Squashing Flat Files Flatter

William DuMouchel, Chris Volinsky, Theodore Johnson, Corinna Cortes & Daryl Pregibon
AT&T Labs-Research
{dumouchel,volinsky,johnsont,corinna,daryl}@research.att.com

Abstract

A feature of data mining that distinguishes it from "classical" machine learning (ML) and statistical modeling (SM) is *scale*. The community seems to agree on this yet progress to this point has been limited. We present a methodology that addresses scale in a novel fashion that has the potential for revolutionizing the field. While the methodology applies most directly to flat (row by column) data sets we believe that it can be adapted to other representations.

Our approach to the problem is not to *scale up* individual ML and SM methods. Rather we prefer to leverage the entire collection of existing methods by *scaling down* the data set. We call the method *squashing*. Our method demonstrably outperforms random sampling and a theoretical argument suggests how and why it works well.

Squashing consists of three modular steps: grouping, momentizing, and generating (GMG). These three steps describe the squashing pipeline whereby the original (very large data set) is sectioned off into mutually exclusive groups (or bins); within each group a series of low-order moments are computed; and finally these moments are passed off to a routine that generates *pseudo data* that accurately reproduce the moments.

The result of the GMG squashing pipeline is a squashed data set that has the same structure as the original data with the addition of a *weight* for each pseudo data point that reflects the distribution of the original data into the initial groups. Any ML or SM method that accepts weights can be used to analyze the weighted pseudo data. By construction the resulting analyses will mimic the corresponding analyses on the original data set.

Squashing should appeal to many of the sub-disciplines of KDD:

statistics: squashing generalizes the *sufficiency principle* to apply across parameter space and across model space,

database research: a squashed data set is a lossy materialized view of a large data set that is tuned for analyses that parallel how data cubes are tuned for fast aggregation queries,

algorithms: the steps in the GMG pipeline can individually be investigated and improved for both speed and accuracy,

machine learning: extending existing ML methods for weighted data and devising new methods that explicitly exploit this structure.

1 Introduction

One of the chief obstacles to effective data mining is the clumsiness of managing and analyzing a very large data set. The process of model search and model fitting often requires many passes over the data, yet it is infeasible to squeeze large data sets into physical memory.

There are two basic approaches to this problem: construct summaries of the large data set on which to base the desired analysis, or to analyze a random sample from the large data set. Each approach has disadvantages. It is hard to devise general-purpose summaries. If a particular model is desired, for example a set of multiple regression analyses, then statistical theory often suggests sufficient statistics that can be computed in a single pass over the large data file without holding the file in memory. But the appropriate summaries will depend on having the desired model fixed in advance. This is a classical chicken and egg problem: to specify the summary, you need a model; to iterate to a well fitting model, you need to mine the data.

The other strategy, drawing a random sample of the rows of the large data set, is easy to achieve. The sample can be analyzed with complete flexibility in choice of statistical method, unrestricted by a possibly unfortunate choice of summary statistics. The biggest disadvantage of this strategy is the inaccuracy introduced by sampling variance.

In this paper we introduce a technique we call *squashing* that attempts to combine the best features of a pre-processed data summary and random sampling.

Data squashing effectively summarizes a large data set with a smaller version (often by several orders of magnitude) having the same variables as the original data set. The squashed data set is constructed to emulate the multivariate distribution of the larger data set more accurately than random sampling. Each element of the squashed data set has a weight, where the sum of the weights equals the number of elements in the original data set. A theory based on a Taylor series approximation to the likelihood function suggests that a weighted analysis of the squashed data can provide accurate approximations to the results that would be found from fitting almost any smooth model to the larger data set.

Data squashing can be seen as a form of lossy database compression. A significant body of recent work in the database literature has examined methods for the lossy compression of databases, and especially data cubes (Barbara 1997). However, data squashing has a somewhat different goal. Lossy data cube compression can be judged acceptable if aggregate queries over ranges of the data cube have a small error. Data squashing is acceptable if a different type of query has a small error, e.g., the fitting of statistical models. Thus, the techniques proposed in this paper are valuable for statistical analysis data warehouses.

In what follows, we assume that the large data set has a simple "flat" structure, consisting of a very large number of rows or elements, each of a fixed number of attributes or variables. The variables are either categorical (nominal scale) or quantitative (measurement scale). The technique of data squashing consists of the following steps (applied sequentially):

Group the data in regions: Categorical variables induce natural bins. For quantitative variables we artificially create bins or regions either from the quantiles of the single variables or from the use of data spheres.

Calculate moments within regions: The number of moments calculated for the elements falling in the single region depends on what degree of Taylor series approximation is desired.

Generate squashed data elements and weights: For each region a set of squashed data elements are created. The squashed data set is constructed so its moments approximate those of the elements of the original data set falling in the region.

The squashed data set is smaller (by orders of magnitude) than the original data set. It preserves analysis flexibility without sacrificing accuracy due to sampling variability. The only restriction on analyzing the squashed data, compared to the methods available to analyze the original file, is that the analysis method or software must be able to make use of the induced weights.

The requirement that the squashed data summarize the large data set as well as possible is taken to mean that for all or almost all statistical models that might be fitted to the data, the result from a weighted model fit of the squashed data is nearly the same as if the same model had been fit to the original data. A naive attempt at creating a squashed data set is to take a 1% random sample of the data set and then add a weight variable equal to 100 for every row in the sample. However, parameter estimates created from this sample will typically differ from the corresponding full data estimates by about 10 standard errors (since standard errors of efficient estimates are inversely proportional to the square root of the sample size). The goal is to create squashed data sets that yield parameter estimates that are within one standard error of the full data estimates.

