Predicting Customer Behavior in Telecommunications

Lian Yan, *Aureon Biosciences*

Richard H. Wolniewicz, *Flatirons Computing*

Robert Dodier, *Independent Researcher*

Predicting customer behavior helps service providers build customer loyalty and maximize profitability. For the success of a project, data preparation is often a critical part of the predictive algorithm.

A wireless service subscriber calls a customer service representative to complain about dropped calls. During the conversation with the customer, the CSR views a display that shows this customer's probability of churn—switching from this service provider to another—as well as the most probable reasons to churn and the best strategy to retain this customer. The CSR then quickly responds to the subscriber according to the system's recommendation. This is an intelligent customer-care system designed to predict customer behavior.

Analysts have estimated that churn costs wireless service providers in North America and Europe more than US$4 billion each year.[1] To build customer loyalty and maximize profitability, intelligent techniques for predicting customer behaviors such as churn become necessary in customer relationship management systems,[2] especially in the highly competitive telecommunications industries: wireless, cable, and satellite.

Some organizations have used advanced machine learning techniques to predict customer churn in the wireless industry and thus to substantially decrease the churn rate.[3] As shown in Figure 1, predicting customer churn is a component in the decision framework for retaining customers and maximizing profitability. Customer data can also help predict the reasons customers switch companies, the revenues generated in a specified period, revenue collection risk, and the response to a marketing offer. Companies can use these probability and revenue estimates in a decision-theoretic framework to determine a churn intervention strategy and a profitability optimization strategy.

For instance, on the basis of estimates for each subscriber of the probability of accepting a marketing offer, the net revenue change after accepting the offer, the revenue collection risk, and how long the subscriber will keep the service, the service provider can estimate the expected lifetime value of this marketing offer for each subscriber. Then the provider can approach each subscriber with the marketing offer of the largest lifetime value estimate to maximize profitability.

Defining the prediction target

The first and most important step during the modeling process is to define the prediction target. Accurately defining target values can be quite involved. The modeler must understand the service provider's business rules and the application scenario because each company has its own understanding of and specific rule for a certain event. For example, a service provider might require only a call from the subscriber to disconnect services, while another might require a subscriber to send a letter to terminate the contract. In general, we use the following principles for defining correct target values.

Published by the IEEE Computer Society

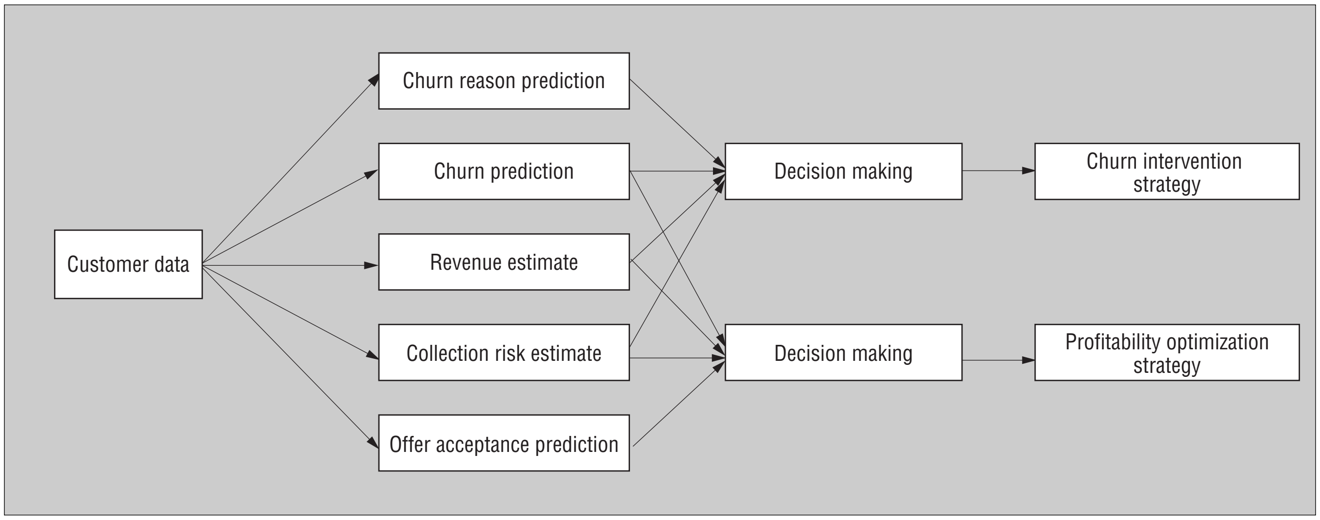

Figure 1. The framework for customer retention and profitability maximization.

First, the target value must be determined by the earliest indicator of the event of interest. For a cable service company, when a subscriber calls to request service disconnection, the CSR immediately enters an open work order for the disconnection, but the completed status of the work order won't appear in the database for several days. In this situation, you can't use the appearance of a *completed* disconnect work order to determine the churn's target value, because it's not the earliest indicator of churn. Moreover, the preceding appearance of the open work order might be a false predictor. Using the earliest indicator, the entry of an open disconnect work order in this example, to determine the target value is also required by the next principle.

Second, you must consider the specific business rule and application scenario so that the prediction can be useful and actionable. In the last example, you must make the prediction before entry of the open disconnect work order so that the CSR can proactively address the subscriber's concern and try to retain the subscriber. Also, in the case of requiring a letter to terminate services, the service provider must make the prediction several days in advance before the letter arrives so that the CSR can act before the subscriber decides to send out the termination letter. Thus, you need a several-day gap between the ending date of the input variable window and the beginning date of the prediction target window. During the model's use phase, the ending date of the input variable window is also the prediction date.

