Notice that this is equivalent to minimizing the following weighted squared loss:

$$\sum_k \sum_i \frac{1}{SST_k} \left(\hat{y}_{ik} - y_{ik}\right)^2 \quad . \tag{5}$$

Through this paper the predicted missing values for pattern *i* and feature *k* are referred as $\hat{y}_{ik}$, the known values as y_{ik}, and N is the sample size.

Speedup [27] is the main metric for measuring distributed algorithms scaling, and it is the ratio between the computational speed of the sequential and the distributed counterpart:

$$\text{Speedup}(\text{N}, \text{P}) = \frac{\text{Tseq}(N, 1)}{\text{Tpar}(\text{N}, \text{P})} \tag{6}$$

where N is the size, P is the number of machines and the denominator represents the running time of the parallel version. For validation purposes, the dataset is split into training and test sets (70% - 30%) and the imputation is performed 10 times to cope with the randomness. Furthermore, the *ML Spark* pipelines [8] are used to ensure correctness and replicability of the experiments.

5. Missing Data Imputation Techniques

5.1 Mean and Median Imputation by Location

The most common techniques used as baselines for comparison and analysis of predictive models are *Mean* and *Median imputation* [6]. The *Mean* (*Median*) substitution replaces the missing values with the global mean (median) of the same attribute of the set without missing values. Instead of the global mean (median), in this work we adopt the *Mean* and *Median Imputation* by Location. This measure should provide more accurate results as geographic information is implicitly used (property in the same region are more likely to share prices and other historical similarities).

$$\hat{y}_{ik} = w_k m_i \, , \tag{6}$$

where w_k represents the vector of location averages and m_i - the one-hot encoded representation of the location ID. This baseline is of fast and straightforward implementation, able to easily scale with the size of the considered dataset. On the other hand, these univariate imputation techniques are generally rejected by the scientific community due to their weak statistical power.

3.2 K-NN Imputation

In the *K-Nearest Neighbours Imputation* (KNNI), each missing observation *i* is imputed with the average over its most similar patterns. The set of neighbours L_i is found by minimizing the *Euclidean Distance* between the pattern *i* and the complete subset [22]:

$$\hat{y}_{ik} = \sum_{l \in L_i} \frac{d_{il}^{-1}}{\sum_{l=1}^{l} d_{il}^{-1}} \quad y_{lk} \, , \tag{7}$$

where d_{il} is the *Euclidean Distance* between patterns *i* and *l*. The KNNI approach comprises three steps: take only the rows without missing data and use this subset to select the nearest neighbours; choose a distance metric and compute the nearest neighbour between each pattern with missing data and the complete subset (Equation 7); impute the data, using the mean or the mode of the chosen neighbours. The only parameter to be selected is the number of neighbours *K* and the authors in [22] argue that the method is fairly insensitive to its choice. In all the simulations carried out in this work, we used a value of K = 10. The *K-Nearest Neighbours* has some advantages: the method can predict both, categorical variables (the most frequent value among the KNN) and continuous variables (the average among the KNN); it has easy interpretability and straightforward implementation; furthermore, it only imputes values in the original range of values of the complete set. Disadvantages of the approach are the low scalability to big datasets, ($O(N^2)$ comparisons needed) and the robustness of the results is questionable (e.g., compared to ensemble methods).

3.3 Linear Regression Imputation

In the *Linear Regression Imputation* (LRI) [7], the variables to be imputed (which are assumed to be in the continuous space) are considered as the dependent variable in a multivariate regression model. The model is fitted to the complete subset and comprises three steps: take only the datasets rows without missing data and use this subset to fit the model using:

$$\hat{y}_{ik} = w_k x_i \, , \tag{8}$$

where w_k is the regression weight vector for feature *k,* and x_i the feature vector of item *i* (note that there is no parameter sharing between the features, so the optimization problem is separable in k); impute each missing value with the fitted model; and repeat steps one and two to generate multiple imputations. Using the linear regression approach, a continuous variable may have an imputed value outside the range of observed values. This problem can be addressed by drawing another value every time this occurs. The main advantages of the LRI are the ready implementation of regression models in *Spark* and the fact that it can easily scale with the size of the dataset. On the other hand, the main drawback of the method is the need of a different model for each feature containing missing values (only one dependent variable can be fit at a time).