Another possible strategy for creating a squashed data set is to perform a cluster analysis on the original data set, where each element of the large data set is in exactly one cluster. Then the squashed file could consist of the centers of each cluster, and the weight variable would be the size of the corresponding cluster. There are two impediments to this approach. First, clustering algorithms are typically $O(n^2)$ or may require many passes over the data so as to render clustering infeasible for very large data sets (see however Bradley, Fayyad, and Reina 1998). Second, replacing each cluster by a single weighted point at the cluster mean implicitly reduces the overall variance of every quantitative variable in the squashed data set and would thus tend to distort the results of most statistical models.

In the following sections we first discuss the theoretical framework for data squashing. We then discuss the computational aspects of the GMG pipeline. Each step can be viewed as an individual module with its own choice of methodology and with its own trade-offs in speed, space, and accuracy. In particular for the grouping step we discuss how to avoid the curse of dimensionality by utilizing data spheres (Johnson and Dasu 1998). We apply data squashing to a real data set and demonstrate the flexibility of the method through the choice of plug-in modules. Finally we discuss related work and future research directions.

2 Theoretical framework for data squashing

In what follows, we assume that the large data set has a simple "flat" structure, consisting of a very large number, N, of rows or elements, each consisting of a fixed number of attributes or variables. The variables are either categorical (nominal scale) or quantitative (measurement scale).

Suppose the large data set has columns $A_1, \ldots, A_C$, $X_1, \ldots, X_Q$. The As are categorical variables, while the Xs are quantitative variables. Let $A = A_{ic}$ and $X = X_{ij}$; $i = 1, \ldots, N$; $c = 1, \ldots, C$; $j = 1, \ldots, Q$, denote the original $N \times C$ and $N \times Q$ data matrices of categorical and quantitative variables, respectively. Let the squashed data set, having $M \ll N$ rows, be represented by corresponding matrices $B = B_{ic}$ and $Y = Y_{ij}$; $i = 1, \ldots, M$; $c = 1, \ldots, C$; $j = 1, \ldots, Q$, and let w_i be the weight assigned to row i, where $\sum_{i=1}^{M} w_i = N$. Suppose further that a statistical model is being fit to the data. That is, a modeling assumption is being made that the large data set is the result of N independent draws from the probability model

$$f(a_1, \ldots, a_C, x_1, \ldots, x_Q; \theta) = \mathrm{P}(A_1 = a_1, \ldots, A_C = a_C, X_1 = x_1, \ldots, X_Q = x_Q | \theta), \tag{1}$$

where θ is a vector of parameters that are estimated from the data. The function f defines the particular model being fit, and we want to have a squashed version of the data that provides nearly identical analyses for a wide choice of f. By "identical analysis" we mean that we want the information about θ to be the same for the full data set (A, X) as for the squashed data set (B, Y, w). This means that we want the product of probabilities in Eq(1) to represent the same function of θ for the two possible data sets, or, in statistical terminology, we want the two samples to have the same *likelihood function*. Equating the two log-likelihoods, we have the following requirement:

$$\sum_{i=1}^{M} w_i \log(f(B_{i1}, \ldots, B_{iC}, Y_{i1}, \ldots, Y_{iQ}; \theta)) = \sum_{i=1}^{N} \log(f(A_{i1}, \ldots, A_{iC}, X_{i1}, \ldots, X_{iQ}; \theta)). \tag{2}$$

We now make the assumption that for every set of fixed values of A and θ, $f(A, X, \theta)$ is a relatively smooth function of $(X_1, \ldots, X_Q)$, so that $\log(f(A_1, \ldots, A_C, X_1, \ldots, X_Q; \theta))$ can be well represented by a Taylor series in the neighborhood of any point $x = (x_1, \ldots, x_Q)$. That is, we have the approximate equality

$$\log(f(A_1, \ldots, A_C, X_1, \ldots, X_Q; \theta)) \simeq \sum_{k=1}^{K} g_k \prod_{j=1}^{Q} (X_j - x_j)^{p_{kj}}. \tag{3}$$

In Eq(3) there are K terms in the Taylor series, the coefficients g_k depend on $(A_1, \ldots, A_C)$, θ, and x, but not on $(X_1, \ldots, X_Q)$, and the power vectors $(p_{k1}, \ldots, p_{kQ})$ are all vectors of Q nonnegative integers satisfying $\sum_j p_{kj} \leq d$, where d is the degree of the approximation. In order to use Eq(3) to solve Eq(2), we divide the space of X into a finite set of neighborhoods and partition the space (A, X) into regions where A is constant and X is confined to a single neighborhood. Suppose there are R such regions and that the large data set has N_r points in region r, where $\sum_{r=1}^{R} N_r = N$. Likewise, we assume that the squashed data set will have M_r points in region r, and that the summation $\sum_{i=1}^{M_r} w_i = N_r$. Within the rth region, the likelihood equivalence Eq(2) can be separately enforced, which amounts to setting every B_{ic} equal to the corresponding constant value of A_c, $c = 1, \ldots, C$, and replacing M and N by M_r and N_r, respectively. Combining Eq(2) and Eq(3), we have the approximate constraints

$$\sum_{i=1}^{M_r} w_i \sum_{k=1}^{K} g_k \prod_{j=1}^{Q} (Y_{ij} - x_j)^{p_{ij}} = \sum_{i=1}^{N_r} \sum_{k=1}^{K} g_k \prod_{j=1}^{Q} (X_{ij} - x_j)^{p_{kj}}, \ r = 1 \ldots, R. \tag{4}$$

