Prediction in the Era of Massive Datasets

Greg Ridgeway (gregr@rand.org)
RAND Statistics Group, Santa Monica, CA 90407-2138

Abstract. Data mining often involves processing massive, retrospective datasets to learn how to make predictions in the future. The researchers that compose the data mining community have approached the prediction problem in their own fashion and have produced a novel set of tools. This article discusses the complexity arising when fitting prediction models to massive datasets, proposes a framework for algorithmic solutions based on bagging and boosting, and poses some ideas for future directions.

Keywords: prediction, data mining, bagging and boosting

1. Introduction

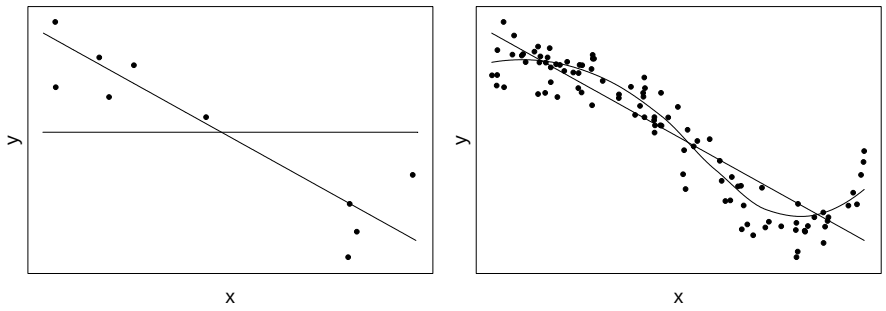
Extracting predictive models from massive datasets is one of the primary data mining efforts. Historically, much of statisticians' effort focused on extracting as much information as possible from relatively small, structured datasets. The arrival of data mining and truly massive datasets changes how we approach data. However, the size of the datasets allows us to examine models that are more complex than ever before. In the last few years researchers working on the boundary between statistics, computer science, and engineering have produced an exciting set of new methods for discovering complex relationships between items in massive datasets.

In this paper I will discuss the prediction problem for massive datasets. When faced with massive datasets we have the blessing that we can increase *model complexity* to gain higher prediction accuracy with the curse of increased *data access complexity*. The task we face is to discover and tune algorithms that provide accurate predictions in spite of the difficulty of accessing the data. I conclude this paper by assembling several recent innovations to produce a framework for solving a variety of prediction problems for massive datasets.

2. Prediction and massive datasets

Massive datasets change the complexity of prediction problems in two main ways. First, having an enormous amount of data allows the analyst to fit more complex models. Second, when the dataset cannot fit into the main memory of a computer, scanning the dataset can be several orders of magnitude slower.


© 2000 *Kluwer Academic Publishers. Printed in the Netherlands.*



a. Sparse data - simple models b. Dense data - more complex models

Figure 1. Model complexity and data density. As the size of the dataset increases the dataset exposes its more complex features allowing for more complex modeling and more accurate prediction.

2.1. Model complexity

Figure 1 shows two graphs displaying data generated from the same data generating mechanism. From this data we wish to learn how to predict y from x. The figure to the left shows a sparse dataset with ten observations and the figure to the right is denser with 100 observations.

With the data available in figure 1a, an extremely parsimonious model would fit the horizontal line to the average of the ten data points. It is stable and rarely provides a disastrous prediction of y. However, visually the data contain more information than the simple horizontal fit, specifically that there is a decreasing trend. The linear model in figure 1a seems to capture that information and provides a reasonable fit to the dataset and improved predictive performance. As the data become more dense, as shown in figure 1b, the linear model seems as poor a model as the horizontal prediction model did in figure 1a.

The first lesson in extracting prediction models from massive datasets is that we can fit more complex models. In particular we can model non-linear terms, explore more potential predictor inputs, and learn how those inputs interact to make predictions.

The main goal in constructing predictors is accurate prediction and, therefore, relying on simple models can severely degrade predictive performance. We often fall back on the simple models for many reasons the common ones being, they are the ones with which we are most familiar, they offer an interpretable prediction rule, and the computational cost of fitting these models might be small. According to Breiman's law

(Breiman, 1997)

$$\text{Interpretability} \times \text{Accuracy} = \text{Breiman's constant} \quad (1)$$

indicating that if we want accurate predictions, interpretation has to be a secondary priority. The evidence from figure 1b demonstrates that, particularly for large values of x, the linear model fails to make good predictions. Having a massive dataset, in terms of having many independent observations, allows prediction models to learn curvature that would be missed in sparse datasets.

