

Adaptive Incremental Framework for Performance-Driven Data Mining

^{1,2}Boštjan Brumen, ¹Tatjana Welzer, ²Hannu Jaakkola

¹University of Maribor, Faculty of electrical engineering and computer science
Smetanova 17, SI-2000 Maribor

²Tampere University of Technology, Pori School of Technology and Economics
Pohjoisranta 11, FIN-28101 Pori

e-mail: {brumen|welzer}@uni-mb.si, {brumen|hj}@pori.tut.fi

Abstract. In this work, we present a framework for performance-driven data mining. Many data mining tasks, such as classification, prediction or estimation, require accurate answers. We can measure a performance of a data mining algorithm during the run and try to draw conclusions based on the recorded performance. It is expected that the performance is increasing when we increase the amount of data used. In our framework, we model the performance by using several function models. The selection of the appropriate model is made on-line. The main contribution is a possibility to model the performance of a learning algorithm and use the model for forecasting how it would behave had it more data available. Additionally, one can estimate both run-time costs and data preparation costs early in the data mining process.

Introduction

Recently, the Data Mining (DM) has been brought to the attention of business community [Berry, 2000]. One main reason for this is the general recognition that there is untapped value in greater collections of data, that such structures are indeed useful not only as repositories of atomic pieces of information, rather than the database seen as a whole provides a lot of information. The concept of Knowledge Discovery in Databases (KDD) [Shapiro, 1991], [Fayyad, 1996] has been defined and researched for some time already. KDD is defined in [Fayyad, 1996] as a "nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data". The data mining constitutes only a subtask in the overall KDD process. Data Mining is used to denote the information extraction itself. It is highly interactive process and requires several decisions to be made by humans. Mining experiments may be tried in many variations, using many approaches and many techniques to solve a single problem. KDD, and data mining in particular, are thus processes with many iterations of a single or several steps.

The input to the data mining is a set of example objects and information about their properties, given in terms of a number and type of attributes. The output can be either a description of data or a model of data. In our work, we will focus on the latter.

When we build a model, we usually measure its "fitness". Several measurement techniques, such as classification error or average of estimation errors, can be used.

In general, there is no crystal ball that could tell us in advance what will be the performance of the induced model. Several general theories exist regarding the number of samples required for a learning algorithm (which is usually used in the DM process) to learn well (e.g. PAC theory of learning), but the theories are either too general or require a deep knowledge about the structure of the algorithm used. Cohen [Cohen, 1995] observes, "...is no longer true that we can predict how a system will behave by looking at its code."

Such a framework would enable a user to see how different algorithms behave and how (and if) the solution was found. When building a model based on the data available, one needs to have training and test data, which need to be prepared in advance. A domain expert is required for the task. Obviously, a human intervention is expensive as well as not too reliable when many data need to be manipulated by hand. Thus, we need an approach where as few data as possible are used for model building. Furthermore, one should be able to have an idea how (and if) the model performance converges early in the process, and at the same time to have an opportunity not to waste the resources on an algorithm (or approach), which does not lead to the desired solution.

In the following section, we present a framework that addresses the issues. In the next section, we present preliminary results of a prototype built and in the last section we conclude with final remarks and outline our future work.

Adaptive Incremental Framework

The problems outlined in the introduction led us to the development of a framework of adaptive incremental approach. At the development of the approach we were guided by the following goals: to develop an approach to use as little data as possible within a data mining process; to aid in assessment of costs of data mining process; to minimize the costs associated with manual data preparation; to assist the data miner in the data mining process; to forecast accurately the performance of a learning algorithm, and to assist data miner in the future data mining processes.

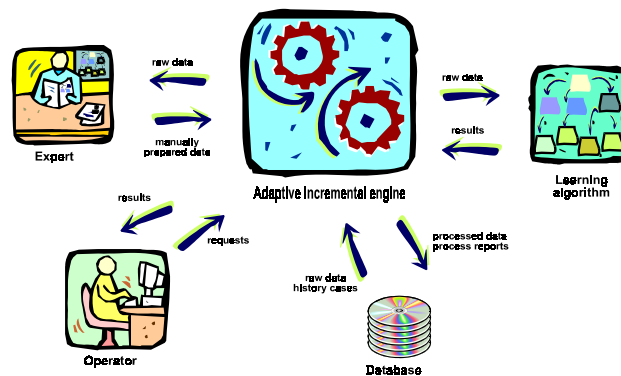


Figure 1: The model of adaptive incremental framework

We present our framework of adaptive incremental approach to the performance-driven data mining as a model, consisting of several parts, as depicted in Figure 1: Operator, Expert, Database, Learning algorithm, and Adaptive incremental engine.