3.4 Neural Networks Imputation

With this model, the missing values are imputed fitting a Neural Network trained on the complete dataset. In particular, the training phase aims to minimize the weighted squared loss (Equation (6)) across all the features containing missing values. The normalization term SST_k allows the model to give the same importance to each feature during the gradient calculation. At imputation time, the missing values are estimated with:

$$\hat{y}_{ik} = f_k(x_i)\sqrt{SST_k}\,, \tag{8}$$

where $f_k(x_i)$ is the k output in the last layer of the network and $\sqrt{SST_k}$ is the de-normalization term used to bring the output to the initial input scale. The training of this model is distributed on *Spark* as discussed in Section 2.2, allowing the imputation of large datasets of multiple features with a single model.

6. Experimentation and Results

The empirical experiment on the Recommender System OTA dataset is carried to test the imputation accuracy and time feasibility of the proposed technique. Before the imputation, the data is split into a training set (70%) and a test set (30%).

The investigated NN topology includes two hidden layers (n-n-n-k), with n being the number of inputs ($n = 657$) and k the number of outputs ($k = 57$). After each hidden layer, a ReLU activation function is used to transform the data (a ReLU is also applied in the output layer as all the missing values belong to features in $\mathbb{R}_{\geq 0}$. The training set is further divided into 80% for training and 20% for validation, and Equation (6) is used to evaluate the learning performance. The stopping condition includes 2000 training epochs, gradient reaching value less than 1.0e-06, or 6 consequent failed validation checks, whichever occurs first. Figure 3 shows the R2 metric across the 57 considered features. As it can be seen, the *D-NNI* outperforms the other techniques (larger R2) for many of the imputed features. The range of R2 for the *D-NNI* spans from 0.07 to 0.95 with a median of 0.56. The second best method (*LRI*) has its lowest imputation performance at 0 (same as all the other compared techniques), the best at 0.87 and a median R2 of 0.38. Following are the *Mean Imputation*, *KNN-I* and *Median Imputation* with 0.27, 0.26 and 0.18 median R2 respectively. It is also worth to notice that the *Mean* and *Median Imputation* techniques have a few outliers reaching an R2 above 0.80, this is happening with features skewed toward one value (low variance in the values), hence closer to the mean (median) of the imputed one.

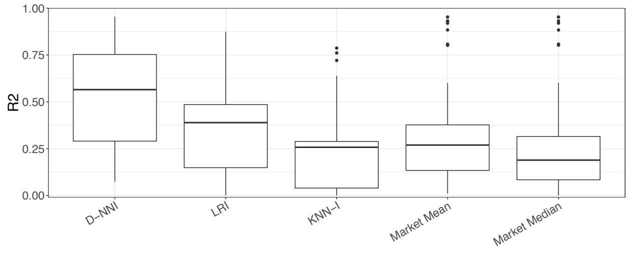


Figure 3 Boxplot for the R2 measure on the 57 imputed variables.

