Scalable Supervised Dimensionality Reduction Using Clustering

Troy Raeder,
Claudia Perlich
M6D Research
37 E. 18th Street
New York, NY 10003
troy,claudia@m6d.com

Brian Dalessandro,
Ori Stitelman
M6D Research
37 E. 18th Street
New York, NY 10003
briand,ori@m6d.com

Foster Provost
New York University
& M6D Research
44 W. 4th Street
New York, NY 10012
fprovost@stern.nyu.edu

ABSTRACT

The automated targeting of online display ads at scale requires the simultaneous evaluation of a single prospect against many independent models. When deciding which ad to show to a user, one must calculate likelihood-to-convert scores for that user across *all* potential advertisers in the system. For modern machine-learning-based targeting, as conducted by Media6Degrees (M6D), this can mean scoring against thousands of models in a large, sparse feature space. Dimensionality reduction within this space is useful, as it decreases scoring time and model storage requirements. To meet this need, we develop a novel algorithm for scalable supervised dimensionality reduction across hundreds of simultaneous classification tasks. The algorithm performs hierarchical clustering in the space of *model parameters* from historical models in order to collapse related features into a single dimension. This allows us to implicitly incorporate feature and label data across all tasks without operating directly in a massive space. We present experimental results showing that for this task our algorithm outperforms other popular dimensionality-reduction algorithms across a wide variety of ad campaigns, as well as production results that showcase its performance in practice.

Categories and Subject Descriptors

I.5.4 [**Computing Methodologies**]: Pattern Recognition—*Applications*

General Terms

Algorithms, Experimentation, Performance

Keywords

supervised dimensionality reduction, clustering

1. INTRODUCTION

Online display advertising, (broadly speaking, the showing of banner ads on websites) is a large and growing industry, with consumer brands expected to spend over $34 billion on display ads in 2013 [22]. The problem of *targeted display advertising* involves determining where, when, and to whom to show a particular display ad on the Internet.

Display advertising decomposes roughly into two distinct tasks: *retargeting*, which involves advertising to people who have had some prior interaction with the brand (e.g., visiting a brand's website) and *prospecting*, which seeks to acquire new customers by advertising to people without an observed prior affiliation with the brand.

At M6D, we operate a large-scale, machine-learning-based display advertising system that delivers tens of millions of display impressions for hundreds of different advertisers every day. Our goal is primarily prospecting, meaning that we want to identify, for each brand, a set of previously unaffiliated individuals who are most likely to respond to the brand's advertising. A response in our case is not a click but rather a "post-view conversion," where a user takes a *brand action* (see below) within a given time period after seeing the ad. In practice, this means that we score hundreds of millions of Internet users as either "good" or "bad" prospects for each of our client brands.

The central component of our targeting technology is a massive-scale machine learning system [12, 16, 17] that simultaneously operates and autonomously updates thousands of classification models in parallel. Our primary predictive modeling learns directly from anonymized browsing history of browsers that allow third-party cookies. An individual data point, for the purposes of our system, is a *browser cookie*. Whenever someone visits our website or interacts with one of our data partners, we get the opportunity to set a cookie on his or her browser. We use these same cookies to maintain a partial history of the URLs on which we interact with the browser, either through our own systems or those of our data partners. Throughout the paper, we use the words **browser**, **cookie**, or **user** to refer to an individual instance of our cookie.

For the purposes of model-building, this history becomes a massive, sparse binary feature space of hashed URLs. A user gets 1 for all URLs in the cookie and 0 for everything else. Currently, we track over 100 million unique URLs in the system, any of which could be used for modeling. For most of our models, the class label during training is simply a binary indicator of whether or not the user has ever taken some sort of **brand action**. This action is typically visiting the brand's home page, downloading target material, or making a purchase from the brand's site.

Our high-dimensional models work quite well (as judged by our internal measurements and by clients' comparisons with other targeters [13]), but there are situations where lower-dimensional models are preferable, such as:

1. **Rare Events**: For some of our models, positive-class examples are very rare. The vast majority of people have never even visited a brand's site, let alone purchased something. As a result, it is difficult to estimate reliable model parameters in millions of dimensions.
2. **Cold Start**: We typically cannot observe brand actions until a campaign has nearly begun (the advertiser has to place a pixel on their website). This exacerbates the rare-event problem for our first models.
3. **Overhead**: High-dimensional models are expensive to build and store. Lower-dimensional models are useful when conducting complex analyses for which high-dimensional models would unduly burden the system.

The goal of this work is to identify a good 'generic' lower-dimensional representation of our high-dimensional feature space that will be used across many different campaigns for predictive modeling. Keep in mind that with insufficient labels, supervised dimensionality reduction and feature selection is unlikely to succeed. Additionally, it would imply that for each campaign we have to maintain a different feature space which burdens the system even further. The existing literature provides a wide variety of unsupervised techniques for dimensionality reduction in large data sets that would be suitable. However, in our particular situation, we have additional information that can be brought to bear: a large library of historical models built on the same, massive feature set. These models encode information about each URL's impact on conversion for hundreds of different campaigns. It seems natural to use as much of this information as possible.

To this end, we have developed a multi-task hierarchical clustering scheme for dimensionality reduction. The algorithm identifies closely related dimensions by finding clusters in the space of *model parameters* from all of the campaigns' models in our system. These clusters produce a much-lower-dimensional space in which similar dimensions combine URLs with similar *predictive information* (strength and direction of affiliation with the class label) across all of our models. As it turns out, models built in this lower-dimensional space are nearly as informative as models built on the original high-dimensional data. Furthermore, we show that our approach outperforms several popular alternatives..