In the following sub-sections, we present each component of the framework in a detail.

Operator

In our model, the operator is an external entity, which controls and monitors the data mining process. The operator's task is to provide acceptable performance parameters (costs, performance rate) and input data.

The operator controls the execution of the process and decides (or confirms) on the parameters of the executions (such as the next sample size in a schedule). The engine displays the intermediate results on-line, so the expert can actively interfere with the procedure, or the procedure runs by itself until the criteria set by the operator are met. If they are not, the process stops at a certain point and the operator is notified of the fact.

Expert

The expert in our framework has the task of supplying the system with the knowledge. The expert knowledge comes in form of examples. The learning is supervised – the examples are fed into a learning algorithm and then the results are checked.

An expert can be a human being or an intelligent agent. Since the task is learning, the expert is most likely to be a human being.

The expert or a group of experts has a task of preparing records from the database. The preparation task itself depends on the data mining problem. In case of classification, the expert has to manually classify a record into a pre-defined class. In case of prediction or estimation, the expert has to assign a value (or several values) to the attribute(s).

One problem that arises with human experts is that they do not always agree. Different experts may, for example, classify the same record differently, causing problems for the classifier to learn properly. For this reason we can expect that the training data are not clean. The same problem is even worse with the case where the values are not bound (e.g. predicting the loss associated with a customer). Thus, the learning algorithm will not achieve 100% accuracy as long as the human experts do not agree [Cohen, 1995].

Database

The database is the data repository of the adaptive incremental framework. In the database several types of data are stored: raw data, data prepared by the expert, data processed by the learning algorithm, and history cases of similar problems together with the description of the problem and the performance curves.

The database part can itself be a computerized system with which the engine part communicates.

In the following sub-sections, we present each part of the database in turn.

Raw data

Raw data are data from an operational database or a data warehouse. It is expected, as is natural for the data mining process, that the data are cleaned. That means that the data are cleaned and prepared for the learning phase. As mentioned, we cannot expect 100% clean training and test data. This fact needs to be taken into a consideration also when selecting a learning algorithm, which is capable of handling such data.

Data processed by the expert

After the raw data are cleansed and stored into the database, it is a task of the expert to classify or otherwise prepare the data manually according to his or her expert knowledge. The cases (records, instances) prepared manually serve as a training set for the learning phase of the algorithm, as well as for the testing phase.

The manual preparation process is considered to be very expensive. This is especially the case when a good and deep knowledge of the problem background is required and when there are many cases to be prepared. It is well known that people start making errors after some time of work and the duller the work is the more errors are made. Thus we take an adaptive and incremental approach to the problem. We want to minimize the number of records the experts have to prepare manually.

After the data are prepared, they are stored into the database to serve as training and testing cases.

Data processed by the learning algorithm

The purpose of the training phase is to “teach” the learning algorithm. Once the learning (also called training) phase is over, the performance of the algorithm is tested on the test cases. Once the performance is satisfactory (that is, when the operator is satisfied with the results), the induced result (e.g. a decision tree or a neural network) can be used for other, new and unseen cases. The new cases are then processed by using the induced system, and stored into the database.

History cases of similar problems

The purpose of the history cases is to keep record of several parameters that were calculated during the process. The history cases together with the description of the problem and the performance curves serve as a repository for analysis of the performance of the framework and to serve as a basis for the future improvements.

Learning algorithm

The task of the learning algorithm is to learn knowledge (patterns) from the data. The inputs to the learning algorithm are the manually prepared data. The outputs are a performance measure¹ and the time required for completing the task. In case the learning algorithm is a classifier, the performance measure can be an error rate, which is a number of wrongly classified instances compared to the number of all instances. Other performance measures depend on the data mining problem. In prediction and estimation problems, the measure of performance can be a distance between the desired and the calculated outcome, or a weighted sum of square errors or, in general, any user-defined (numeric) measure.

