Original Article

Dealing With Data Streams

An Online, Row-by-Row, Estimation Tutorial

Lianne Ippel, Maurits Kaptein, and Jeroen Vermunt

Methodology and Statistics, Tilburg University, The Netherlands

Abstract: Novel technological advances allow distributed and automatic measurement of human behavior. While these technologies provide exciting new research opportunities, they also provide challenges: datasets collected using new technologies grow increasingly large, and in many applications the collected data are continuously augmented. These *data streams* make the standard computation of well-known estimators inefficient as the computation has to be repeated each time a new data point enters. In this tutorial paper, we detail *online learning*, an analysis method that facilitates the efficient analysis of Big Data and continuous data streams. We illustrate how common analysis methods can be adapted for use with Big Data using an online, or "row-by-row," processing approach. We present several simple (and exact) examples of the online estimation and discuss Stochastic Gradient Descent as a general (approximate) approach to estimate more complex models. We end this article with a discussion of the methodological challenges that remain.

Keywords: Big Data, data streams, machine learning, online learning, Stochastic Gradient Descent

The ever-increasing availability of Internet access, smart phones, and social media have led to many novel opportunities for collecting behavioral and attitudinal data. These technological developments allow researchers to study human behavior at large scales and over long periods of time (Swendsen, Ben-Zeev, & Granholm, 2011; Whalen, Jamner, Henker, Delfino, & Lozano, 2014). Because more data are made available for research, these technological developments have the potential to advance our understanding of human behavior (Barrett & Barrett, 2001) and its dynamics. However, these novel data collection technologies also present us with new challenges: If (longitudinal) data are collected from large groups of subjects, then we may obtain extremely large datasets. These datasets might be so large that they cannot be analyzed using standard analysis methods and existing software packages. This is exactly one of the definitions used for the buzz term "Big Data" (Demchenko, Grosso, De Laat, & Membrey, 2013; Sagiroglu & Sinanc, 2013): datasets that are so large that they cannot be handled using standard computing machinery or analysis methods.

Handling extremely large datasets represents a technical challenge in its own right, moreover, the challenge is amplified when large datasets are continuously augmented (i.e., new rows are added to the dataset as new data enter over time). A combination of these challenges is encountered when – for example – data are collected continuously using smart-phone applications (e.g., tracking fluctuations in happiness, Killingsworth & Gilbert, 2010) or when data are mined from website logs (e.g., research into improving e-commerce, Carmona et al., 2012). If datasets are continuously augmented and estimates are needed at each point in time, conventional analyses often have to be repeated every time a new data point enters. This process is highly inefficient and frequently forces scholars to arbitrarily stop data collection and analyze a (smaller) static dataset. In order to resolve this inefficiency, existing methods need to be adapted and/or new methods are required to analyze streaming data. To be able to capitalize on the vast amounts of (streaming) data that have become available, we must develop efficient methods. Only if these methods are widely available we will be able to truly improve our understanding of human behavior.

Failing to use appropriate methods when analyzing Big Data or data streams could result in computer memory overflow or computations that take a lot of time. In favorable cases, the time to compute a statistic using standard methods increases linearly with the amount of data entering. For example, if computing the sum over n data points requires t time (where the time unit required for the computation is dependent on the type of machine used, the algorithm used, etc.), then computing the sum over $n + 2$ data points requires $t + 2c$ time, where c is t/n. Thus, the time increase is linear in n and is every time increasing as the data stream grows. In less fortunate and more common cases, the increase in time complexity is not linear but quadratic, or worse, amplifying the problems. Regardless of the exact scaling however, if the data are continuously augmented both the required computation time and memory use eventually will become infeasible.

The aim of this tutorial paper is to introduce *online learning* (or row-by-row estimation), as a way to deal with Big Data or data streams. Online learning methods analyze the data without storing all individual data points, for instance by computing a sample mean, or a sum of squares without revisiting older data. Therefore, online learning methods have a feasible time complexity (i.e., the time required to conduct the analysis) and they require a feasible amount of computer memory when analyzing data streams or Big Data. In the latter case, a very large static dataset is treated as if it were a data stream by iterating through the rows without having all data points available in memory.

Online estimation methods continuously *update* their estimates when new data arrive, and *never revisit* older data points. Formally online learning can be denoted as follows:

$$\theta_n = f(\theta_{n-1}, x_n),$$

or equivalently and a shorthand

$$\theta := f(\theta, x_n), \tag{1}$$

which we will use throughout the paper. In Equation 1, θ is a set of sufficient statistics (not necessarily the actual parameters of interest), which is updated using a new data point, x_n. The second equation for updating θ does not include subscript n because we use the update operator ":=", which indicates that the updated θ is a function of the previous θ and the most recent data point, x_n.

A large number of well-known conventional estimation methods used for the analysis of regular (read "small") datasets can be adapted such that they can handle data streams, without losing their straightforwardness or interpretation. We provide a number of examples in this paper. Furthermore, we will also introduce Stochastic Gradient Descent, a general method that can be used for the (approximate) estimation of complex models in data streams. For all the examples introduced in this paper, we have made [R] code available at http://github.com/L-Ippel/Methodology.

The paper is organized as follows. The next section discusses conceptual approaches for the estimation of parameters in Big Data and/or streaming data. We focus primarily on online learning, the method further illustrated in the remainder of this paper. In the following section we first illustrate how often-used estimators such as sample means, variances, and covariances can be estimated using online learning. Here the benefits of online learning to deal with data streams are illustrated by comparing the computational times of online and offline estimation methods. This section is followed by an introduction of Stochastic Gradient Descent (SGD) as a general (approximate) method to estimate more complex models in data streams. Next we provide an example of an application of SGD in social sciences. We continue the paper with a discussion of the limitations of the online learning approach and end the paper with directions for further research on data streams and Big Data.

Dealing With Big Data: The Options

In the recent years, data streams and the resulting large datasets have received the attention of many scholars. Diverse methods have been developed to deal with these vast amounts of data. Conceptually, four overarching approaches to handle Big Data can be identified:

1. sample from the data to reduce the size of the dataset,
2. use a sliding window approach,
3. parallelize the computation, or
4. resort to online learning.

The first option, to sample from the data, solves the problem of having to deal with a large volume of data simply by reducing its size. Effectively, when the dataset is too large to process at once, one could "randomly" split the data into two parts: a part which is used for the analyses and a part of the data that is discarded. Even in the case of data streams, a researcher can decide to randomly include new data points or let them "pass by" to reduce memory burden (Efraimidis & Spirakis, 2006). However, when a lot of data are available, it might be a waste not to use all the data we could potentially use.

The second option, using a sliding window, also solves the issue of needing increasingly more computation power by reducing the amount of data that is analyzed. In a sliding window approach the analysis is restricted to the most recent part of the data (Datar, Gionis, Indyk, & Motwani, 2002; Gaber, Zaslavsky, & Krishnaswamy, 2005). Thus, the data are again split into a part which is used for the analyses and a part which is not used for the analysis. The analysis part (i.e., also coined "the window") consists of the m most recent data points, while the second part contains older data which is discarded. One could see a sliding window as a special case of option 1, where the subsample only consists of new data points. When new data enter, the window shifts to include new data (i.e., a (partially) new subsample) and ignores the old data. Although a sliding window approach is feasible in computation time and amount of memory needed, the sliding window approach has the downside that it requires domain knowledge to determine a proper size of the window (e.g., determine m). For instance, when studying a rare event, the window should be much larger than in the case of a frequent event. It is up to the researcher's discretion to decide how large this window ought to be. Also, when analyzing trends, a sliding window approach might not be appropriate since historical data are ignored.

The third option, using parallel computing, is an often-used method to analyze static Big Data. Using parallel computing, the researcher splits the data in chunks, such that multiple independent machines each analyze a chunk of data, after which the results of the different chunks are combined (see, e.g., Atallah, Cole, & Goodrich, 1989; Chu et al., 2006; Turaga et al., 2010). This effectively solves the problem of memory burden by allocating the data to multiple memory units, and reduces the computation time of static datasets, since analyses which otherwise would have been done "sequentially" are conducted "parallel." However, parallelization is not very effective when the dataset is continuously augmented: since all data are required for the analyses, computation power has to eventually grow without bound for as long as the dataset is augmented with new data. Also, the operation of combining the results obtained on different chunks of data might itself be a challenge.