The remainder of the paper is organized as follows: Section 2 gives a brief overview of dimensionality reduction techniques and the particular techniques that we consider in this paper. Section 3 describes our supervised multi-task dimensionality reduction technique in detail. Section 4 experimentally compares our technique with several popular existing techniques, and shows its performance within the M6D production system, and Section 5 presents conclusions and some observations on the nature of dimensionality reduction in large production data.

2. DIMENSIONALITY REDUCTION FOR ONLINE DISPLAY ADVERTISING

The goal of dimensionality reduction is to take a high-dimensional feature vector $\mathbf{x} = (x_1 \dots x_n)$ and project it into a k-dimensional space, with $k << n$, such that the resulting lower-dimensional vector $\mathbf{x}'$ captures the vast majority of the relevant information in $\mathbf{x}$. Prior research has produced a number of very popular techniques for dimensionality reduction (e.g. [6, 8, 9, 21]), and we cannot consider all of them. Instead, we focus in particular on techniques that are well-suited to our problem domain.

We prefer techniques that are feasible in large, sparse binary data. This rules out, for example, PCA [21] and ICA [8], which require rescaling the columns to have zero mean. Such mean-centering destroys the sparsity of the data, dramatically increasing the space and time required.

Additionally, we prefer techniques that will produce a *single* low-dimensional feature space across multiple classification tasks, since it is prohibitively expensive to maintain thousands of projection matrices for thousands of models. All unsupervised dimensionality reduction techniques trivially satisfy this requirement, but it rules out many existing supervised approaches [6, 9].

Subject to these constraints, we have experimented with a number of different dimensionality-reduction approaches, which we outline briefly here:

Feature Hashing: Perhaps the simplest way to reduce a massive binary feature space is via *feature hashing* [20]. Feature hashing transforms a bag of words into a bag of hashed IDs. Given a set of tokens and a hash function $h()$, we apply the hash function to each of the tokens and the new feature space is simply the set of hashed values. Dimensionality reduction results from hash collisions. For example, if a cookie contains {`m6d.com`, `nytimes.com`, `nyu.edu`}, and we have h("`m6d.com`") = 6, h("`nyu.edu`") = 6 and h("`nytimes.com`") = 8 then, in the new space, the cookie has values for features 6 and 8. Hash functions are typically 32-bit or 64-bit, and to project into an arbitrary k-dimensional feature space, we compute $h(\cdot) \mod k$.

The method of [20] also employs a second one-bit hash, which assigns a *sign* (1 or -1) to each URL. The new feature space is integer-valued rather than binary, and example i's value for feature j in the new feature space is given by:

$$\mathbf{x}'_{ij} = \sum_{u|h(u)=j} h_2(u). \tag{1}$$

where the summation runs over all the URLs u in example i for which $h(u) = j$.

Contextual Categories: The web has a number of sources, both proprietary and free, that *categorize* specific web pages by their content. These categories serve as content-based groupings that can be used to reduce the dimensionality of the data. For the purposes of this paper, we have obtained two sources of category data: a freely available web ontology called the Open Directory Project (ODP)[1], as well as a proprietary categorization from a commercial data provider. With category data, our original feature space of URLs becomes a feature space of categories. A given browser gets 1 for any category in which he has visited at least one website and 0 for all other categories.

Singular Value Decomposition: The reduced-rank *sin-*

[1]`www.dmoz.org`

gular value decomposition (SVD) approximates an $m \times n$ data matrix $\mathbf{X}$ by a rank-k matrix $\mathbf{X}'$, defined as the product of three matrices:

$$\mathbf{X}' = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \tag{2}$$

where $\mathbf{U}$ and $\mathbf{V}$ are $m \times k$ and $n \times k$ orthonormal matrices of *singular vectors* respectively and $\mathbf{\Sigma}$ is a $k \times k$ diagonal matrix of *singular values.*

SVD is theoretically attractive because it produces the *best* (in the least-squares sense) reduced-rank approximation to the original data, and recent work has led to fast, block-wise algorithms for low-rank SVD'[7] that can operate in a distributed fashion on large data sets like ours.

The matrix $\mathbf{U}\mathbf{\Sigma}$ forms the reduced-rank data set used for classification. A new data set $\mathbf{X}_2$ in the original n-dimensional feature space can be converted to its low-rank representation by:

$$\mathbf{X}_2' = \mathbf{X}_2\mathbf{V} \tag{3}$$

SVD is unique among techniques that we discuss in that it produces a *dense* output matrix rather than a sparse one. The implications for storage and computability make this approach less appealing to us, but the predictive performance observed using SVD in recent data mining competitions [10] on sparse data makes it an attractive technique for comparison.

Supervised Clustering: Each of the prior techniques will work for our particular problem domain, since they can reduce dimensionality in large, sparse binary data. However, as unsupervised techniques, they cannot incorporate one of the richest sources of information that we have: the modeling experience on hundreds of campaigns over the years. We develop in the next section a multi-task clustering approach that utilizes this knowledge by using models' parameters as the features describing the URLs.

