

Pervasive Parallelism in Data Mining: Dataflow Solution to Predicting Customer Behavior

Srivatsava Daruru
Department of Computer Sciences
The University of Texas at Austin
The Ideal Laboratory
Austin, Texas, USA
(512) 471-8980
vatsava@cs.utexas.edu

Larry Schumacher, Nena Marín
Innovations Laboratory
Pervasive Software, Inc.
Austin, Texas, USA
(512) 231-6026
nmarin@pervasive.com
lschumacher@pervasive.com

Joydeep Ghosh
Department of Electrical and Computer
Engineering
The University of Texas at Austin
Austin Texas, USA
(512) 471-8980
ghosh@ece.utexas.edu

ABSTRACT

Churn prediction and management is critical for companies in the fast and competitive telecommunication market. In this paper, we introduce the dataflow computational model in the context of data and computationally intensive high performance parallel data mining. We present a highly scalable and robust model capable of scoring "propensity-to-churn" at the rate of 50,000 customers in a 1.6GB test set (Orange Labs France Telecom, KDD Cup) in 3 minutes on commodity 16-core CPUs. This is an effective scoring runtime of 3.6 milliseconds per customer, orders of magnitude faster than some systems. As a competitor in this year's KDDcup data mining competition, this speed enables more iteration towards improved performance; while the research focus was speed, resulting predictive accuracy ranked higher than 70% of competitors.

Categories and Subject Descriptors

H.2.8 [Database Management]: Database Applications—
Data Mining.

General Terms

Algorithms, Management, Performance, Design, Verification.

Keywords

Predictive Modeling, Dataflow Machines, Scalability.

1. INTRODUCTION

Customer relationship management (CRM) analytics are known for powerful synthesis of historical customer data. However, the data's potential often goes untapped. Traditional marketing strategies focus on customer segmentation and identifying "good customers". But retention and loyalty has an element of timing. In the book, *The Ultimate Question*, [1] Frederick Reicheld of Bain & Co. says every 5% increase in retention equals a 25% to 100% increase in profitability. The KDD Cup 2009 offered the

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

PAW '09, Oct 20–21, 2009, Washington, DC.
Copyright 2009 PAW

opportunity to work on large CRM database from the French Telecom company to predict the propensity of customers to switch provider (churn), buy new products or services (appetency), or buy upgrades or add-ons proposed to them to make the sale more profitable (up-selling). "The participants in the KDD Cup 2009 were given 50,000 records of CRM data from a telecommunications company and tasked with predicting three target variables: churn, appetency and up-selling. In 2009 KDD Cup, there were both a "fast challenge" (to be completed in five days) and a "slow challenge" (with a deadline of about a month from dataset availability). In the slow challenge, the entrants had the option of using either the full dataset of 15,000 variables or a smaller subset with 230 variables." [2] The winning entry for each track was decided by an average of AUC scores for each of the three target variables. The production system was developed by Orange Labs [2]; the research division of France Telecom. This in-house system builds knowledge on customers in the form of a model which once applied produce scores. A score (the output of a model) is an evaluation for all instances of a target variable to explain propensity to churn, appetency or up-selling. Scoring engines predict on a given population quantifiable information computed using input variables which describe instances. Scores characterize customers and are therefore used to personalize the customer relationship. A requirement of such scoring engines is the ability to scale on very large datasets with hundreds of thousands of instances and thousands of variables. The rapid and robust detection of the variables that have most contributed to the output/prediction can be a key factor in a marketing application. A large portion of this knowledge discovery challenge is dealing with a very large database that includes heterogeneous noisy data (numerical and categorical variables), and unbalanced class distributions. Automated data mining techniques coupled with the unprecedented volumes of electronically stored historical data can revolutionize the speed and quality of these predictions. Before one can fully profit from this revolution, data mining faces the challenge of mapping well established algorithms to parallel architectures. The simple approach to parallelization is to choose an embarrassingly parallel data mining algorithm and deploy this single program to multiple data (SPMD) on multiple computers (parallelizing in space). For all other non-embarrassingly parallel algorithms, re-architecture using some combination of parallelization in time and in space is the alternative. Our research focuses on shared memory multi-core processors, a parallel computing architecture where processor communication happens