In this paper we will focus on a fourth method: online learning (e.g., Bottou, 1998; Shalev-Shwartz, 2011). As introduced in the previous section, online learning uses all available information, but without storing or revisiting the individual data points. Online learning methods can be used in combination with parallel computation (for instance, see Chu et al., 2006; Gaber et al., 2005), but here we discuss it as a unique method that has large potential for use in the social sciences. This method can be thought of as using a very extreme split of the data; the data is split into a part consisting of $n - 1$ data points, where n is the total number of observations, and only 1 data point on the other hand. Additionally, in online learning methods, the $n - 1$ data points are summarized into a limited set of parameters or estimates of interest, which take all relevant information of previous data points into account (Opper, 1998). The summaries required to estimate the parameters of interest (often the sufficient statistics) are stored in θ. Subsequently, θ is updated using some function of the previous θ and the new data point; historical data points are not revisited.

Note that in this paper, we focus on the situation where parameters are updated using a single (most recent) data point. There are also situations where one rather uses a "batch" of data points. This is known as *batch learning*. See for a discussion of batch learning in SGD, Wilson and Martinez (2003), or Thiesson, Meek, and Heckerman (2001) about choosing block (or batch) sizes for the expectation-maximization (EM) algorithm.

The two characteristics of online learning – including all the data in the estimate and not revisiting the historical data – jointly make online learning a very suitable approach to analyze data streams. However two downfalls remain, like sliding windows, online learning also requires domain knowledge to judge which information should be gathered beforehand; the researcher needs to choose the elements of θ and their update functions up front. Second, although this issue is not unique for online learning, the researcher often needs to choose starting values for the elements of θ. In the next section we further detail online learning and how to choose starting values by providing the online adaptation of a number of conventional statistics.

From Conventional Analysis to Online Analysis

In this section we discuss online analysis by providing several examples of the online computation of standard (often computed *offline*) estimators. We discuss the online estimation of the following parameters:

1. the sample mean,
2. the sample variance,
3. the sample covariance,
4. linear regression models, and
5. the effect size η^2 (in an ANOVA framework)

The online formulations we discuss in this section are exact reformulations of their offline counterparts: the results of the analysis are the exact same whether one uses an offline or online estimation method. Note that for each of these examples, small working examples as well as ready-to-use functions are available on http://github.com/L-Ippel/Methodology.

Sample Mean

The conventional estimation of a sample mean ($\bar{x}$) is computationally not very intensive since it only requires a single pass through the dataset,

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i. \tag{2}$$

However, even in this case, online computation can be beneficial. The online update of a sample mean is computed as follows:

$$\begin{aligned} \theta &= \{\bar{x}, n\}, \\ n &:= n + 1, \\ \bar{x} &:= \bar{x} + \frac{1}{n}(x_n - \bar{x}). \end{aligned} \tag{3}$$

where we again use the update operator ":=" and start by stating the elements of θ that need to be updated: in this case these are n (a count) and $\bar{x}$ (the sample mean). Note that, appropriate starting value(s) for all the elements θ need to be chosen. This also holds for all the other examples provided. In the case of the mean one can straightforwardly choose $n = 0$ and $\bar{x} = 0$ as this starting point does not impact the final result – this regretfully will not generally hold. Also note that an online sample mean could also be computed by maintaining $n := n + 1$ and $Sx := Sx + x_n$, where Sx is the sum over x, as the elements of θ; in this case the sample mean could be computed at runtime using $\bar{x} = \frac{Sx}{n}$. This latter method however (a) does not actually store the sought for statistic as an element of θ, and (b) lets Sx grow without bound, which might lead to numerical instabilities.

We implemented the online formulation of the sample mean in [R] code, *mean_online()*, which can be found at http://github.com/L-Ippel/Methodology/Streaming_functions, and is ready to use to update the mean. Below we present [R] code, which gives a demonstration of the use of the online implementation of the sample mean. In the [R] language, the "#" denotes a comment.

```
> # create some data:
> # number of data points = 1000,
> # mean of the data is 5 and standard deviation is 2:
> N <– 1000
> x <– rnorm(n=N, mean=5, sd=2)
> # create an object for the results:
> res <– NULL
> # the res object is needed such that you can feed back
> # the updates into the function
> # all sufficient statistics that are required for
  mean_online
> # are created within the function, at the first call
> for(i in 1 : n )
+ {
+   res <– mean_online(input = x[i], theta=res)
+ }
>
```

Sample Variance

In case of the sample variance (often denoted s^2) more is to be gained when moving from offline to online computation as the conventional method of computing a sample variance requires two passes through the dataset:

$$\begin{aligned} s^2 &= \frac{1}{n-1}\sum_{i=1}^{n}(x_i-\bar{x})^2, \\ &= \frac{SS}{n-1}, \end{aligned} \tag{4}$$

where SS is the sum of squares. Here, the first pass is used to estimate the mean $\bar{x}$, while the second pass is used to compute the sum of squares.

A numerically feasible online method to compute a sample variance in a data stream is Welford's method (1962), which, to keep notation consistent, we denote as:

$$\begin{aligned} \theta &= \{\bar{x}, SS, n\}, \\ d &= x_n - \bar{x}, \\ n &:= n+1, \\ \bar{x} &:= \bar{x} + \frac{1}{n}(x_n - \bar{x}), \\ SS &:= SS + d(x_n - \bar{x}). \end{aligned} \tag{5}$$

Note the use of auxiliary variable d which is used since the online update of the sum of squares uses both the deviation from the current sample mean as well as from the previous sample mean:

$$SS_n = SS_{n-1} + (x_n - \bar{x}_{n-1})(x_n - \bar{x}_n). \tag{6}$$

In order to obtain the actual sample variance, we compute $s^2 = SS/(n-1)$. The function to compute the sum of squares is coined *SS_online* and *var_online* uses the online sum of squares function to compute the variance. In order to obtain the standard deviation directly, *sd_online* function can be used. Note that in order to compute the variance and the standard deviation, starting values of $n = 1$ and a $\bar{x} = x[1]$ (which is the first data point) are required due to the fact that the sum of squares is divided by $(n - 1)$. The values $\{n = 1, \bar{x} = x[1]\}$ are provided as default, in case the user does not provide starting values.

Sample Covariance

Next we turn to the estimation of quantities which depend on multiple variables, for instance the sample covariance between x and y, which is often computed using:

$$\begin{aligned} s_{xy} &= \frac{1}{n-1}\sum_{i=1}^{n}(y_i - \bar{y})(x_i - \bar{x}), \\ &= \frac{SC}{n-1}, \end{aligned} \tag{7}$$

where SC is the sum of cross products. Again, making use of Welford's method (1962), we can estimate the sample covariance online:

$$\begin{aligned} \theta &= \{\bar{x}, \bar{y}, SC, n\}, \\ n &:= n+1, \\ \bar{x} &:= \bar{x} + \tfrac{1}{n}(x_n - \bar{x}), \\ SC &:= SC + (y_n - \bar{y})(x_n - \bar{x}), \\ \bar{y} &:= \bar{y} + \tfrac{1}{n}(y_n - \bar{y}). \end{aligned} \tag{8}$$

Note that, contrary to the online computation of the sample variance, we do not need auxiliary variables in this case since we can alternate updating of $\bar{x}$ and $\bar{y}$. The choice of which of the two means is updated first, is arbitrary (Pèbay, 2008). Similar to the case of the sample variance, to compute the sample covariance, we compute $s_{xy} = SC/(n-1)$.

In Appendix A we present [R] code to compute covariances and correlations online. Since computing a correlation entails the estimation of sample means, variances, and a covariance all of these are also included in this code snippet. For readers wanting to compute the sum of cross products, the covariance, or the correlation during their analysis we have implemented the online estimation procedures in [R], and these can be found in the Streaming_functions file on github as, respectively, *SSxy_online, cov_online*, and *cor_online*. Note that, unlike the previous statistics, these functions require two inputs, one for each variable.

