environment, and correct underperforming models all with minimal or no human intervention.

2. DESIGN PRINCIPLES FOR AUTONOMOUS DATA MINING SYSTEMS

In developing principles for autonomous system design, we are concerned primarily with systems that maintain a large and potentially dynamic set of independent models and operate in near real-time. Examples would include systems for electronic stock trading [17], fraud detection [6], churn prediction [18], and online display advertising [8, 4].

As a result of their size and the diversity of their problem space, these systems need to function largely autonomously even in the face of changing data and populations. Specifically, a massive robust production system should:

Yield fail-safe predictions: The system must run well for every stakeholder or user, maintaining acceptable worst-case performance. Worst-case performance here has two different components: all models must perform well under normal operating conditions, and they must *continue* to perform well under reasonable deviations in the operating conditions. That is, small changes in data distribution should not destroy model performance.

In an ad-targeting system, it is better to provide solid performance on every campaign than excellent performance for apparel ads and poor performance for fast-food ads. A fraud detection system needs to be able to deal with ordinary activity fluctuations, such as those due to weekends and holidays without issuing a bunch of false alarms.

Scale and easily extend: The system should be designed for growth and be able to scale as the application requires.

Exactly what it means to grow depends on the application, but at the very least the system should be able to add new users and models gracefully. In many cases the system must also be able to incorporate new data.

Minimize human intervention: The system must be able to hold all of the above properties without excessive human intervention.

In other words, the processes that ensure fail-safe predictions, scalability, and extensibility should be as automatic as possible. Ideally, the system would be self-correcting but where that is not possible, it should be self-diagnosing. This way, when human intervention is required because of extraordinary changes in the operating environment, the intervention is minimal. For example, the system might automatically look for significant changes in data distribution or performance and issue alerts when they occur.

3. THE M6D TARGETING ENGINE

At m6d, we run an automated online display ad targeting platform. Our customers are consumer brands; they pay us to identify Internet browsers well-suited to their product and target them with display ads. In order to provide the highest quality of targeting to our customers, our system is *browser specific*, *dynamic*, and *brand specific*.

Browser-specificity means that we target individual browsers rather than static buckets of browsers, which allows for very fine-grained targeting. It is worth noting that when we speak of *browsers*, both here and throughout the paper, we are referring to an individual browser cookie. If an individual web user deletes his or her cookies or uses a different computer, the user becomes a completely different browser for the purposes of our system. We use the words "browser" and "cookie" interchangeably throughout the paper, and they always refer to the same thing.

Brand-specificity means that we build a separate classification model, using separate data, for each client,[1] and "dynamic" here means that we regularly re-evaluate and re-score browsers. A browser who is a good target today may be a bad target next week.

In order to achieve browser-level targeting, we collect browser-level data. Specifically, we target browsers based on the existence, recency, and frequency of their visits to various sites around the web. For each campaign we run, we place pixels on the customer's website. These pixels call back to `m6d.com` and allow us to create or modify cookies for any user who visits the customer site[2]. Additionally, we have pixels on non-brand sites around the web, as well as data from some third-party providers, which together present a snapshot of the browser's general web activity. It is worth noting that we only store anonymized identifiers, rather than actual URLs, at every point in our system.

Our primary classification model [4] computes a brand-affinity score based on the user's browsing activity. If many of the URLs in a browser's cookie correlate positively with conversion, the browser gets a high score, and if the user's browsing history correlates negatively with conversion, he or she gets a very low score. The targetable audience for a given campaign is some number of the highest-scoring browsers, depending on the campaign's budget. What exactly counts as a "conversion" is different for different campaigns, but usually it requires a purchase from the customer web site.

3.1 System

The above-described classification procedure runs as part of a large-scale, end-to-end data mining system. Our system responds to billions of ad-targeting opportunities each day and maintains well over 1,000 classification models while operating hundreds of different advertising campaigns. We interact with hundreds of millions of browsers per day, each of which is scored for every campaign that we run.

A diagrammatic overview of our targeting system appears in Figure 1. Our raw input data comes from two major external sources: *mapping* pixels and *action* pixels. Action pixels are placed on individual customer web sites, and we use them both to define positive browsers and to track the conversion rates associated with their campaigns. Mapping pixels are placed on content-generating sites (e.g. blogs) throughout the web, and they provide the visit history data that forms the core of our classification models. A more detailed discussion of action and mapping pixels and their role in our system appears in [4]. We discuss two quality controls on this data in Section 3.2.

These data form the basis of our low-level classification model: a linear model over the sites that a browser has visited. Specifically, mapping data define the **features** for our low-level model and action data define the **class**. The output of the low-level model, as well as some other features

[1]This is both a performance issue and a client privacy issue: data collected for one client should not not inadvertently help its competitors.

[2]This assumes, of course, that the user accepts cookies. We make no attempt to track users who have disabled cookies or opted out of our tracking.