Over the years, statisticians and computer scientists have developed a handful of Bayes risk consistent procedures, predictors that converge to the optimal predictor as the sample size becomes large. These large datasets almost make optimal predictors a reality rather than merely a theory... if it were not for data access complexity.

2.2. Data access complexity

So far I have distinguished between "simple" and "complex" models without solidifying their traits. Simple models generally make strong assumptions about the relationship between the outcome and the input variables. These assumptions often make computation and interpretation manageable (e.g. linear regression models, the best separating hyperplane classifier). In the absence of a strong theory, which is generally the case in data mining, we wish to avoid specifying a functional form for the prediction model, allowing models that have greater capacity for fitting non-linear and input interaction terms.

Massive datasets, defined to be those that cannot fit into the main memory of a computer, can complicate the process of fitting non-linear and interaction terms to data. The simplest models tend to require few scans of the dataset to produce the best fit. In the case of linear regression, the algorithm needs only one scan. Any other method that permits non-linear terms and non-linear interactions almost certainly requires many more scans of the dataset. If the dataset could fit into main memory then multiple laps might not cause problems. On the other hand, retrieving data from the disk is about one million times slower than retrieval from main memory!

The second lesson is that in order to be useful for massive datasets, the construction of prediction models must rely on algorithms that either require very few scans of the dataset or can sequentially process the dataset in blocks that can reside in main memory. The main objective of constructing prediction models for massive datasets is to fit models of appropriate *model complexity* with algorithms that can handle the issues arising from *data access complexity*.

3. Prediction methods

In this section I will discuss the nature of the prediction problem and some of the methods used to construct prediction models and how they relate to the issues of model complexity and data access complexity.

Let y be the quantity that we want to predict. It may be continuous (profit from a particular customer, time until a part fails) or categorical (whether or not a customer defaults on a loan, whether a patient has a malignant, benign, or no tumor).[1] Let x be a set of features measured on the unit of observation. In the case of disease prediction this may be a set of observed symptoms and patient characteristics. In the case of credit approval this may be a customer's payment history.

Regardless of the situation, the problem is always to learn a function, $f(x)$, that minimizes the expected loss

$$J(f) = \mathrm{E}_{x,y} L(y, f(x)) \tag{2}$$

where $L(y, f(x))$ measures the cost associated with predicting $f(x)$ when the true value is y. For least-squares regression $L(y, f(x)) = (y - f(x))^2$ or for binary classification $L(y, f(x))$ is the cost of labeling an observation with $f(x)$ when the true label is y. Equation (2) is known as the *generalization error* of $f(x)$. The expectation is with respect to future observations drawn from the same distribution of (x, y).

Without knowing the exact distribution of (x, y) we try to infer the $f(x)$ that minimizes $J(f)$ using a set of previously encountered examples, $(x_1, y_1), \ldots, (x_n, y_n)$. We can approximate the average in (2) using our sample as

$$J(f) \approx \hat{J}(f) = \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i)). \tag{3}$$

The functional $\hat{J}(f)$ is the *training error* of $f(x)$.

For regression and classification problems, the $f(x)$ that minimizes the training error is one that predicts the training dataset exactly, so that $f(x_i) = y_i$ for all i. Although this minimizes $\hat{J}(f)$, the approximation in (3) breaks down. That is, if we choose $f(x)$ so that $\hat{J}(f) = 0$ by perfectly fitting the training data then almost certainly $J(f) > 0$. Various methods for constraining the capacity of $f(x)$ prevent overfitting the training data. These methods may impose smoothness constraints, limit the number of model parameters, or otherwise control the size of the set of candidate prediction models.

[1] The outcome can also be unknown, the usual setting of a clustering problem.

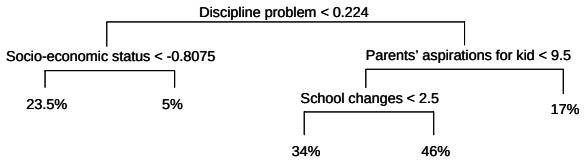


Figure 2. Decision tree produced by running the CART algorithm on the first half of the dataset

The prediction problem reduces to finding the $f(x)$ that minimizes the generalization error using only the information available from the training dataset.