Generally, any learning algorithm, whose output is its performance and the time needed for the completion of the learning/testing phase, can be used in our framework. The only limitation is, as mentioned earlier, that the algorithm is capable of handling non-clean data.

Several learning algorithms may be used in our framework. Most popular ones are decision tree induction, neural networks, naïve Bayes, genetic algorithms, regression methods, and classification rules.

¹ Actually, the performance of model induced by an algorithm on the given data is measured.

Adaptive incremental engine

The engine is the heart of the model. It contains several inseparable decision-making structures. It has the following parts (to be described in detail later): Core engine; Sampling algorithm; Learning curve monitoring unit; Prediction unit and Cost estimator.

Core Engine

In order to get a reliable performance measurement, one could repeatedly train and test an algorithm, then average the scores obtained during the testing phases and derive parametric confidence intervals by t-tests, as described in [Cohen, 1995]. For example, suppose we have a data set of 1000 samples, to each of which a human expert had manually assigned a corresponding class. Then, we draw 100 samples from the total data set, train the classifier on 90, test it on the remaining 10, and repeat this process ten times, using up all the data in the dataset. At the conclusion of the experiment, we can average the performance scores obtained during each of the ten tests with 100 (10 times 10) samples and obtain the confidence intervals. Overall, 900 samples would have been used for training and 100 samples – none of which were in the training set – would constitute the test sets. The advantage of this scheme is its separation of training and test sentences; it guards against the possibility of training a system to respond correctly to idiosyncrasies of the training set. Unfortunately, the scheme is very expensive. There is a better way to use training sets.

A more efficient training and testing scheme is called *k-fold cross validation* (Cohen, 1995). It requires a set of problem-solution pairs.

When a classifier is tested with a ten-fold cross-validation, its original set of n samples is divided into ten sets, and each is used nine times for training and once for testing. This is efficient in the following sense: We want to train a learning algorithm with all the available data, because its performance is expected to improve as it encounters more items. If we train on n items and immediately test on the same items, however, we will not find out how our algorithm performs on items it hasn't seen during training. Cross-validation allows us to train a system on almost all the items (90% for ten-fold cross-validation), test it on all the items, and still discover how the system performs on unseen items.

Cross-validation is a resampling technique that reminds of the bootstrap testing. The common idea that underlies these techniques is to estimate the true (population) performance by treating a sample as if it is the population and repeatedly drawing samples from it. A true bootstrap approach, of course, involves resampling with replacement.

To get good estimates of performance from bootstrap resampling, it is recommended that k should be at least 200 [Weiss, 1989]; the bootstrap procedure thus involves a lot more computation than, say, a ten-fold cross validation. Not only that it requires much more computation, it is also much more expensive regarding the manual preparation of the training and test sample.

K -fold cross-validation and bootstrapping test performance of a classifier after a given amount of training. To chart a learning curve, however, it is desirable to measure the performance repeatedly as the amount of training is increased. For this purpose, Lehnert and McCarthy [Lehnert, 1993] introduced a variant of a cross validation, called *incremental k-fold cross-validation*, which is a widely adopted method by other researchers (see e.g. [Frey, 1999], [John, 1996]):

Unlike the previous cross-validation procedures this one tests an algorithm (McCarthy and Lehnert used OTB) twenty-six times before training is completed, but it doesn't clear

memory until then, so we get to see the cumulative effects of training. The procedure trains OTB on just one item, then tests it on all fifty from the test set; then it trains on a second item and tests again, and so on, until it has trained OTB on ten items. After that, the procedure adopts increments of 10 training items until OTB has trained on 100 items, and increments of 50 thereafter. Finally, the procedure clears memory, forgetting everything learned during training, and repeats itself.

At the end of the procedure, McCarthy and Lehnert average k performance measures taken after 1, 2, ..., 10, 20, ..., 100, 150, ..., 450 items of training, respectively. The performance measure is the proportion of the items in fifty test cases that were correctly classified. They knew the size of their database *in advance*, so they chose the one tenth of the size for the testing.