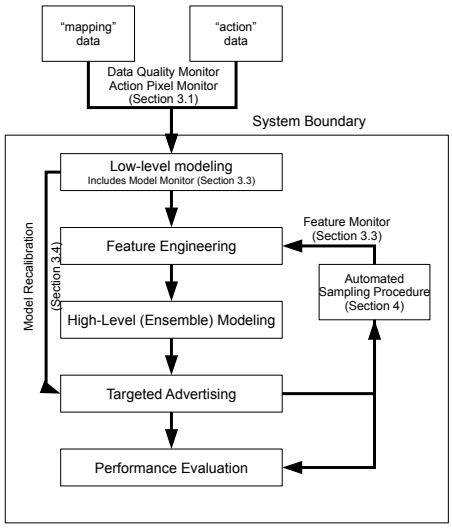


Figure 1: Our targeting system

computed from the user's cookie, form the basis of a "high-level" linear ensemble model. Both the low-level model and the ensemble model are available for user targeting. We discuss quality controls on the structure of the models themselves in Section 3.3. Once models are deployed, we evaluate their performance on a sample set that is carefully designed to represent the targetable population. The construction of the sample set and its appropriateness for model evaluation are discussed in Section 4.

It is worth briefly mentioning the *metrics* we use for evaluation, since we will need to quantify performance later in the paper. The most important feature of our classification model is its ability to rank. If the model ranks perfectly, then the optimal campaign-management strategy is to target as many browsers from the top of the list as the campaign's budget allows. Since most of our campaigns target only a small fraction of the browser population, metrics, (like ROC area) that evaluate ranking over the *entire* space of model scores can be misleading [9]. As a result, we measure performance using a more focused metric: *lift* at 5% of the population. Lift measures how many more converters (positive-class browsers) appear in the top 5% of model scores than would be expected from a random classifier.

We also track the *conversion rate* within the targeted browser population. The conversion rate for a campaign is the proportion of the targeted population (i.e. the browsers we have shown an ad) who convert within a short time of seeing the ad. Conversion rates depend on many factors other than raw classification performance, but since conversion rate is important to our customers, it serves as a measure of overall *system* performance.

In the absence of quality control, drastic changes in these performance metrics are often the only way to detect problems. The processes outlined in the rest of the paper allow us to notice changes before they affect performance.

3.2 Monitoring External System Interfaces

Our targeting system processes and incorporates a large portion of its data from sources that are either partly or entirely beyond our control. The pixels that we place on client websites and throughout the web, as well as the data we incorporate from external partners, are all critical to the functioning of our system. Changes in our client and partner systems, as well as incorporation of new partners, can lead to system disruptions if problems are not detected early on.

To this end, we have developed two separate processes aimed at monitoring this data stream. The first of these simply monitors the rate of activity on each of our action pixels. Visits to action pixels are critical elements of our model training because they form the "positive" browser population for a particular campaign. However, since this visibility depends on web servers outside of our control, the data stream could be disrupted at any time. Monitoring these streams allows us to detect disruptions before they affect system performance (ensuring *fail-safe predictions*) and reduces the human effort needed to investigate problems (*minimal intervention*).

Figure 2 shows performance statistics for a campaign that was flagged by our action pixel monitor system. Figure 2(a) plots the conversion rate among targeted browsers for one of our campaigns over time. This is a commonly-tracked business performance metric. Notice that it drops precipitously around the middle of the graph and remains low throughout.

Figure 2(b) plots conversion rate against our model's rank-order performance and Figure 2(c) plots conversion rate against action pixel activity for the same campaign and time window. Taking the two plots together, we see that both conversion rates and action pixel activity drop at exactly the same time, but that model rank-order performance (measured by lift) is not affected. The monitoring system was able to detect that external data, and not model performance, caused the drop in measured conversion rate.

In a smaller system, this type of anomaly could be detected fairly easily by human observers. As the system scales to handle more and more models, the *cause* of poor performance (or apparently poor performance) for any single model becomes increasingly difficult to identify. The installation of automated monitors such as these is an effective way to scale the system rapidly while controlling the amount of human effort actually required.

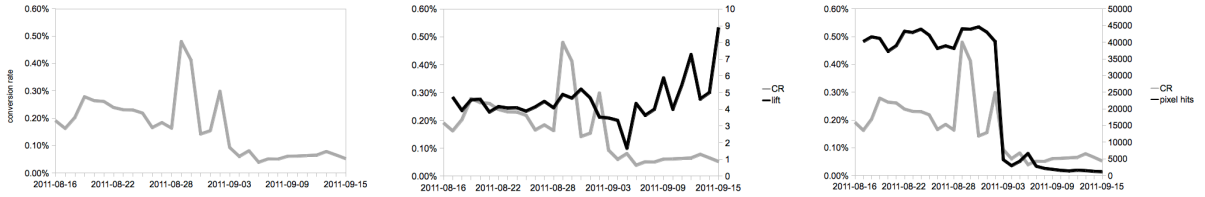
Monitoring external data sources.