3.1. Decision trees

A decision tree learns a set of conjunctive rules to make predictions. The most popular decision tree algorithms in use for data mining applications include CART (Breiman et al., 1984) and C4.5 (Quinlan, 1993). The CART algorithm initially tries to partition the dataset into two groups such that each group is more homogeneous on the outcome. CART considers all possible splits on each of the features and selects the one that does the best job at reducing the training error. Imagine using CART for predicting whether a student will drop out of high school (at age 17 or 18) using information gathered at age 13. On the first iteration, CART selects the split that best discriminates between the dropouts and graduates. Subsequently, the algorithm recursively splits the newly formed partitions until it meets some stopping criteria designed to avoid overfitting (usually an ideal number of terminal nodes selected by cross-validation). Figure 2 shows an example of a tree constructed using CART. To estimate the probability of dropout for a newly observed student, CART compares the features of the student with the series of questions that the tree poses, true answers moving to the left and false answers moving the right. The dropout probability estimate for the newly classified student is the percentage of dropouts in the training data that are in the same terminal node.

The benefits of decision trees are that they model non-linear relationships with variable interactions, accept continuous, ordinal, and nominal features, and are relatively fast to construct from data and use for future prediction. The fact that trees accept a variety of types of data is especially important in data mining applications and few other methods have this feature. In addition, the divide-and-conquer nature of the algorithm is appealing when faced with a massive dataset. Each recursion of the algorithm only has to deal with a fraction of the observations that the parent processed (usually 25% to 75% less).

Furthermore, specialized algorithms can take advantage of the fact that choosing the split variable and the split point depends on inspecting the input variables only one at a time (Gehrke et al., 2000). In this way much of the data access can occur in main memory.

On the surface decision trees appear easy to interpret, however, they are structurally unstable. Slight variations in the dataset can result in completely different structures making their interpretable appearance very deceiving.

Although decision trees might not be interpretable, the main feature that removes them as candidates for prediction models for massive datasets is that they simply do not predict as well as other methods. Decision trees can proficiently reduce training error but generally have poor generalization performance. Even the simple linear model frequently outperforms CART. I believe, however, that research over the last few years on merging multiple trees has produced the most promising algorithms. The next section describes these methods.

3.2. Boosting, bagging, and variations

Section 3.1 enumerated the decision tree properties desirable for building prediction models from massive datasets. Their ability to handle various types of input variables and their scalability are essential for our purposes. Although they can model complex, non-linear interactions, other methods often exceed their predictive ability.

A series of papers by various authors on learning prediction models from data (not necessarily massive datasets) have generated what I believe to be breakthrough concepts for fitting models to massive datasets. Among these ideas are bagging (Breiman, 1996) and pasting (Breiman, 1999a), boosting algorithms (Freund and Schapire, 1997), and formulating boosting as gradient descent in function space (Friedman, 1999a). I will describe these concepts and suggest a way to assemble them into a unified framework for modeling massive datasets.

Bagging and pasting. Bagging (*B*ootstrap *agg*regat*ing*) simulates a continuous stream of datasets drawn from the joint distribution of x and y. The first step in the process is to sample with replacement from the original dataset to produce a "bootstrap" dataset with the same number of observations as the original. Due to the random sampling with replacement, this new dataset almost certainly contains some duplicate observations so that about 37% of the original observations are not in the bootstrap dataset. But the bootstrap dataset in some respects resembles a draw from the joint distribution of x and y. Empirical evidence shows that if we construct several bootstrap datasets, fit a

decision tree to each, and produce predictions for new observations by averaging the predictions from each model then we can achieve a substantial improvement in accuracy over predicting with a single decision tree. The heuristic argument is that the averaging reduces variance in the predictions.

A natural question to ask is whether the bootstrap datasets need to be as large as the original one. Pasting algorithms average across models fit to randomly selected subsets of the original dataset. Empirical evidence shows that substantial subsampling can still produce an aggregated predictor with accuracy comparable to bagging. This is great news to the data mining community that struggles with fitting data into main memory!

A secondary benefit that bootstrapping or subsampling offers is that the fit of the prediction model is independent from those observations not sampled. If we fit the model to a randomly selected half of the dataset, then the remaining half can provide an unbiased estimate of $J(f)$. This will be an important feature for automating the model fitting process.

Boosting. Boosting came to the forefront of classification research after various empirical studies concluded that boosting offered some of the best performance on standard test datasets (Bauer and Kohavi, 1999). The original algorithms proceeded by first fitting a model to the original dataset. In the next boosting iteration the observations that were previously misclassified receive higher weight and the correctly classified observations receive less weight. Subsequent iterations pile more weight on the hard to classify observations so that most of the weight ends up on those observations near the decision boundary. The final prediction model is a weighted average of all of the models.

The observation reweighting suggested a nice heuristic for boosting's uncanny classification performance. However, for generalizing the boosting framework to the more general prediction problem it hardly gave a satisfactory explanation.