through variables stored in a shared address space. Concurrency issues like threading, synchronization and consistent memory models make shared memory parallel programming difficult. We present a naturally parallel programming model called dataflow [3][7]. The essence of dataflow programming is the concept of execution as the streaming of data flows through a graph. The graphs are composed using standard operators in a java library in support of maximum programmer productivity. The composed graphs are auto-tuned to available resources like number of cores and heap size in support of infinite data volumes. Dataflow exploits multiple models of parallelism; such as pipeline, horizontal and vertical parallelism in support of maximum application efficiency. Orange Labs provided KDD Cup participants with a baseline solution based on a Naïve Bayes implementation. The solution performance measured in AUC (area under the curve) was good. However each batch scoring task took 12 hours, too expensive to operate in real-time on a large-scale system. In this paper, we first review dataflow process networks; a method of computation used in signal processing systems that operates on infinite streams of data values[1], to realize concurrent processing of functional parallelism. We adapt the scalable Winnow algorithm by Littlestone [7] to use dataflow networks. The dataflow Winnow solution transforms the large scale CRM training dataset into streaming dataflows. It operates on these streams to build a linear discriminant function that is mistake driven. The Winnow weight vector is updated only when the algorithm is not able to correctly classify a training sample. Predictions are then generated quickly by applying the Winnow weights to new samples.

In this paper, we make two main contributions:

1. We propose a “real-time” customer behavior prediction approach based on balanced Winnow and the dataflow computational model. We compose a dataflow graph that exploits horizontal and vertical parallelism.
2. We demonstrate linear scaling on the highly dimensional and noisy CRM dataset.

2. PROBLEM DEFINITION

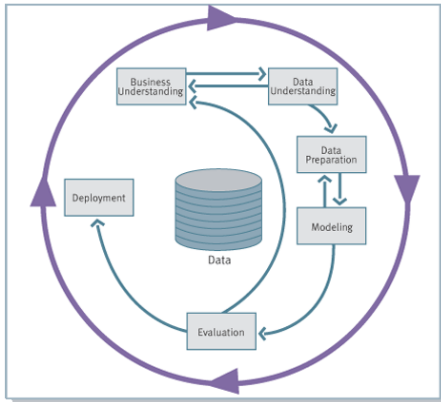


Figure 1 - CRISP-DM Methodology

Figure 1 shows the phases of the methodology defined by the CRISP-DM [11] Cross Industry Standard Process for Data Mining. It is important for problem definition to identify the phases needed in order to prepare the data for analysis, build a model, apply and evaluate the model. We will set aside the first (business discovery) and last phase (deployment) which are not part of the KDD Cup 2009 requirements. The KDD Cup

challenge is about building a model capable of predicting customer behavior based on historically well defined customer choices. The datasets provided contain CRM records for real customers who have previously exercised their choices in churning, appetency and upselling.

2.1 Data Understanding:

During the first month of the KDD Cup, the datasets were released with toy labels. This allowed us to perform data discovery in order to ascertain the nature of the data. We could determine sparseness, outliers, frequency distributions, basic stats but not relevancy to the target. This discovery defined the amount of preprocessing required in order to prepare the data for analysis.

2.2 Data Preparation

Given the large number of attributes/variables (15000), we expected Feature Selection to be the key to winning the competition. The data was also very noisy and heterogeneous. We had to address data preparation for both categorical and numerical variables. The standard deviation in a large number of variables was zero. The range of values on numerical attributes indicated the need for normalization. We considered several normalization strategies and tried all of them for best results. The plan for pre processing steps included:

- 1) Normalize the numerical data by $(x-\text{mean})/\text{std}$ also known as ZScore. We also tried normalization by range (Max – Min) and Log scaling of features with high skewness (mean-median/mean).
- 2) Sort by standard deviation and remove the attributes having standard deviation close to zero.
- 3) Remove the attributes and the features with entries mostly missing. Compute missing value counts by both columns and rows.
- 4) Get the number of distinct values for categorical features
 - a. if they are small, the features can be represented in binary coding scheme (for winnow below)
 - b. else if it contains all distinct values, it can be removed.

2.3 Modeling

We seek a model for a production environment where volumes of data are very large. We cannot afford several passes over the data during training. Additionally there are many (15,000) variables and many are irrelevant. The CRM database feeding the stream of data available to train the model is dynamic. New variables and even entities/tables can be added online. The model must therefore be able to select features on the fly based on the total list of current variables presented during training. We are also interested in a solution that parallelizes all independent steps in the analysis in order to be feasible for real-time production environments. We are considering a voting scheme in addition to the traditional best hypothesis learned approach. The voting or averaging approach is expected to produce more stable models with less over fitting. After careful consideration, the modified balanced Winnow [5][6][8][9] algorithm was selected because of the following reasons:

- 1) It takes each point at a time and hence is highly parallelizable. Easy to learn weights on each dataset partition and combine them.
- 2) It automatically learns weights for each feature and thus automatically does feature selection (by weighting of features).
- 3) It is proven to be good in high dimensional and noisy data.
- 4) Since the dimensions are high and comparable with the number of data points, the data is hopefully linearly separable and it converges fast.
- 5) We also considered using Naïve Bayes for the purpose of classifying categorical variables with too many distinct values.