Linear Regression

In applied research, often the aim is to estimate group differences or the effect of a certain independent variable x on a dependent variable y. In such cases the computation of a sample mean of a sample variance will not necessarily suffice to answer the research question. One often-used approach to answer research questions about the relationship between one or more independent variables and one dependent variable is using a linear regression model:

$$\mathbf{y} = \mathbf{X}\beta + \varepsilon, \tag{9}$$

where $\mathbf{y}$ is the vector containing the data of the dependent variable, β is a vector of the regression coefficients of the q independent variables (including an intercept), and $\mathbf{X}$ is the matrix ($n \times q$) with observed data, including a column of 1's for the intercept. Finally ε denotes the error or noise. When assuming $\varepsilon \sim \mathcal{N}(0, \sigma^2)$, the regression coefficients β are conventionally estimated as follows:

$$\beta = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}, \tag{10}$$

where $\mathbf{X}'$ denotes the transpose of $\mathbf{X}$.

Computing this row-by-row works as follows: We can define $\mathbf{A} = \mathbf{X}'\mathbf{X}$ and $\mathbf{B} = \mathbf{X}'\mathbf{y}$ and compute the update as follows:

$$\begin{aligned} \theta &= \{\mathbf{A}, \mathbf{B}\}, \\ \mathbf{A} &:= \mathbf{A} + \mathbf{x}_n\mathbf{x}_n', \\ \mathbf{B} &:= \mathbf{B} + \mathbf{x}_n y_n. \end{aligned} \tag{11}$$

To obtain the regression coefficients, β, one computes $\beta = \mathbf{A}^{-1}\mathbf{B}$. This method is well known in the parallel computing literature and is, for example, described in Chu et al. (2006).

Although fairly simple, computing the regression coefficients this way has a disadvantage: Every time that β is computed, a matrix inversion is required. Especially when the number of independent variables q is large, this itself can be a computationally intensive operation. We can address this by computing the regression coefficients, β, directly online, using the Sherman-Morrison formula (Escobar & Moser, 1993; Plackett, 1950; Sherman & Morrison, 1950):

$$\begin{aligned} \theta &= \{\mathbf{A}_{\text{inv}}, \mathbf{B}\}, \\ \mathbf{A}_{\text{inv}} &:= \mathbf{A}_{\text{inv}} - \frac{\mathbf{A}_{\text{inv}}\mathbf{x}_n\mathbf{x}_n'\mathbf{A}_{\text{inv}}}{1 + \mathbf{x}_n'\mathbf{A}_{\text{inv}}\mathbf{x}_n}, \\ \mathbf{B} &:= \mathbf{B} + \mathbf{x}_n y_n, \end{aligned} \tag{12}$$

where $\mathbf{A}_{\text{inv}}$ is the inverted matrix $\mathbf{A}$. Using this formulation we directly update the inverted matrix. In practice one would use a small part of the data to create matrix $\mathbf{A}$, invert this matrix, after which the original matrix $\mathbf{A}$ can be discarded from computer memory. The "small" part of the data that is used should at least have $n > q + 1$ for $\mathbf{A}$ to be non-singular. Obtaining the regression coefficients using Equation 12 only requires a matrix multiplication: $\beta = \mathbf{A}_{\text{inv}}\mathbf{B}$. In Appendix B, we implement online linear regression in [R] using the Sherman-Morrison formula. At the github page mentioned before, the function is named *lm_online*. The function requires two separate inputs, one for the dependent variable and one for the independent variables. The latter can obviously be a vector of multiple variables.

Computation Time of Linear Regression

To illustrate the difference between online and offline methods, Figure 1 presents a comparison of the computational time required to compute the regression coefficients β in a data stream between the three estimation methods discussed above. The x-axis denotes the number of data points seen so far. While the scale of the y-axis (time) will heavily depend on the size of the model (the number of parameters q) and the type of computing system used, the qualitative results presented here will hold in general: the computational time needed to obtain an estimate of β, at each value of n, will grow quite quickly (quadratic) for the offline method, while it grows only slowly for the online methods (linear). This result clearly illustrates the computational benefits of online methods over offline methods. It can also be seen that the direct online computation of the inverted matrix $\mathbf{A}_{\text{inv}}$ is faster than inverting the matrix at each time point. This latter difference however only affects the slope of the linear computation time.

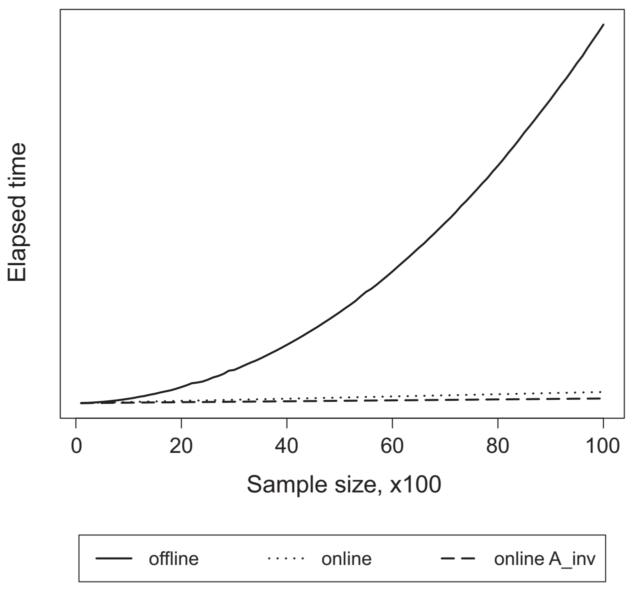

Figure 1. Computation time of regression coefficients using offline estimation (solid line), online estimation by inverting the matrix (online A_{inv}, dotted line), or online estimation by online updating the inverted matrix (dashed line).

Effect Size η^2 (ANOVA)

In many studies in sociology and psychology it is of interest to examine whether *k* distinct groups differ from one another, for instance because one group of participants received a treatment while the other group of participants did not. When such experiments are carried out using modern interactive technologies, such as on social media platforms, sample sizes can grow very quickly. Traditionally researchers often analyze the data from such group comparisons using an ANOVA approach. Between-subjects ANOVAs can be computed fully online. Here we focus on the computation of the effect size η^2 which is given by:

$$\begin{aligned}\eta^2 &= \frac{SS_b}{SS_b + SS_w}, \\ &= \frac{SS_b}{SS_t}, \\ &= 1 - \frac{SS_w}{SS_t},\end{aligned} \tag{13}$$

where SS_b equals the sum of squares *between* the *k* groups, SS_w is the sums of squares *within* each of the *k* groups, and SS_t is the total sum of squares. The last expression of Equation 13 shows that computing both SS_w and SS_t in a stream suffices to compute the desired effect size.

Equation 6 already presented how sums of squares can be computed in data streams. The only complexity introduced in the ANOVA example is the computation of the sums of squares within each of the *k* groups. This requires computing the average within group *k*:

$$\bar{x}_k := \bar{x}_k + \frac{x_{k,n} - \bar{x}_k}{n_k}. \tag{14}$$

which is only updated once a data point $x_{(k,n)}$ originates from group *k*. Subsequently, Equation 6 is used within each group *k*, substituting $\bar{x}$ with $\bar{x}_k$ to compute SS_w. The computation of the effect size or proportion of variance explained (η^2) in a data stream thus requires the following parameters:

$$\theta := \left\{ \begin{array}{c} \bar{x}_k, n_k \\ \bar{x}, n, SS_t, SS_w \end{array} \right\}, \tag{15}$$

where the top row of θ indicates the parameters at the *group* level (and hence need to be kept in memory for each group *k*) and the bottom row indicates the global parameters, which are only single parameters which need to be stored in memory. Thus, in total θ contains 2*k* + 4 elements. We have implemented the online computation of η^2 in a function named *etasq_online*. This function will compute η^2 when two or more groups are available. Note that this function also requires two inputs: the data point and to which group this data point belongs. New groups can easily be included during the stream, without the data analyst interfering in the analysis.

In order to compute the *F*-statistic online, one can use the information that is already available:

$$F = \frac{(SS_t - SS_w)/(k-1)}{SS_w/(n-k)}. \tag{16}$$

It is important to note that repeated testing until a certain small *p*-value is found, which might be attractive if results are available for each new data point, is considered a questionable research practice (Armitage, McPherson, & Rowe, 1969; John, Loewenstein, & Prelec, 2012; Simmons, Nelson, & Simonsohn, 2011), due to inflation of Type 1 error. For instance, when a researcher decides to collect more data based on whether the ANOVA is significant 10 times with significance level of 5% while new data are entering, the actual Type 1 error equals

$$\alpha = 1 - (1 - 0.05)^{10} = 0.401,$$

instead of the 5% she started with. Perhaps the most common correction of this inflated Type I error is known as Bonferroni correction (Armstrong, 2014). Effectively this correction decreases "α" as a function of the number of tests to prevent an increase of Type I error.