The boosting algorithm was later found to be equivalent to gradient descent in function space. These results showed that boosting is a greedy optimization algorithm for finding a classifier that minimized an upper bound on the misclassification error. The sample reweighting is just a feature of the optimization method. The culmination of this insight produced Friedman's gradient boosting algorithm for minimizing a wide class of loss functions for least-squares regression, robust regression, as well as classification. It is a simple step to further expand boosting to the class of exponential family and survival regression models (Ridgeway, 1999).

The gradient boosting algorithms proceed as follows. Initially set the predictor $f(x) = c$, where c is the constant that minimizes $\hat{J}(c)$. For least-squares regression c is the average of the y_i. The next iteration proposes improving $f(x)$ by adding a new function to it. With $f(x)$ fixed, find a relatively simple $g(x)$ such that $\hat{J}(f+g) < \hat{J}(f)$. For many useful loss functions finding such a $g(x)$ can reduce to fitting a decision tree. We update our current prediction model as $f(x) \leftarrow f(x) + g(x)$. Iterating in this fashion, the algorithm guides our f through the space of prediction models on a path toward minimizing $\hat{J}(f)$.

Slight variations to this algorithm can further improve the performance. The first variation is to reset $g(x)$ as $g(x) \leftarrow \lambda g(x)$, where $0 < \lambda \leq 1$, to shrink the proposed modification toward 0 (Friedman, 1999a). The shrinkage smoothes the path through the space of prediction models. The final model also tends to be smoother as well. The second variation, inspired by a related algorithm known as adaptive bagging (Breiman, 1999b), is to propose the modifications using a randomly selected subset of the dataset (Friedman, 1999b). Results using the stochastic gradient boosting algorithm have shown that proposing the modification based on around 50% of the dataset can result in enormous improvements in the generalization error. Regardless of the variations, to avoid overfitting the algorithm needs to halt before too many components enter the model. We can use a hold-out dataset to help determine the halting point although an automated stopping criteria is much more desirable.

While bagging's intention is to reduce prediction variation, gradient boosting is explicitly reducing prediction bias. In empirical studies boosting has consistently outperformed bagging. However, somewhere in between there must be a place where these two methods meet to produce a low variance, low bias predictor.

3.3. Putting the pieces together

The previous section presented prediction methods with some very desirable features.

- Decision trees capture non-linear interaction effects.
- Decision trees can handle continuous, ordinal, and nominal variables as well as missing values.
- Decision trees are scalable to massive datasets.
- Bagging reduces prediction variance.
- Pasting provides an implementation of bagging that reduces the size of the dataset that needs to be in main memory.

- Bagging and pasting preserve an independent test set that can give unbiased estimates of generalization error.
- Boosting reduces prediction bias and is applicable to a large class of prediction problems.

Borrowing the principles from each of these methods forms a framework for building predictive models from massive datasets. Consider the following framework.

1. Initialize $f(x) = c$ where c minimizes $\hat{J}(c)$.
2. Subsample k observations from the dataset. Save the other observations for a validation set.
3. Propose modifying $f(x)$ by finding a simple and stable function $g(x)$ such that $\hat{J}(f + g) < \hat{J}(f)$, using only the k subsampled observations to compute $\hat{J}$.
4. Shrink the proposed improvement function as $g(x) \leftarrow \lambda g(x)$ where $0 < \lambda \leq 1$.
5. Estimate the improvement $g(x)$ makes in generalization error using the other half of the dataset.

$$\Delta J = \sum_{i \in \text{validation}} L(y_i, f(x_i) + g(x_i)) - L(y_i, f(x_i)) \tag{4}$$

6. If ΔJ is positive then update the current predictor as $f(x) \leftarrow f(x) + g(x)$ and return to (2). Otherwise if on the previous several iterations ΔJ has been consistently negative, exit the algorithm.

Step one initializes the predictor to the best constant predictor. As in bagging, pasting, and stochastic gradient boosting, step two uses a fraction of the dataset in order to decrease the variance of the predictor modification and to preserve some observations for validation. In practice, half-sampling seems to work well. Step three relates to boosting's greedy step by finding an improvement that seems to offer a decrease in the training error. Shrinking the proposed $g(x)$ in step four requires the algorithm to perform more iterations and therefore more computation time. But since using $\lambda < 1$ empirically seems essential to obtaining the best prediction models, the rule of thumb is to make it as small as computationally possible. Step five takes advantage of the fact that learning $g(x)$ did not involve some of the observations so that we can compute a nearly unbiased estimate of the improvement in generalization error. This automates the process in step six of deciding when to halt the iterations. Plots comparing ΔJ with the generalization error show that the iterations for which ΔJ begins to go below zero

correspond to minimal values of the generalization error. Note that another path may have reduced the generalization error further, but at least on the greedily selected path this framework describes we can regularly find the best predictor.