Third, the information for determining the target values must be available before the ending date of the prediction target window. For some service providers, the open work order might not appear in the database for several days. In this case, if we use the appearance of the open work order to determine the target value, it will mislabel some positive samples (churn) as negative samples (nonchurn) for the training data set, and will adversely affect the timely evaluation of prediction accuracy as well.

Fourth, we consider the duration of the target window. When we can update the model and prediction frequently—for example, weekly—the target window's size is important only for training the model. Intuitively, when the target window is large, there will be more positive samples in the training set, which benefits the training especially when the positive sample rate is small. However, at the same time, this results in a model trained over stale, out-of-date data.

We've found that a training set with a four-week target window yields greater prediction accuracy, as measured by the area under the *receiver operating characteristic* (ROC) curve,[4] compared to one-week or eight-week target windows. In general, you must determine the optimal target window length by searching for the greatest classification accuracy over a range of candidate window lengths.

Extracting customer data

Figure 2 shows a simplified version of a typical wireless-customer database. A data source such as this forms the basis for predictions. Some data tables, such as the Account and Demographic tables, provide a single raw data field per column. Other tables, such as Services, are variable in length and require summarization. Time series data might be regular in frequency (such as Bill) or intermittent (such as Call_Detail_Record and Customer_Service_Contact).

A common problem in predictive analytics is how to properly extract training data to train a model that will reliably forecast future events. Several problems can occur here, most of which result in a trained model that performs better on cross-validation and historical data sets than can be achieved in a production environment where future events are being predicted. We have found this typically results from one or more independent variables falsely appearing to be correlated in a particular way to the dependent variables, on the basis of the training data. For that reason, we refer to these false relationships as *false predictors*.

In our experience, we've seen three main sources of false predictors: errors in assembling training sets, failure to detect the first instance of the target event, and inclusion of post-event information in independent variables.

Errors in assembling training sets

More than once, we've seen basic errors in assembling training sets, where the training set is no longer statistically similar to the production environment. In one case, an international wireless carrier conducted a vendor comparison of analytic tools for churn prediction. The vendor requested a training set of 100,000 subscribers, with 30 percent churners and 70 percent nonchurners. After generating this training set, the carrier used the identical training set with other vendors to compare results through cross-validation.

The carrier's actual monthly churn rate was approximately 30,000, so the carrier constructed the training set using all the churners in a single month. To create the

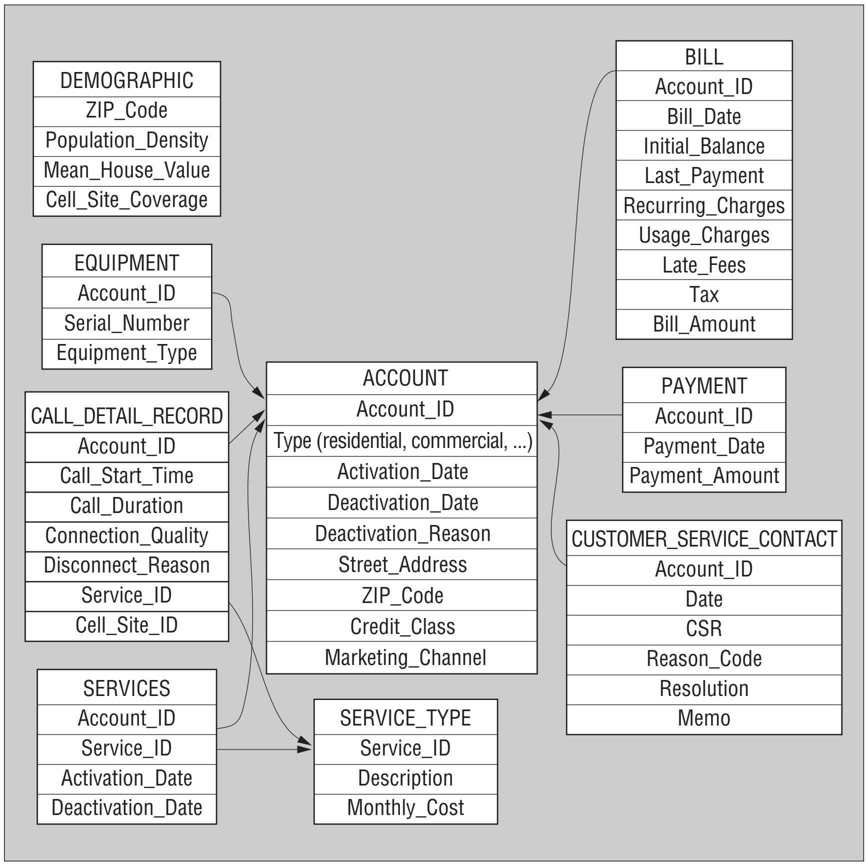

Figure 2. Simplified tables in a typical wireless-customer database.

nonchurner set, the carrier extracted the first 70,000 nonchurner accounts from their database, based on account number. Account numbers were principally assigned in order from lowest to highest, so in this training set no nonchurner's tenure was less than 10 months. All vendors' models quickly learned the dominant relationship—if short-tenure, then churner—and this result held up well under cross validation. The false predictor would appear only in production deployment.

Although this is a simple error, it has occurred often enough for us to conclude that good data extraction entails extracting complete subsets of data, not relying on customer staff untrained in statistics to perform statistical operations such as up-sampling, filtering, and random selection. We discussed this problem with the carrier after the trial was complete. In those discussions, the carrier informed us that, for whatever reason, most data-mining vendors failed to mention the statistical problems in the training set and simply presented their model's predictive accuracy results.

