How to Gauge Accuracy of Processing Big Data: Teaching Machine Learning Techniques to Gauge Their Own Accuracy

Vladik Kreinovich, Thongchai Dumrongpokaphan, Hung T. Nguyen, and Olga Kosheleva

Abstract When the amount of data is reasonably small, we can usually fit this data to a simple model and use the traditional statistical methods both to estimate the parameters of this model and to gauge this model's accuracy. For big data, it is often no longer possible to fit them by a simple model. Thus, we need to use generic machine learning techniques to find the corresponding model. The current machine learning techniques estimate the values of the corresponding parameters, but they usually do not gauge the accuracy of the corresponding general non-linear model. In this paper, we show how to modify the existing machine learning methodology so that it will not only estimate the parameters, but also estimate the accuracy of the resulting model.

Vladik Kreinovich
Department of Computer Science, University of Texas at El Paso, 500 W. University, El Paso, Texas 79968, USA, e-mail: vladik@utep.edu

Thongchai Dumrongpokaphan
Department of Mathematics, Faculty of Science, Chiang Mai University, Thailand, e-mail: tcd43@hotmail.com

Hung T. Nguyen
Department of Mathematical Sciences, New Mexico State University, Las Cruces, NM 88003, USA
and
Faculty of Economics, Chiang Mai University, Chiang Mai 50200 Thailand, e-mail: hunguyen@nmsu.edu

Olga Kosheleva
Department of Computer Science, University of Texas at El Paso, 500 W. University, El Paso, Texas 79968, USA, e-mail: olgak@utep.edu

1 Need to Gauge Accuracy of Big Data Processing

Need for data processing. In many practical situations, we are interested in the value of a quantity y which is difficult – or even impossible – to measure directly. For example, we may be interested:

- in tomorrow's temperature y, or
- in the distance y to a faraway star.

Since we cannot measure this quantity y directly, we have to measure it *indirectly*; namely:

- we measure easier-to-measure quantities $x_1,\dots,x_n$ whose values determine y, i.e., for which $y = f(x_1,\dots,x_n)$ for some function $f(x_1,\dots,x_n)$, and then
- we use the results $\widetilde{x}_i$ of measuring x_i and the known dependence $f(x_1,\dots,x_n)$ to estimate y as $\widetilde{y} = f(\widetilde{x}_1,\dots,\widetilde{x}_n)$.

The corresponding computations are known as *data processing*.

Need to find the corresponding dependence. To be able to perform data processing, we need to know the dependence $y = f(x_1,\dots,x_n)$ between the corresponding quantities x_i and y.

- In some cases, this dependence can be determined based on the fundamental physical principles.
- However, in many other cases, this dependence needs to be determined experimentally, based on the known observation results.

Traditional approach to finding a dependence: a brief reminder. The traditional statistical approach to finding the desired dependence has been designed for situations in which the number of available observations is reasonably small; see, e.g., [4]. In such situations, we can usually fit the data with some simple model – e.g., with the linear regression model in which the dependence is linear: $f(x_1,\dots,x_n) = a_0 + \sum\limits_{i=1}^{n} a_i \cdot x_i$. For such models, the traditional statistical approach provides both:

- the estimates for the values of the corresponding parameters, and
- a good description of the accuracy of the corresponding estimated model $\widetilde{f}(x_1,\dots,x_n)$, i.e., of the probability distribution of the approximation error $\Delta y \stackrel{\text{def}}{=} y - \widetilde{f}(x_1,\dots,x_n)$.

The emergence of big data. In the last decades, the progress in computer and computer-based measurement technologies enabled us to get huge amounts of data, amounts far exceeding the sample sizes for which we can apply the traditional statistical techniques.

This phenomenon is known as *big data*; see, e.g., [3].

For big data, simple models are rarely possible. Real-life phenomena are usually very complex. When we have a reasonable small amount of data, we can still have simple approximate models that describe this data well. However, as we increase the amount of data, we can no longer use simple models.

This phenomenon is easy to explain. In general in statistics, based on a sample of size n, we can determine each parameter of the model with accuracy $\sim 1/\sqrt{n}$.

- When the sample size n is reasonably small, the resulting inaccuracy usually exceeds the size of the quadratic and higher order terms in the actual dependence. So, within this accuracy, we can safely assume that the dependence is linear.
- However, as the sample size increases, the accuracy $1/\sqrt{n}$ becomes smaller than the quadratic and higher order terms – and thus, these terms can no longer be ignored if we want to have a model that fits all the data.

This phenomenon is well known. For example:

- When a comet appears and we have only few of its observations, we can safely use simplified Newton's equations – that assumes that only the Sun and the Jupiter have to be taken into account – and get a good description of all the observed data.
- However, as the number of measurements increases, we have to take into account the gravitational influence of other planets to get a good fit with observations.