In order for Eq(4) to hold for arbitrary functions f, and thus for arbitrary coefficients g_k, the factors multiplying each g_k must be equated separately. This leads to the set of equations

$$\sum_{i=1}^{M_r} w_i \prod_{j=1}^{Q} (Y_{ij} - x_{ij})^{p_{kj}} = \sum_{i=1}^{N_r} \prod_{j=1}^{Q} (X_{ij} - x_j)^{p_{kj}}, \ r = 1, \ldots, R; \ k = 1, \ldots, K. \tag{5}$$

This result suggests that within each region (where the categorical variables are constant and the quantitative variables are confined to a compact subregion) and for each power vector $(p_{k1}, \ldots, p_{kQ})$, the weighted mixed moment of the squashed data set should approximately equal the corresponding mixed moment of the original data set. If $d = 2$, only means, variances and covariances are preserved within each region. Although this would be sufficient to preserve the results of many linear statistical models such as multiple regression, the squashed data set might not be a good enough approximation for the nonlinear models used often in data mining applications. In later sections we discuss the issues involved in relating the degree of approximation in Eq(5) and the size of the squashed data set.

3 Grouping the data in regions

The goal of a binning strategy is to partition the data into compact subregions. Within each of these subregions we then calculate moments, and create pseudo points via the squashing algorithm. An obvious choice for creating bins is to use all qualitative variables in the data set, since the categorical nature of these variables will induce a partition. Sometimes, this may be the best approach, so as not to introduce subjective partitions in the data set that do not already exist.

We propose two techniques to create bins from the quantitative variables:

Hyper-rectangles: We can create bins by categorizing the continuous variables creating hyper-rectangles in data space. However, an efficient choice of bin boundaries is often difficult to specify in advance. With a large number of bins per variable there is a tradeoff; more total bins are created giving more precision, but it results in less data reduction. Since the total amount of bins grows exponentially with the number of bins per variable, it is wise to keep the number of categories relatively low. For the example in this paper, we split each continuous variable into 3 bins, based on the 25th and 75th percentiles, and found this to give reasonable results.

Data Spheres: Data Spheres are a way of inducing a categorical value on a set of continuous variables based on the multivariate distribution of those variables. Data Spheres also avoid the curse of dimensionality imposed by the hyper-rectangle technique. Given a point $p = (x_1, \ldots, x_Q)$, we transform it into $p' = (y_1, \ldots, y_Q)$ by subtracting the center of the data set (for example, the mean) and dividing by the standard deviation. For every point p' in the data set, we compute the distance of p' from the origin. We then partition the data set by the distance of the points from the origin. Each of these partitions (or *layers*) can contain a desired fraction of points, i.e. we use an approximate quantiling algorithm to compute the distance cutoffs.

We also partition the data set into *pyramids*. Informally, a pyramid consists of the set of points for which a particular attribute has the greatest deviation from the origin, and in a particular direction (positive or negative). In three dimensions, a pyramid has the familiar pyramidal shape. More formally,

$$p' \in P_{i+} \quad \text{if } | y_i |>| y_j |,\ y_i > 0 \quad j = 1, \ldots, Q\ ; j \neq i,$$
$$p' \in P_{i-} \quad \text{if } | y_i |>| y_j |,\ y_i < 0 \quad j = 1, \ldots, Q\ ; j \neq i.$$

Layers and pyramids are orthogonal partitioning strategies. In combination, they define data sphere partitions. In analogy to data cubes, it is possible to roll up or drill down data sphere partitions. For example, we can collapse all pyramids in a layer. Or, we can further partition the pyramids into sub-pyramids, creating a hyper-pyramid partitioning. For example, P_{i+j-} represents the region in which y_i has the greatest deviation from the origin, y_j has the second greatest deviation from the origin, $y_i > 0$, $y_j < 0$.

For the experiments in this paper, we use the following strategy. We create three layers. The central layer contains 50% of the data points, and the cutoff between the middle and outer layer is twice the distance of the central layer cutoff. The central layer is not partitioned by pyramids, the middle layer is further partitioned by pyramids, and the outer layer is partitioned by pyramids and sub-pyramids. Up to $4Q^2 - 2Q + 1$ partitions will be created, but many outer-layer partitions will be empty.

Deciding which binning technique to use is an important element of data squashing. It affects both the level of data reduction realized and the computational time needed to do the reduction. In the example we introduce later, we compare the two binning strategies, which we give the following abbreviations:

HypRect: The bins are all combinations of categorical variables, and within each categorical bin construct hyper-rectangular bins from the continuous variables,

DSphere: The bins are all combinations of categorical variables, and within each categorical bin use Data Sphere partitioning of the continuous variables.

4 Moment calculations

Once the bins have been chosen the next step in data squashing is to calculate the moments of the continuous variables for the elements falling in each bin. These moments are the sufficient statistics that will be used to create pseudo data. But how many and which moments should we calculate?