The first instance of the target event

As we discussed earlier, it's important to define dependent variables on the basis of the earliest instance of the target event being predicted. Where this isn't the case, earlier indicators of the target event might become false predictors. A simple example of this is scheduled disconnects. For a telecommunications provider, most subscriber disconnects (churn) occur on the same day the subscriber calls to cancel service. Occasionally, a subscriber will schedule a disconnect at some point in the future, resulting in a work order appearing in the operations database prior to the churn event. Assuming that pending work-order information appears among the independent variables—generally a good idea for telecommunications churn prediction models—then there is the potential for a false predictor to appear if the churn event itself is dated to the disconnect date rather than the work order entry date.

The resolution of this issue rests with the modeler evaluating the actual prediction event and all the ways in which that event might manifest in the available data. In the case of churn, the target event we wish to predict is the moment a subscriber decides to discontinue service. That information will never appear as such in any data warehouse; it's incumbent on the modeler to identify all evidence for the event.

Including post-event information

The final common source of false predictors is a result of querying data sources for independent variables for training sets, and receiving values that in some way reflect changes to the data after the target event has occurred. When constructing a training data set, the time at which the data set is constructed—call it t_{query}—must be at or later than the end of the dependent-variable window t_{dvend}. To construct the data set, we need the independent variables to reflect the state at some earlier time, t_{ivend}, prior to the dependent variable window:

$$t_{\text{query}} \geq t_{\text{dvend}} > t_{\text{ivend}} .$$

Therefore, we must query the t_{ivend} state of all independent variables at the later time t_{query}. False predictors can occur when the state of the independent variables reflects information that wasn't truly available at t_{ivend}.

To illustrate this relationship, consider a churn model trained on 18 October 2003 to identify churners in the following 28 days. The training set will use the known churners from the preceding 28 days—those since 20 September 2003. In this case, t_{query} and t_{dvend} are both 18 October 2003, while t_{ivend} is 20 September 2003. To avoid false predictors, we must ensure that all independent-variable queries return the state of the data warehouse as of 20 September rather than 18 October. For the prediction data set, t_{ivend} is set to 18 October; t_{dvend} isn't relevant to prediction data set queries. In production situations where periodic batch updates occur to the data warehouse queried for independent-variable data, it's often convenient to synchronize t_{ivend} and t_{dvend} to the batch update periods.

In the case of time series independent-variable data, you should use historical time series of identical duration for both training and prediction sets. Continuing the previous example, if you use six months of bills in the training set with t_{ivend} equal to 20 September 2003—that is, bills for March through

August—then six months of bills also should be used in the prediction set with t_{ivend} equal to 18 October 2003 (bills for April through September).

Retrieving prior states. In practice, there are three approaches to retrieving prior states of independent variables. First, you can query a data store (typically a data warehouse) that isn't explicitly designed for predictive analytics for independent variables, with past states of values inferred from information stored in the data store. This is the most common approach but is often problematic.

Second, you can store prior snapshots of independent variables as you generate them, and use them in future training sets. This approach is common and workable but is limited in the range of possible t_{ivend} states to those for which a snapshot was generated. In addition, changes to independent variables—additions or changes to transformations—will be problematic because you must allow changed snapshots to age before constructing a training set. This delay often has a negative impact on the system's ability to respond to changes in the predictive environment.

Finally, you can use a temporal database in which all source data is stored with temporal information, allowing past states to be reliably queried.[5] This is our approach, as it offers flexibility in changing the independent-variable data environment and snapshot timestamps, as well as assuring the reliability of past-time queries.

Common sources of errors. Because the first approach is the most common and is often unavoidable when primary data sources were designed for purposes other than predictive analytics, we discuss three of the most common sources of data extraction errors that result in false predictors. First, some variables reliably change when a target event occurs. A common example in the telecommunications churn problem is the count of equipment records—cell phones, set-top boxes, and so forth—associated with a subscriber account. In general, subscribers with more equipment items are less likely to churn than subscribers with a single equipment item, and it helps to include this information in a churn prediction model.

Unfortunately, source systems often store only current equipment information, and subscriber churn generally results in the reassignment of equipment to new subscribers. For example, cable TV set-top boxes are reassigned to new subscribers when returned. Therefore, using the count of equipment records in the source data is often unreliable and can result in models that learn the relationship

$$\textit{Equipment-record-count} = 0 \Rightarrow \textit{Churn-probability} = 100\%.$$

In the absence of a reliable means of querying equipment records as they existed prior to the churn event, this otherwise-useful independent variable must be discarded. In addition to this variable, there are other sources of false predictors. For some fields, the independent variable's state at t_{query} might vary systematically from the state at t_{ivend} in at least some instances. For example, you might adjust billing records for past months after a churn event. Also, you might update call detail records in batches at some time later than the events they record.

Our approach offers flexibility in changing the independent-variable data environment and snapshot timestamps, as well as assuring the reliability of past-time queries.

As a final observation, operational changes might result in variables becoming false predictors that previously were not problematic. For this reason, we recommend that production systems requiring periodic retraining use a temporal data store for generating training data to eliminate most instances of false predictors.

Preprocessing the raw data

From a general point of view, preprocessing is a mapping or re-expression of an input space into a modified space in which the boundaries between classes are easier to find. Although, theoretically speaking, a sufficiently flexible model—such as a neural network with hundreds of hidden units—should be able to compute a representation equivalent to any of these preprocessing steps, purpose-built preprocessing is still much more efficient. This remains an important consideration even for data sets that contain millions of records.

Related to the topic of data preprocessing, but not as a transformation of the input data, we also preprocess the target data. We typically change the binary target values 0 and 1 to −1 and 1 during training. We have empirically found that training neural networks with target values of −1 and 1 achieves greater classification accuracy. We speculate that the error surface for target values of −1 and 1 might avoid some local minima.

Representing categorical data

There are several purposes for preprocessing raw customer data. The first is to represent the categorical variable $C = \{c_1, c_2, \ldots, c_d\}$ or other nonnumerical data in a numerical form. It's easy to represent a categorical variable with a low cardinality d, such as sales method, as a vector of d binary variables. However, when d is large, this binary representation dramatically augments the input feature space.