The R2 for each missing feature is also presented in Figure 4. As it can be seen, the prediction accuracy of the *D-NNI* is always greater of those provided by the LRI and KNNI. For the *Mean* (*Median*) *Imputation*, the D-NNI is better in 51 out of 57 cases, while still being comparable in the six remaining features (pct_gt_ly_amer_posa_h, pct_gt_ly_emea_posa_h, etc. (see Figure 4)). Figure 5 shows one of the best (gt_ly_h) and worst (pct_gt_stay_month_4_h) imputed features for the *D-NNI.* It illustrates the NN ability to predict with high accuracy continuous features (Figure 5a), while struggling with the zero inflated ones (Figure 5b) (but still showing better imputation accuracy compared to the other models). Furthermore, in Table 6 the running times over 10 runs for the five imputation techniques are displayed when using the following *Amazon Web Service* cloud cluster configuration: Master (r4.xlarge, 30gb, 4 cores) and 8 Workers (15gb, 8 cores). The average and standard deviation in minutes are reported, showing the D-NNI as the slowest method to impute, with a large variance across different runs (depending on the convergence speed for the training phase). The second slowest method is the LRI. Here, the 57 features containing missing values are independently fitted, reason for the achieved imputation speed. It would be expected a linear decrease in speed using up to 57 workers (where each node of the cluster is fitting a different feature). The fastest imputation models appeared to be the *Mean* (*Median*) *Imputation*, where the prediction only finds the average (median) of each missing feature. The KNN-I speed is not reported as the imputation was not possible with the considered cluster configuration (not enough memory to fit the N^2 pair matrix). When increasing the cluster size to 20 nodes, the KNN-I imputation took 40 minutes, with a standard deviation of 8 minutes, mainly due to communication latency among the workers.

Table 6 Average and std run time (in minutes) over 10 runs

	D-NNI	LRI	KNNI	Mean Imputation	Median Imputation
Avg	24	12	-	< 1	< 1
(Std)	(±10)	(±5)	-	(± 0)	(± 0)

To measure the *D-NNI* sensitivity to the batch size and the number of workers, we consider a grid of values from 10 to 70 for the first parameter, and 2 to 8 for the second one. For each training run we compute the ratio given with Equation 6. This represents the achieved speedup, relative to training on a single node (sequential NN with SGD). In Table 7 we report the imputation *speedup* for the *D-NNI* model under 21 different settings. Table 7 exhibits several trends, with the top row representing the case of two

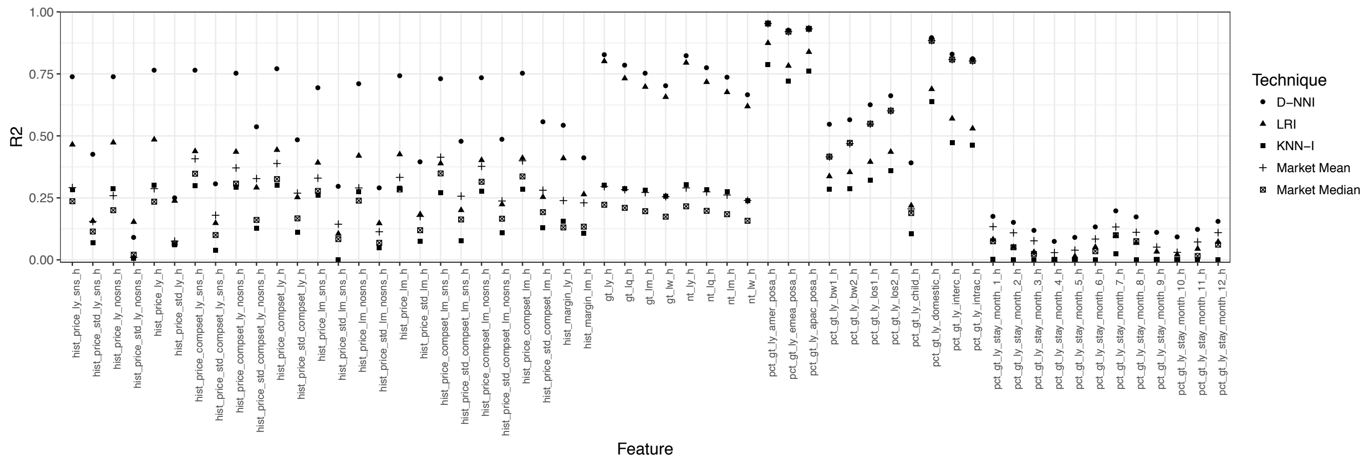


Figure 4 R2 metric for the 57 missing features. The prediction accuracy of each feature is independently showed for the five techniques.

machines. As it can be seen, the *speedup* decreases when incrementing the batch size, which is due to the training taking the largest part of the total computational time, while the communication between nodes is negligible. The same trend still holds in the case of 4 machines (row 2). In the 3rd row, the trend shows reduction of the *speedup* up to the batch size of 50%. The subsequent increase of the ratio (for 60% and 70%) could be due to the time being evenly split between training and communication, and from randomness due to fluctuation in the convergence of the optimization process. Another interesting trend can be observed when inspecting the table by columns. Is true almost for all cases that the use of 4 nodes gives the best speedup over 2 and 8.