3. MULTI-TASK CLUSTERING

More so than in data mining environments where an objective performance metric can be optimized, the effectiveness of clustering is driven by deliberate setup choices. The core of clustering is the distance metric and with it the representation of the entities of interest. We are trying to group together URLs such that using only the groups retains the maximal amount of predictive performance. Initially, there are a number of 'natural' ways of thinking about the similarity between websites:

- Similarity of a site's content using, for instance, tf-idf after crawling the page and identifying the contained text. Aside from the obvious challenges of crawling and extracting text (permissions and computational effort), we would also need to account for the internal structure of sites and aggregate their content in some consistent way. Practically, M6D is neither willing nor able to do so for a project that is meant to simplify our computation.
- Co-visitation patterns of browsers visiting the sites: Given our data sources, it is straightforward to derive a co-visitation graph. Using some measure of overlap between the audiences of two sites one could project that there should be semantic relationships that suggest some form of similarity. However, whether or not this translates to predictive information is less clear. Another issue is scale. Calculating overlap when the (visitation) base rates vary by orders of magnitude is known to be tricky [15].

	F1	F2	F3	F4	F5	F6	F7	F8
Task 1	1.3	0.9	1.1	0.1	-0.2	0	-2.3	-3.2
Task 2	0.4	0.5	0.2	-6.4	-5.3	-5.9	-5.7	-6.1
Task 3	1.9	2.1	1.7	-0.1	0.2	0.3	1.3	1.5
Task 4	-1.0	-1.2	-1.4	5.1	5.4	4.7	0.3	0.2

Table 1: Example representation where each feature represents the coefficient of a linear model that was estimated for a given task. We include here four hypothetical tasks. The curious reader should be able to identify 3 distinct groups of features (the answer is in the text).

M6D has another, much more relevant set of information about URLs in our system: how they affected the conversion-likelihood scores modeled across many past campaigns. It is likely that future campaigns share similarities with those of the past in terms of industry and product types. So intuitively, URLs that consistently affect the predictions in a similar way can be grouped together safely without losing predictive information. Consider Table 1 as an example. It shows the parameters of a linear model for 8 different features on 4 different tasks and there are 3 intrinsic feature groups.

3.1 Distance Measure

Having decided on the representation—using the parameters from previous modeling tasks as features, the question of selecting a good distance measure is less difficult than for a typical clustering task. We are looking to group features with similar impact on the predictions across the different tasks. While it is less important with our binary indicator features, we would prefer a metric that is somewhat indifferent to the scaling of the features and mostly directional. In a linear model if two features were nearly identical except that one is a multiple of the other, the parameters are likely to be similar subject to the scalar difference (the parameter scales counter-proportional to the scaling of the feature). As a result we used correlation similarity (specifically 1-correlation as the distance function). Looking at the example in Table 1, the tree groups that become apparent using correlation similarity are features 1-3, 4-6 and 7-8.

3.2 Hierarchical Clustering

There are a number of well-known clustering methods including k-means, hierarchical clustering, and (also applicable in this case) co-clustering. To maintain flexibility in the following step of cluster assignment we choose hierarchical clustering. Figure 1 shows an example result on a small subset of 70 URLs.

The clustering for the production system was done for the 15,000 URLs with the *highest visitation rates* during a one-week training period. We use URLs with the highest visitation rates because these are the URLs whose coefficients will have the least variance across all campaigns. The choice to limit the clustering to 15,000 URLs is mostly due to memory constraints in R. If we ever want to increase the number of URLs covered, a simple approach would be to assign any new URL to the cluster with which it is most correlated. So

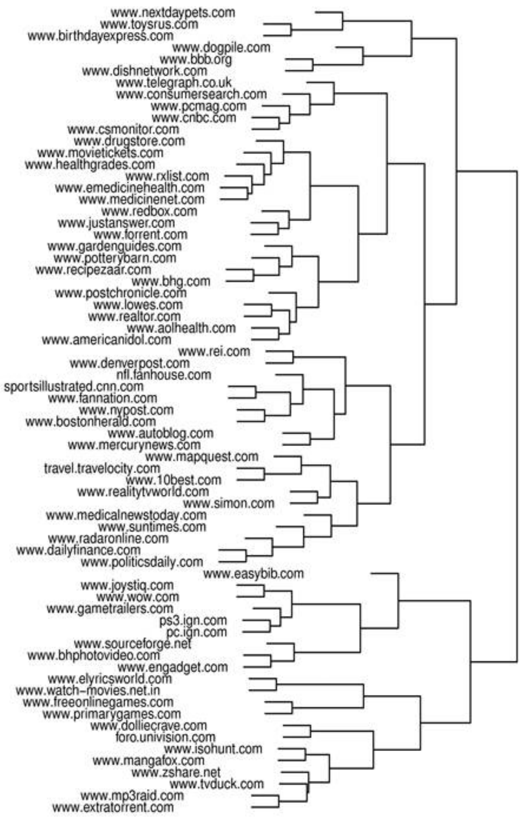


Figure 1: Small example of clustered URLs.

it is not strictly necessary to include all 100 million URLs in the actual clustering algorithm. There is some danger in including rarer web sites into the clustering step as their parameters are likely to suffer more from variance during the prior model estimation step. Having a majority of entities with basically random features would likely reduce the effectiveness of the approach. We use the parameters for 100 campaigns with most positive examples that were running at the time. The parameters are estimated using a variant of naive Bayes that calculates a likelihood ratio [14]. In production, a large number of our models are built as logistic regressions using SGD [3] for estimation. For clustering, we deliberately use naive Bayes models as they maintain each individual feature's signal more distinctly. For example, if there were a set of n URLs with nearly identical relationships to the targets, the logistic models would adjust the individual parameters accordingly (ideally but not necessarily reducing each by a factor of $1/n$, depending on the regularization), whereas the class-conditional independence assumptions lead naive Bayes to overestimate the parameters. While this may be detrimental to predictive performance, it is beneficial for identifying similar URLs. Naive Bayes also tends to produce *stable* parameter estimates, since each individual parameter estimate unaffected by the others. The hierarchical clustering was performed in R using 1-correlation as the distance metric and the *hclust* package as shown in Appendix A. The small dendrogram in figure 1 shows a nice even breakup of the space. Upon investigation we find many meaningful groupings. This confirms our initial intuition that using the parameters from campaign models provides a promising representation that does not suffer from size-induced artifacts so commonly observed in clustering.