2.4 Evaluation

At the end of the modeling phase, The Winnow model is in the form of a set of weights. Applying the model is as simple as:

$$\text{Predict: } \hat{y}_t = \text{sign}(w_t^+ \cdot x_t - w_t^- \cdot x_t)$$

Equation 1

The predicted values are then compared to the actual value. The performance evaluation metric as defined by the KDD Cup was AUC (area under the curve). The submission web site allows the upload of predicted values of both train and test datasets. Upon an upload of predictions for both training and test datasets, the web site evaluates the predictions using ROC analysis and posts a response AUC score on the full trainset and an AUC testset score based on only 10% of the submitted results. AUC is the area under the ROC curve. The ROC curve is the curve resulting from plotting Sensitivity (Se) on the y-axis versus 1 – Specificity (1 – Sp) on the x-axis. The basic algorithm implemented to calculate AUC was as follows:

- 1) Apply the winnow model to predict targets. Instead of using the binary predicted scores we used Wtx as a probability or discriminant values to build the ROC curve.
- 2) Figure 2 depicts the frequency distribution of positives and negatives class target values based on actual for upselling trainset. The x-axis corresponds to the values of Wtx . The y-axis corresponds to customer counts that fall into the bin at x value.
- 3) In order to build the ROC curve, we need to calculate Se and Sp value pairs based on changing the threshold value for Wtx that would separate class target into positive and negative. The black vertical line shows one such threshold value.

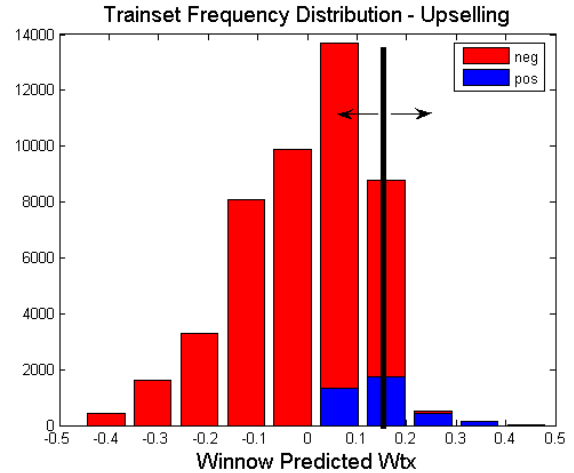


Figure 2 - Frequency Distribution for ROC Analysis

- 4) True Positives Tp corresponds to the count of “pos” (blues) that are greater or equal to the threshold.
- 5) False Negatives Fn corresponds to the count of “pos” (blues) that are less than the threshold.
- 6) True Negatives Tn corresponds to the count of “neg” (reds) that are less than the threshold.
- 7) False Positives Fp corresponds to the count of “neg” (reds) that are greater or equal to the threshold.
- 8) Sensitivity = $tp/(tp+fn)$ and Specificity = $tn/(fp + tn)$
- 9) Repeat steps 3) thru 8) to build ROC curve
- 10) Use trapezoidal rule to calculate area under the curve

3. BALANCED WINNOW FOR CUSTOMER BEHAVIOR

Littlestone's Winnow algorithm [5][6] for learning disjunctions of features where most attributes are irrelevant is one of the most fundamental approaches in learning theory. In this paper, we consider the binary classification problem to determine a label $y \in \{-1, 1\}$ associated with an input vector x .

Figure 3 shows the pseudo-code for the balanced Winnow algorithm implementation used here. For each new example x presented, the current model will make a prediction

$$\hat{y}_t \in \{-1, 1\}$$

and compare it to the true class y .

The prediction will be based on the score function f , on the example x_t and on the current weight vector w_t positive and w_t negative. In the case of a prediction mistake, the model will be updated. Positive weights vector are updated by multiplying by

the exponential of η times vector x times predicted class \hat{y}_t .

Negative weights vector are updated by dividing by

the exponential of η times vector x times predicted class \hat{y}_t .

η is the learning rate for the algorithm. A larger value of η will cause the algorithm to converge faster but may result in overshooting the solution.

For non-separable problems, a generalization of optimal hyper-plane was proposed in [9] by introducing a “soft-margin” loss term that minimizes the Winnow mistake bound (much like the Perceptron mistake bound and SVM).