We will continue our discussion of online learning methods with Stochastic Gradient Descent (SGD). SGD is an optimization method which is useful to estimate more complex models when an analytical solution is not available.

Online Estimation Using Stochastic Gradient Descent

In the previous section we have discussed how to estimate, fully online, a number of statistics and models that are often used in the analysis of sociological and psychological data. We have also illustrated the computational advantages of online estimation for very large datasets and data streams. However, for each of the methods discussed above we could derive exact summation methods; using simple algebra it was possible to transform standard estimation methods to online variants. Unfortunately, this is not always the case. Many estimation methods, especially those that require multiple iterations through a static dataset, cannot exactly be implemented online, in part because even when using conventional offline analysis the estimation is approximate. Examples are logistic regression or multilevel models. However, this does not mean that we can only estimate very simple models online: there is a multitude of methods available for the online estimation of more complex models. In this section we focus on Stochastic Gradient Descent (SGD), a general online estimation method that can be used for the estimation of more complex statistical models.

To explain SGD, we will first discuss Gradient Descent (GD), an optimization method that is often used in conventional offline analysis and provides a logical starting point for SGD. We provide a general intuition to GD/SGD and subsequently provide the technical details. Lastly, we will provide an applied example of fitting a logistic regression model using SGD, for which [R] code is provided in Appendix C.

Offline Gradient Descent

There are multiple ways to obtain estimates for parameters of statistical models, for instance using a least-squares approach (see, e.g., Bretscher, 1995), using Maximum Likelihood (ML) estimation, or using the method of moments (e.g., Arvas & Sevgi, 2012). In the social sciences we often use the maximum likelihood framework (see, for an introduction, Myung, 2003). In the maximum likelihood framework, we want to obtain the parameter values which maximize the probability of the data. Assuming independent observations, the likelihood function for many models takes the following form:

$$\mathscr{L}(\zeta|x_1,\ldots,x_n)=\prod_{i=1}^{n} f(x_i|\zeta), \tag{17}$$

where ζ is a set of parameters, $f()$ is a probability density function (PDF) (or probability mass function in the discrete case), and as before $x_1, \ldots, x_n$ denote the observations. In words, Equation 17 states that the likelihood of ζ given the observed data is a product of the individual probabilities of each of the data points. In practice it is often (much) simpler to obtain the maximum likelihood estimates by taking the logarithm of the likelihood:

$$\ell(\zeta|x_1,\ldots,x_n)=\sum_{i=1}^{n}\ln f(x_i|\zeta), \tag{18}$$

which effectively replaces the product term with a sum and gives the same solution for the maximum since the logarithm is a monotonic function. For some models, obtaining a maximum likelihood estimate analytically, after the log-likelihood, is defined as straightforward: we take the derivative of the log-likelihood, set it to zero, and solve for the parameters to obtain the required estimates. Effectively this has already been demonstrated: the estimation of the sample mean, and of linear regression models as discussed in previous sections, actually *are* analytical maximum likelihood estimates given the appropriate models (see, e.g., Gelman, 2007).

However, exact analytical solutions are not always available. In such cases one can resort to *approximate* methods, which are also frequently applied in offline analysis. One such approximate algorithm is called Gradient Descent, or actually Gradient *Ascent* because we use it in the context of a likelihood function which we want to *maximize*. Gradient Descent is the name most often used in the machine learning literature and classically used to *minimize* the error.

The GD algorithm can be stated as follows:

$$\zeta := \zeta + \lambda\nabla\ell(\zeta|x_1,\ldots,x_n), \tag{19}$$

where λ is a learn rate (also known as step size) chosen by the researcher and $\nabla\ell()$ denotes the *gradient* (vector of first-order derivatives) of the log-likelihood function. Intuitively this algorithm states that one chooses starting values for each parameter and evaluates the gradient using these values. In the simple case where ζ is scalar, and the gradient simplifies to the derivative, this evaluation gives information regarding the slope of the log-likelihood function: if the slope is positive[1] then the maximum can be found at higher values of ζ and we can make a step toward higher values of ζ. If the slope at ζ is negative, we need to step in the opposite direction: we need to choose a lower value. Using this intuition, GD iteratively – passing through the dataset multiple times – takes steps toward the maximum of the log-likelihood function. In the case that ζ is a vector, GD takes a step in q dimensions: for each parameter (i.e., dimension)

[1] Here we are assuming the log-likelihood function to be well behaved.

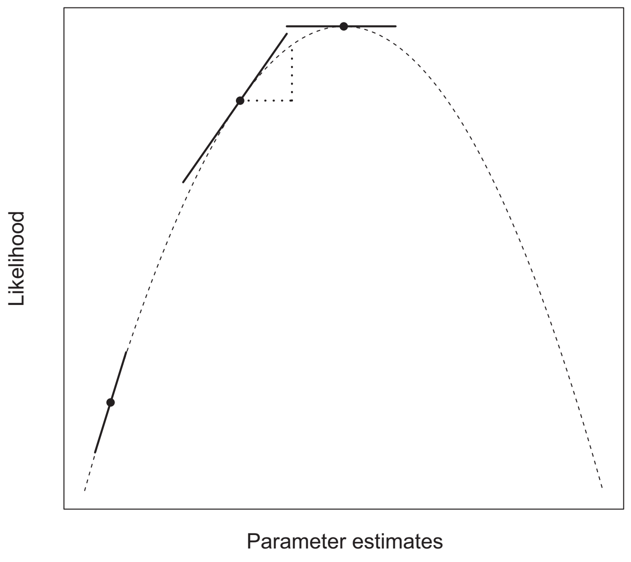

Figure 2. Graphical display of the likelihood function. In cases where the direct maximization of the likelihood function is difficult, an algorithm such as GD can be used to find the maximum. GD uses the slope of the tangent and a learn rate to make steps toward the maximum of the function.

GD determines whether the slope of the derivative is positive or negative, accordingly GD takes a step in the q dimensional space which causes the steepest ascent toward the maximum of the likelihood function.

Figure 2 provides an illustration of a gradient in a single dimension (e.g., the derivative). On the x-axis are possible parameter values and on the y-axis is the likelihood. The dashed curve is the likelihood of a given parameter value. At each point on the curve we can evaluate the first-order derivative. Figure 2 presents three evaluations of the first-order derivative, including the tangent lines at each of the three points (solid black lines). When the derivative has a positive number, the likelihood increases by increasing the parameter value. Opposite, if the derivative has a negative value, the slope is negative, and the likelihood increases by decreasing the parameter value. Obviously, the aim is to find the parameter value where the derivative is equal to zero, in order to find the maximum. The second evaluation of the derivative in Figure 2 contains two dotted lines. These dotted lines illustrate how the next evaluation is chosen. GD increases the value of the parameter when the result of the evaluation is positive and visa versa when the result is negative. The magnitude of the derivative together with the learn rate influence how much the parameter value is changed.

Gradient Descent can be a very effective method of finding the maximum likelihood value of ζ, although it is not without difficulties. For example, the parameter λ controls the size of the steps and has to be chosen carefully: a learn rate which is too large can be problematic since the algorithm could make jumps over the maximum likelihood solution. A learn rate which is too small causes the algorithm to take very small steps, and thus many iterations will be needed to obtain the maximum likelihood estimate. It depends on the model (e.g., complexity of the model, complexity of the likelihood function, etc.) as to what learn rate will be appropriate. One can choose for either a *fixed* learn rate or a learn rate which is adaptive, for instance one could choose to let the learn rate decrease with the number of iterations. A more extensive discussion on choosing the appropriate learn rate for complex models can be found in Wilson and Martinez (2003) and Bottou (2010).

Online or Stochastic Gradient Descent

Gradient Descent provides an iterative, approximate method to find maximum likelihood estimates. Deriving an effective *online* version of GD is as follows: instead of iterating over the full dataset multiple times and updating ζ each iteration, the algorithm takes a small step to a more likely parameter value every time a data point enters:

$$\zeta := \zeta + \lambda \nabla \ell_n(\zeta | x_n), \tag{20}$$

where we use ℓ_n to denote that we are evaluating the log-likelihood function. Hence, instead of updating our parameters based on iterations using all data, we update based on each arriving data point. SGD will converge to an unbiased estimate of the parameters as long as the order in which the data points arrive is random (Bottou, 2010). This means that the process that generates the data, does not change over the period in which the data are arriving.