Intuitively, the grouping should reveal **both** contextual similarity and co-occurrence. One example is information substitutes—some people read the financial section of the NY Times and others the financial section of the Wall Street Journal. Only a few read both. The clustering identifies them as substitutes and by grouping them we improve the sparsity and reduce variance. But while context could cause content to cluster together, it does not have to be the case. The connection is 'discovered' by the predictive modeling and may work in a way we do not understand. Our notion of context may be different from the indication of consumer interest. In this sense, seeing `www.toysrus.com` and `www.birthdayexpress.com` together indicates that the consumer is probably in the market for children's gifts. While the two sites might have co-occurrence of people shopping around (co-visitation), the intuitive 'consuming intent' relationship between these sites can be observed by the clustering even without co-visitation.

3.3 Cutting the Tree And Assigning URLs

The most common way to assign entities to hierarchical clusters is to cut the tree at a suitable height. However, this may not lead to the best set of features as some of the clusters will have many fewer (and less commonly visited) URLs than others. A good feature needs to have useful coverage. Thus, rather than a static cutoff on the dendrogram, we designed a recursive cut function that traverses the cluster tree and cuts a branch once the URLs below reach a minimum coverage (number of browsers visiting them). The code is provided in Appendix A where the input data format has the number of browsers for each site in the first column. In essence, the recursive procedure attempts to balance the sparsity and the similarity of the elements in the final clusters. We assign a goal coverage of 1% resulting in a total of 4,318 final clusters, where each URL is assigned to exactly one cluster.

3.4 Two Technical Notes

Let us quickly clarify how we convert the original features (indicators of URL visitation) into the new feature space. The two obvious options are: the count of number of original URLs in each cluster (the sum of the original indicator features), or the creation of another simple indicator when the user visits at least one URL in the cluster. For our production system we have chosen the latter as it is consistent with the representation used by our other models. We have never found increased predictive performance when replacing the indicators with counts or other more complex expressions. Ultimately, keeping it simple also adds to the robustness of the system implementation [17] and avoids potential artifacts from the heavy-tailed Poisson distributions. Furthermore, some of our models use L1 regularization, so keeping the features on a similar scale is beneficial.

One more technical note is important. Clusters are constructed as a group of websites that consistently get similar model parameter estimates across a wide range of tasks. By grouping them as one feature, we force all of these websites "into" the same parameter and that may induce some bias in the individual estimates. On the other hand, the reduced dimensionality reduces the variance of the estimates for the models built on the new clusters. Choosing to use dimensionality reduction always displays a preference for bias over variance. It is an empirical question as to whether proposed clustering approach is more effective at limiting this bias compared to other approaches. We have not examined the bias/variance decomposition in detail—largely because ours are ranking tasks, for which the decomposition is non-trivial. We do believe that leveraging the learning from 'similar' tasks might indeed be beneficial, because we minimize the bias by using other models to tell us which websites can 'safely' be put together without reducing the predictive information content.

3.5 Related Work

Our clustering solution relates to a number of existing techniques and strategies in machine learning.

- **Stacking and Ensembles:** We are proposing a 3-step process where we first estimate a number of models across multiple tasks and use them for a second modeling stage. The main difference between classical stacking [4, 19] and our method is that we are NOT using the predictions of the first level as features in the next stage. Rather, we cluster the features based on the learned models' parameter estimates, and then use the resultant clusters as the second-level feature space.
- **Transfer learning:** Transfer learning in the most general sense [11] suggests leveraging knowledge derived from one task onto a different one where either the domain (feature space and sampling) or the dependent variable are different. In this sense, our clustering approach fits this definition very well. We utilize what is informative across a variety of potentially similar tasks to generate a new (simpler) feature space for the next task.
- **Multi-task learning:** Being an instance of inductive transfer, multi-task learning [5, 2] is also related to our clustering-based design. We are exploiting the commonality among a set a tasks (via the parameters estimated for a common set of features) to build a model for the task at hand.
- **Hierarchical Bayesian Modeling:** The intuition of using insights from related tasks (transfer learning) has also been used in a Bayesian framework to define multivariate Gaussian priors on parameter estimates [18]. The algorithm of [18] also uses other similar learning problems to estimate the covariance of pairs of individual parameters. The main difference of our work is the larger grouping of arbitrarily many features into a distinct set of clusters and a much stronger constraint of enforcing essentially the same parameter rather than just having a prior. The advantage of the more restrictive setting is the desired reduction of the feature space and model size.

4. EMPIRICAL RESULTS

In order to demonstrate the utility of our proposed cluster method for dimensionality reduction, we report three distinct evaluation settings. The first is a set of fully controlled experiments where we compare our clustering approach to the set of comparable techniques outlined in Section 2 'in vitro' under identical settings (training data, model estimation, and campaign mix). The second scenario compares our production models (models estimated using our production system on potentially different date ranges) rather than the controlled experiments. We are trying to answer the question how much better the high-dimensional production models perform. We compare models estimated on the cluster and the original high-dimensional feature space 'in vitro' in a controlled sandbox (identical test set) used by M6D to track the quality of our models internally. Finally we look at the real campaign performance when we show ads to the audience selected by the model in the M6D production environment for a small set of campaigns that were running both cluster and high-dimensional models 'in vivo'.