- Initialize: $w_{1,i}^+ = w_{1,i}^- = \frac{1}{2N}$
- Predict: $\hat{y}_t = \text{sign}(\mathbf{w}_t^+ \cdot \mathbf{x}_t - \mathbf{w}_t^- \cdot \mathbf{x}_t)$
- Update
 - if no mistake, $\mathbf{w}_{t+1}^+ = \mathbf{w}_t^+$, $\mathbf{w}_{t+1}^- = \mathbf{w}_t^-$;
 - else,

$$w_{t+1,i}^+ = w_{t,i}^+ \frac{e^{\eta y_t x_{t,i}}}{Z_t}$$

$$w_{t+1,i}^- = w_{t,i}^- \frac{e^{-\eta y_t x_{t,i}}}{Z_t}.$$

Figure 3 - Balanced winnow algorithm.

Balanced Winnow implementation depicted in Figure 3 was modified by adding a margin. According to [9], a prediction is considered mistaken, not only when y_t (predicted) is different from y (actual), but also when the score function f multiplied by y_t is smaller than the “margin” M , where $M \geq 0$.

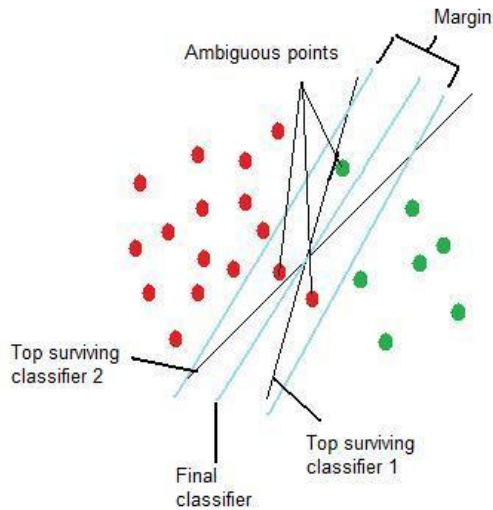


Figure 4 - Weighted Voting and Margin

Figure 4 shows the margin separating the two class values. For example, the green dots may depict churners ($y_t = 1$) and the red dots may depict non-churners ($y_t = -1$). The soft-margin is the region of separation between the two classes. Finally, top K weights during training were saved with the goal of using weighted voting on longest/most iterations surviving solution/weights. Figure 4 depicts the top surviving solutions/weights and the final voted winning solution.

4. SCALABLE CUSTOMER BEHAVIOR PREDICTION

4.1 Dataflow Programming

Dataflow[3][7] is a model of computation particularly well suited to concurrency. It arranges a program into a set of processes communicating only by way of unidirectional ports that act as FIFO queues. Data is transformed into dataflows. Processes accept data from dataflows via input ports, construct results based upon it, and push the results onto output ports [13]. Because the processes share no state, they can operate concurrently, allowing dataflow applications to take advantage of multiple processor cores. Further, the process developer need not be concerned with threads, deadlock detection, starvation, or concurrent memory access since parallel scheduling and synchronization is handled external to the process. The essence of dataflow programming is the concept of execution as the streaming of data flows through a graph [13]. As the data is streaming, only data required by any active operation need be in memory at any given time, allowing very large data sets to be analyzed. Besides offering the potential for scaling to problems larger than what the heap would otherwise permit, dataflow graphs exploit multiple forms of parallelism. By its very nature, a dataflow graph exhibits pipeline parallelism. If each operator generates output incrementally, dependent operators can execute simultaneously, just a few steps behind. Also, if the results of an operator are independent for each piece of data, the operator can be replaced with multiple copies, each receiving a portion of the original input. This is called horizontal partitioning. Finally, the output of an operator might undergo multiple sets of processing and later be merged (this is most prevalent with record data) as input to another operator. The different branches can execute in parallel; this is vertical parallelism [13].

Pervasive [4] DataRush™ is a library and dataflow engine to construct and execute dataflow graphs in Java. All threading and synchronization is handled by the framework as data is only shared through inputs and outputs. An operator is an extension of an existing class in the framework. A library of common operators is already implemented as part of Pervasive DataRush, in addition to the dataflow engine. A dataflow graph is composed by adding operators to the graph. Operators require their input sources in order to be constructed, so the wiring of outputs to inputs is done as you build the graph. Once you are finished composing, just invoke the run method and the graph begins execution. Because this is all done in Java, composition can be done conditionally based on pre-execution processing. Scalability refers to the ability to speed up your applications linearly with the number of resources available. Therefore, the most common practice would be making adjustments to the graph based on the number of available processors.