Note that in the case that the dataset is no longer augmented, SGD can still be a useful tool: Analyzing static Big Data using SGD circumvents that the entire dataset needs to be available in memory. By simulating that the data enter a point at a time and letting the data stream in repeatedly, SGD can obtain unbiased estimates while still estimating the parameters without seeing all data at once.

Logistic Regression: An Example of the Usage of SGD

We present an example to illustrate SGD in which we are interested in the effect of independent variables on a binary dependent variable. In applied research, dependent variables are often binary, examples of which include whether and how people intend to vote (democrat vs. republican, Anderson, 2000), or whether or not people smoke cigarettes (Emmons, Wechsler, Dowdall, & Abraham, 1998). In the case of a binary dependent variable, often a logistic regression model is chosen to describe the relationship between a binary dependent variable and continuous independent variables:

$$\Pr(y = 1|\mathbf{X}) = p(\mathbf{X}) = \frac{\exp(\mathbf{X}\beta)}{1 + \exp(\mathbf{X}\beta)}, \tag{21}$$

where p is the probability to score a 1 on y, $\Pr(y = 1|\mathbf{X})$ and is modeled as a function of $\mathbf{X}$. Unlike linear regression, logistic regression does not have a closed form solution to estimate the parameters β using a maximum likelihood approach, and hence even for offline analysis approximate methods are used. Estimating the parameters online can be done using SGD as follows. First, we specify the log-likelihood:

$$\ell(\beta) = \sum_{i=1}^{n} y_i \log p(\mathbf{x}_i) + (1 - y_i)\log(1 - p(\mathbf{x}_i)). \tag{22}$$

Second, we compute the gradient (see Agresti, 2002, for more details):

$$\frac{\partial \ell}{\partial \beta} = \sum_{i=1}^{n} (y_i - p(\mathbf{x}_i))\mathbf{x}_i, \tag{23}$$

which in the case of offline estimation would be evaluated for all the data at once. When we use SGD to estimate the β's, the following online algorithm is obtained:

$$\begin{aligned} \theta &= \{\beta, \lambda\}, \\ \lambda &:= \lambda + f(\lambda, \mathbf{x}_n), \\ \beta &:= \beta + \lambda(y_n - p(\mathbf{x}_n))\mathbf{x}_n. \end{aligned} \tag{24}$$

Here we include λ, the learn rate, in θ. The inclusion of lambda will not be necessary for a fixed value of λ, but it highlights that the learn rate could be a function of the data stream. Given an appropriate choice of λ, and a large enough data stream, SGD will correctly estimate the parameters of interest (Bottou, 2010). See Appendix C for an implementation of SGD for the estimation of logistic regression in [R]. We implemented SGD for logistic regression in [R], the function is called *sgd_log* and can be used to estimate logistic regression in a stream.

Online Learning in Practice: Logistic Regression in a Data Stream

Switching to a Safe Well

To illustrate a logistic regression in a data stream, we use an example dataset, described in Gelman (2007). The dataset contains information regarding households in Bangladesh and whether or not they switch to a safe well to collect water. The wells were labeled safe if the arsenic level was low enough. Five years after the labeling of the wells, researchers collected data to study how many households had switched from their own unsafe well to another safe well. Switching to another well was dependent on whether owners of a safe well were willing to share their safe well and whether the households that did have an unsafe well were willing to make some extra effort to go to the safe well. The relatively small dataset consists of N = 3,020 households. Among other variables, the dataset includes the distance in meters to a safe well (X_{dist}) from the household, and arsenic level that is present in the water (X_{ars}).

In practice, choosing which variables to include from often a large set of variables, could be a challenging task on its own. Methods to deal with variable selection in a data stream are for instance the online Lasso (Yang, Xu, King, & Lyu, 2010) or based on Ridge regression (Tarrès & Yao, 2014). For the current example, we simulate that the data enter a point at a time by analyzing the data row-by-row using both offline and online implementations to predict whether the household switched to a safe well (coded 1) or did not switch (coded 0). The model we estimate contains two independent variables and an interaction term:

$$\begin{aligned} &\Pr(y = 1|X_{\text{dist}}, X_{\text{ars}}) \\ &\quad = \frac{\exp(b_0 + b_1 X_{\text{dist}} + b_2 X_{\text{ars}} + b_3 X_{\text{dist}} X_{\text{ars}})}{1 + \exp(b_0 + b_1 X_{\text{dist}} + b_2 X_{\text{ars}} + b_3 X_{\text{dist}} X_{\text{ars}})}. \end{aligned} \tag{25}$$

We thus estimate the four coefficients b_0, b_1, b_2, and b_3. The starting values for all four β's are zero.

Results

In Figure 3 we present the results of fitting a logistic regression in a data stream with four coefficients and an adaptive learn rate, $\lambda = \frac{1}{\sqrt{n}}$. The x-axes present the data stream and the y-axes present the estimated parameter values. During the stream we monitored the estimated parameter values using a moving average of 100 data points. These moving averages are presented in Figure 3.

The estimates of the effect of the arsenic level (b_2) and the interaction term (b_3) are very accurate from the beginning of the data stream. The estimates of the intercept (b_0) and the effect of the distance to the next safe well (b_1) require some more data. The dashed line is fluctuating, even toward the end of the dataset: this is due to the fact that the learn rate is still quite large ($\frac{1}{\sqrt{3020}}$ =0.018) for a dataset this size. A smaller learn rate (or one that decreases more rapidly) would stabilize the SGD algorithm more, but increases the risk of introducing more bias.

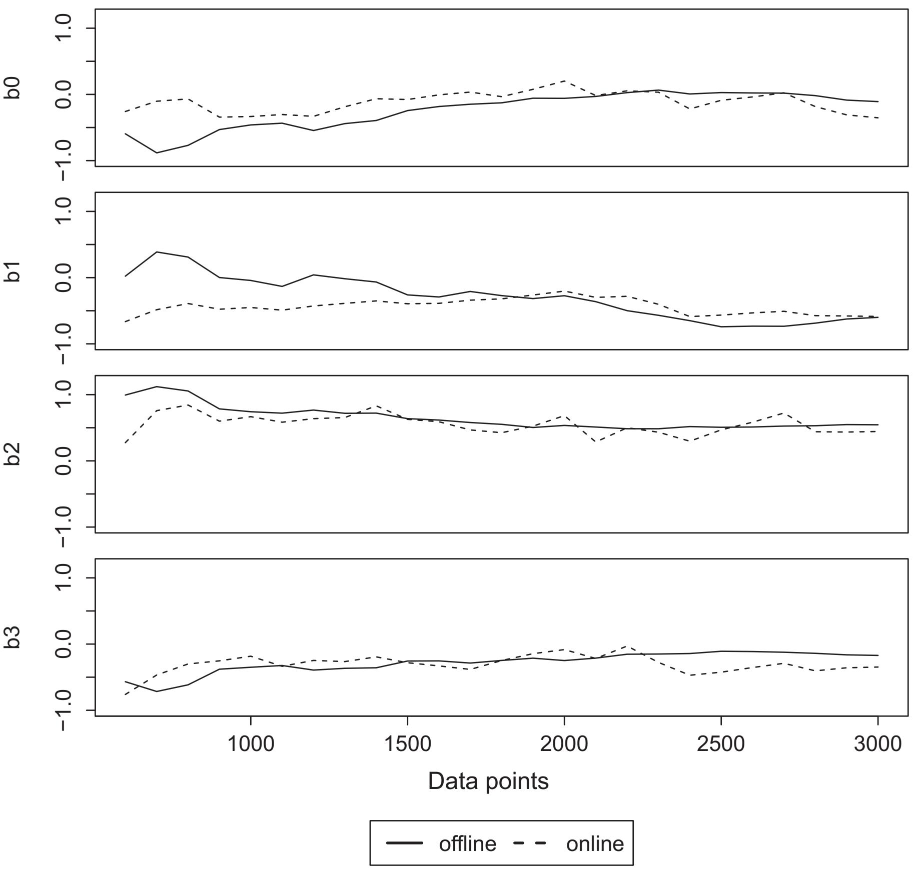

Figure 3. Online (dotted) and offline (solid) estimated beta coefficients of logistic regression as more data enter.