4.1 Controlled Experiments

In order to evaluate dimensionality-reduction algorithms, we need to compare them with respect to the performance of a particular classification algorithm. For the purposes of this paper, we use one of our production algorithms: a logistic regression trained by stochastic gradient descent [3]. Given the speed and workload constraints in our production system, more complex classifiers like random forests are generally not applicable, so we do not discuss alternatives here.

Experimental Setup:
We chose a set of 28 campaigns that are reasonably representative of our overall advertiser mix and built separate low-dimensional models for each campaign. We use real data from seven days in January 2013 for training and the following day's data for evaluation.

For each campaign, we transformed both the training set and the evaluation set into the different feature spaces described in Section 2. For both contextual categories and feature hashing, the transformation is trivial: for the category data it is simply a lookup table, and for feature hashing it is provided by the hash function (`hash()` in Python, in this case). Recall from Section 3 that we only use the 15,000 most popular URLs in our system to train the cluster model. For a fair comparison, we reduce the exact same 15,000-dimensional feature space for all techniques. The transformation into the cluster space was described in Section 3.4.

For SVD, the transformation is derived from data, and we learn the transformation from a subset of our negative-class training data. Since we use the same negative set across all campaigns, this results in a single transformation matrix. Despite using the efficient stochastic SVD algorithm, subsampling was necessary in order to run it within our memory constraints.

There are exactly 4,318 dimensions in the low-dimensional feature space produced by the clustering (and running in production), so for comparison, we tried to get as close to this number as possible with the other techniques. Since it is trivial to hash into any number of buckets, we use exactly 4,318 features for the hash models. With Open Directory, our existing dimensions map into 5,594 distinct groups, and this is what we use. For the commercial category data, the number is 1,183.

SVD is considerably more difficult. The algorithm of [7] and its Hadoop implementation are relatively fast, but it operates in blocks to control memory consumption. As a result, computation time can be an issue. More constraining, however, is the production of the training and test sets, where each combination of user and URL joins to k different factor values in order to produce the product $\mathbf{XV}$. Even after downsampling the training set to reduce compu-

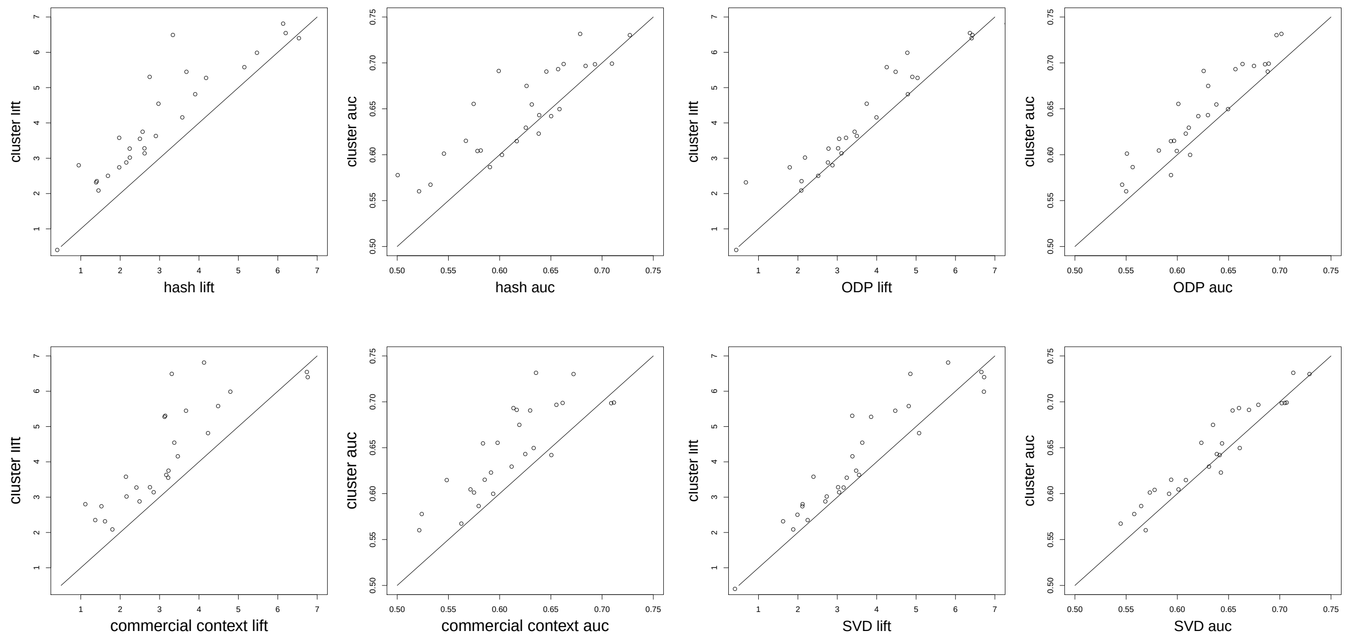


Figure 2: Performance comparison between our algorithm and competitors across all 28 campaigns. Dots above the line indicate superior performance for our clustering algorithm.

tation time, it proved prohibitively expensive to expand the feature set beyond 1,000 dimensions, as several different attempts to scale up to 2,000 ran our machines out of memory.[2] Whenever we present SVD results here, we use 1,000 dimensions since that is the strongest performance we have. We briefly explore the performance of lower-dimensional projections later in the paper.