4.2 Dataflow Data Preparation

We seek a solution to Customer Behavior Prediction for the KDD Cup 2009 problem. We have chosen to preprocess the data in two passes. The first pass allows us to calculate basic statistics, distinct values discretization, missing value counts, correlation coefficients. Figure 5 depicts the data preparation dataflow application graph using 2-cores. The partitions are determined based on the number of cores. For the first pass of data

preparation, we chose to vertically partition the data. Since statistics and categorical variable discretization happens along columns/variables, we chose to partition the data into groups of columns (vertically partitioned) instead of groups of rows (horizontally partitioned).

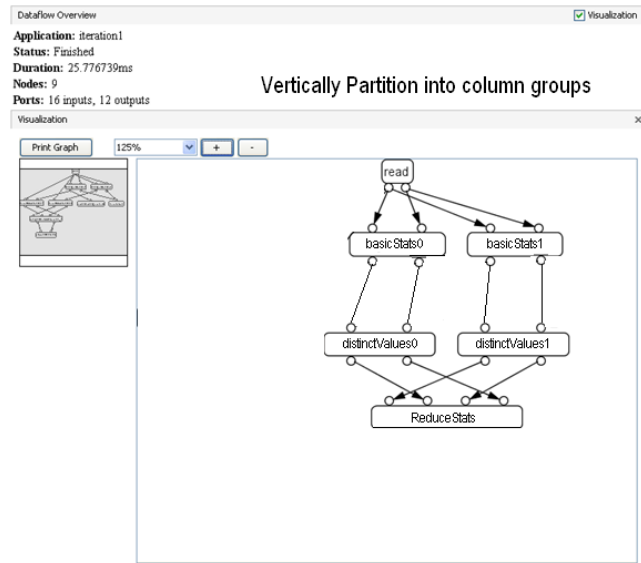


Figure 5 - Dataflow Data Preparation Application Graph.

During the second pass, we use the basic statistics and counts from the first pass to replenish missing values (optional), clean and normalize the data. The various normalization and missing value handling options could be turned on or off at runtime to facilitate algorithm tuning.

4.3 Dataflow Balanced Winnow Training

We seek a solution to Customer Behavior Prediction for the KDD Cup 2009 problem. We have chosen a linear threshold classifier for the prediction of the binary targets (e.g. churn vs. no-churn, appetency vs. no-appetency and upselling vs. no-upselling). The approach to parallelism is to first horizontally partition the data, learn weights on each data partition and then reduce by combining the weights. The data is preprocessed, cleaned and normalized prior to algorithm training. After each full pass over the data, the preprocessed, clean and scored data is staged using Datarush staging internal storage. DataRush staging datasets are high-performance binary alternative to temp database tables or text/flat files. These staging datasets carry their own metadata facilitating their consumption and transformation into DataRush record flows.

Figure 6 shows the application graph that includes reading the KDD Cup original training and test datasets. The customer records are read, parsed and extracted using a parallel delimited text reader. This application graph was generated using a 2-core affinity value. The *rushPreproc* operator performs the tasks described in the previous section of data preparation section. Then the numerical and discretized categorical variables are staged onto the “.preprocess” folders (training and testset). The *rushFeatSel* operator can perform feature selection based on Pearson Correlation coefficients. This step reads from the previously staged datasets and can (optional) do correlation to target and

cross-correlation to eliminate redundancy in features. We also evaluated mutual information among other feature selection methods before settling for Winnow as the feature selection approach to “winnow” out irrelevant features. The selected features are then staged onto the “.clean” folders (training and testset).

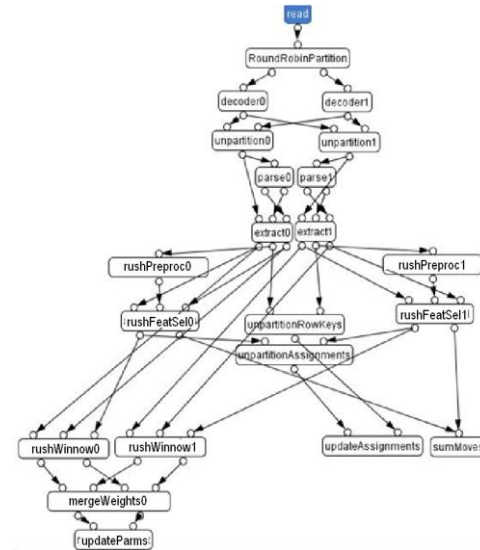


Figure 6 - Dataflow Winnow Application Graph.