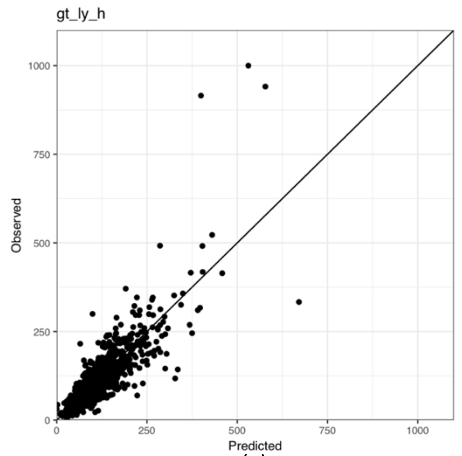
This could be explained by the quantity of data used for training. When a certain threshold for the number of workers is passed, the overhead for communication and synchronization can become larger than the actual processing time. This is enforced by the fact that the trend is less accentuated when moving toward bigger batches. For 40% and 50%, the *speedup* for 4 and 8 nodes is almost identical, while for 8 nodes the *speedup* is higher when using more than 50% of data in each batch.

Table 7 Speedup ratio of the NN compared to the sequential model, for number of samples in batch (10% to 70%) against number of workers (2 to 8)

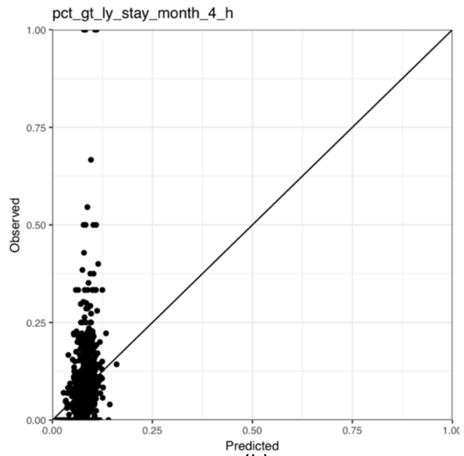
	10	20	30	40	50	60	70
2	2.89	2.40	1.69	1.45	1.27	1.21	1.12
4	3.85	3.62	2.13	2.21	1.97	1.93	1.68
8	3.35	2.95	1.96	2.12	1.71	3.06	2.63

7. Conclusion

The missing data problem in big data context is investigated and a novel imputation approach using Distributed Neural Networks is proposed. The D-NNI framework is implemented as an additional stage in the *Spark* pipeline, ensuring that the missing data are imputed before applying the machine learning techniques. The D-NNI is tested on a real world Recommender System dataset composed of 400 thousand samples and more than 600 features (of which 57 historical characteristics containing missing values). The approach showed improved performance (R2 metric over the 57 missing features) when compared to univariate benchmarks (*Mean* and *Median*



(a)



(b)

Figure 5 Predicted (x-axis) and observed (y-axis) values for the D-NNI. Figure 5a (left) shows the historical yearly gross transactions of each property, while Figure 5b (right) depicts the yearly percentage transactions for one month (April).

imputation by Location) and state-of-the-art techniques (KNN-I and LRI). Furthermore, a speedup analysis has also been carried to test the D-NNI scalability when using 10% to 70% of the data as mini-batches for the training phase, over 2 to 8 machines when compared to the sequential Neural Networks implementation. The reported results indicate that the D-NNI method is a viable option for the imputation of missing data when the considered datasets do not fit in the memory of one machine.

References

[1] C. K. Enders, Applied missing data analysis, Guidford: Guidford Press, 2010.