Figure 2 compares the performance of our logistic classifier on clusters against each of the other reduced feature spaces. We report two performance metrics here: area under the ROC curve (AUC) and *lift* at a threshold of 5%. This is simply the number of positive examples in the top 5% of model scores divided by the number that would be expected from a random classifier (i.e., 5% of all positives). We often report this number internally because it reflects a typical campaign setup, where given a fixed budget that can reach only a small proportion of the targetable population, M6D is expected to return as many conversions as possible. All of our performance estimates are averaged over 100 bootstrap estimates in order to reduce the variance in the results.

Each plot compares the performance of our algorithm against the performance of a single competitor across all 28 campaigns as measured by either lift or AUC. Values above the identity line indicate that our algorithm performed best and values below the line mean that the challenger performed best. Several conclusions are immediately evident from the figure. First, our clusters outperform all of the other methods most of the time. Furthermore, when they do lose, it is generally by a small margin.

The clusters seem to be stronger, relative to other methods, under lift than under AUC. This is perfectly fine for our purposes, since performance on the highest-scoring browsers is most important for our business.

In order to quantify the performance differences presented visually in Figure 2, Table 2 summarizes the average performance and average relative performance, as well as statistical interpretations of the performance differences. More specifically, average relative performance is the *average ratio* between our clustering and a competing method. We report a difference in performance as a "win" if our clusters perform significantly better than a competitor with 95% confidence using means and variances estimated from the bootstrap samples. Similarly, we report a loss if they perform significantly worse.

The supervised clusters outperform all competitors, in both lift and AUC, over our set of test campaigns. It has more statistically significant wins, fewer losses, and higher average performance. The most significant improvement is relative to feature hashing, where we perform 42% better on average.[3] Even if we remove the most extreme outlier, the number drops only to 35%.

SVD performs admirably in a much smaller feature space. At 1,000 dimensions, SVD had the best relative performance in terms of both lift and AUC and clearly outperformed the commercial category feature space, which is its closest competitor in terms of dimensionality. One possible explanation is that we were not able to obtain contextual data for all 15,000 of our original URLs from either our commercial or free category sources, whereas SVD can make use of any information that we feed into it. This is a fundamental limitation of any third-party data source for dimensionality

[2]This was run on a cluster of 30 machines, each with 8 cores and 16 GB of memory. We distributed the computation across dozens of machines (30,000 examples per input file; 65 reduce tasks) and limited each instance to 2 GB of memory.

[3]Relative differences in lift are more meaningful to us than relative differences in AUC. A 42% increase in lift means 42% more conversions at the same targeting budget.

(a) AUC

	average performance	average relative improvement	win	loss	tie
Cluster	0.640	-	-	-	-
vs ODP	0.619	1.034	24	2	2
vs Commercial	0.611	1.061	22	3	3
vs Hash	0.615	1.043	19	4	5
vs SVD (1000)	0.629	1.018	16	5	7

(b) Lift at 5%

	average performance	average relative improvement	win	loss	tie
Cluster	4.024	-	-	-	-
vs ODP	3.643	1.185	17	1	10
vs Commercial	3.195	1.403	24	2	2
vs Hash	3.035	1.428	26	1	1
vs SVD (1000)	3.539	1.162	20	4	4

Table 2: Comparison of dimensionality reduction techniques over 28 campaigns. Wins and Losses are computed using 95% confidence intervals from bootstrap estimates.

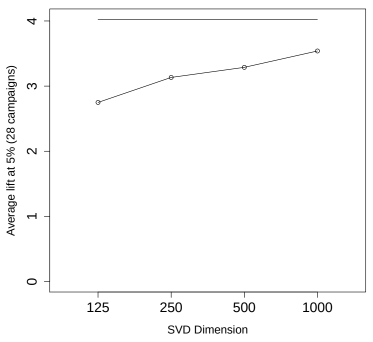


Figure 3: SVD-based model performance as a function of dimensionality in terms of average Lift over 28 campaigns.

reduction: the power of the low-rank feature space is limited by the reach of the source data.

Even with this limitation, the human-curated contextual categories from Open Directory perform very well, just 18% behind our clusters on average. The Open Directory categories were statistically indistinguishable from our algorithm over 1/3 the time, but significantly outperformed our algorithm only once. It is worth mentioning that the OpenDirectory feature space is the largest in our study, which may play a part in its strong performance.

To the best of our knowledge [10], SVD is typically used to compress data into a much smaller dimensionality than we have attempted here, but we found these extremely low-dimensional models to be tremendously ineffective in our domain. Figure 3 plots the average performance of the SVD models against the average performance of cluster models as we scale up the dimensionality of the SVD. SVD model performance improves considerably from 125 features to 1,000 features (about 30% on average) and in lower dimensions compares very unfavorably against our cluster method, shown for reference as a black line in the figure.

4.2 'In Vitro' Model Performance

We currently build $\tilde{9}00$ cluster -based models in production, serving hundreds of thousands of display ads per day. Like all of our other models [17] they are built automatically, with absolutely no human curation. While humans do ultimately decide what proportion of each campaign's budget to allocate to a particular model, cluster models exist, and we score users against them, for the majority of our campaigns.

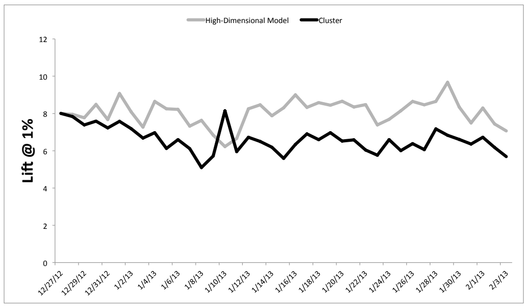
One of the quality control tools in the M6D system [17] is a 'sandbox' test dataset of random browsers for which we track brand actions and evaluate our models. This happens independently of whether or not we showed an ad for the campaign. We have established previously [17] that performance in this sandbox correlates with actual campaign performance. The goal of this metric is to isolate the model performance from the circumstances (such as bid price, scale, or audience) of the campaign and allows for more rigorous testing and quality control of the automated modeling.