A second monitoring process tracks the quality of mapping pixel (low-level feature) data from our external data providers. As we mentioned above, we incorporate browser-activity data from many different external sources. These data improve our visibility into browser activity on the web and provide most of the data that informs our models.

Monitoring our incoming data stream at the data-source level has two goals. First, we need to ensure that each source continues to provide quality data. Second, we need to understand the *risk* that would be posed to our system by losing a data source.

We incorporate data from dozens of external sources of varying sizes. The most relevant statistics we track are:

- *Reach*: the proportion of our browser population on which the source provides data. Since our data collection process seeks in part to increase the size of our targetable audience, we track both the proportion of the population that the source reaches in isolation and the proportion of the population reached the source's data removed.



(a) Anomalous Conversion Rate (b) Model Performance vs. Conversion Rate (c) Pixel Activity vs. Conversion Rate

Figure 2: Anomaly detected with action pixel monitoring.

	reach	Prop Good	avg loss	worst loss	reach without
1	0.21	0.20	0.9964	0.9657	0.9999
2	0.22	0.00	0.9927	0.9649	0.9984
3	0.24	**0.45**	0.9928	0.9409	0.9998
4	0.26	0.05	0.9944	0.9642	0.9994
5	0.32	0.30	0.9951	0.9653	0.9996
6	0.37	0.40	**1.000**	0.9806	0.9997
7	0.89	0.80	0.9166	0.6659	0.9764
8	0.90	0.75	0.9109	0.5904	0.9862
9	0.89	0.70	0.9896	0.8445	0.9842

Table 1: Example data source quality report.

- *Performance*: In order to assess the quality of the data provided by an individual source, we estimate the performance of our model using only data from that source. Similarly, to understand the risk of losing a source, we estimate our model's performance with data from that source removed.

In combination, these reports show us both the extent to which a particular source's data is *predictive*, and the extent to which it is *redundant*. Following these reports over time assures us that each of the sources we use is continuing to add value, even when all others are considered.

Table 1 shows some typical results. Each metric that we compute is an aggregate over several campaigns. We present both average-case and worst-case analyses because we need to be mindful of any change that could potentially devastate one of our campaigns.

We present an evaluation of several of our sources on a diverse set of 20 campaigns. Data sources are anonymized for privacy reasons and numbered in order of reach. The other four columns are:

1. The proportion of campaigns on which the source's data provides a lift above a minimum performance threshold. This is useful for spotting sources that are especially predictive for a few campaigns.
2. The average performance loss after removing the data source, measured by lift at 1% (1.000 indicates no loss, 0.900 would be a 10% loss).
3. The worst-case loss over all campaigns after removing a data source. In larger studies, we sometimes report 5th or 10th percentile loss. Either way, this metric is designed to ensure that there will be no crises if we were to lose a source.
4. The proportion of our browser population that we still reach when removing the source.

Most of the columns correlate relatively strongly with reach, which is to be expected. The more data that comes from a particular source, the more browsers it will reach, the more it will contribute to performance, and the more it will hurt to lose. Of particular interest to us are sources that deviate from the general trend, as these sources are discernably better or worse than their comparably-sized counterparts. In the table above, for example, source 3 contributes significantly to many more campaigns than others of similar size and source 6 has the lowest average loss in the table.

Our application of data source tracking is in many ways more relative than absolute. If a particular data provider costs more than its comparably-performing peers, the report provides a quantitative basis for renegotiation or termination. If we notice a source's performance consistently degrading, we may try to engage the data provider and work toward a technical solution.

Our data quality reporting improves both *extensibility* and *scale*, and *stability* in our system. The ability to add new data seamlessly is a major growth driver for the reach of our targeting. Tracking the report's contents over time helps ensure that we are properly insulated against the loss of a data provider and gives us the knowledge and freedom to cut data providers, if necessary, if they become too expensive. As a result, we can easily take on new providers on a provisional basis and evaluate their impact on our ability to scale.

3.3 Monitoring Data Distributions

Once the raw input data that is discussed in the prior sections has been transformed into labeled feature vectors, it is available to train our models. At this point, we monitor both the input data and the output models for significant changes that may indicate systematic problems.

Model Monitor.

The quality-control process for our low-level models consists of a series of tests on model coefficients. As we mentioned above, each feature in our low-level classification model is a distinct (anonymized) URL and the model learns a single coefficient for each feature.

Each time one of these models is re-trained, it must pass two different quality checks before entering the production system. The first is simply a paired t-test of the model coefficients. The paired T-test tests the null hypothesis that the *mean difference* between the paired observations is zero. In our particular case, it tests whether the average change in model coefficients between two successive model builds is statistically indistinguishable from zero.

In an ideal world, where our model coefficients are unbiased estimators of some true quantity, the null hypothesis is exactly what one would expect: changes in model coefficients will amount to measurement error and the mean difference should be zero. Thus, the t-test serves as a safe-

guard against significant and unexpected bias in the model coefficients.