[2] P. Schmitt, J. Mandel and M. Guedj, "A comparison of six methods for missing data imputation," *Journal of Biometrics & Biostatistics,* vol. 6, no. 1, pp. 1-6, 2015.

[3] A. Petrozziello and I. Jordanov, "Column-wise Guided Data Imputation," in *17th International Conference on Computational Science*, Zurich, 2017.

[4] J. L. Schafer and J. W. Graham, "Missing data: our view of the state of the art.," *Psychological methods,* vol. 7, no. 2, p. 147, 2002.

[5] J. W. Graham, "Missing data analysis: Making it work in the real world," *Annual review of psychology,* vol. 60, pp. 549-576, 2009.

[6] C. M. Musil, C. B. Warner, Y. P. K. and S. L. Jones, "A comparison of imputation techniques for handling missing data," *Western Journal of Nursing Research,* vol. 24, no. 7, pp. 815-829, 2002.

[7] C. Anagnostopoulos and P. Triantafillou, "Scaling out big data missing value imputations: pythia vs. godzilla," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014.

[8] Y. Sun, X. Wang and X. Tang, "Hybrid deep learning for face verification," in *Computer Vision (ICCV), IEEE International Conference on*, 2013.

[9] L. e. a. Deng, "Recent advances in deep learning for speech research at microsoft," in *Acoustics, Speech and Signal Processing (ICASSP), IEEE International Conference on*, 2013.

[10] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010. Physica-Verlag HD*, 2010.

[11] e. a. Chilimbi, "Building an efficient and scalable deep learning training system," in *11th USENIX Symposium on Operating Systems Design and Implementation*, 2014.

[12] L. e. a. Mu, "Scaling distributed machine learning with the parameter server," in *11th USENIX Symposium on Operating Systems Design and Implementation*, 2014.

[13] C. Cheng-Tao, "Map-reduce for machine learning on multicore," *Advances in neural information processing systems,* 2007.

[14] T. White, Hadoop: The definitive guide, O'Reilly Media, Inc., 2012.

[15] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker and I. Stoica, "Spark: cluster computing with working sets," *HotCloud,* vol. 10, pp. 1-7, 2010.

[16] A. Petrozziello and I. Jordanov, "Data Analytics for Online Travelling Recommendation System: A Care Study," in *MIC 2017*, Vienna, 2017.

[17] M. Odersky, L. Spoon and B. Venners, Programming in scala, Artima Inc, 2008.

[18] M. Xiangrui, "Mllib: Machine learning in apache spark," *The Journal of Machine Learning Research ,* vol. 17, no. 1, pp. 1235-1241, 2016.

[19] "Neuron," 28 11 2017. [Online]. Available: https://github.com/bobye/neuron. [Accessed 28 11 2017].

[20] P. Moritz, R. Nishihara, I. Stoica and M. Jordan, "Sparknet: Training deep networks in spark," in *arXiv preprint arXiv:1511.06051*, 2016.

[21] G. Zervas, D. Proserpio and J. Byers, "Therise of the sharing economy: Estimating the impact of airbnb on the hotel industry," in *Boston U. School of Management Research Paper*, Boston, 2016.

[22] O. Troyanskaya, "Missing value estimation methods for dna microarrays," *Bioinformatics,* vol. 17, no. 6, p. 520–525, 2001.

[23] C. Robusto, "The cosine-haversine formula," *The American Mathematical Monthly,* vol. 64, no. 1, pp. 38-40, 1957.

[24] C. Willmott and K. Matsuura, "Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance," *Climate research,* vol. 30, no. 1, pp. 79-82, 2005.

[25] T. Chai and R. Draxler, "Root mean square error (RMSE) or mean absolute error (MAE)?--Arguments against avoiding RMSE in the literature," *Geoscientific Model Development,* vol. 7, no. 3, pp. 1247-1250, 2014.

[26] N. Draper and H. Smith, Applied Regression Analysis., Wiley-Interscience, 1998, pp. 505--553.

[27] J. Hennessy and D. Patterson, Computer architecture: a quantitative approach, Waltham: Elsevier, 2011.