Figure 4(a) summarizes the performance of all our currently active cluster models, compared against the corresponding high-dimensional models, regardless of whether they are currently being used for targeting. We report the *median* lift at 1% over the 120 different models, since median model performance is our preferred internal summary metric of system performance. Cluster performance tracks fairly consistently at 15%-20% below the performance of our high-dimensional model. The average performance difference over the entire time window is 17.4% and the median is 19%. In summary, our low-dimensional cluster models achieve a significant fraction of our main model's performance, but the full model has typically a distinct advantage.

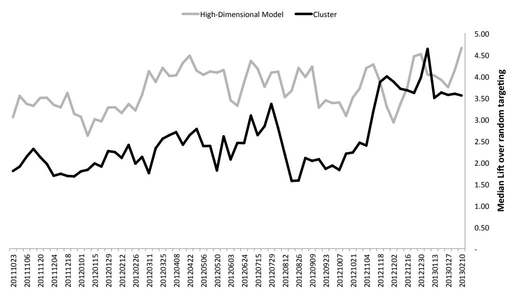
4.3 'In Vivo' Campaign Performance

Figure 4(b) shows median targeting performance, over all active campaigns, of both the cluster model and our high-dimensional production model over time. The metric here is post-impression conversion rate relative to random targeting. In order to provide "control group" baselines for comparison, we continually serve a relatively small number of untargeted (random) impressions for each of our campaigns. The y-axis in Figure 4(b) is simply the (median) conversion rate of the targeted population divided by the conversion rate of the untargeted population. The plot runs from October 2011 to February 2013, and the big increase in cluster performance corresponds to a redefinition of the underlying clusters. The main difference between Figures 4(a) and 4(b) is that since the latter shows actual targeting performance, it only includes models that *actually delivered* impressions in the wild. As a result, models that are allocated zero targeting budget, presumably due to poor performance, are excluded.

While the previous 'in vitro' results depend only on the quality of the model, the 'in vivo' performance of relative post-impression conversion rates depends on many other factors, including the price that we bid for a particular campaign, the level of competition for good browsers, and the scale of the campaign. It is interesting to observe that while our high-dimensional model clearly outperforms the cluster model on average 'in vitro', the difference diminishes



(a) 'in vitro' model performance



(b) 'in vivo' campaign performance

Figure 4: Median model performance over time across all active campaigns.

considerably after taking into account our human budget-allocation decisions in the time period after the most recent re-clustering was done.[4] Here we see evidence of the campaign managers' ability to 'cherry pick' only the best models based on their historical performance, and in particular to use cluster models only when they perform well.

5. CONCLUSION AND DISCUSSION

We present an algorithm for supervised, multi-task dimensionality reduction using hierarchical clustering and demonstrate its effectiveness in both laboratory and production environments. In particular, we show that classification models built in the low-dimensional space derived by our algorithm perform nearly as well, in production, as models trained on a massive feature space of about 30 million URLs. Additionally, we show that our algorithm outperforms other reasonable alternatives for classification in high-dimensional sparse binary data. In the course of this study, we have experimented with a variety of dimensionality reduction techniques in a wide variety of classification scenarios. In the process, we have noticed a number of implementation issues that may be of interest to practitioners.

Feature hashing is extremely fast and very easy to implement, but not really suitable for dimensionality reduction in our case. Most applications of feature hashing (e.g., [1]) hash into a much larger feature space than we have used here. It is possible that our 15,000-dimensional feature space used for clustering is not sparse enough for the collisions induced by hashing to be harmless.

Singular Value Decomposition performs reasonably well, even though we never reached 4,000 dimensions. It is interesting to note that while prior studies [10] have reported success reducing to much smaller spaces, we see considerable benefit in going all the way to 1,000 dimensions.

However, it became clear over the course of the study that SVD would be very difficult to implement in our production system. Transforming our training data into the learned feature space requires a massive database join that takes hours even when distributed across dozens of machines. Furthermore, the resulting training data files (the input to our learning algorithm) are huge compared to what we usually encounter (1,000 nonzeros per example, compared to 20 or 30 in a typical sparse model). This dramatically increased the strain on the system in terms of the time it takes to train models, the disk space used to store input files, and the I/O demands that we place on the system. Scoring new browsers against existing models requires this same expensive transformation, which severely limits the ability to keep scores up-to-date.

One of the key features of our targeting system [17] is that we continuously and autonomously rebuild thousands of classification models in order to keep pace with changes in people's browsing behavior. Furthermore, we are constantly re-scoring browsers in order to incorporate the newest information from the browser's cookies. As such, an approach that dramatically increases training or scoring time is undesirable. Our results may provide further support the continued investigation of SVD for situations where model-building is infrequent, the number of models to build is small, and near-real-time scoring is not required.

Category Data is unique to our domain (learning from web histories), but is not totally unreasonably to include, because there are a lot of similar applications where people learn from web histories or web traversal patterns. Our results indicate that human-curated category data have some promise as a proxy for the individual URLs themselves, but such data will always be limited by the completeness of the associated database. Furthermore, the effective incorporation of such data is not necessarily trivial. For example, the site that we know as `health.yahoo.net` is encoded in OpenDirectory as `health.yahoo.com`. By contrast, automated techniques like hashing, SVD, and our own cluster algorithm utilize data we already have.