The five-digit US postal zip code is a categorical variable with a very large cardinality and cannot be represented in a binary vector. We have used the class probability $P(T|c_i)$, calculated from the training set, to represent c_i, $i = 1, \ldots, d$, where T is the target. For a binary target $T = \{0, 1\}$, c_i is replaced by $P(T = 1|c_i)$. This representation not only lets us incorporate information from the categorical variables but can also be applied to continuous variables after discretization to improve model performance. This approach is more effective when the number of samples per category is relatively large. If each category contains only a few samples—for example, as with the nine-digit US zip code—the target will be encoded with little loss in the input variable, and the classification accuracy on out-of-sample data will be substantially less than on the training data.

A related transformation is the discretization of continuous data to construct a naive Bayes classifier. We selected a bin width for discretization according to a heuristic, proposed in the kernel density literature, that makes the bin width narrower as the number of data increases. As density estimation is a difficult problem, in general, we sidestepped the density estimation by reformulating the naive Bayes output in terms of the per-input class probability, $P(T|x_i)$, instead of the per-input class-conditional density, $p(x_i|T)$. This approach also makes it possible to use neural-network outputs and other kinds of classifier outputs as inputs in the naive Bayes model. All the same, we have found that neural-network ensembles are more accurate than naive Bayes models.

Incorporating domain knowledge

The second goal of preprocessing the raw data is to incorporate domain knowledge into the model. For many machine-learning models—such as multilayer perception neural networks—you can't effectively incorporate domain knowledge into the model during training. A more effective way to incorporate domain knowledge into the model is through data representation based on domain knowledge. For example, there are many possible service changes in the cable industry, and how to represent all of them in the inputs is challenging. We tried to use clustering techniques to preprocess the service change data, but representing all possible service changes in clusters was quite difficult.

Fortunately, not all the changes are informative to the prediction. Still, it's equally difficult for the model itself to find the most informative service changes. In this case, domain experts can provide useful insights. For example, a recent upgrade to particular cable packages, such as HBO and Showtime, might be negatively related to the adoption of Internet broadband service. We can then add a binary variable indicating this specific service change into the input variables for predicting acceptance of the broadband service. We also found that a weighted average of monthly data (specifically billing data) is a useful input transformation. We significantly improved model accuracy by incorporating domain knowledge via such data representations.[6]

We can also incorporate domain knowledge through the overall organization of the model. We've found it useful to embed prediction models for different tasks in a larger model, called a *decision network* or *influence diagram*, which brings together notions of utility (preference) and action as well as beliefs about variables such as subscription status.[7] Making predictions is just a step toward a larger goal, such as selecting which intervention (attempt to save, attempt to upsell, attempt to cross-sell) is best for a particular subscriber.

We can model the choice of intervention as an action or decision on the part of the service provider, and the value or preferability of a choice as a utility. The structure of the decision network depends strongly on the service provider's business rules. Thus, we can build specific information into the decision network structure to reflect domain knowledge relevant to a particular service provider.

While a decision network's structure is application-specific, the probability and utility computations involved are general. Our software deployment consists of a decision network containing the application-specific information and the decision network processor that is the same from one application to the next. The knowledge engineer can focus on the decision network as the application-specific knowledge representation; new applications require new decision networks, but not new software.

For many machine-learning models—such as multilayer perception neural networks—you can't effectively incorporate domain knowledge into the model during training.

Missing and inconsistent data

In our applications, data cleaning means dealing with missing values and reducing noise. We deal with missing values by modifying the predictive model input representation, although a more sophisticated approach that takes advantage of the decision-network structure is also possible. For categorical features, we expand the cardinality of the feature by one, making "missing" a distinct value of the feature. For continuous features, we expand the dimensionality of the feature space by one, adding a binary feature that indicates whether the feature value is missing. At the same time, the mean or median of the feature imputes the missing value. Both these approaches essentially assume "informative missingness." That is, whether the data is missing depends on the target variable. These methods also have the correct behavior for random missing data.

We also can handle the missing-data problem in a more principled way by building a missing-data model into the decision network. The most correct approach, from a theoretical point of view, is to average predictions over the probability distribution of each missing variable, given known values of some or all of the other variables. For example, if a discrete value is missing, we can average a prediction output by substituting each possible value into the prediction equation and weighting each such output by the probability the variable takes on that value. The decision network machinery can automatically organize such computations so that the knowledge engineer need only think about how to state the relationships among the variables involved.

The decision network structure can also encode models for coping with inconsistent data, although we used simpler means of handling inconsistent data. Variables with many inconsistent values will be less useful for prediction and therefore will be omitted during the input selection and model-building phase. As with missing values, we can exploit the decision network structure to handle inconsistent data by computing probability distributions that put less weight on values that are improbable, given known values for other variables.

To some extent, the machine-learning models we use are naturally robust in the presence of inconsistent data. In these cases, the model will learn to output an average value, which is the best answer in the absence of further information. We also found that excluding multimembership samples for training a churn reason model can achieve better accuracy. Multimembership samples belong to more than one class and implicitly consist of inconsistent data.

Feature selection

Input-feature selection achieves both data cleaning and data reduction by selecting important features and omitting redundant, noisy, or less informative ones. We do the initial feature selection when we decide to extract certain raw data from the database after understanding the company's business. At this stage, we can typically have 200 to 300 input features.

The ROC curve is a useful measure for classifier performance in many real-world applications. We've used the area under the ROC curve (AUC) to conduct feature selection in most of our applications. We can measure the AUC by the Wilcoxon-Mann-Whitney statistic in the form

$$U = \frac{\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} I(x_i, y_j)}{mn}$$

where

$$I(x_i, y_j) = \begin{cases} 1 & x_i > y_j \\ 0 & \text{otherwise} \end{cases}$$

is based on pairwise comparisons between a sample x_i, $i = 0, \ldots, m-1$, of random variable X and the sample y_j, $j = 0, \ldots, n-1$, of random variable Y.