Learn Rates

To gain some insight into the sensitivity of SGD to its learn rates we also present the results of four different rates: .1, .01, .001, and $1/n$. We present the results of the four learn rates for the intercept, though the learn rates were equal for all coefficients. Again, the x-axis presents the data stream and the y-axis the estimated parameter value. Figure 4 presents the moving average of 100 data points of the parameter estimate during the stream. Clearly the curve with the largest learn rate shows much more fluctuation. Much of this fluctuation is already gone when we lower the learn rate to .01, although the online estimation of the intercept remains close to the offline estimation of the intercept. All fluctuation has gone for the two smallest learn rates. These two are a clear example of learn rates that are too low. In such cases the estimates do not, or hardly change.

Starting Values

Lastly we present the results of starting the analysis with different starting values in Figure 5. On the x-axis is the data stream presented, the y-axis is the estimated parameter value, and the lines are the moving average of 100 data points of the estimated parameter values. While the intercept had starting values {−2, −1, 1, 2}, the remaining coefficients had starting values equal to zero and the learn rate remained $1/\sqrt{n}$. Here we present the influence of the starting values on the final parameter estimate. Although there is some difference between the four lines, all four of them result in very similar parameter estimates. This illustrates that SGD does not really depend (given an appropriate learn rate) on the starting values and that the data dominate the results quickly.

For larger datasets (and for continuous streams, which is what we primarily focused on in this paper) the performance of SGD is often accurate.

Considerations Analyzing Big Data and Data Streams

In this paper we have discussed online learning as a way to deal with Big Data. However, some important issues remain. Here we discuss two practical and two conceptual issues related to analyzing Big Data.

Practically, it has to be noted that at this moment not many off-the-shelf statistical packages are available to

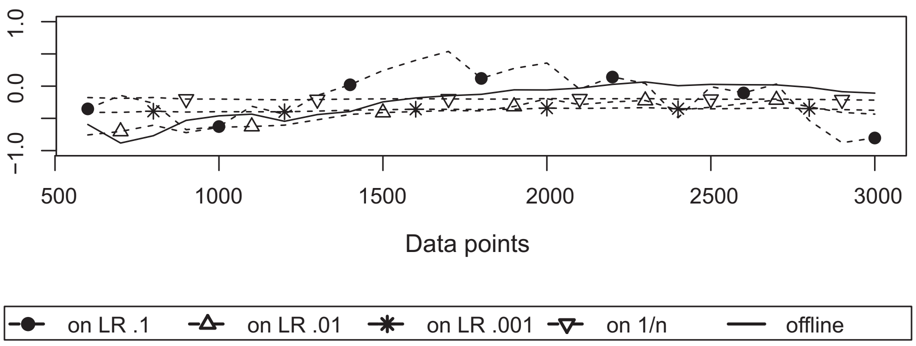

Figure 4. Online (dotted) and offline (solid) estimated intercepts for learn rates: .1, .01, .001, and 1/*n* coefficients of logistic regression as more data enter.

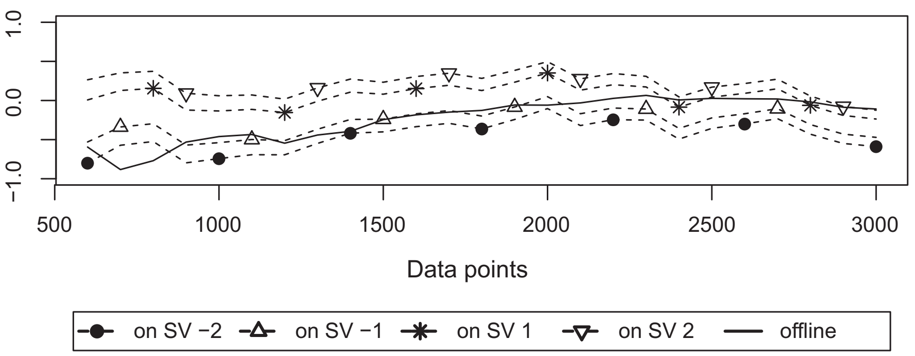

Figure 5. Online (dotted) and offline (solid) estimated intercepts for starting values: −2, −1, 1, and 2 coefficients of logistic regression as more data enter.

actually analyze data streams. The currently available software, for instance (and not exhaustive) *Apache Storm* (Toshniwal et al., 2014) *Apache Spark* (Karau, Konwinski, Wendell, & Zaharia, 2015), *RStorm* (Kaptein, 2014), *S4* (Neumeyer, Robbins, Nair, & Kesari, 2010), *RapidMiner* (Hofmann & Klinkenberg, 2013), *KNIME* (Berthold et al., 2009), and *MOA* (Bifet, Holmes, Kirkby, & Pfahringer, 2010), often require extensive programming knowledge and focus mainly on the infrastructure of analyzing large datasets. There is still a large gap between the methods and software developed by computer scientists, and those that can be used by social scientists to analyze their data streams using models that they are accustomed to.

Second, we have to stress that for the application of online methods the analyst has to know beforehand what type of analysis and model is required to answer the research question. Online learning methods make use of a limited set of quantities – referred to the elements of θ throughout this text – to store the relevant information and to subsequently estimate model parameters. This means that it is important to know what information is required *before* the analysis. Any information that is not stored is forgotten and is impossible to retrieve if the data themselves are not stored.

A solution to this latter issue could be to run simultaneously different analyses and/or models, such that at a later point in time a decision can be made as to which analysis or model to use. This, of course, does require that enough computer memory is available to store the sufficient statistics of multiple models. A frequently adopted practical solution to this in the computer science literature is to adopt a so-called λ-architecture (Marz & Warren, 2013): the data stream is operated online (for those computations that were specified in advance), but also stored and can thus be analyzed offline at a later point in time (often using parallelization methods to deal with the size of the dataset).

From a conceptual point of view, we do explicitly mention that we are not promoting repeated null hypothesis significance testing in data streams; this should be avoided. When a researcher decides to stop the data collection once she obtains a significant result of the hypothesis test, the Type I error rate increases above nominal level (i.e., too many false positives, Strube, 2006). It is considered a questionable research practice to repeatedly test for a significant effect and stop data gathering once the effect yields a $p < .05$ (John et al., 2012). When adopting an online learning approach we encourage researchers to focus on obtaining precise estimates of the size of the effects of interest, in adherence to the APA guidelines (Affairs & the Task Force on Statistical Inference, 1999), as opposed to null hypothesis testing.

Finally, it is not always feasible to translate all analyses from the offline framework to the online framework. For instance, the analysis of binary dependent data, data that are nested within units, which are in the offline case often analyzed using logistic multilevel (or random effects-) models, have not yet found a proper online synonym. Therefore future research should be aimed at translating complex models, such as logistic multilevel models, to the online learning framework. Note that active research work is carried out in this field, with, for example, recent publications describing online approximations of the well-known Expectation-Maximization algorithm (Cappé & Moulines, 2009).

Discussion

Using new data collection methods and technologies, for instance experience sampling (Barrett & Barrett, 2001), to collect social and psychological data has made data streams more apparent and more prevalent in recent years. In this paper we discussed how social scientists can deal with these large datasets, and how regular estimation methods can be applied in the context of continuous data streams. We hope to have contributed to opening up the possibilities to answer both existing research questions as well as new types of research questions using large datasets or continuous data streams. Note that we have only touched upon a few methods which are used to analyze data streams; there are many more techniques available to analyze data streams, for instance the Bayesian framework can in some cases also be used to update the estimated parameters (e.g., Gelman, Carlin, Stern, & Rubin, 2004). In the case of conjugate priors, the posterior can be updated relatively easily. However, in the situation where the prior is not conjugate, other methods such as particle filtering are required to update the posterior (Robert, 2015).