In summary, the designed cluster approach shows better predictive performance compared to any of the (feasible) alternative dimensionality reduction approaches. The reduction in predictive performance in many campaigns is limited compared to using the full models with two to three orders of magnitude more parameters. Due to their lower dimensionality, cluster-based models are notably more efficient to build and maintain and serve in our system. Further, the clusters themselves can provide additional insights to the brands (an interesting direction for future research).

[4]The clustering itself runs infrequently in production.

6. REFERENCES

[1] D. Agarwal, R. Agrawal, R. Khanna, and N. Kota. Estimating rates of rare events with multiple hierarchies through scalable log-linear models. In *Proceedings of KDD*, pages 213–222, 2010.

[2] A. Argyriou, T. Evgeniou, and M. Pontil. Convex multi-task feature learning. *Mach. Learn.*, 73(3):243–272, Dec. 2008.

[3] L. Bottou. Large-scale machine learning with stochastic gradient descent. In Y. Lechevallier and G. Saporta, editors, *Proceedings of the International Conference on Computational Statistics*, pages 177–187, Paris, France, August 2010. Springer.

[4] L. Breiman. Stacked regressions. *Machine learning*, 24(1):49–64, 1996.

[5] R. Caruana. Multitask learning: A knowledge-based source of inductive bias. In *Proceedings of ICML*, pages 41–48. Morgan Kaufmann, 1993.

[6] X. Geng, D. Zhan, and Z. Zhou. Supervised nonlinear dimensionality reduction for visualization and classification. *IEEE Trans. on Systems, Man, and Cybernetics, Part B*, 35(6):1098–1107, 2005.

[7] N. Halko, P. Martinsson, and J. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288, 2011.

[8] A. Hyvarinen. Fast and robust fixed-point algorithms for independent component analysis. *IEEE Trans. on Neural Networks*, 10(3):626–634, 1999.

[9] G. Karypis and E. Han. Fast supervised dimensionality reduction algorithm with applications to document categorization & retrieval. In *Proceedings of CIKM*, pages 12–19. ACM, 2000.

[10] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, Aug. 2009.

[11] S. Pan and Q. Yang. A survey on transfer learning. *Knowledge and Data Engineering, IEEE Transactions on*, 22(10):1345–1359, 2010.

[12] C. Perlich, B. Dalessandro, R. Hook, O. Stitelman, T. Raeder, and F. Provost. Bid optimizing and inventory scoring in targeted online advertising. In *Proceedings of KDD*, pages 804–812. ACM, 2012.

[13] C. Perlich, B. Dalessandro, O. Stitelman, T. Raeder, and F. Provost. Machine learning for targeted display advertising: Transfer learning in action. Technical Report CeDER-13;01, Stern School of Business, New York University, 2013. http://hdl.handle.net/2451/31710.

[14] C. Perlich and F. Provost. Distribution-based aggregation for relational learning from identifier attributes. *Machine Learning*, 62(1-2):65–105, 2006.

[15] C. Perlich and S. Rosset. Identifying bundles of product options using mutual information clustering. In *Proceedings of the Seventh SIAM International Conference on Data Mining*, 2007.

[16] F. Provost, B. Dalessandro, R. Hook, X. Zhang, and A. Murray. Audience selection for on-line brand advertising: privacy-friendly social network targeting. In *Proceedings of KDD*, pages 707–716. ACM, 2009.

[17] T. Raeder, B. Dalessandro, O. Stitelman, C. Perlich, and F. Provost. Design principles of massive, robust prediction systems. In *Proceedings of KDD*, 2012.

[18] R. Raina, A. Ng, and D. Koller. Constructing informative priors using transfer learning. In *Proceedings of the 23rd international conference on Machine learning*, ICML '06, pages 713–720, New York, NY, USA, 2006. ACM.

[19] G. Valentini and F. Masulli. Ensembles of learning machines. *Neural Nets*, pages 3–20, 2002.

[20] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of ICML*, pages 1113–1120. ACM, 2009.

[21] S. Wold, K. Esbensen, and P. Geladi. Principal component analysis. *Chemometrics and Intelligent Laboratory Systems*, 2:37–52, 1987.

[22] 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.

APPENDIX

A. CLUSTERING R CODE

```
size=dat[,1]
dat[is.na(dat)]=0;
co=cor(t(dat[,2:ncol(dat)]),use="pairwise.complete.obs")
co[is.na(co)]=0;
cl=hclust(as.dist(2-co))
mergesize=1:(nrow(dat)-1)
mergesize[]=0
for (i in 1:(nrow(dat)-1)) {
        c1=cl$merge[i,1]
        c2=cl$merge[i,2]
        if (c1<0){c1s=size[-c1]}else{c1s=mergesize[c1]}
        if (c2<0){c2s=size[-c2]}else{c2s=mergesize[c2]}
        mergesize[i]=c1s+c2s;
}

extract=function(ind,clusterid) {
        if (ind<0){res[-ind,2]<<-clusterid}
        else {
                extract(clu[ind,1],clusterid)
                extract(clu[ind,2],clusterid)
                clu[ind,3]<<- floor(maxsize)*100
        }
}

clu=cbind(cl$merge,mergesize)
res=dat[,1:2]
res[,2]=0
i=nrow(dat)-1
n=0;
while(i>0) {
        if (clu[i,3]<maxsize) {
                n=n+1;
                extract(i,n);
        }
        i=i-1;
}
```