The *rushWinnow* operator applies the modified balanced Winnow algorithm previously described in detail. The data partitioning for Winnow was done horizontally by groups of rows. The “.clean” training staged dataset is read, and a set of weights is calculated for each partition. The reduction of weights after Winnow is applied concurrently to each partition. Two reduction methods were evaluated towards best AUC results: averaging the Winnow weights from all partitions and weighted averaging the Winnow weights from each partition based on their performance on a validation set.

4.4 Dataflow Balanced Winnow Prediction

In this section we will compose a dataflow graph that implements Equation 1. The first step is to read from the staged preprocessed testset to be scored using the Winnow model. The staged datasets are already partitioned by subsets of rows. The *applyWinnow* operator is applied to each partition to generate the following results:

- wTx: is the total linear combination of Winnow weights and variables for one customer
- wTx_voted: is the total linear combination of the Winnow voted weights and variables for one customer
- yhat: is the Math.signum of wTx
- yhat_voted: is the Math.signum of wTx_voted

There is no reduction step necessary after the *applyWinnow*. The results are written to a “.resu” flat file according to KDD Cup submission requirements.

5. EXPERIMENTAL RESULTS

In this section we present our experimental results in support of the quality of predictions and runtime performance of our dataflow solution.

5.1 Datasets and Algorithms

Balanced Winnow algorithm was implemented using Pervasive DataRush Java Library version 4.0.1.21 [4] and JVM Sun64.jdk1.6.0_07. The IDE used to implement the Pervasive DataRush™ Winnow application was Eclipse 3.4.

Hardware specifications:

Processor: Intel Xeon™ CPU L5310 1.60 GHz (2 processors, quad cores each)

Memory: 16 GB System type: 64-bit Operating System.

The dataflow Winnow algorithm was trained using the KDD Cup 2009 Orange Labs CRM Large Dataset. The data includes 50,000 samples or customer records with 15,000 variables. The target values or classes (.labels files) have one example per line in the same order as the corresponding data files. The row id on both data and labels file is implicit.

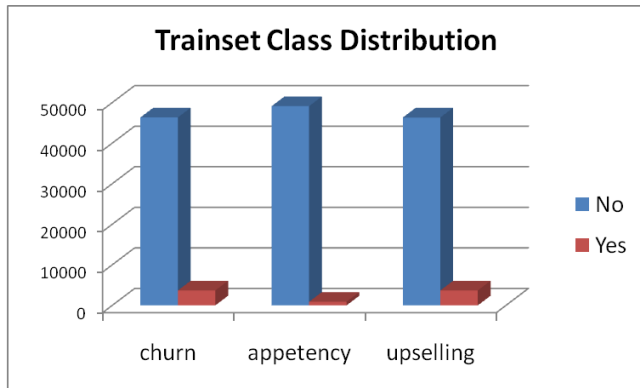


Figure 7 - Trainset Class Distribution.

Figure 7 shows one of the challenges with this dataset. The class distributions for all labels are very unbalanced. There are less than 2% of positive appetency training samples. There are less than 8% of positive churn training samples. And there are less than 8% of positive upselling training samples. The cost of misclassifying a minority class here is high since we would lose revenue or the customer altogether. There are various approaches to ameliorate this problem including sampling.

5.2 Results and Discussion

Scaleup

Relative Scaleup is defined as the time taken on a single processor by the problem divided by the time taken on p processors by p*problem. Figure 8, shows the average of total application runtimes during training Winnow algorithm using the KDD Cup large training dataset over 1 through 8 cores. The error-bars correspond to the standard deviation over five runs for each of 1, 2, 4 and 8 cores on the same hardware. The ideal curve is shown based on linear scaling across cores based on one-core runtime. Total time shown is in milliseconds for training with the KDD Cup large dataset. The target for this experiment was “upselling”.

The learning rate eta was set to 12 and the margin value was 0.0005.

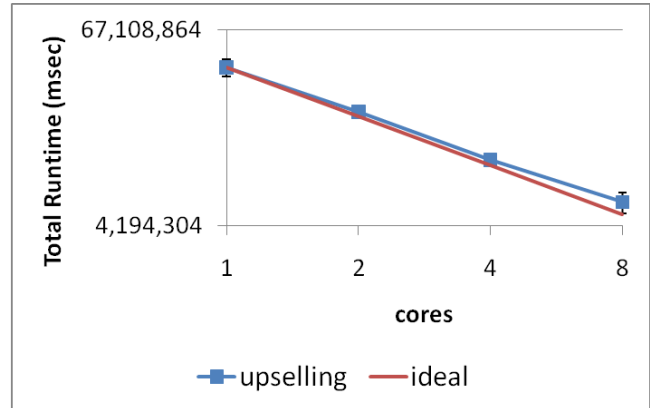


Figure 8 - Variation of Total Time with number of Processors under KDD Cup training dataset and “upselling” labels.