Second, we compute the rank correlation between sets of model coefficients in successive builds. A drastic change in the rank-ordering of the model coefficients indicates a higher-than-expected *variance* in our coefficient estimates.

A failure in either of the tests simply indicates that the model has changed appreciably and flags it for human examination. At this point, the system compiles a test failure report including the relevant statistics (t-score and rank correlation) as well as performance estimates for each (old and new) model on a recent hold-out sample. If the examiner approves the new model (usually because its performance is similar or improves) then the model passes into production. Otherwise, the most recent passed model remains until the source of the change is thoroughly investigated.

It would certainly be possible to skip the t-test and the rank test altogether and automatically pass any model whose performance stays "roughly the same" or improves by some measure, but there are advantages to our approach. The t-test and rank test are much faster than computing model performance. The latter requires scoring a sample of browsers and then ranking them while the former needs only mathematical operations on coefficients. As such, it is much more efficient to compute performance only for those models that fail the test.

Table 2 shows some results from one of our automated modeling runs. The thresholds for anomaly-flagging were a T-statistic greater than 2 in absolute value or a rank correlation less than 0.7. Boldfaced models showed significant improvement in model performance and so were allowed to pass. The ability to identify potentially problematic models cheaply, (both in terms of human and computer effort) is a big part of our ability to maintain thousands of effective models at once.

Feature Monitor.

A second similar process tracks the distributions and rank-ordering ability of each of the features in our high-level model as well as the model scores themselves. Specifically, we track *feature values*, *conversion rates*, and *lift* at various points on the distribution. This tracking allows us to see changes in both the distributions of features and their relative importance to model performance over time. Since the outputs or scores of our low-level models are also features of our high-level model, those outputs get tracked as a part of this process as well.

The ability to track changes in data distribution has many well-documented benefits. If one can identify situations where the input data have changed "significantly" in some sense, it becomes easier to determine when to re-train models or develop new ones. Our primary use of distribution monitoring, however, is to learn about the behavior of our system.

For example, Figure 3 plots the 98th percentile low-level model score for one of our campaigns over time. Since the top 2% of our audience is approximately what we target for the average campaign, this represents the approximate bottom of our targetable range. The black vertical lines (on 11-14 and 1-11) mark dates when the low-level model was retrained. Notice that the distribution changes dramatically a couple days after model retraining and then stabilizes a short time after. In our experience, stabilization takes about

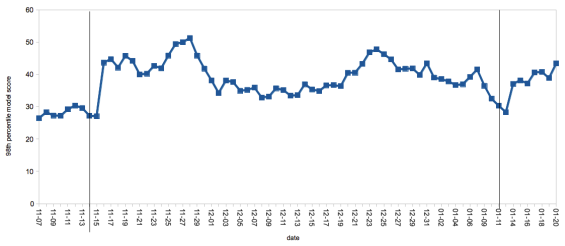


Figure 3: 98th percentile of a model score distribution over time. Black lines represent times of model retraining.

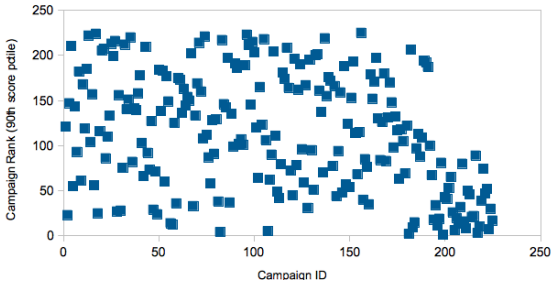


Figure 4: An emerging relationship between ID and model score can indicate a system change.

three to seven days, indicating that we should refresh our classification thresholds about a week after any retraining.

Figure 4 presents another interesting result. It shows the 90th percentile low-level model score (calculated across a sample of about one million targetable browsers) plotted against the internal ID number for the campaign associated with the model. The campaign ID is a meaningless primary key in our database, but since it is auto-incrementing, it is monotonic in the age of the campaign. That is, newer campaigns have higher IDs.

For the vast majority of campaigns, (up to about 200) there is absolutely no correlation between campaign ID and model score, exactly as we would expect. Conversely, campaigns with IDs greater than 200 appear to cluster and give very high model scores.

The phenomenon turned out to be a bug in the model-builder that affected any very-new campaigns that were also using very-new data. We fixed it very quickly after it was discovered. This result brings up an interesting point about the role of identifiers in maintaining data mining systems. We stressed earlier that automated systems need to remain stable, behaving relatively consistently in the face of "ordinary" changes in the operating environment. Figure 4 is a very clear violation of the stability principle. Our input data changed slightly (we were simply bringing on more data than we had in the past), but the behavior of the system changed substantially. The emergence of correlation between identifiers and system outputs strongly suggests that something fundamental in the system has changed.