If we identify $\{x_0, x_1, \ldots, x_{m-1}\}$ as the classifier outputs for m positive samples, and $\{y_0, y_1, \ldots, y_{n-1}\}$ as the classifier outputs for n negative samples, we obtain the AUC for our classifier.[8] For the purpose of feature selection, we identify $\{x_0, x_1, ..., x_{m-1}\}$ as a feature's value for m positive samples, and $\{y_0, y_1, \ldots, y_{n-1}\}$ as the feature's value for n negative samples. We then obtain the AUC for a linear classifier using only this single feature as the input. We use the larger value between this AUC and an alternative value obtained by replacing $I(x_i, y_j)$ with $I(-x_i, -y_j)$ for ranking this feature.

For categorical features, we compute the AUC over the class probabilities. We could compute the AUC for continuous features on the basis of the class probabilities too. This value is generally larger than the one directly based on the feature value because this represents the AUC for a nonlinear classifier with this single feature as the input. However, this process typically doesn't change the selected features significantly. We can further simplify U and quickly calculate it if we manage $\{x_0, x_1, \ldots, x_{m-1}\}$ and $\{y_0, y_1, \ldots, y_{n-1}\}$ in ascending order, yielding

$$U = \frac{1}{mn}\left(\sum_{i=0}^{m-1} r_i - \frac{m(m-1)}{2} \right)$$

where $r_i \in \{0, 1, \ldots, m+n-1\}$, $i = 0, \ldots, m-1$, is the rank of x_i. We choose the features with a value of U above an empirically determined threshold. This method, which considers one input feature at a time, doesn't consider the correlation between the input features; features that are strongly correlated—such as payment amounts in successive months—might all appear to be relevant if any one of them is relevant. The presence of correlated input features might require additional work to identify and remove redundant features. On the whole, we found that using the Wilcoxon-Mann-Whitney statistic for feature selection is very effective.

Another approach to input feature selection is to use a genetic algorithm[9] or other form of heuristic search to evaluate sets of inputs and find the best sets. This approach does take input correlation into account, although it's somewhat time-consuming if the number of records or input features is large. We also considered evaluating input feature relevance by means of mutual information. However, mutual information doesn't directly measure the same notion of relevance as the Wilcoxon statistic; it's possible to construct examples of variables for which the mutual information is relatively low but the Wilcoxon statistic is relatively high.

The model needs frequent updating to overcome data staleness and inconsistency. We typically retrain the model weekly over the newly extracted data.

The discrepancy between the Wilcoxon statistic and mutual information becomes especially pronounced for classification problems in which the positive sample rate is relatively low. We've worked with several such problems. Because the Wilcoxon statistic is easy to compute and directly related to the area under the ROC curve, we've mostly worked with the Wilcoxon statistic to assess the relevance of input features.

Taking data characteristics into account

Prediction data have several identifiable characteristics. We can improve model accuracy by building our data-processing algorithms around these known characteristics. For starters, subscriber data is typically nonstationary because of changing behaviors over time, the inconsistency of data available at different times, and changes in the environment. These factors result in several consequences. First, the commonly used evaluation based on cross validation is a somewhat less accurate estimate of out-of-sample accuracy because the model will be used over a forward time-shifted data set, which can have a different distribution than the training data, leading to decreased accuracy.[10]

Compensating for data nonstationarity

The second consequence is that the model needs frequent updating to overcome data staleness and inconsistency. We typically retrain the model weekly over the newly extracted data. Still, the inconsistency of data available at different times—for example, updating billing data only once at a certain time of the month—brings cyclical model performance. We've proposed several algorithms to overcome the data nonstationarity problem.[10] The most effective and simple way is to reuse the older models to a certain point in the past and combine the outputs of the models by unweighted average. Averaging models up to six weeks old increases prediction accuracy as measured by AUC.

To some extent, the deployment environment imposes this nonstationarity on the data. In particular, we expect the prediction model to produce predictions about acceptance or rejection of offers made by the service provider to its subscribers. The service provider marketing department introduces and retires these offers on a frequent but irregular schedule. We have two mechanisms for handling this issue. One is to set up, within the decision network, a component model or submodel for each currently active offer. The other is to use previously collected observations of acceptance and rejection to train models for new offers.

Compensating for data imbalance

An important characteristic of most data sets in our applications is the highly imbalanced class distribution. For example, the positive sample rate for accepting some marketing offers is as low as 0.5 percent. Changing the prior class distribution by up-sampling positive samples or down-sampling negative samples is the most common way to compensate for the imbalanced distribution. Research on this topic concludes that there is no single class prior that is the optimum for all data sets during training, and that the optimal class prior for training must be empirically determined for each data set.[11] Instead of trying to find the best class prior for training a model, we propose a new ensemble algorithm specifically for imbalanced data sets.