Figure 8, clearly shows how well the dataflow Winnow application on the KDD Cup training dataset using “upselling” labels as target scales linearly across 1 through 8 cores.

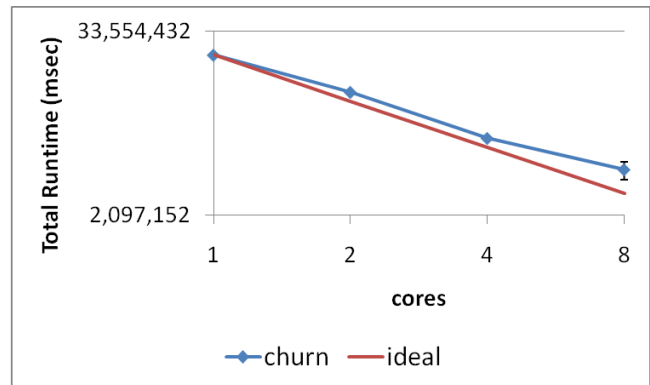


Figure 9 - Variation of Total Time with number of Processors using KDD Cup training dataset and “churn” labels.

Figure 9 demonstrates linear scaling across 1,2 and 4 cores using KDD Cup training dataset and “churn” labels. At 8 cores, we observed nearly-linear scaling. Runtime parameters for the Winnow algorithm included a learning rate of 6 and a margin of 0.004. The margin is high relative to the one used for appetency and upselling experiments. When the margin is high, we only consider misclassified points outside the margin and therefore finding a solution is easier. This translates into faster runtimes for churn experiments.

Table 1 - Runtime parameters, auc and total runtime.

target	eta	margin	AUC	runtime (min)
churn	6	0.004	0.61	51.91
appetency	12	0.0005	0.72	82.09
upselling	12	0.0005	0.83	97.90

Table 1 lists the runtime parameters for experiments with the best results in both runtime and AUC scores. The total end-to-end runtimes are in minutes. In the case of appetency and upselling, we chose a smaller margin and a larger learning rate.

Convergence runtime was much slower but the accuracy was higher. Upselling was easier to predict for all contestants as well as for our Winnow classifier.

Another set of experiments not included in the scalability results were performed using an affinity of 16 cores. The purpose of these experiments was to profile the application graphs to determine the runtime breakdown across all data mining tasks.

Table 2 - Runtime breakdown by Data Mining Task.

	Input (Gb)	Output (Gb)	Minutes
Data Preparation	3	12	6
Clean and Normalize	12	12	6
Build Model	18	1Kb	6
Apply Model/Score	6	245Kb	3
Total	39	24	21

Table 2 shows the results of profiling the end-to-end dataflow Winnow implementation. The table includes the data volumes on input and output ports. The total runtime of 21 minutes corresponds to experiments run on a 16-core, 16 Gb RAM cpu. The two data preprocessing steps necessary to prepare the data for analysis take one pass through the data each at about equal time slices of 6 minutes each. Build model also takes one pass per iteration of the Winnow algorithm. The fastest task is evaluating the model on new unseen samples. For the purposes of iterating through the modeling phase, we evaluated using misclassified counts and true positive counts. During the application of the model on the testset we generated the predictions (wTx , wTx_voted , $signum(wTx)$ and $signum(wTx_voted)$) and used the wTx values for each sample as probabilities. We then varied a threshold value for wTx to separate positive and negative predicted class values. The resulting counts of true positives, false negatives, true negatives and false positives led to Se and Sp pairs and the ROC curve.

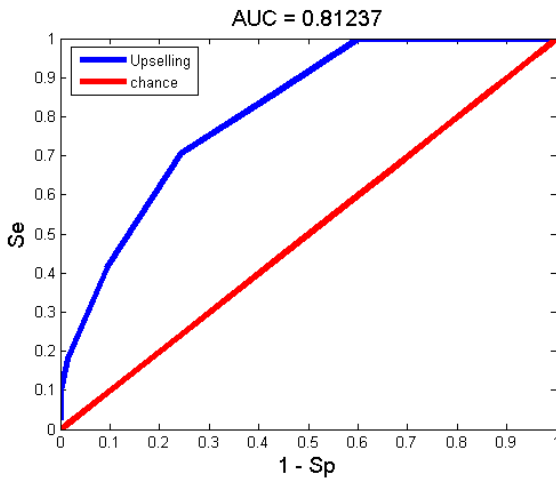


Figure 10 - ROC Curve for Upselling Predictions.