In order to provide a good fit for a range of machine learning and statistical models, we propose calculating the following moments on the continuous variables (note that all of the following variables have been centered and scaled): means, minima and maxima, second order moments (e.g., X_i^2, X_iX_j), third order moments (e.g., $X_iX_jX_k, X_i^2X_j$, and X_i^3), fourth order moments (e.g., $X_iX_jX_kX_l, X_i^2X_jX_k, X_i^2X_j^2, X_i^3X_j$, and X_i^4), marginal fifth moments (e.g., X_i^5). These moments are computed separately for each bin defined by the binning strategy. However not all moments are necessarily used for every bin.

The number of moments that are actually used depends on the number of pseudo points fit to each bin. Fewer pseudo points are able to match fewer moments. So, as with the binning procedures, the number of moments fit can be adjusted so as to control the amount of data reduction desired, and to reduce computational complexity if necessary. In order to match the number of moments to the number of pseudo points, we define degrees of freedom as $df = m(Q+1)$, where m is the number of pseudo points in a bin. Note that df is the number of free parameters (values of w and Y) that the pseudo data contain. We choose the $K \approx df$ lowest order moments from among those listed above.

Choice of pseudo sample size. Within the rth bin, the number of points, M_r, in the pseudo sample depends on the corresponding number of points, N_r, in the original data. The choice of M_r depends on how much data reduction is desired, and M_r should increase slowly with N_r. We use the somewhat arbitrary formula

$$M_r = \max(1, [\alpha \log_2 N_r]),$$

where α is a number that determines how much data reduction is achieved. In our example, we compare $\alpha = 0, \frac{1}{2}, 1$, and 2. When $\alpha = 0$, only one point is used to approximate each bin, positioned at the bin mean for each variable. When $\alpha > 0$, the method uses more points and moments as α increases for a better-fitting pseudo sample. The overall data reduction factor is thus approximately $\sum_r N_r / \sum_r M_r \approx N/\alpha \sum_r \log_2 N_r$.

The moment calculations have a strong advantage in data mining applications. They can be calculated sequentially, so that when a new set of data arrives, they can be used to update the moments without recalculating the moments for the whole set. This is important when we are dealing with millions of records which are streaming in daily; we do not want to have to re-compute moments each time we receive fresh data.

5 Generating squashed data elements and associated weights

We search for an approximate solution to Eq(5) separately for each region indexed by r. Within each region, we can assume that the x_j in Eq(5) are equal to the mean of the X_{ij} contained in the region, and, by re-centering the Xs within each region to have mean 0, assume that every $x_j = 0$. Therefore, for each r (in what follows, we do not bother to add an index r to many variables that are defined separately for each region), we search for an $M_r \times Q$ matrix Y and an M_r-length vector w to approximately satisfy the equations

$$\sum_{i=1}^{M_r} w_i \prod_{j=1}^{Q} Y_{ij}^{p_{kj}} = z_k, \qquad k = 1, \ldots, K; \qquad (6)$$

$$\text{where } z_k = \sum_{i=1}^{N_r} \prod_{j=1}^{Q} X_{ij}^{p_{kj}}.$$

This is a system of K equations in $M_r(Q+1)$ unknowns. As mentioned above, we choose K for each r by selecting enough low-order moments so that $K > M_r(Q+1)$ up to a maximum K as discussed in Section 8. Even if $K \leq M_r(Q+1)$ there may often be no exact solutions to Eq(7), because we enforce the constraints

$$w_i \geq 0; \ \min_i X_{ij} \leq Y_{ij} \leq \max_i X_{ij}. \qquad (7)$$

We rule out negative weights and variable values outside the ranges of variables found in the corresponding region of the original data set because we want the squashed data set to be similar to the original data set. However, we do not insist that the values of Y be restricted to values that occur in the corresponding X. For example, if a given X_j is "number of children in the family", and varies from 0 to 10 in the original file, the value 8.5 would be an acceptable value for Y_j, but not 10.5 or -1. If it is desired to maintain the set of actually occurring values, then we make the variable categorical (a column of A). The search for (Y, w) is treated as the search for a least squares estimate to minimize

$$S(Y, w) = \sum_{k=1}^{K} u_k (z_k - \sum_{i=1}^{M_r} w_i \prod_{j=1}^{Q} Y_{ij}^{p_{kj}})^2. \qquad (8)$$

Finding a least squares solution. In Eq(8), the positive multipliers u_k are used to ensure that the lower-order moments are approximated more accurately. Since it is trivial to scale w and each column of Y so that the moments of order 0 and 1, and the pure squares of order 2, are fit exactly, the corresponding values of k have $u_k = 1000$. All the other u_k sum to 1, with moments of order 2 having larger u_k than those of order 3, which are in turn larger than the u_k for moments of order higher than 3. (In addition to being centered at 0, the Xs are originally scaled to have variance 1, so that moments of different orders are comparable.) Computation of the moments z_k is done using an updating algorithm, so that the centered and scaled moments within each region can be computed in a single pass over the large data set with a minimum of round-off error. Since $S(Y, w)$ in Eq(8) is a simple polynomial function of the unknowns, it is easy to compute the required derivatives for a second-order Newton-Raphson iterative scheme. In order to maintain the constraints of Eq(7) at each update step, we transform the unknowns using a logistic transform, so that the finite range is converted to an unbounded range and out-of-bound updates are prevented. In addition, at each update the step size is reduced if necessary to ensure that Eq(8) gets smaller at each update. There will rarely be a unique local minimum of Eq(8). However, in our experience the multiple local minima often have very similar values of $S(Y, w)$, and will serve equally well for the purpose of creating a squashed data file. The Newton-Raphson search can be repeated with different random starting points to guard against the occasional solution that is much poorer than average.