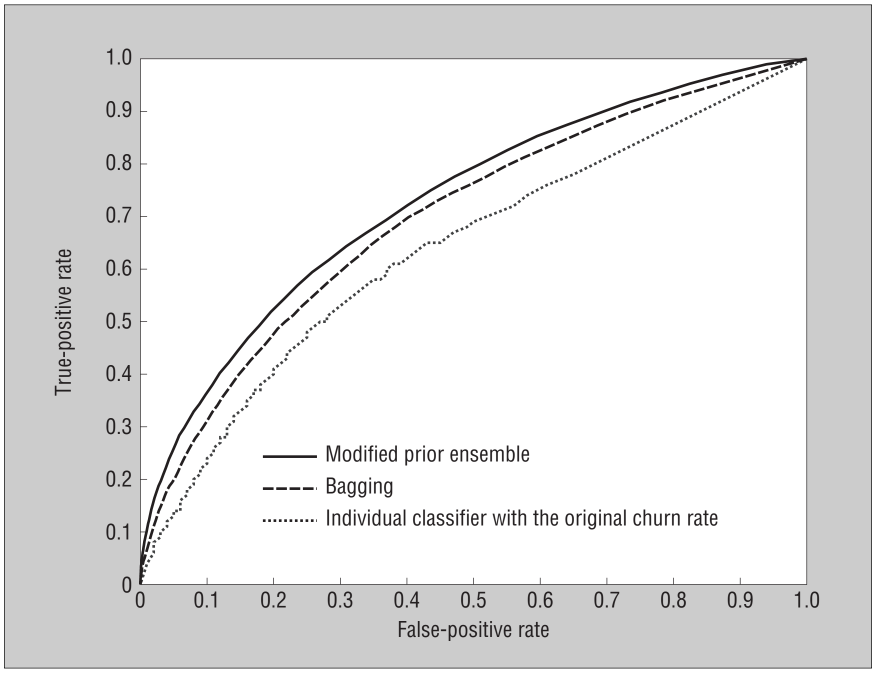

Figure 3. The new ensemble algorithm for imbalanced data sets outperforms conventional bagging and the model trained over the original prior.

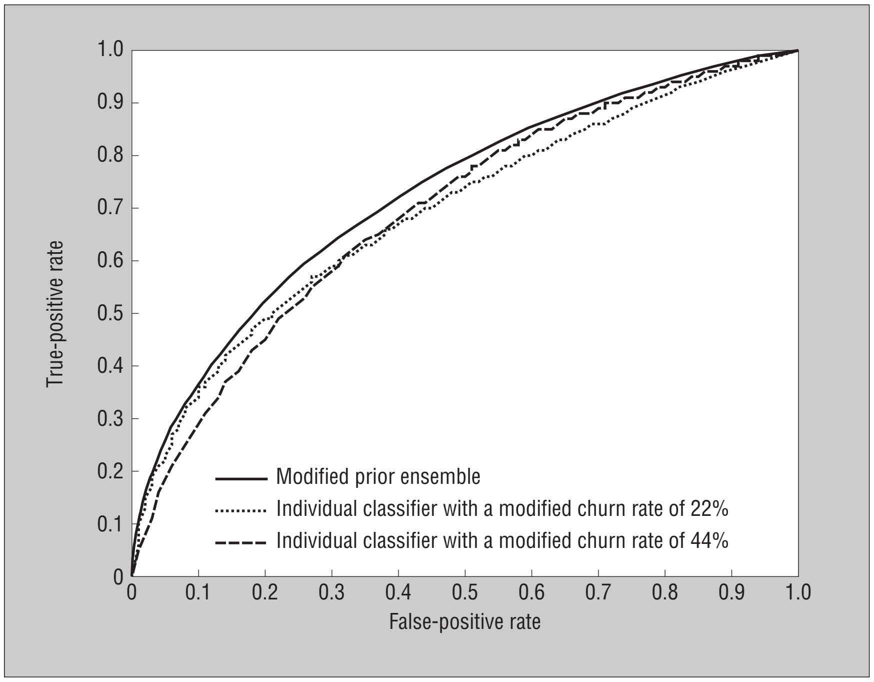

Figure 4. The receiver operating characteristic (ROC) curve of the combined outputs is higher than both the individual curves in all parts of the ROC space.

Given a training set of size *N* and a positive sample rate *p*, we train an ensemble of *M* classifiers, such as neural networks. For each classifier *i*, *i* = 1, …, *M*, we construct a training set of size *N* and with a positive sample rate q_i by random sampling (with replacement) separately from the positive and negative samples. Here,

$$q_i = \frac{(q_{\max} - q_{\min})(i-1)}{M-1} + q_{\min} ,$$

and $q_{\max}$ and $q_{\min}$ are predetermined maximum and minimum priors, respectively. Typically, we choose $q_{\max} = 0.5$, $q_{\min} = 0.02$, and $M = 25$. Because each classifier in the ensemble is trained over a different class prior, you must adjust the outputs to the same original prior *p* before the combination of the individual outputs by unweighted average. The distribution in the input space within each class is unchanged by resampling, so we can derive the following formula to adjust the output s_i from the *i*th classifier:

$$f(s_i) = \frac{s_i \frac{p}{q_i}}{s_i \frac{p}{q_i} + (1 - s_i)\frac{1-p}{1-q_i}} .$$

We compare the ROC curves over a forward time-shifted churn prediction data set for a wireless service provider in Figure 3. The new algorithm substantially outperforms the model trained over the original prior and as well as the conventional bagging ensemble.[12] Figure 4 shows the ROC curves for two individual classifiers in the ensemble together with the ROC curve of the combined outputs. The two individual classifiers have greater accuracy at different classifier thresholds—different points on the ROC curve. This can shed light on why the ROC curve of the combined outputs is higher, or better, than the individual curves in all parts.

Compensating for data sparsity

Unlike the large data sets used for predicting churn, we have much less data on predicting acceptance of a marketing offer. At the initial launch of a new marketing offer, little or no data exists about its acceptance. In this case, we can modify prediction for targets with few training samples by combining results of models trained for other offers. The combination is based on the distance between offers, which is determined by business-specific data such as prices and costs of the offers. In addition to the distance, the training samples' size also affects the combining weights. The model trained by more training samples will contribute to a larger combining weight.