Despite the versatility of online methods as displayed in this tutorial, many challenges remain: common methods such as (latent) factor analysis, mixture models, or multilevel models are not easily estimated online (see for a discussion and online approaches for multilevel models, Ippel et al., 2016). A possible way to deal with these types of analyses is to alter for instance the EM algorithm (Dempster, Laird, & Rubin, 1977). Suggestions for parallel computations and more efficient procedures for the EM algorithm have already been proposed (Cappé & Moulines, 2009; Neal & Hinton, 1998; Wolfe, Haghighi, & Klein, 2008), and this work should be extended to make the EM algorithm applicable for streaming data.

We hope that the current article motivates applied researchers to explore new research areas that are opened up by the technological opportunity to monitor individuals in a data stream. We believe that data streams can provide social scientists with many new insights into human behavior and can provide new research areas to study human emotions and attitudes.

References

Affairs, L. W., the Task Force on Statistical Inference. (1999). Statistical methods in psychology journals. *The American Psychologist, 54*, 594–604. doi: 10.1037/0003-066X.54.8.594

Agresti, A. (2002). *Categorical data analysis*. Gainesville, FL: Wiley Series in Probability and Statistics.

Anderson, C. J. (2000). Economic voting and political context: A comparative perspective. *Electoral Studies, 19*, 151–170. doi: 10.1016/S0261-3794(99)00045-1

Armitage, P., McPherson, C., & Rowe, B. (1969). Repeated Significance Tests on accumulating data. *Journal of the Royal Statistical Society, 132*, 235–244. doi: 10.2307/2343787

Armstrong, R. A. (2014). When to use the Bonferroni correction. *Ophthalmic & Physiological Optics: The Journal of the British College of Ophthalmic Opticians (Optometrists), 34*, 502–508. doi: 10.1111/opo.12131

Arvas, E., & Sevgi, L. (2012). A tutorial on the method of moments. *Antennas and Propagation Magazine, IEEE, 54*, 260–275. doi: 10.1109/MAP.2012.6294003

Atallah, M. J., Cole, R., & Goodrich, M. T. (1989). Cascading divide-and-conquer: A technique for designing parallel algorithms. *SIAM Journal on Computing, 18*, 499–532. doi: 10.1137/0218035

Barrett, L. F., & Barrett, D. J. (2001). An Introduction to computerized experience sampling in psychology. *Social Science Computer Review, 19*, 175–185. doi: 10.1177/089443930101900204

Berthold, M. R., Cebron, N., Dill, F., Gabriel, T. R., Kötter, T., Meinl, T., . . . Wiswedel, B. (2009). KNIME: The Konstanz Information Miner - Version 2 and beyond. In *SIGKDD Explor. Newsl. 11*(1), 26-31. doi: 10.1145/1656274.1656280

Bifet, A., Holmes, G., Kirkby, R., & Pfahringer, B. (2010). MOA: Massive online analysis. *The Journal of Machine Learning Research, 11*, 1601–1604.

Bottou, L. (1998). Online algorithms and stochastic approximations. In D. Saad (Ed.), *Online learning and neural networks*. Cambridge, UK: Cambridge University Press.

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In Y. Lechevallier & G. Saporta (Eds.), *Proceedings of the 19th International Conference on Computational Statistics (COMPSTAT'2010)* (pp. 177–187). Paris, France: Springer. doi: 10.1007/978-3-7908-2604-3_16

Bretscher, O. (1995). *Linear algebra with applications* (4th ed.). New Jersey, USA: Pearson.

Cappé, O., & Moulines, E. (2009). Online expectation-maximization algorithm for latent data models. *Journal of the Royal Statistics Society: Series B (Statistical Methodology), 71*, 593–613. doi: 10.1111/j.1467-9868.2009.00698.x

Carmona, C. J., Ramírez-Gallego, S., Torres, F., Bernal, E., Del Jesus, M. J., & García, S. (2012). Web usage mining. *Expert Systems With Applications, 39*, 11243–11249. doi: 10.1016/j.eswa.2012.03.046

Chu, C., Kim, S. K., Lin, Y., & Ng, A. Y. (2006). Map-reduce for machine learning on multicore. In B. Schölkopf, J. C. Platt, & T. Hoffman (Eds.), *Advances in neural information processing systems*. (Vol. 19, p. 281). Vancouver, British Columbia, Canada: Massachusetts Institute of Technology.

Datar, M., Gionis, A., Indyk, P., & Motwani, R. (2002). Maintaining stream statistics over sliding windows. *SIAM Journal on Computing, 31*, 1794–1813. doi: 10.1137/S0097539701398363

Demchenko, Y., Grosso, P., De Laat, C., & Membrey, P. (2013). Addressing Big Data issues in Scientific Data Infrastructure. *Proceedings of the 2013 International Conference on Collaboration Technologies and Systems, CTS 2013*, 48–55. San Diego, CA, USA: IEEE. doi: 10.1109/CTS.2013.6567203

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society: Series B (Methodological), 39*, 1–38. doi: 10.1.1.133.4884

Efraimidis, P. S., & Spirakis, P. G. (2006). Weighted random sampling with a reservoir. *Information Processing Letters, 97*, 181–185. doi: 10.1016/j.ipl.2005.11.003

Emmons, K. M., Wechsler, H., Dowdall, G., & Abraham, M. (1998). Predictors of smoking among US college students. *American Journal of Public Health, 88*, 104–107. doi: 10.2105/AJPH.88.1.104

Escobar, L., & Moser, E. (1993). A Note on the Updating of Regression Estimates. *The American Statistician, 47*, 192–194. doi: 10.1080/00031305.1993.10475974

Gaber, M. M., Zaslavsky, A., & Krishnaswamy, S. (2005). Mining data streams: A review. *SIGMOD, 34*, 18–26. doi: 10.1145/1083784.1083789

Gelman, A. (2007). Rich state, poor state, red state, blue state: What's the matter with Connecticut? *Quarterly Journal of Political Science, 2*, 345–367. doi: 10.1561/100.00006026

Gelman, A., Carlin, J., Stern, H., & Rubin, D. (2004). *Bayesian data analysis* (2nd ed.). London, UK: CRC Press.

Hofmann, M., & Klinkenberg, R. (2013). *RapidMiner: Data mining use cases and business analytics applications*. Boca Raton, Florida, USA: Chapman & Hall/CRC.

Ippel, L., Kaptein, M., & Vermunt, J. (2016). Estimating random-intercept models on data streams. *Computational Statistics and Data Analysis, 104*, 169–182. doi: 10.1016/j.csda.2016.06.008

John, L. K., Loewenstein, G., & Prelec, D. (2012). Measuring the prevalence of questionable research practices with incentives for truth telling. *Psychological Science, 23*, 524–532. doi: 10.1177/0956797611430953

Kaptein, M. (2014). fRStormg: Developing and testing streaming algorithms in fRg. *The R Journal, 6*, 123–132.

Karau, H., Konwinski, A., Wendell, P., & Zaharia, M. (2015). *Learning spark*. Sebastopol, CA, USA: O'Reilly Media.

Killingsworth, M. A., & Gilbert, D. T. (2010). A wandering mind is an unhappy mind. *Science, 330*, 932. doi: 10.1126/science.1192439

Marz, N., & Warren, J. (2013). *Big Data: Principles and best practices of scalable realtime data systems*. New York, USA: Manning Publications.

Myung, I. (2003). Tutorial on maximum likelihood estimation. *Journal of Mathematical Psychology, 47*, 90–100. doi: 10.1016/S0022-2496(02)00028-7

Neal, R., & Hinton, G. E. (1998). A view of the Em algorithm that justifies incremental, sparse, and other variants. In M. I. Jordan (Ed.), *Learning in Graphical Models* (pp. 355–368). Netherlands: Springer.

Neumeyer, L., Robbins, B., Nair, A., & Kesari, A. (2010). S4: Distributed stream computing platform. In *Proceedings – IEEE International Conference on Data Mining, ICDM* (pp. 170–177) Sydney, Australia: IEEE. doi: 10.1109/ICDMW.2010.172

Opper, M. (1998). A Bayesian approach to online learning. In D. Saad (Ed.), *On-line learning in neural networks* (pp. 363–378). Cambridge, UK: Cambridge University Press.

Pèbay, P. (2008, September). Formulas for robust, one-pass parallel computation of covariances and arbitrary-order statistical moments. *Sandia Report*1–18. (SAND2008-6).