4. Discussion

The framework described in section 3.3, borrowing strength from each of the algorithms discussed, can generate algorithms that are accurate, scalable, and completely automated. By substituting the loss function of our choice and a scalable prediction model like decision trees, the framework produces algorithms that seem ideal for massive datasets. Future research will need to tune the steps in the framework and adapt it for special applications. Already this framework has produced an efficient algorithm for high-dimensional density estimation for massive datasets (Ridgeway, 2000). Furthermore, research at RAND is learning the relationship between cost of health care and patient characteristics from large Medicare datasets. Algorithms derived from this framework are producing the best estimates we have seen to date.

To conclude, in this paper I have argued that when extracting prediction models from massive datasets

- the analyst can fit more flexible and accurate prediction models
- and the computational cost of fitting those models can be enormous due to complexity in storing and accessing the data.

The task before us is to discover and tune algorithms that provide accurate predictions in the face of data access complexity. The papers referred to earlier show extensive empirical evidence that these innovations show remarkable improvement in prediction accuracy. The framework presented here, is modular so that we can tune parts of the derived algorithms for massive dataset applications. Specialized subsampling algorithms, such as delegate sampling (Breiman and Friedman, 1984) and data squashing (Madigan et al., 2000), and even more efficient tree algorithms are likely to provide workable solutions. I believe the next line of empirical research will continue to verify that these algorithms can produce some of the best predictive and computational performance to date.

References

Bauer, E. and R. Kohavi: 1999, 'An Empirical Comparison of Voting Classification Algorithms: bagging, boosting, and variants'. *Machine Learning* **36**(1/2), 105–139.

Breiman, L.: 1996, 'Bagging predictors'. *Machine Learning* **26**, 123–140.

Breiman, L.: 1997, 'The uncertainty principle in statistics'. Lecture notes from the Generalisation in Neural Networks and Machine Learning - NATO ASI, at the Isaac Newton Institute for Mathematical Sciences, Cambridge University.

Breiman, L.: 1999a, 'Pasting Small Votes for Classification in Large Databases and On-Line'. *Machine Learning* **36**(1/2).

Breiman, L.: 1999b, 'Using Adaptive Bagging to Debias Regressions'. Technical Report 547, University of California, Berkeley, Statistics Department.

Breiman, L. and J. Friedman: 1984, 'Tools for large data set analysis'. In: E. Wegman and J. Smith (eds.): *Statistical Signal Processing*. Marcel Dekker, Inc., pp. 191–197.

Breiman, L., J. H. Friedman, R. A. Olshen, and C. J. Stone: 1984, *Classification and Regression Trees*. Wadsworth.

Freund, Y. and R. Schapire: 1997, 'A decision-theoretic generalization of on-line learning and an application to boosting'. *Journal of Computer and System Sciences* **55**(1), 119–139.

Friedman, J.: 1999a, 'Greedy Function Approximation: A Gradient Boosting Machine'. Technical report, Stanford University, Statistics Department. Available from http://www-stat.stanford.edu/~jhf.

Friedman, J.: 1999b, 'Stochastic Gradient Boosting'. Technical report, Stanford University, Statistics Department.

Gehrke, J., R. Ramakrishnan, and V. Ganti: 2000, 'RAINFOREST - A framework for fast decision tree construction of large datasets'. *Data Mining and Knowledge Discovery* **4**(2/3), 127–162.

Madigan, D., N. Raghavan, W. DuMouchel, M. Nason, C. Posse, and G. Ridgeway: 2000, 'Likelihood-based Data Squashing: A Modeling Approach to Instance Construction'. In: H. Liu and H. Motoda (eds.): *Instance Selection and Construction - A data mining perspective*. Kluwer Academic Publishers, Chapt. 12.

Quinlan, J. R.: 1993, *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

Ridgeway, G.: 1999, 'The state of boosting'. *Computing Science and Statistics* **31**, 172–181.

Ridgeway, G.: 2000, 'Looking for lumps: boosting and bagging for density estimation'. In: *Proceedings of Non-linear Methods and Data Mining*.

Author's Vitae

Greg Ridgeway
is an Associate Statistician at the RAND Corporation. He received his Ph.D. in Statistics from the University of Washington. His research interests include statistical modeling of massive datasets and boosting and bagging for prediction.