Adaptive incremental k-fold cross-validation

We modify the McCarthy's and Lehnert's procedure to be able to cope with the following requirements: in general, the database size is not known in advance; only the current sample size is known; the learning phase, and especially the testing phase should use as many items as possible; and, the training procedure should stop if the current costs are exceeded, or the performance is on the appropriate level, or the user (or the system) aborts the procedure because the estimated costs for the performance level are too high.

For this reason we developed an adaptive incremental k-fold cross-validation method (Procedure 1), which takes into the account the requirements set in the framework.

Procedure 1: Adaptive incremental k-fold Cross-Validation

Repeat

1. Shuffle the items in the training set
2. Divide the training set into k equal parts of size n
3. Do $i = 1$ to k times:
 - a. Call the i^{th} set of n samples the test set and put it aside.
 - b. Train the system on the remaining $k - 1$ sets; test the system on the test set, record the performance.
 - c. Clear memory, that is, forget everything learned during training.
4. Calculate the performance of training averaged over the k test sets.
5. Increment the size of the training set (i.e. add additional data to the *existing* training set) based on the properties of the performance curve

Until (performance is satisfactory) or (costs are exceeded)

Unlike the *incremental k-fold cross-validation* procedure, this one tests a classifier for an unknown number of iterations. The number of iterations is defined *adaptively*, based on the properties of the performance curve built in run-time (as described in step 5). The main modification we made to the incremental k-fold cross-validation is the adaptive increment and the stopping criteria.

We still get to see the cumulative effects of training. The procedure repeatedly (k times) trains a classifier on $n-n/k$ items, then tests it on n/k from the test set (and records the i^{th} performance). The overall performance on n items is calculated by averaging the k intermediate results.

Afterwards, the size of the *original* sample is incremented by a factor, which is calculated *adaptively*, based on the properties of the performance curve built in run-time. This new sample is composed of old items (used in the previous iteration) plus some new

ones, picked randomly from the samples. We added some “new knowledge” to the knowledge base. When the classifier is run on the new sample, we get to see a cumulative effect of the new knowledge on the performance. The effect can be either positive (e.g. the error rate decreases) or negative (e.g. the error rate increases).

The adaptive incremental cross-validation is an improved version of a standard incremental cross validation because we do not need to know the exact number of items in a sample in advance. This is especially advantageous when the items for the sample need to be prepared or pre-processed. Additionally, we can observe the cumulative effect of training on-line (after each set of k -fold cross-validations), instead of at the end of the whole incremental cross-validation procedure. Our approach increases the test (and training) sample size every time, while the original ones keeps the test size constant, and only increases the training sample size.

If the performance curve does not behave well, we can conclude that the sample size is too small. If there is no training effect, the performance curve might have flattened out and our goal is accomplished. Last, while the adaptive cross-validation process is running, some other processes can run in parallel, such as observation of learning curve, prediction of future behavior and others.

Sampling algorithm

The task of the sampling algorithm is to take a sample of raw data from the database and hand it over to the expert to prepare. The expert returns the prepared data to the control unit, which in turn, initiates the learning algorithm.

A central component we use is a *sampling schedule* (Equation 1)

$$S = \{n_0, n_1, n_2, \dots, n_k\}, n_i < n_j \text{ for } i < j \quad (1)$$

where each n_i is an integer that specifies the size of a sample to be fed into a learning algorithm. We start with a simple arithmetic schedule S_a (Equation 2):

$$S_a = \{n_0 + (i*k)\}; 1 \leq i \leq m \quad (2)$$

where n_0 and k are arbitrary numbers and i is an index; m denotes the number of schedule points and depends on the outcome of subsequent runs of the algorithm (thus it is adaptive). The parameter k denotes the step and depends on the performance curve monitoring unit.

Sampling from the raw data database is done without the replacement. This means that once an item is selected, it is handed over to the expert to prepare it manually. After it is prepared, it is not returned to the raw data database, but to the database of the data prepared by the expert. In the next sampling, this item will not be re-selected for the manual preparation. This way the costs of manual preparation are reduced since an item is prepared only once. Additionally, the item will be prepared only once. If the expert preparation is done by a group of experts, the same item could be prepared in different ways at different time points.

Performance curve monitoring unit

The performance curve depicts the performance of the learning algorithm. Monitoring of the performance curve is a very important task, because the performance curve explains the behavior of the learning algorithm.