Plackett, R. (1950). Some theorems in least squares. *Biometrika, 37*, 149–157. doi: 10.1093/biomet/37.1-2.149

Robert, C. P. (2015). *The Metropolis-Hastings algorithm (Mcmc)*, 1–15. doi: 10.1002/9781118445112.stat07834

Sagiroglu, S., & Sinanc, D. (2013). Big Data: A review. In *International Conference on Collaboration Technologies and Systems (CTS)* (pp. 42–47). San Diego, CA, USA: IEEE. doi: 10.1109/CTS.2013.6567202

Shalev-Shwartz, S. (2011). Online learning and online convex optimization. *Foundations and Trends® in Machine Learning, 4*, 107–194. doi: 10.1561/2200000018

Sherman, J., & Morrison, W. J. (1950). Adjustment of an Inverse Matrix Corresponding to a Change in One Element of a Given Matrix. *The Annals of Mathematical Statistics, 21*, 124–127. doi: 10.1214/aoms/1177729893

Simmons, J. P., Nelson, L. D., & Simonsohn, U. (2011). False-positive psychology: Undisclosed flexibility in data collection and analysis allows presenting anything as significant. *Psychological Science, 22*, 1359–1366. doi: 10.1177/0956797611417632

Strube, M. J. (2006). SNOOP: A program for demonstrating the consequences of premature and repeated null hypothesis testing. *Behavior Research Methods, 38*, 24–27. doi: 10.3758/BF03192746

Swendsen, J., Ben-Zeev, D., & Granholm, E. (2011). Real-time electronic ambulatory monitoring of substance use and symptom expression in schizophrenia. *The American Journal of Psychiatry, 168*, 202–209. doi: 10.1176/appi.ajp.2010.10030463

Tarrès, P., & Yao, Y. (2014). Online learning as stochastic approximation of regularization paths. *IEEE Transactions on Information Theory, 60*, 5716–5735. doi: 10.1109/TIT.2014.2332531

Thiesson, B., Meek, C., & Heckerman, D. (2001). Accelerating EM for large databases. *Machine Learning, 45*, 279–299. doi: 10.1023/A:1017986506241

Toshniwal, A., Donham, J., Bhagat, N., Mittal, S., Ryaboy, D., Taneja, S., . . . Fu, M. (2014). Storm@twitter. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data – SIGMOD '14* (pp. 147–156). New York, USA: ACM. doi: 10.1145/2588555.2595641

Turaga, D., Andrade, H., Venkatramani, C., Verscheure, O., Harris, J. D., Cox, J., . . . Jones, P. (2010). Design principles for developing stream processing applications. *Software: Practice and Experience, 40*(12), 1073–1104. doi: 10.1002/spe.993

Welford, B. (1962). Note on a method for calculating corrected sums of squares and products. *Technometrics, 4*, 419–420. doi: 10.1080/00401706.1962.10490022

Whalen, C. K., Jamner, L. D., Henker, B., Delfino, R. J., & Lozano, J. M. (2014). The ADHD spectrum and everyday life: Experience sampling of adolescent moods, activities, smoking, and drinking. *Child Development, 73*, 209–227. doi: 10.1111/1467-8624.00401

Wilson, D., & Martinez, T. R. (2003). The general inefficiency of batch training for gradient descent learning. *Neural Networks, 16*, 1429–1451. doi: 10.1016/S0893-6080(03)00138-2

Wolfe, J., Haghighi, A., & Klein, D. (2008). Fully distributed EM for very large datasets. In *Proceedings of the 25th international conference on Machine learning – ICML '08* (pp. 1184–1191). New York, USA: ACM.

Yang, H., Xu, Z., King, I., & Lyu, M. R. (2010). Online learning for group lasso. In J. Fürnkranz & T. Joachims (Eds.), *Proceedings of the 27th international conference on machine learning* (pp. 1191–1198). Haifa, Israel: Omnipress.

Received December 8, 2015
Revision received August 5, 2016
Accepted August 19, 2016
Published online December 5, 2016

Lianne Ippel is working as a PhD in methods and statistics at Tilburg University, since 2013. Her research focuses on how traditional statistical models – especially those dealing with dependencies between observations – can be used in the context of data streams.

Maurits Kaptein is an assistant professor of statistics at Tilburg University. Previously, worked at the Aalto School of Business, and at Stanford University. Maurits' research explores methods for content personalization. His work has been published in leading journals such as Behavior Research Methods, and Bayesian Analysis.

Jeroen K. Vermunt is a full professor in the Department of Methodology and Statistics at Tilburg University. In 2005, he received the Leo Goodman early career award from the methodology section of the American Sociological Association. He is the co-developer (with Jay Magidson) of the Latent GOLD software package.

Lianne Ippel
Methodology and Statistics
Tilburg University
Warandelaan 2
postbus 90153
5000 LE Tilburg
The Netherlands
G.J.E.Ippel@uvt.nl

Appendix A

Online Correlation

```
> N <– 1000                                   #number of observations
> x <– rnorm(N, 5, 2)                         #generate data
> y <– 1.5*x+rnorm(N)
> # because a correlation requires at least 2 points we start with n=1
> n = 1; xbar = x[1]; ybar = y[1]; SC = 0; SSx = 0; SSy = 0;
> for (i in 2:N)
+ {
+     dx    <– (x[i]–xbar)                    #deviance x
+     dy    <– (y[i]–ybar)                    #deviance y
+     n     <– n+1                            #update number of observations
+     xbar  <– xbar+(x[i]–xbar) / n           #update mean x
+     SSx   <– SSx+dx*(x[i]–xbar)             #update sum of squares for x
+     SC    <– SC+(x[i]–xbar)*(y[i]–ybar)     #update sum of cross products
+     Sxy   <– SC / (n–1)                     #compute covariance
+     ybar  <– ybar+(y[i]–ybar) / n           #update mean y
+     SSy   <– SSy+dy*(y[i]–ybar)             #update sum of squares for y
+     sx    <– sqrt(SSx / (n–1))              #estimate std.dev. x
+     sy    <– sqrt(SSy / (n–1))              #estimate std.dev. y
+     rxy   <– Sxy / (sx*sy)                  #estimate correlation
+ }
>
```

Appendix B

Online Linear Regression

```
> N  <– 1000          # generate data
> x0 <– rep(1, N)
> x1 <– rnorm(N, 5, 2)
> x  <– matrix(c(x0, x1), nrow=N)
> y  <– 3+1.5*x[, 2]+rnorm(N)
> A  <– matrix(0, nrow=2, ncol=2); B <– c(0, 0)
> #the as.matrix and as.numeric are required to get [r] running
> for (i in 1:N)
+ {         #fit linear regression:
+    if (i<3)
+    {         #update A as long as it is not invertible
+       A          <− A+x[i,]%*%t(x[i,])
+    }             #update B
+    B             <− B + as.matrix(x[i,]) %*%y[i]
+       if (i==3)
+       {          #invert A when n>p
+          A_inv <− solve(A)
+       }
+       if (i>=3) #update inverted matrix A_inv
+       {         #C is a scalar
+          C      <− as.numeric((1+x[i,]%*% A_inv%*% [i,]))
+          A_inv <− A_inv − ((A_inv%*%x[i,]%*%x[i,]%*%A_inv) / C)
+          beta  <− A_inv%*%B #compute coefficients
+       }
+    }
>
```

Appendix C

Logistic Regression Using Stochastic Gradient Descent

```
> N       <−3000
> x       <−rnorm(N, 1, 1)
> e       <−rnorm(N)
> y       <−rbinom(N, 1, (exp(−2+1.5*x+e) / (1+exp(−2+1.5*x+e))))
> beta <− c(0, 0)
> for(i in 1:N)
+ {
+    p    <− exp(beta[1]+beta[2]*x[i]) / (1+exp(beta[1]+beta[2]*x[i]))
+    beta<−beta + lambda*(y[i]− p) %*%(1, x[i])
+ }
>
```

Appendix D

Wells Data Example: Logistic Regression Using Stochastic Gradient Descent

```
> beta             <− c(0, 0, 0, 0)
> for(i in 1:nrow(wells.dat))
+  {
+       n        <− n+1
+       x        <− c(1, wells.dat[i, c(dist, ars, dist*ars)])
+       y        <− wells.dat$switch[i]
+       p        <− exp(sum(beta*x)) / (1+exp(sum(beta*x)))
+       beta     <−beta+ 1 / sqrt{n}*(y − p) %*%x
+  }
>
```