However, it might be possible to identify a certain relationship between a marketing offer and the services involved. For example, a cable service company tries to launch a marketing campaign to promote its broadband Internet service. To compensate for the data sparsity, we can now try to identify the subscribers who spontaneously adopted broadband Internet service in the past and use these subscribers as positive samples for the training. Here, we implicitly assume that the spontaneous adopters have similar characteristics to those who will accept the marketing offer associated with the same service.

This assumption isn't necessarily correct, but the empirical results strongly support using the spontaneous adoption data for offer acceptance modeling. Figure 5 shows the substantially improved ROC curve for predicting acceptance of a new service for a cable service provider. The results emerge over a time-shifted data set that consists of samples of acceptance and rejection of the offer. To get well-calibrated probability estimates, you must rescale the results by the formula for prior adjustment because the incorporation of spontaneous adoption data changes the class priors in the training set.

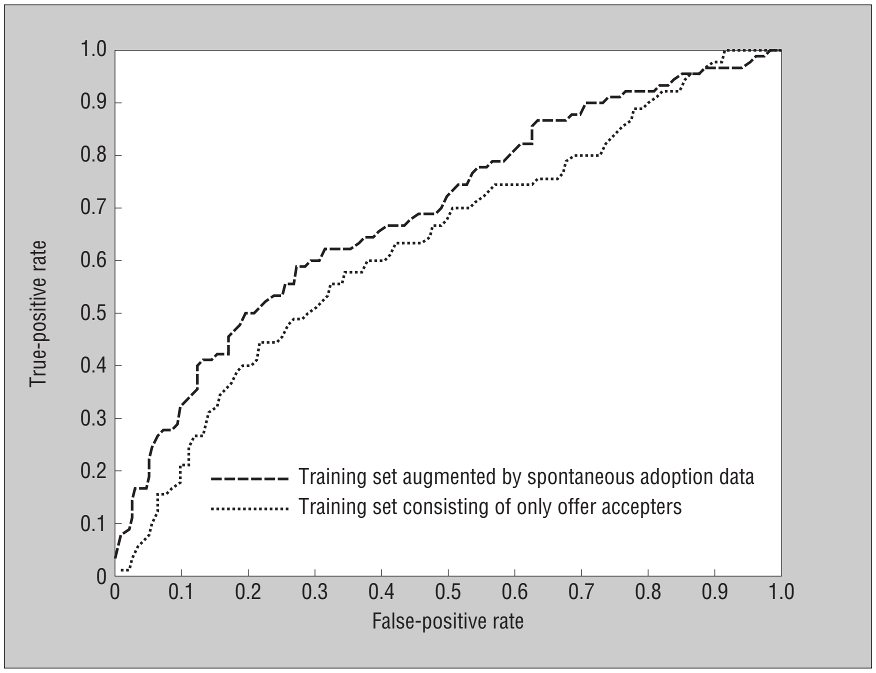

Figure 5. The model trained over the training set augmented by spontaneous adoption data substantially improves the acceptance prediction accuracy.

Accounting for competing risks

The kind of churn of greatest interest to a service provider is typically "voluntary" churn—that is, churn initiated by subscribers for their own reasons. The service provider also initiates churn, typically due to subscribers' failure to pay for their subscriptions. Depending on the provider's business rules, other outcomes can occur as well. From a survival-analysis viewpoint,[13] these outcomes are all *competing risks*: any of these outcomes might terminate a subscriber's tenure. And for any given outcome, the others cannot occur later.

Thus, the training data set that we can construct for, say, voluntary churn must have the characteristic that a record shows voluntary churn only if it doesn't show any other outcome. So, the predictions computed from a model trained on such a data set are conditional on the absence of outcomes other than voluntary churn. However, predicting voluntary churn and not any other outcome—that is, a *joint* prediction as opposed to a *conditional* prediction—is more useful.

We can show that the joint probability for voluntary and not nonvoluntary churn satisfies

$$p(\text{vc},\neg\text{nvc}|x)=\frac{p(\text{vc}|\neg\text{nvc},x)(1-p(\text{nvc}|\neg\text{vc},x))}{1-p(\text{vc}|\neg\text{nvc},x)p(\text{nvc}|\neg\text{vc},x)},$$

denoting voluntary and nonvoluntary churn by "vc" and "nvc," and denoting the input fields collectively by x. We calculate the quantities $p(\text{vc}|x)$ and $p(\text{nvc}|x)$ by models trained on the competing-risks data. Essentially, $p(\text{vc},\neg\text{nvc}|x)$ amounts to a corrected prediction for voluntary churn to be used in place of $p(\text{vc}|x)$.