6 Example

At AT&T we routinely see data sets with millions, hundreds of millions or even billions of records. Many of these data sets have a fixed number of variables, which are somewhat restricted by the space we have to keep them or the processing time it takes to update them. These types of data sets are the ones we feel will get the most benefit from squashing, since we can keep a smaller version of the data set handy for quick analyses, and we can update the data set as new data come in through sequential updating of the moments. Here we describe a relatively small data set and use it to illustrate a variety of squashing strategies and their associated performance.

The goal of our analysis is to see if we are able to detect customers who have switched to another long distance carrier. Because almost all information about our customers goes through the local phone companies, sometimes we do not find out about the departing customers for weeks. It is to our benefit to be able to detect these customers as soon as possible.

We collected a set of variables describing customer behavior which we thought might provide information about those who have left, whom we'll call *Defectors*. Two of these variables were 3 level categorical variables (A_1, A_2), and five others are continuous $(X_1, \ldots, X_5)$.

We gathered a training set of 744,963 records. A data set this size is large enough to show the benefits of squashing, but also small enough so that we can do a non-linear analysis and compare the results of the full data to the results from a variety of squashed versions. Our goal is classification and we will use logistic regression to predict *Defectors* using the 7 other independent variables.

We stress that our purpose in presenting this example is not to show how to solve a particular classification problem. Rather, we assume that we are required to solve a nonlinear problem that has no acceptable traditional solution computable via one or two passes over the data set. Estimation of logistic regression coefficients is intended to serve as a prototype of such a problem. We assume that examination of coefficients from a complicated model will answer an important business problem.

As discussed earlier, we have many different squashing options with respect to number and type of bins and the number of pseudo points fit in each bin. These two options will determine the number of moments used and the computational complexity involved. In general, using a method that creates many bins decreases the number of pseudo points per bin in order to keep the same level of data reduction. We investigated the performance of 6 different squashing techniques defined by the binning strategies and data reduction parameter α. Table 1 summarizes the binning and moment strategies for these methods. As mentioned earlier, we use fewer moments per bin when a bin size (based on $\alpha \log_2 N_r$) is small.

As the table shows, we combine the values $\alpha = 0, 1, 2$ with DSphere grouping ($R = 394$ regions), and the values $\alpha = 0, .5, 1$ with HypRect grouping ($R = 3710$ regions), to provide pseudo data set sizes of $M = 394, 2183, 4475, 3710, 8373, 17386$, respectively. These are compared to a random 1 % sample of size 7576. The sample reduction factor is N/M, which is 98 for the random sample, and ranges from 43 to 1891 for the squashed samples. The table also shows that the number of pseudo points per bin and the number of moments used per bin are larger for DSphere than for HypRect, and also that they increase with α.

We fit logistic regressions predicting the *Defectors* from the other 7 variables to the full data set, the sampled data and the 6 squashed data sets. One regression consists of only main effects while the other adds all second order terms, including interactions. To assess the performance of the methods, we assume that the coefficients from the full data set are the truth, and look at how close the other methods approximate this truth.

To quantify how close the reduced data sets approximate the full data, we look at standardized residuals from the true coefficients (Figures 1 and 2). The last two columns of Table 1 summarize the plotted values in the Figures. Each MSE in the table is an average squared value of standardized coefficient errors (Coef − True)/StErr for a particular sample. Note that, theoretically, we expect MSE = Reduction Factor for a random sample, as in row 2 of the Table. A squashing method works well if MSE $\ll$ Reduction Factor. Table 1 shows that this is true whenever $\alpha > 0$, but this is not true for $\alpha = 0$. In other words, basing the logistic regression solely on bin means is worse than taking a random sample of the same size. This is probably related to the fact that the bin means have much reduced variances compared to the original data. (Note that the figures do not show the results for $\alpha = 0$, since to include them would force the vertical range of the figures to increase so much that comparative detail for the other methods would be lost.) To summarize, for the first order model, all of the squashing techniques except for Means (i.e. $\alpha = 0$) do significantly better than 1% sampling. Most of the estimated coefficients from the squashed data sets are within one standard error of the true value, whereas the majority of the sampled-data coefficients are greater than 5 standard errors away. For the second order model, which has 48 coefficients to estimate excluding the intercept, the improvement of

Method (α)	*Bins* (R)	*Pseudo Pts. per Bin*	*Moments per Bin*	*Total Points*	*Reduction Factor*	*Comp Time*	*MSE Main*	*MSE Quadratic*
Full Data	1	-	-	744,963	1	-	0	0
Random Sample	1	-	-	7,576	98	-	115.5	129.5
DSphere(0)	394	1	5	394	1,891	-	889,507	1,922,062
DSphere(1)	394	1-16	5-120	2,183	341	50	0.52	29.0
DSphere(2)	394	1-33	5-130	4,475	166	119	0.60	44.6
HypRect(0)	3,710	1	5	3,710	201	-	389.0	2,177
HypRect($\frac{1}{2}$)	3,710	1-6	5-45	8,373	89	373	0.24	10.5
HypRect(1)	3,710	1-12	5-80	17,386	43	562	0.09	0.5

Table 1: Comparison of the different types of squashing, sampling, and the full data set. Computer Time is in minutes and refers to the computation of pseudo points on an SGI Origin workstation using a fitting algorithm programmed in Splus. MSE refers to the mean squared value of (Coef - True)/SE for the main effects model (10 coefficients) or the quadratic effects model (48 coefficients), in a logistic regression, where True and SE are the coefficients and standard errors, respectively, using the full data set.

squashing over sampling is less pronounced, but still apparent, especially for the HypRect method with $\alpha = 1$.