Figure 10 shows the ROC curve for upselling predictions on the training dataset. The red line depicts chance. The area under the

curve was calculated using an approximation of the definite integral known as the trapezoidal rule. The testset labels were not released with the KDD Cup data.

6. CONCLUSION and FUTURE WORK

This paper presents a novel dataflow implementation of a parallel Winnow algorithm, demonstrating runtime scalability with the number of cores. Our experiments apply this code to the KDD Cup 2009 Orange Lab Large training dataset, a domain with enough data and dimensions to make such scalability highly desirable. Although we considered only a few of many strategies for parallelizing the target algorithm, we do so using a framework that emphasizes development-time productivity.

Balanced Winnow proved very effective in winnowing out irrelevant features and homing in on the relevant ones. This makes it a great candidate for online learning where new samples can be easily used to incrementally update the model. And new features/variables are included or excluded via adjusted weights based on their relevancy. The modified balanced Winnow algorithm using wTx outperformed the wTx voted scheme. The learning rate combined with the margin provided flexibility when tuning for tight yet linearly separable classes. The modified balanced winnow provides a simple, efficient and naturally suited scheme for online customer behavior prediction.

Future extensions to our dataflow solution would be to compare to two popular online learners like linear SVM and Naïve Bayes. And also to investigate non-linear classifiers for the case of churn where AUC score was low.

On data parallelism, Pervasive DataRush allocates one thread per node in a dataflow graph and the cost of synchronization across dataflow queues is ameliorated via batching. This is not generally well suited for fine-grained parallelism. In our Winnow implementation, we do not fully exploit the parallelism inherent in cross-validation during modeling and tuning for best AUC score. We also considered a topology in which Winnow was parallelized vertically rather than horizontally. Since horizontally partitioning the dataset of 50000 samples results in partitions where the number of rows is similar to the number of columns, we were concerned with over fitting within each horizontal partition. The modified Winnow use of a margin ameliorated this problem. Further, these represent only a few among a large number of possible topologies. The highest performance topology can only be determined through experimentation.

The single most valuable contribution emerging from our dataflow framework for parallelism is the power for experimentation resulting from the amazing speedup in processing time for any parallelizable algorithm. Despite the temporary reprieve that 64 bit memory addressing has brought to large scale Data Mining, the in-memory data staging for predictive analytics and modeling has hit its wall. Pervasive DataRush dataflow architecture provides a Java framework to develop highly scalable and massively parallel data-intensive applications while providing a high degree of stability and usability.

7. ACKNOWLEDGMENTS

This research was funded by Pervasive Software of Austin, Texas. We want to acknowledge Matt Walker and Sreangsu Acharyya (Alex) for their contributions to this research.

8. REFERENCES

- [1] F. Reichheld, "The Ultimate Question", © 2006-2009, Bain & Company, Harvard Business School Publishing Corporation, 2006.
- [2] <http://www.kddcup-orange.com/> KDD Cup 2009.
- [3] G. Kahn, The semantics of a simple language for parallel programming, Proc. of the IFIP Congress 74, North-Holland Publishing Co., pp. 471-475, 1974.
- [4] Pervasive Software, Inc., Austin, TX, USA
<http://www.pervasivedatarush.com/downloads>
- [5] N. Littlestone, Learning quickly when irrelevant attributes abound: a new linear threshold algorithm, Machine Learning 2, pp 285-318, 1988.
- [6] N, Littlestone, Mistake bounds and logarithmic linear algorithms, Ph.D. Thesis, Report USCSC-CRL-89-11, University of California, Santa Cruz, CA, 1989
- [7] E. Lee, T. Parks, Dataflow Networks, In Proceedings of the IEEE, vol. 83, no. 5, pp. 773-80, May, 1995.
- [8] V. Carvalho, W. Cohen, Single-pass online learning: Performance, Voting Schemes and Online Feature Selection. International Conference on Knowledge Discovery and Data Mining. 12th ACM SIGKDD, 548-553, 2006.
- [9] T. Zhang, Regularized Winnow methods, In Advances in Neural Information Processing Systems 13, 2001.
- [10] I. Dagan, Y. Karov, D. Roth, Mistake-driven learning in text categorization. In EMNLP, 55-63, Aug 1997.
- [11] CRISP-DM, <http://www.crisp-dm.org/Process/index.htm>.
- [12] R.O. Duda, P.E. Hart, D.G. Stork, 2001 Pattern Classification, 526-528, Second Edition, John Wiley & Sons, Inc.
- [13] K. Irwin, M. Walker, October 2008 "Four Paths to Java Parallelism", Java Developer's Journal, Vol. 13 Issue 9.