Machine learning techniques: the big-data analogs of the traditional statistical data processing. For large data sizes, we cannot use simple few-parametric models, we need complex models with large number of parameters. In most situations, it is not possible to guess the exact form of the corresponding non-linear dependence. So, we need to use techniques that do not assume any specific form, but try to extract the form of the dependence from the data itself.

Such techniques are known as *machine learning*; see, e.g., [1, 2]. For most of machine learning approaches such as neural networks, there are *universal approximation* results that show that every possible non-linear function can be, with any given accuracy, approximated by the corresponding computational model.

Machine learning techniques: successes and limitations. Machine learning techniques have been very successful in many applications [1, 2]. In many practical applications, they use the observations to come up with a dependence $\widetilde{f}(x_1,\ldots,x_n)$ that provides a good fit for the observed data.

However, what is lacking in most machine learning techniques is a good understanding of how accurate is this model, i.e., what are the probabilities of different values of the approximation error $\Delta y = y - \widetilde{f}(x_1,\ldots,x_n)$.

To be more precise, we can estimate overall characteristics of such an accuracy: e.g., we can take all the values of the approximation error corresponding to all the input observations $(x_1,\ldots,x_n,y)$ and find the standard deviation and the whole distribution. But this will be the overall distribution.

We know that in many cases, the approximation accuracy depends on the inputs $x_1,\ldots,x_n$:

- for some inputs, the model $\widetilde{f}(x_1,\ldots,x_n)$ provides a more accurate approximation, while
- for other inputs, the model provides less accurate approximation.

Traditional statistical methods enable us to find the distribution of measurement errors corresponding to each individual tuple $x = (x_1,\ldots,x_n)$.

It is desirable to have something similar for machine learning techniques as well.

What we do in this paper. In this paper, we show how we can teach the existing machine learning techniques

- to not only the approximate dependence model, but
- also to automatically gauge the accuracy of the resulting model.

2 Our Main Idea

Analysis of the problem. We would like to be able, for each possible tuple $x = (x_1,\ldots,x_n)$, to generate not only a single numerical estimate for the corresponding value y, but also the whole conditional probability distribution describing possible values y corresponding to this tuple.

From the computational viewpoint, a natural way to simulate a probability distribution with a given cumulative distribution function (cdf) $F(Y) = \mathrm{Prob}(y \le Y)$ is:

- to start with the standard random number r which is uniformly distributed on the interval $[0,1]$ – and whose generation is supported by most programming languages and programming environments – and
- apply the inverse function $F^{-1}(p)$ to this random number.

The resulting values $y = F^{-1}(r)$ are indeed distributed according to the given probability distribution $F(Y)$.

The inverse function $F^{-1}(p)$ has a direct probabilistic meaning; namely,

- for each $p \in [0,1]$,
- the value $x = F^{-1}(p)$ is the p-th *quantile*, i.e., the value x for which $F(x) = p$.

For $p = 0.5$, we get the median, for $p = 0.25$ and $p = 0.75$, we get the lower and upper quartiles, etc.

From this viewpoint, what we want is to be able,

- for every possible tuple $x = (x_1,\ldots,x_n)$ and
- for every possible value p,

to come up with the p-th quantile of the conditional y-distribution corresponding to this tuple. In precise terms, for the function $G_x(y) \stackrel{\text{def}}{=} F(y\,|\,x_1,\ldots,x_n)$, we want to be able to generate the quantile $q = G_x^{-1}(p)$ for which

$$\text{Prob}(y \le q \,|\, x_1, \ldots, x_n) = p.$$

Let us denote this quantile q by $G(x_1, \ldots, x_n, p)$.

Thus, we want to generate a function $G(x_1, \ldots, x_n, p)$ of $n+1$ variables. Once we have this function, we will be able to find, for each tuple $x = (x_1, \ldots, x_n)$, the cdf corresponding y-distribution $F_x(y)$ – as the inverse function $F_x(p) = q_x^{-1}(p)$ to the function $q_x(p) \stackrel{\text{def}}{=} Q(x_1, \ldots, x_n, p)$.

Historical comment. The idea of combining stochastic processes with neural networks was originally proposed and actively promoted by Paul Werbos; see, e.g., [6, 7] (see also [5]).

How can we use machine learning to find the desired function $Q(x_1, \ldots, x_n, p)$? In the ideal world, for each tuple $x = (x_1, \ldots, x_n)$, we would have several different observations in which we have

- these same values of x_i and
- different values of y.

In this case, from this sample of different values of y corresponding to the given tuple x, we will be able to determine the conditional probability distribution corresponding to this tuple x.

Specifically, once we have N such y-values, we can sort them into an increasing sequence $y_{(1)} \le \ldots \le y_{(N)}$. Crudely speaking, these values correspond to the quantiles $p = 1/N$, $p = 2/N$, etc. We therefore expect each predicted quantile $Q(x_1, \ldots, x_n, j/N)$ to be close to the corresponding value $y_{(j)}$.

In practice, we do not have such "same-x" observations: different observations correspond, in general, to different tuples x. Since we cannot have different observations corresponding to the *exact same* tuple x, a natural idea is to use observations corresponding to *nearby* tuples x.