As discussed in [Brumen, 2000a], the performance curves are not well behaved. They have sudden drops or rises which are hard to explain, but in general, on a larger scale they do follow some model.

The performance curve monitoring unit has a task of observing the shape and the quality of the learning curve². The most important task is to detect the point where the learning curve starts to behave well.

We found out that in many cases the performance curve starts behaving well when it becomes monotonically decreasing and concave up (in case the learning curve depicts the diminishing error, it is then monotonically increasing and concave down) for a given number of points.

Let us define the concavity for the case where we have discrete points.

Definition 1: Let f be a set of ordered points (x, y) and m be the number of points in the set. The graph of f is concave upward if $y_{i-2} > y_{i-1} > y_i$ and

$$\frac{y_{i-1} - y_{i-2}}{x_{i-1} - x_{i-2}} < \frac{y_i - y_{i-1}}{x_i - x_{i-1}} \quad (3)$$

for $2 \leq i \leq m$. The graph of f is concave downward if the inequity sign in Equation 3 is turned.

The learning curve monitoring unit's task is to observe whether the conditions of discrete concavity are met. If they are, the prediction unit is invoked. The prediction unit predicts the performance.

Prediction Unit

The prediction unit has the task of predicting the performance of the learning algorithm for a given sample size n . It can predict the performance of the learning algorithm, the time needed for execution or the cost of expert classification.

For the purpose of predicting the performance the unit takes as input the points (n_i, a_i) where n denotes the sample size in i^{th} schedule point and a_i denotes the performance in i^{th} schedule point. The points (n_i, a_i) were calculated so far in the process. The prediction unit tries to fit a best-fitting function to these points. Based on the best-fitting function one can predict the future value of a for any n .

The foundation of the prediction unit for the performance is the fact that the best fitting function for the learning algorithm is the power law [Frey, 1999]. Although this is true for most of the data sets, this is generally not true. For this reason we propose in our framework a dynamic function fit. The purpose of dynamic function fit is to find the best function to fit the obtained data based on the Chi-Squared test. We propose the following functions, while others can easily be added:

- power: $y = a + b \cdot x^c$ (4)

- linear: $y = a + b \cdot x$ (5)

- log-linear: $y = a + b \cdot x \cdot \log(x)$ (6)

- exponential: $y = a + b \cdot e^{c \cdot x}$ (7)

² A learning curve depicts the dependency between the data size used for learning and the performance output; the performance curve and the learning curve are synonyms.

In Equations 4-7, the y (bound variable) denotes the performance and the x (free variable) denotes the sample size.

The reason we added the constant a in Equations 4-7 is because we believe that the learning algorithm never acquires the 100% correct performance on the data. We believe the performance of the learning algorithm approaches asymptotically to a given (but unknown) number. Additionally, the forms we use are more general even in the case when the performance actually reaches 100%.

Of course, several other functions can be observed whether they are best fit for the obtained points. Other functions can easily be added to the framework.

The method and the details of finding the parameters for the functions are left for the implementation, although the least-square method has proven to be working adequately well [Brumen, 2001].

Estimating the cost in any task is a very important part of the overall process. One can – based on the estimation – decide whether to continue with the task or not. The basic idea is that for each task, some resources are available, and if the hint is that the resources would be extinguished without the goal to be accomplished, the process can be stopped early enough to spare the (unnecessary) costs. The costs are estimated in very similar way as the performance – we observe the cost-vs.-number of samples, draw the points and try to find the function that fits best.

Conclusion and future work

In the paper, we presented a framework of adaptive incremental approach. The main contribution is a possibility to model the performance of a learning algorithm and use the model for forecasting how it would behave had it more data available. Additionally, one can estimate both run-time costs and data preparation costs early in the data mining process.

We have partially implemented the framework of adaptive incremental approach in a prototype. Currently, the prototype is limited to the estimation of future performance based on partial results [Brumen, 2001], and the estimation of the sample size needed to obtain the desired performance [Brumen, 2000b]. Due to the lack of space, we will not sketch the results. Interested reader is referred to the works mentioned above.

The results show that the performance can be reliably estimated by using only a small subset of the total data available, in many cases (with majority of datasets) within one percent. In one case, the data set was so large that it was unmanageable with the selected learning algorithm. Despite that, we were able to build a very reliable model with a small subset.