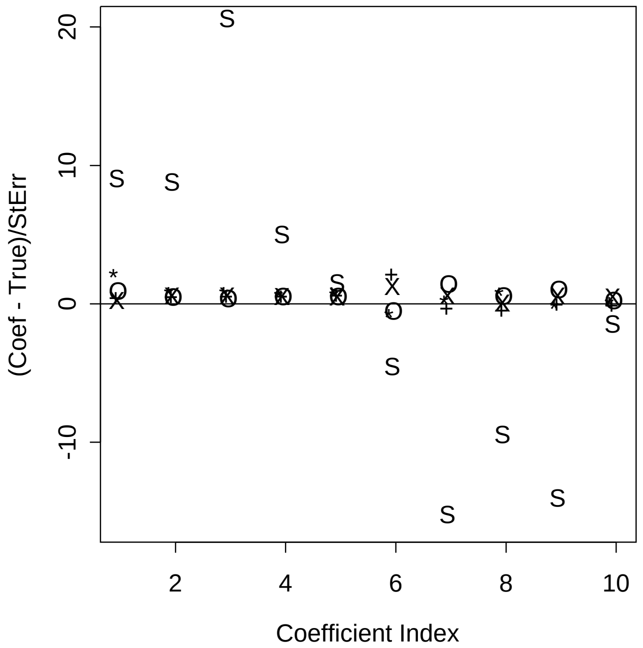

Figure 1: Standardized errors for logistic regression coefficients from reduced samples, main-effects model. Plot symbol denotes method. [+: DSphere(1) *: DSphere(2), O: HypRect($\frac{1}{2}$), X: HypRect(1), S: Random Sample]

Another way of checking the efficacy of squashing is to see how well it emulates the full data set in the output of the model. In a logistic regression model, each element of the data set is assigned a probability of being a *Defector*. We used the coefficients from

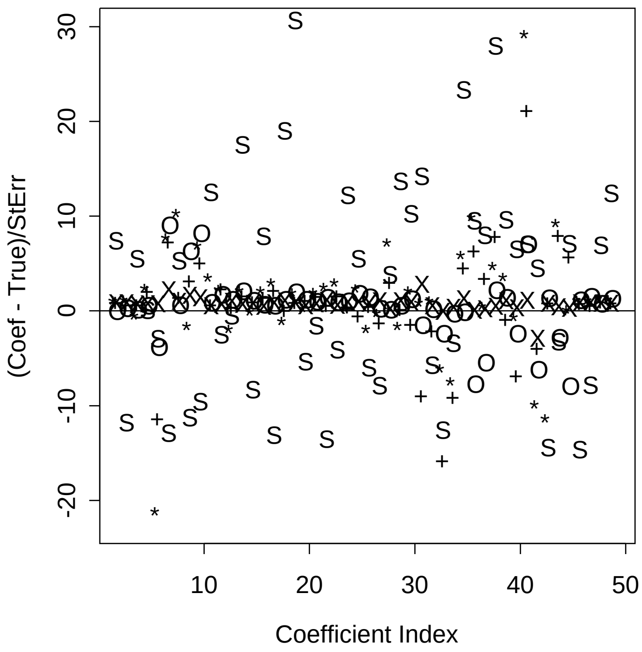

Figure 2: Standardized errors for logistic regression coefficients from reduced samples, second order model. Plot symbol denotes method. [+: DSphere(1) *: DSphere(2), O: HypRect($\frac{1}{2}$), X: HypRect(1), S: Random Sample]

the 6 squashed data sets and the 1% sample , as fit to the first order logistic regression, to assign this probability. This probability was then compared with the "true" probability of being a *Defector* from the full model. For each row of the full data set, a residual is defined as (Probability based on reduced data set) - (Probability based on full data set), multiplied by 10000 for descriptive purposes. Table 2 describes the

Method (α)	*Mean*	*StDev*	*Min*	*Max*
R. Sample	-32	207	-824	1260
DSphere (0)	-807	4012	-8875	7109
DSphere (1)	-1	14	-56	41
DSphere (2)	-3	13	-69	30
HypRect (0)	-67	310	-844	1632
HypRect ($\frac{1}{2}$)	-2	9	-37	34
HypRect (1)	-2	5	-19	12

Table 2: Comparison of predicted probabilities from the main-effects logistic regression model. For each reduced data set, the N = 744,963 residuals are defined as (Probability based on reduced data set – Probability based on full data set) * 10000 . Each row of the Table describes the distribution of the corresponding residuals for a given reduction method.

distributions of the residuals for all of the methods. The squashed data sets having $\alpha > 0$ perform at least an order of magnitude better than the random sample.

The results in this section give a little more information to compare the different squashing techniques. In all of the results for this data set, HypRect performed closest to the full data set, sometimes dramatically so. Since the binning strategy for HypRect categorizes each continuous variable independently, HypRect is most vulnerable to the curse of dimensionality. With many continuous variables, DSphere may be preferred. The Means strategy ($\alpha = 0$) gave comparatively poor results in our example, performing worse than a random sample. The method with $\alpha = 1$, having about twice as many pseudo points, performs much better than with $\alpha = .5$ for the HypRect binning strategy. On the other hand, surprisingly, taking $\alpha = 2$ did not improve the results compared to $\alpha = 1$ for the DSphere binning strategy. Further research is needed to better understand the relationship between binning strategy and pseudo sample size on the results.