3.4 Correcting underperforming models

Up to this point, we have discussed quality controls that take place prior to model construction. One might think of

(a) T-test

ID	T Statistic	Old lift	New lift
2829	2.7595	4.7579	2.1362
3132	5.0807	1.0830	**1.4904**
3579	2.1367	6.7901	4.7616
3934	2.4460	1.3513	**2.0831**
659	4.4058	8.7298	3.4879

(b) Rank correlation

ID	Rank Corr	Old lift	New lift
2679	0.6581	0.7860	0.7718
274	0.6726	1.9479	0.9114
2829	0.6293	4.7579	2.1362
3132	0.6925	1.0830	**1.4904**
3577	0.5651	10.7136	5.4022

Table 2: Sample model monitoring results.

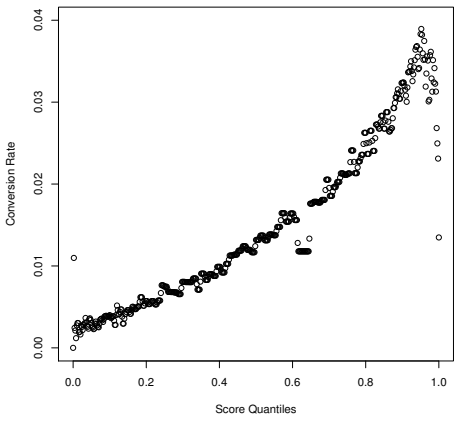


Figure 5: Binned estimates of actual conversion rate vs. quantile of score for a recalibration candidate in our system. The model mostly rank-orders well, but the highest-scoring browsers perform relatively poorly.

these controls as guarding against the pollution of innocent models by bad data or broken processes. The final quality control in our pipeline, however, goes beyond this: actually correcting for some degree of error in the models.

Both our low-level and ensemble classification models are ultimately linear functions of their constituent features. This decision has a lot of practical advantages. In addition to relatively strong performance, the storage requirements for the models are very low (just tables of coefficients) and scoring large groups of browsers is very efficient, requiring a single database join.

The downside to this approach is that, occasionally, the relationship between the conversion rate and model features is decidedly non-linear. In this case, our classification models will rank-order sub-optimally, with the highest-scoring browsers converting at less-than-optimal rates while lower-scoring browsers perform better. Similar behavior has been observed elsewhere, including the 2008 KDD Cup [14].

Since it is critical for us to perform well on every campaign (*fail-safe predictions*), we need to fix sub-optimal performance whenever we can. However, since we lack the computational resources to perform expensive searches over complicated hypothesis spaces to find the best model, we develop a novel re-calibration procedure that allows us to adjust the rank-order of suboptimal models when necessary without affecting any of the other models in the pipeline.

To see why such a procedure is necessary, simply observe Figure 5, which shows conversion probability, estimated in equal-frequency bins, against score quantiles from one of our low-level models. The top few quantiles of classifier score convert at a lower rate than the preceding quantiles despite the fact that the majority of the classifier scores rank order nicely. In this case the quantile-based conversion rates in Figure 1 could now be used to recalibrate the algorithm by re-ranking the browsers by estimated conversion rate. For most of our models, however, conversion rates are much smaller than 1% - 4%, and often smaller than 1 in 1,000. In this case, the bucketed quantiles will not provide good estimates of conversion rates. For this reason we use a generalized additive model (GAM) [10] with smoothing splines.

Figure 6(a) shows conversion rate as a function of score, for the campaign in Figure 5, as predicted by a GAM.[3] We recalibrate the scores by using the probabilities predicted by the GAM as the scores rather than the scores from the initial classifier. In this example those individuals in the top 1% as scored by the smoothing algorithm (the scores between the dotted lines on the plot) converted at a rate of about 4.0% while the top 1% of people scored by the initial algorithm (those scores to the right of the solid vertical line) converted at a rate of 2.6%. Thus, by using the re-calibrated scores we end up with a group of people that convert at a 50% higher rate than by using the initial algorithm alone. Figure 6(b) shows a campaign where the recalibrating algorithm does little to change the ranking of the browsers since those ranked in the top 1% by the GAM (between the dotted lines) have a large amount of overlap with the top 1% according to the initial algorithm (to the right of the solid vertical line). The difference in the conversion rate in this case is negligible (.159% vs .157%). In many other cases the score itself is monotonic in conversion rate and as a result the recalibration does not affect the ranking at all.

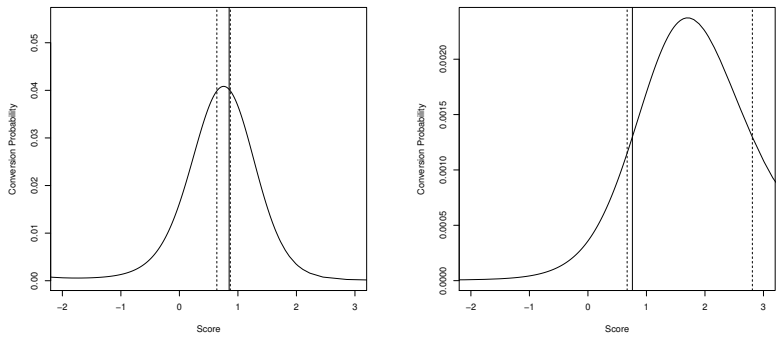
The use of GAM for re-calibration has a significant advantage over other recalibration approaches in the literature, such as Logistic [15] or isotonic [13, 5] regression in that its fit is potentially non-monotonic, allowing it to smooth over and reorder poor-performing regions. However the use of GAM is not magical. In practice, any smoothing method which can predict probabilities from a continuous feature will do.