The cost estimation part is very sensitive to the granularity level of the desired performance. In our case, we were measuring the amount of data needed to obtain a pre-defined accuracy of a classifier. When the granularity was larger than two percents, the results were extremely satisfactory, but within one percent, the variations in performance caused unreliable estimations.

The framework of adaptive incremental approach can be used in data mining projects that are performance-driven, i.e. when a model is built, and expectations regarding the performance of a model are set. By using this approach, a data miner can assess the costs associated with the manual data preparation (usually done by a domain expert) for the desired performance level. This way the costs can be minimized. Further, a user gains an insight into the possible future behavior of the learning algorithm used already early in the

process, when not many data are used and when the run-times are relatively low. Additionally, the results of each project are stored together with all the accompanying parameters, so that the next project can use the data available for comparison and better decisions regarding which approach and learning algorithm to use.

In our future work, we plan to implement the framework into a fully-functioning data mining suite with all the support described.

References

- [Berry, 2000] Berry, Michael A. J.; Linoff, Gordon: *Mastering Data Mining: The Art and Science of Customer Relationship Management*, John Wiley & Sons, Inc., New York, USA, 2000
- [Brumen, 2000a] Brumen, Boštjan; Jaakkola, Hannu; Welzer, Tatjana; Rozman, Ivan: Predicting Minimum Sample Size in Data Mining Tasks: Additive Approach. In (Jaakkola, Hannu; Kangassalo, Hannu Eds.): *Proceedings of The 10th European-Japanese Conference on Information Modelling and Knowledge Bases*, Saariselkä, Finland, pp. 264-270, 2000
- [Brumen, 2000b] Brumen, Boštjan; Welzer, Tatjana.: Predicting data mining costs: additive approach, In (Lasker, George, Ed.): *Proceedings of the 12th International conference on systems research, informatics and cybernetics - InterSymp-2000*, Baden-Baden, Germany, 2000
- [Brumen, 2001] Brumen, Boštjan; Welzer, Tatjana; Golob, Izidor; Jaakkola, Hannu: Convergence Detection in Classification Task of Knowledge Discovery Process, In *Proceedings of Portland international conference on management of engineering and technology*, Portland, Oregon, USA, 2001
- [Cohen, 1995] Cohen, Paul R.: *Empirical Methods for Artificial Intelligence*, Mit Press, Cambridge, MA, USA, 1995
- [Fayyad, 1996] Fayyad, Usama; Shapiro-Piatetsky, Gregory; Smyth, Padhraic; Uthurusamy, Ramasamy; (Eds.): *Advances in Knowledge Discovery and Data Mining*, AAAI Press, USA, 1996
- [Frey, 1999] Frey, Lewis J.; Fisher, Douglas H. Jr.: Modeling Decision Tree Performance with the Power Law, In (Heckerman, David; Whittaker, Joe; Eds.): *Proceedings of the Seventh International Workshop on Artificial Intelligence and Statistics*, San Francisco, Morgan Kaufmann, Inc., USA, 1999
- [John, 1996] John, George H; Langley, P.: Static Versus Dynamic Sampling for Data Mining, In (Simoudis, Evangelos; Han, Jiawei; Fayyad, Usama, Eds.): *Proceedings of the Second International Conference on Knowledge Discovery in Databases and Data Mining*, AAAI/MIT Press, USA, p. 367-370, 1996
- [Lehnert, 1993] Lehnert, Wendy G.; McCarthy, Joseph; Soderland, Stephen; Riloff, Ellen; Cardie, Claire; Peterson, Jonathan; Feng, Fang Fang; Dolan, Charles; Goldman, Seth: Umass/Hughes: Description of the CIRCUS system as used for MUC-5. In *Proceedings of Fifth Message Understanding Conference*, Morgan Kaufmann Publishers Inc., San Mateo, USA, pp. 277-291, 1993
- [Shapiro, 1991] Piatetsky-Shapiro, Gregory; Frawley, William J. (Eds.): *Knowledge Discovery in Databases*, AAAI/MIT Press, USA, 1991
- [Weiss, 1989] Weiss, Sholom M.; Kulikowski, Casmir A.: *Computer Systems that learn*, Morgan Kaufmann Publishers Inc., San Mateo, CA, USA, 1989