7 Related work

The squashing technique presented in this paper provides a principled methodology for reducing a large data set to a smaller data set, that will provide accurate approximation to the result obtained from fitting most smooth models on the original large data set. It is to our knowledge the first attempt in the KDD literature to produce such a pseudo set of points that locally mimic the statistical properties of the original multivariate data set beyond mean values.

The methods generalizes the squashing method in one dimension that was used to speed parameter estimation in DuMouchel (1999). In that paper the likelihood was a mixture of negative binomial regressions:

$$L(A, X, \theta) = \prod_{i=1}^{N} [p\, f(A_i, \alpha_1, \beta_1, X_i) + (1-p) f(A_i, \alpha_2, \beta_2, X_i)] \tag{9}$$

where $\{A_i\}$ and $\{X_i\}$ are univariate integer and continuous variables respectively, $\theta = (p, \alpha_1, \alpha_2, \beta_1, \beta_2)$ is a vector of 5 parameters, and

$$f(a, \alpha, \beta, x) = (1 + \beta/x)^{-a}(1 + x/\beta)^{-\alpha}\Gamma(\alpha + a)/\Gamma(\alpha)a!$$

is the negative binomial probability density. In DuMouchel (1999) A and X are each vectors with $N >$ 1.3M. Squashing X separately for each unique value of A produced a reduced data set with only a few thousand elements, which sped up the maximization of $L(A, X, \theta)$ with respect to the five-dimensional parameter θ by a factor of 600.

Squashing can be viewed as an extension of the many clustering algorithms appearing in the KDD literature (Bradley, Fayyad, and Reina 1998; Zhang, Ramakrishnan, and Livny 1997). These algorithms will, in a linear pass over a large data set, cluster elements into mutually exclusive bins. The original data set can thereafter be replaced with the cluster centers and associated weights (determined by how many elements fall in the cluster). Further analysis can be performed on these weighted cluster centers. In squashing terminology such an approach corresponds to generating a distribution that only mimics the moments of the original data set to first order. Our results suggests that a squashed data set consisting of weighted cluster centers is inferior to that of a squashed data set that matches higher order moments within each cluster. Of course, the size of this second squashed data set is larger than one consisting solely of weighted cluster center so this factor should be taken into account when deciding on a squashing strategy.

Due to the modular design of the squashing methodology, clustering algorithms and the squashing can however borrow strength from each other. Any clustering algorithm will induce a grouping of the data and can be used in the first step of GMG pipeline. It is a topic for further research to experiment with alternative clustering algorithms for the grouping step.

8 Computational Complexity of Data Squashing

We believe that data squashing as described will scale up quite well with the number of rows in the data set, and moderately well with the number of variables. The computational burden needs to be separately broken down into sources due to the grouping, the moments, and to the pseudo points generated. Suppose there are B bins due to categorical variables, and Q quantitative variables, with N points in the large data set. We always assume that the data set has been sorted according to the quantitative variables, and the overhead for this sorting step, roughly $QN\log(N)$ operations, is part of every version of squashing in this paper. The squashing is done independently within each of the B bins, and can be parallelized. Suppose the total number of populated regions (combinations of quantitative variable bins within categorical variable bins) is R.

Cost of Grouping For either of the two binning strategies, it takes one pass over the data to assign each point to one of the bins (the Data Sphere method requires an extra pass over the data to determine bin boundaries). Memory requirements are proportional to NQ and CPU is proportional to NQ.

Cost of Computing Moments The total number of moments considered is $K = 5Q + 3Q(Q-1) + 2Q(Q-1)(Q-2)/3 + Q(Q-1)(Q-2)(Q-3)/24$, which is asymptotically equivalent to $Q^4/24$, but for moderate Q the lower-order terms in the expression for K predominate. Using a one-pass algorithm, the memory requirements are proportional to RK and CPU is proportional to NK. However, if the data are first sorted according to the R regions, then only $O(K)$ units of memory are required.

Cost of Computing the pseudo points If there are M_r pseudo points to estimate in region r, the task is to estimate $M_r(Q+1)$ unknowns as the least squares solution to K_r equations, where K_r is the number of moments used in region r. The task must be repeated R times. The dependence on N, the total number of points in the large data set, comes from the relations $M_r = [\alpha\log_2(N_r)]$ and $K_r = M_r(Q+1)$. The total CPU is proportional to the sum over r of $\mathrm{CPU}(\alpha, N_r, Q)$, where $\mathrm{CPU}(\alpha, N_r, Q)$ is the cost of iterating to a solution when $M_r = [\alpha\log_2(N_r)]$ and $K_r = M_r(Q+1)$. The resulting total CPU depends on the distribution of N_r across the R regions. When R is large, the vast majority of the M_r are quite small and the total CPU is virtually proportional to R, the number of regions, with relatively little dependence on N. Examining our estimation algorithm in detail, each iteration is dominated by function evaluations that involve $O(M_r K_r)$ operations. Thus the total CPU is approximately $O(\sum_{r=1}^{R} M_r K_r) = O(\alpha^2 Q \sum_{r=1}^{R}(\log(N_r))^2) = O(\alpha^2 RQ(\log(N/R))^2)$. Note that whenever $M_r = 1$ there is a trivial solution to the equations found by equating the pseudo points to the means of the original points.