Good news is that when we have big data, i.e., a *very large* amount of data, it is highly probable that a few of the observations will be close to x – even when the probability of being close to x is small.

Once we pick N such nearby tuples, we can sort the corresponding N y-values into an increasing sequence $y_{(1)} \le \ldots \le y_{(N)}$. Crudely speaking, these values correspond to the quantiles $p = 1/N$, $p = 2/N$, etc. In other words, each of these values $y_{(j)}$ correspond to $p = j/N$. We therefore expect each predicted quantile $Q(x_1, \ldots, x_n, j/N)$ to be close to the corresponding value $y_{(j)}$.

Thus, we arrive at the following algorithm for reconstructing the desired function $Q(x_1, \ldots, x_n, p)$.

Resulting algorithm. We start with the list of observations $\left(x_1^{(k)}, \ldots, x_n^{(k)}, y^{(k)}\right)$. Based on this list, we will form the extended tuples $\left(x_1^{(k)}, \ldots, x_n^{(k)}, p_j, y_j^{(k)}\right)$ as follows:

1°. We fix some number N – e.g., $N = 10$.

2°. Then, for each original observation $\left(x_1^{(k)}, \ldots, x_n^{(k)}, y^{(k)}\right)$, we do the following:

2.1° We find $N-1$ observations $\left(x_1^{(\ell)},\ldots,x_n^{(\ell)},y^{(\ell)}\right)$ in which the x-tuple $x^{(\ell)}=\left(x_1^{(\ell)},\ldots,x_n^{(\ell)}\right)$ is the closest to the original x-tuple

$$x^{(k)}=\left(x_1^{(k)},\ldots,x_n^{(k)}\right).$$

2.2° In the resulting $1+(N-1)=N$ observations, we have N different y-values:

- the original y-value $y^{(k)}$ and
- $N-1$ values $y^{(\ell)}$ corresponding to $N-1$ "nearest" observations.

Let us sort them into an increasing sequence

$$y_{(1)}^{(k)}\le y_{(2)}^{(k)}\le\ldots\le y_{(N)}^{(k)}.$$

2.3° Then, we form N extended tuples

$$\left(x_1^{(k)},\ldots,x_n^{(k)},\frac{j}{N},y_{(j)}^{(k)}\right).$$

3°. After that, we combine all the extended tuples corresponding to different original tuples into a single list.

4°. To the resulting single list, we apply a machine learning algorithm and thus, construct the desired function $Q(x_1,\ldots,x_n,p)$ that provides,

- for each tuple $x=(x_1,\ldots,x_n)$ and
- for each value $p\in[0,1]$,

the p-th quantile of the y-distribution corresponding to this tuple x.

Which value N should we use: a comment.

- The larger N, the more detailed is the resulting information about the probability distributions.
- On the other hand, if we take N to be too large, then some of the "closest" tuples $x^{(\ell)}$ may be too far away from the original tuple $x^{(k)}$ and thus, this description will not be very accurate.

So, here, we have a usual trade-off between accuracy and details – similar to what we have, e.g., when we divide the real line into bins to build a histogram approximating the actual probability density function $\rho(x)$:

- if we use smaller bins, we get more details, but
- these details are at the expense of accuracy of approximating $\rho(x)$: the smaller bins, the fewer observations are in each bin, and thus, the less accurately the corresponding frequencies represent the desired probabilities.

The actual value N should be determine empirically – or, if we have some prior information, by using this prior information. As a rule of thumb, we suggest using $N=10$, since in our experience, this is where we get the largest number of reliable details.

Acknowledgments

This work was supported by Chiang Mai University, Thailand. This work was also supported in part by the National Science Foundation grants HRD-0734825 and HRD-1242122 (Cyber-ShARE Center of Excellence) and DUE-0926721, and by an award "UTEP and Prudential Actuarial Science Academy and Pipeline Initiative" from Prudential Foundation.

One of the authors (VK) is thankful to Paul Werbos for inspiring talks and discussions.

References

1. C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer Verlag, New York, 2006.
2. I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, Cambridge, Massachusetts, 2016.
3. V. Mayer-Schönberger and K. Cukier, *Big Data: The Essential Guide to Work, Life, and Learning in the Age of Insight*, John Murray Publ., London, UK, 2017.
4. D. J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures*, Chapman & Hall/CRC, Boca Raton, Florida, 2011.
5. C. Turchetti, *Stochastic Models of Neural Networks*, IOS Press, Amsterdam, 2004.
6. P. Werbos, "A brain-like design to learn optimal decision strategies in complex environments", in: M. Karny, K. Warwick, and V. Kurkova (eds.), *Dealing with Complexity: A Neural Networks Approach*, Springer, London, 1998.
7. P. Werbos, "Intelligence in the brain: a theory of how it works and how to build it", *Neural Networks*, 2009, Vol. 22, No. 3, pp. 200–212.