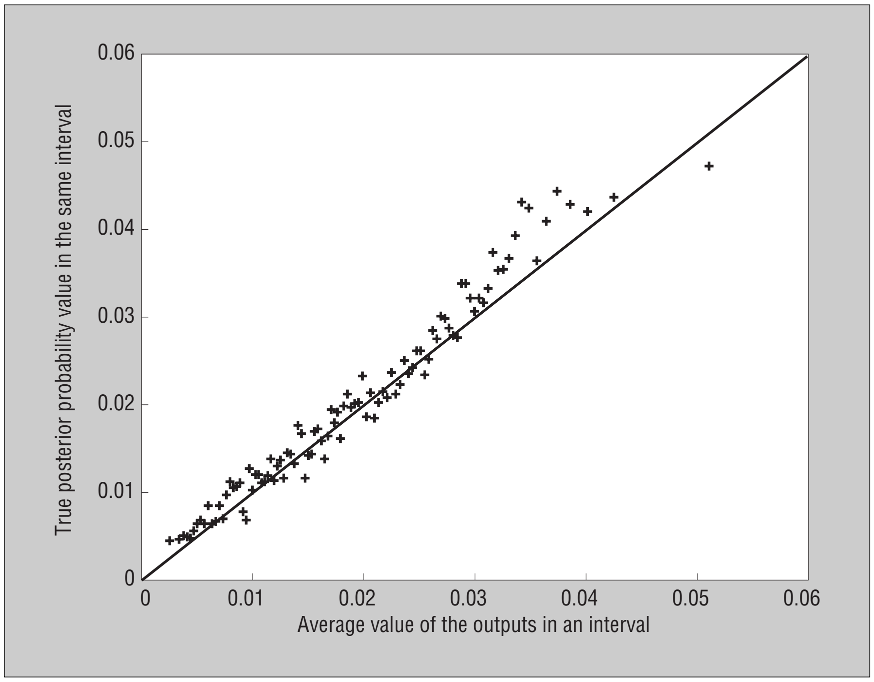

Figure 6. A calibration curve with the *x* axis as the average value of the outputs in a certain interval and the *y* axis as the true posterior probability value in the same interval. Perfect calibration should have all the points lying on the diagonal line.

Improving probability calibration

We use the term *calibration* here to refer to the agreement of the average score within a set of cases to the rate of positive examples in that set. If the average score matches the rate of positive examples, we call the scores *well calibrated*. In our decision-theoretic framework, the outputs from the churn or offer acceptance prediction models must be well-calibrated probability estimates, so that expected utility values can be accurately calculated from them.

We've found that ensembles of neural networks yield better calibration than any single neural network. Figure 6 shows a calibration curve for churn predictions from an ensemble of neural networks. In the figure, scores below about 0.03 are well calibrated, while higher scores are not too far off. To improve calibration in the higher scores, we applied a linear transformation over them so that the transformed scores have better agreement with observed churn rate.

Empirically, we found that using ranks rather than the original scores as inputs to the linear function is more robust for improving calibration. Improving calibration in the relatively high scores is especially important because the cases with high churn scores are economically more interesting.

Predicting customer behavior can help service providers build customer loyalty proactively and maximize profitability. Customer data used for prediction can be massive and yet sparse, and is dynamic, noisy, and sometimes faulty. For project success, data preparation is often a critical part of the predictive algorithm.

Several large-scale customer relationship management applications for telecommunications service providers have used the techniques we describe here, but many areas of data preparation still require substantial future work. One such topic is related to survival analysis for the lifetime value estimate. Most companies keep currently active customer data but store very little data about inactive customers. Thus, the survival data is strongly right-censored. One approach we are investigating is to use the steady-state assumption to derive a penalty term related to the censored proportion for the survival model. ■

The Authors

Lian Yan is a senior machine learning scientist at Aureon Biosciences. His research interests include machine learning and its applications in predictive data mining in various industries. He received his PhD in electrical engineering from Pennsylvania State University. Contact him at Aureon Biosciences, 28 Wells Ave., Bldg. 3, Yonkers, NY 10701; lian.yan@aureon.com.

Richard H. Wolniewicz is a partner at Flatirons Computing. His research interests include processing very large data sets with numeric and statistical analytics, and the application of these technologies to current business problems. He received his PhD in computer science from the University of Colorado. He is a member of the IEEE and ACM. Contact him at 5560 Colt Dr., Longmont, CO 80503-8604; richard@wolniewicz.com.

Robert Dodier is an independent researcher. His research interests include belief and decision problems and the theory and applications of machine learning. He received his PhD in civil engineering from the University of Colorado. Contact him at 60 South Boulder Cir. #6031, Boulder, CO 80303; robert_dodier@yahoo.com.

References

1. Anderson Consulting, "Battling Churn to Increase Shareholder Value: Wireless Challenge for the Future," *Anderson Consulting Research Report*, 2000.
2. A. Berson, S. Smith, and K. Thearling, *Building Data Mining Applications for CRM*, McGraw Hill, 2000.
3. M.C. Mozer et al., "Predicting Subscriber Dissatisfaction and Improving Retention in the Wireless Telecommunications Industry," *IEEE Trans. Neural Networks*, vol. 11, no. 3, 2000, pp. 690–696.
4. D.M. Green and J.A. Swets, *Signal Detection Theory and Psychophysics*, John Wiley & Sons, 1966.
5. S. Navathe and R. Ahmed, "A Temporal Relational Model and a Query Language," *Information Sciences*, vol. 49, no. 2, 1989, pp. 1–3.
6. R. Wolniewicz et al., "Enhancing the Value of Your CRM Solution through the Use of Data Representation," *Proc. CRM Infrastructure*, IBC Global Conferences, 2000.
7. D.W. North, "A Tutorial Introduction to Decision Theory," *IEEE Trans. Systems Science and Cybernetics*, vol. 3, no. 4, 1968.
8. L. Yan et al., "Optimizing Classifier Performance via an Approximation to the Wilcoxon-Mann-Whitney Statistic," *Proc. 20th Int'l Conf. Machine Learning*, AAAI Press, 2003, pp. 848–855.
9. C. Guerra-Salcedo, "Feature Subset Selection Problems: A Variable-Length Chromosome Perspective," *Proc. 5th Int'l Conf. Artificial Neural Networks and Genetic Algorithms*, Springer-Verlag, 2001, pp. 260–263.
10. L. Yan et al., "Improving Prediction of Customer Behavior in Nonstationary Environments," *Proc. Int'l Joint Conf. Neural Networks*, IEEE/Omnipress, vol. 3, 2001, pp. 2258–2263.
11. G.M. Weiss and F. Provost, *The Effect of Class Distribution on Classifier Learning: An Empirical Study*, tech. report ML-TR-44, Dept. of Computer Science, Rutgers Univ., 2001.
12. L. Breiman, "Bagging Predictors," *Machine Learning*, vol. 24, no. 2, 1996, pp. 123–140.
13. R.C. Elandt-Johnson and N.L. Johnson, *Survival Models and Data Analysis*, John Wiley & Sons, 1980.