The CPU cost of converging to an iterative solution is impossible to specify in advance and depends on the particular estimation algorithm and the data. We typically perform fewer than 100 function evaluations for each solution. Note that for the timings given in Table 1, the total CPU went up somewhat slower with α and faster with R than the above formula suggests. The four ratios $1000\ \mathrm{CPU}/(\alpha^2 RQ(\log(N/R))^2)$ are 0.4, 0.3, 2.9, and 1.1, respectively.

The storage required for our Newton-Raphson algorithm is proportional to $\alpha^2 Q^2 \max_r(\log(N_r))^2$.

Cost Summary With respect to memory, all the methods and steps of the squashing technology are extremely economical and scale very well in both N and Q. With respect to arithmetic operations, the grouping and momentizing steps involve an effort no more onerous than sorting and a very few passes over the data set of similar effort. The pseudo point generation step is much more computationally intensive, but has the tremendous advantages that it increases only with $\log(N)$ and can make efficient use of parallel processing. In our example, the grouping and momentizing steps took a matter of minutes, while the pseudo point generation took hours on workstations with no parallel processing. Had there been one hundred times as many rows, but the same number of columns, in the original data, the entire squashing procedure would still have taken hours, rather than days.

9 Conclusion

We have demonstrated that it is possible to scale down large data sets while preserving micro-structure that might be important for subsequent analysis. We view squashing as a breakthrough since it immediately opens up the possibility of applying all current machine learning and statistical techniques to large data sets — all that is required is that these techniques accept element weights. Our method is novel in several respects:

- it constructs a replacement for the original data set that has the same column structure (i.e., variables) but dramatically reduced row structure (i.e., elements),
- it consists of modular pieces that can be individually tuned and improved,
- it has a theory to guide further refinements and to provide possible error bounds

Squashing is demonstrably more powerful than taking a simple random sample from the large data set. It is possible that special-purpose samples might do better than a simple random sample. For example, if classification of a very rare response category is the goal, taking a case-control sample, in which the rare category is oversampled from the database, will lead to more efficient estimates than a simple random sample. Squashing has the advantage that the sampling method does not have to be tuned to the intended analysis, which is particularly advantageous when multiple diverse analyses are planned.

A more subtle feature of squashing can be a significant advantage in applications where customer or patient confidentiality is an important concern. Samples (random, special-purpose, or otherwise) contain actual records from the underlying database and clever data sleuths can sometimes identify customers or patients from unlabeled micro-data (usually through joins with other databases). A squashed dataset defeats the data sleuth since the squashed records faithfully represent only the aggregate behavior of the large data set and not the individual records themselves. So squashing naturally lends itself to data publishing with minimal non-disclosure risk.

Just how big a problem can squashing handle? We believe that the limiting factor is *not* the number of rows (N). Given a grouping/moment strategy, as N increases, the pseudo data does not increase, it just gets updated. Thus the complexity of squashing has little to do with N. The biggest challenge we see is in dealing with the number of columns, as this can lead to exponential bin growth. In practice however, we almost always have correlations between variables, so that a large fraction of the bins are empty and thus require neither moments nor pseudo points.

We expect further research in each of the components of the GMG pipeline:

- grouping into regions: this step is critical to both the accuracy of squashing and its application to data sets with huge column spaces. Linear time clustering methods have enormous potential here. Note that clustering is being used here as a preprocessor — the resulting clusters have no import whatsoever on later analyses, and moreover, the more clusters the merrier (up to a limit!). Further research is also needed to handle the case of very many categorical variable combinations.

- momentizing: the effect of outliers on moments is well known; robust moments can be used instead of ordinary moments, but does this violate the spirit of squashing in that we *want* to track the original data outliers included? In data streaming applications, simple updates of moments, while possible, are probably not desirable since data from ancient time periods are still being reflected in the moments. Should exponential averaging be used to update moments in such cases?

- generating pseudo elements: nonlinear least squares has served us quite well (we think!) but this step of squashing could benefit by serious numerical analysis to ensure the stability of the solutions.

Finally we wish to draw attention to an aspect of modeling that is arguably important, but one where extensive research needs to be done in order to leverage squashing, namely visualization. It is clear that visualization methods suffer from data overload just as modeling methods do. Squashed data should help but it is not at all clear how the weights should be incorporated into visualization methods. We hope to stimulate our colleagues to undertake this important challenge.

References

Barbara, D. (1997). The New Jersey data reduction report. *Bulletin on the Technical Committee on Data Engineering 20*(4), 3–45.

Bradley, P. S., U. Fayyad, and C. Reina (1998). Scaling clustering algorithms to large databases. In *Proc. 4th Intl. Conf. on Knowledge Discovery and Data Mining (KDD)*, pp. 9–15.

DuMouchel, W. (1999). Bayesian data mining in large frequency tables, with an application to the FDA spontaneous reporting system, with discussion. *The American Statistician, in press.*

Johnson, T. and T. Dasu (1998). Comparing massive high dimensional data sets. In *Proc. 4th Intl. Conf. on Knowledge Discovery and Data Mining (KDD)*, pp. 229–233.

Zhang, T., R. Ramakrishnan, and M. Livny (1997). Birch: A new data clustering algorithm and its applications. *Data Mining and Knowledge Discovery 1*(2).