In summary, we created a two stage classification algorithm with the following stages:

1. Fit an initial classification model on the training data.
2. Recalibrate the initial estimator using a smoothing algorithm, such as GAM, and use the predicted probability as the new score.

In situations where the relationship between the scores and

[3]Estimates were generated using the mgcv library in R.



(a) (b)

Figure 6: GAM-predicted conversion rate vs. model score. Dashed lines enclose the top 1% of the population according to the GAM, while the black line marks the 99th percentile model score.

the conversion probabilities are monotonic this method does not change the ordering, but it will reorder those classifiers where the relationship is non-monotonic. By running this algorithm in production after the initial classifier is fit we can recalibrate its scores automatically and provide higher-quality models for the affected campaigns.

The ability to do this automatically and on demand has significant practical benefits for our system. It improves our ability to scale the system (we can take on more campaigns without changing models), maintains the *stability* of the system, since we can re-calibrate scores when things change, and helps make model predictions more fail-safe by allowing us to re-order model scores.

4. CONSTRUCTING REPRESENTATIVE BROWSER SAMPLES

Several different pieces of our system rely on having a "representative" sample of our reachable browser population. The construction of such a sample is complicated by a couple of factors. First, any sample of Internet browsers is naturally prone to activity bias [1]. Care must be taken to ensure that active browsers are not overrepresented in the sample. The second complicating factor is cookie decay. Since we respect a browser's right to delete his or her cookies (i.e. do not fingerprint or reconstruct cookies), a browser gets a new identifier in our system every time it clears its cookies. In the worst case (where it rejects cookies entirely), a browser may get a new identifier for every interaction. As a result, a large majority of browser identifiers are assigned to cookies that we see exactly once.

Broadly speaking, there are two different methods for correcting these types of bias: modeling and sampling. The Heckman correction [11] and its variants, for example produce unbiased estimates of the quantity of interest (in this case conversion rate) by explicitly modeling the selection process and including an estimate of the *selection probability* in the final classification model. Prior work [2, 19] has shown that this approach is effective in classification settings, but the size of our data and the number of factors influencing selection make the construction of an effective model difficult.

The second option is to sample more intelligently. The ultimate goal of any procedure we implement is to produce a representative sample of *targetable browsers*: browsers to which, broadly speaking, we could conceivably serve an ad. The set of such browsers changes over time, of course, as users get new computers or clear their cookies, but a very simple proxy for the targetable population is the set of browsers that *actually saw an ad* on a given day.

This approach is attractive on several levels. The set of browsers that saw an ad is certainly a subset of the targetable browser population. If any targetable browser is just as likely to see an ad as any other targetable browser, it is a perfectly representative sample of the population.

Unfortunately, this is not the case. Some browsers in our system qualify for dozens of campaigns, and these browsers are much more likely to see an ad than browsers who qualify for only a few campaigns. Additionally, this kind of impression-based sampling is heavily influenced by external factors. Whether a qualified browser actually sees an ad depends, among other things, on the amount of competition for the browser and how many ads we have already shown that day.

Our sampling algorithm accounts for these difficulties in the following manner. Each day, our system selects a random sample of browsers that meet the following criteria:

1. The browser is qualified for at least two of our campaigns. This guarantees that the cookie has survived long enough and accumulated enough information to be targetable.
2. We interacted with the browser on the day in question. This guarantees that the cookie is still active.

We find that this sampling methodology produces training and evaluation samples that generally outperform impression-based sampling. A fairly straightforward explanation appears in Figure 7, which shows distributions of a few high-level model features over both impression and qualification samples. It is easy to see that the qualification sampling gives access to a wider range of feature values, providing a more inclusive sample than impression-based sampling. In particular, the qualification sampling tends to get more representation from lower-valued regions of the feature space, which tend to contain less active browsers.

Performance estimates on this sample of browsers correlate surprisingly well with the "real" performance on which

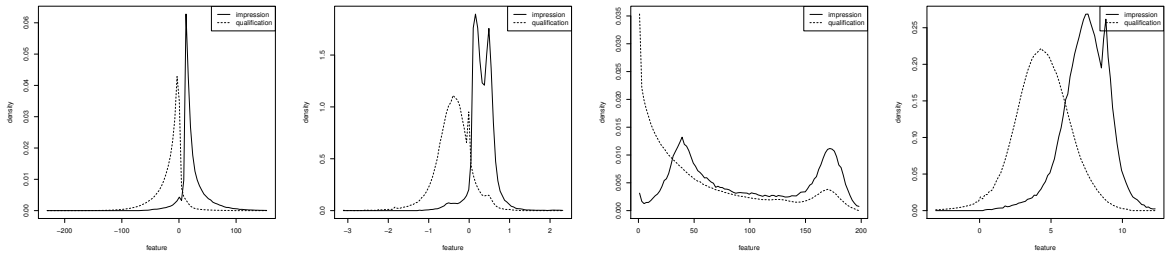


Figure 7: Distributions induced by different sampling schemes. Impression-based sampling tends to exclude less active browsers (i.e. those with lower feature values).

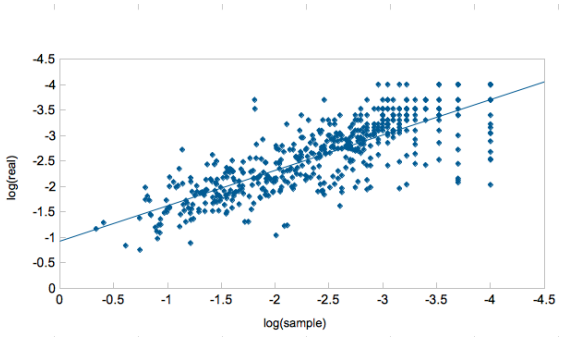


Figure 8: Correlation of sample performance estimates and real performance.

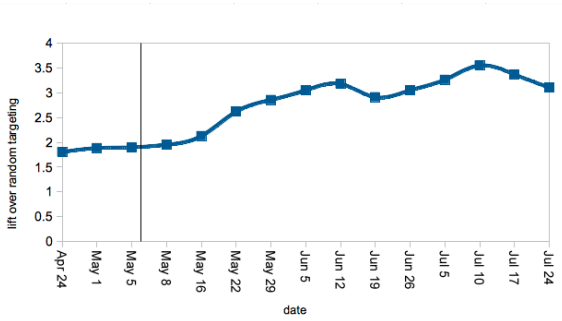


Figure 9: High-level model performance before and after switching to qualification-based sampling (black line).

we are ultimately evaluated. Figure 8 shows a plot of predicted performance against real-world performance on our entire browser population. The two quantities are correlated with $r^2 = 0.75$ and a Spearman rank coefficient of 0.8.

Further validation is available in Figure 9, which shows shows a plot of our median high-level model performance over time. On the date marked by the black line, we began training our high-level models on a qualification-based rather than an impression-based sample. The cutover was gradual, but after fully implementing this sampling procedure for training, the performance of the high-level model increased more than 50% from 1.88x (lift over untargeted-browser performance) to sustained performance above 3.0x going forward.

A consistent, automated, representative sampling procedure is, in our opinion, a critical component in a large-scale data mining system. Having a readily available automated sample ensures not only that the evaluation of new methods is consistent and correct and contributed to the development of most of the other quality-control systems described here.

5. RELATED WORK

We are not aware of any papers that discuss quality control in deployed systems as large as ours. Sculley et al [16] present a system for detecting malicious ads that works in a semi-automated fashion (consistent with "minimal intervention") and trains thousands of models (consistent with "scale"). The authors say that automated monitoring and recalibration tools are important aspects of their system, but they do not give any implementation details.

Ghani and Kumar [7] discuss an interactive quality-control improvement to a health-insurance-auditing application. They augment an existing classifier (which predicts overpayments in health insurance claims) with a "more-like-this" feature to reduce the time human auditors spend auditing claims.

Moreno-Torres et al [12] report on a situation where data generated for the same classification task from two different research labs were incompatible. In other words, the classifier trained on one lab's data was completely useless for the other lab even though the features and label were theoretically the same. They develop a genetic-programming-based feature-engineering algorithm that solves the problem in their case.

The problem of change detection has attracted considerable attention, although most of it in a research setting. The most relevant of these deal with change detection in production systems. Curry et al [3] develop a change-detection system for identifying anomalies in credit card payment records and Fawcett and Provost [6] describe a system for fraud detection from cellular phone records. Both of these systems serve to automatically detect adverse system changes with minimal human intervention. In both of these cases, however, change detection is a problem in and of itself, rather than a quality control within a production system.

6. PRACTICAL GUIDANCE

So far we have given a very general set of principles along with a very specific description of our own production system. The purpose of this final section is to offer a few practical suggestions for someone trying to implement our principles from scratch. The following simple suggestions, based

on our experience, should help practitioners trying to get a new system off the ground.

- **Closely monitor data feeds** from any external source, including the Internet. Unless you control the data-generating process *completely*, it will eventually change.
- **Monitor interfaces** between critical internal systems. In a large evolving system, even intentional changes may have unintended consequences. It is best to detect these as quickly as possible.
- **Work, then scale, then excel**: It is easier to improve a system that provides adequate performance at scale than to scale a system that was not designed to grow. For example, simple models are fast to train, fast to score, and can be enhanced on a case-by-case basis as necessary.
- **Build a sandbox**: A consistent, automatically-generated data sample saves time when experimenting with new ideas, but also helps diagnose problems with the existing system. Data that is pulled the exact same way for two months makes identifying changes much easier.
- **Bound your error**: The simplest realization of the *fail-safe predictions* principle simply is to ensure that even dreadful model predictions cause limited damage to your system. We do this ourselves in a number of ways, for example by limiting the number of ad impressions any one browser can see.

7. CONCLUSION

Operating an end-to-end data mining system at scale requires more than just clever feature engineering or brilliant algorithmic design. The consistent need to grow and change the system while maintaining or improving performance means that *quality control* becomes increasingly important as the system matures.

In an effort to energize and focus the discussion of quality control in the data mining community, we have proposed three desirable criteria for a self-sufficient data mining system. Next, we present the quality-control processes that we have developed for our online display-ad targeting system in accordance with the principles outlined above. Many of these controls developed because of our need to maintain thousands of classification models simultaneously, but they should be useful in any system where the number of distinct components makes direct human oversight impractical.

We have automated monitoring of all of our external data, which allows us to scale and extend the system without increasing the quality-control burden on human operators. We monitor and flag significant changes in our model coefficients, which helps ensure that our system is *stable*: that is, that it continues to operate well in a changing environment. We have developed an algorithm, based on generalized additive models, which re-calibrates the output of a classification model and improves its rank-ordering performance. This naturally keeps performance "fail-safe", but also helps the system *scale*, as we can add new customers and be confident that our models will continue to perform well.

8. REFERENCES

[1] D. Chan, R. Ge, O. Gershony, T. Hesterberg, and D. Lambert. Evaluating online ad campaigns in a pipeline: causal models at scale. In *Proceedings of KDD*, pages 7–16. ACM, 2010.

[2] N. Chawla and G. Karakoulas. Learning from labeled and unlabeled data: An empirical study across techniques and domains. *Journal of Artificial Intelligence Research*, 23(1):331–366, 2005.

[3] C. Curry, R. Grossman, D. Locke, S. Vejcik, and J. Bugajski. Detecting changes in large data sets of payment card data: a case study. In *Proceedings of KDD*, pages 1018–1022. ACM, 2007.

[4] F. Provost et al. Audience selection for on-line brand advertising: privacy-friendly social network targeting.

[5] T. Fawcett and A. Niculescu-Mizil. PAV and the ROC convex hull. *Machine Learning*, 68(1):97–106, 2007.

[6] T. Fawcett and F. Provost. Adaptive fraud detection. *Data mining and knowledge discovery*, 1(3):291–316, 1997.

[7] R. Ghani and M. Kumar. Interactive learning for efficiently detecting errors in insurance claims. In *Proceedings of KDD*, pages 325–333. ACM, 2011.

[8] A. Goldfarb and C. Tucker. Online display advertising: Targeting and obtrusiveness. *Marketing Science*, 2010.

[9] D. Hand. Measuring classifier performance: a coherent alternative to the area under the roc curve. *Machine learning*, 77(1):103–123, 2009.

[10] T. Hastie and R. Tibshirani. *Generalized additive models*. Chapman & Hall/CRC, 1990.

[11] J. Heckman. Sample selection bias as a specification error. *Econometrica*, pages 153–161, 1979.

[12] J. Moreno-Torres, X. Llorà, D. Goldberg, and R. Bhargava. Repairing fractures between data using genetic programming-based feature extraction: A case study in cancer diagnosis. *Information Sciences*, 2010.

[13] A. Niculescu-Mizil and R. Caruana. Predicting good probabilities with supervised learning. In *Proceedings of ICML*, pages 625–632. ACM, 2005.

[14] C. Perlich, P. Melville, Y. Liu, G. Świrszcz, R. Lawrence, and S. Rosset. Breast cancer identification: Kdd cup winner's report. *ACM SIGKDD Explorations Newsletter*, 10(2):39–42, 2008.

[15] J. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. *Advances in Kernel Methods: Support Vector Learning*, 208(MSR-TR-98-14):1–21, 1998.

[16] D. Sculley, M. Otey, M. Pohl, B. Spitznagel, J. Hainsworth, and Y. Zhou. Detecting adversarial advertisements in the wild. In *Proceedings of KDD*, pages 274–282. ACM, 2011.

[17] H. Stoll. Electronic trading in stock markets. *J. Economic Perspectives*, 20(1):153–174, 2006.

[18] C. Wei, I. Chiu, et al. Turning telecommunications call details to churn prediction: a data mining approach. *Expert systems with applications*, 23(2):103–112, 2002.

[19] B. Zadrozny. Learning and evaluating classifiers under sample selection bias. In *Proceedings of ICML*, page 114. ACM, 2004.

[20] ZenithOptimedia. Global ad expenditure to return to pre-recession peak level this year. `http://www.zenithoptimedia.com/files/media/image/news/PressReleasefiles/2011/July/AdspendforecastsJuly2011.pdf`, 2011.