To appear in *Proceedings of the 2014 ACM International Conference on Information and Knowledge Management*, 2014.

On Building Decision Trees from Large-scale Data in Applications of On-line Advertising

Shivaram Kalyanakrishnan*
Indian Institute of Science
Bengaluru Karnataka 560012 India
shivaram@csa.iisc.ernet.in

Deepthi Singh, Ravi Kant
Yahoo Labs Bangalore
Bengaluru Karnataka 560071 India
{drsingh, rkant}@yahoo-inc.com

ABSTRACT

Decision trees have been used for several decades as simple and effective solutions to supervised learning problems. Their success extends to tasks across a variety of areas. Yet, data collected today through web-domains such as on-line advertising presents many new challenges: sheer size, the prevalence of high-arity categorical features, unknown feature-values, "cold starts", sparse training instances, and imbalance in the class labels. We argue that decision trees remain an ideal choice for applications of on-line advertising as they naturally construct higher-order conjunctive features; we then contribute two ideas to improve tree-building accordingly. First, to handle high-arity categorical features, we introduce a method to cluster feature-values based on their output responses. The result is more "data-dense" trees with relatively small branching factors. Second, we employ cross-validation as a principled approach to derive splitting and stopping criteria: thereby we identify splits that generalize well, and also curb overfitting. Evaluated on three distinct probability-estimation tasks in on-line advertising, our method, "CCDT", shows significant improvements in the accuracy of prediction.

Categories and Subject Descriptors

I.2.6 [**Computing Methodologies**]: Artificial Intelligence—*learning*

General Terms

Algorithms

Keywords

Decision trees; categorical features; on-line advertising; clustering; cross-validation

*Shivaram Kalyanakrishnan contributed to this paper when he was at Yahoo Labs Bangalore.

CIKM'14, November 3–7, 2014, Shanghai, China.
Copyright 2014 ACM 978-1-4503-2598-1/14/11 ...$15.00.
http://dx.doi.org/10.1145/2661829.2662044.

1. INTRODUCTION

Supervised learning is perhaps the most common and typical form of machine learning [5, see Chapter 1]. Given a set of "key-response" pairs $\langle \mathbf{X}, y\rangle$, the objective is to learn a mapping that can be applied to a new key $\bar{\mathbf{X}}$ to predict its response $\bar{y}$. For example, a bank could apply supervised learning in order to detect (and thereby prevent) fraudulent credit card transactions. Here the key could be represented by *features* such as the time of the transaction, the area code, the amount, and the number of transactions from the account in the last 24 hours; the response would be either "fraud" or "no fraud".

Being a very general setting, supervised learning has received a great deal of attention over the years: numerous approaches have been proposed to model and learn the mapping between key and response. Examples include linear regression, logistic regression, decision trees, neural networks, support vector machines, and nearest-neighbor techniques [5, 16]. Naturally every learning *method* has strengths and weaknesses with respect to different types of *problems* [21], and so it becomes critical for a practitioner to (1) select the most appropriate method for his/her problem and (2) adapt the chosen method specifically for the task at hand.

In this paper, we argue that the inbuilt capability of decision trees to handle categorical features makes them a natural choice in applications of on-line advertising, where such features are predominant. Specifically, tree-building amounts to constructing discriminative higher-order conjunctions of the atomic features. This defining aspect of decision trees relieves much of the burden of domain analysis and feature engineering, which are far more critical to the success of contrasting approaches such as regression. In a domain such as on-line advertising, where new features become available periodically, and response patterns drift over time, the "representation discovery" that comes "for free" with decision tree-building is especially appealing.

A key technical challenge remains to be surmounted to scale decision tree algorithms to the on-line advertising domain: the categorical features typically encountered therein can take a very large number of values (sometimes tens of thousands). Canonical decision tree algorithms are not designed with such "high-arity" features in mind. The first contribution of this paper is a method to recursively cluster feature-values based on their output responses, thereby reducing the branching factor of trees and keeping them "data-dense". High-arity features and the size of web data also imply that the resulting trees are large, often with millions of nodes. To minimize suboptimal splits and arrest their compounding effects as large trees get built, our second contribution is a cross-validation-based rule to explicitly promote good splitting and stopping decisions at each node.

This paper is organized as follows. In Section 2, we provide a brief review of decision trees. Section 3 presents an overview of on-line advertising, highlighting the relevance of decision trees to tackle supervised learning problems in this domain. Section 4 provides a detailed description of our improved decision tree-building algorithm, denoted "CCDT". We present an experimental evaluation of the algorithm in Section 5, and an accompanying discussion in Section 6. Section 7 serves as the conclusion.

2. DECISION TREES

Decision trees [32] are a natural way to map keys to responses. A decision tree partitions the universe of keys and associates a predicted response with each cell of the resulting partition. Thus, the predicted response for a (test) key is simply the prediction associated with the cell to which the key belongs. The partition is learned and represented hierarchically, as a tree, wherein each cell corresponds to a leaf. Figure 1 presents a schematic depiction of a decision tree for the credit card fraud-detection example described in Section 1.

There exist well-established algorithms, such as ID3 [32] and C4.5 [33], for inducing decision trees from training data. These algorithms work in a top-down fashion, initialized with a root node containing all the training data. Nodes are recursively split until some termination condition is met. As in the illustration in Figure 1, splits typically correspond to conditioning on a value of some categorical feature (example: value "31451" of feature "area code"), or on a range into which a real-valued feature (example: range "< 3" of feature "transactions in last day") could fall. At a given node, it is usual to evaluate the splits that apply and to pick the one that separates the node's data into the most "homogeneous" sets. Depending on the application, homogeneity is suitably defined: information gain and gain ratio [32] are common criteria when the responses are categorical.

Splits are typically chosen *greedily*, based on a 1-step look-ahead. While multi-step look-ahead can potentially yield better splits, it is usually ruled out because of the added computational cost. Even so, a greedy approach typically

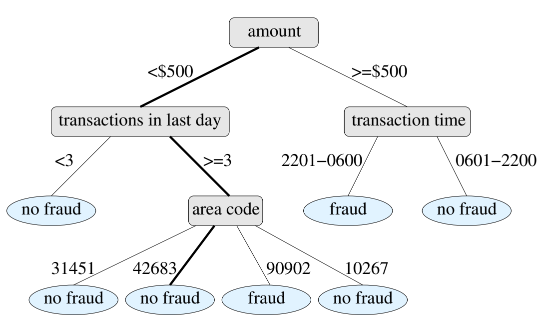

Figure 1: Illustrative decision tree for credit card fraud-detection (see Section 1). The highlighted path-to-leaf is interpreted as follows: if the transacted amount is less than \$500, <u>and</u> there have been three or more transactions in the last 24 hours, <u>and</u> the area code is 42683, then "no fraud" is predicted.

produces trees where "informative" features occur close to the top, splitting the training data into more homogeneous regions in which other features might then become discriminative. Nodes are not split further when the responses of the training data reaching the node become sufficiently homogeneous: the majority class in the set of responses could be taken as the prediction in classification tasks, and the mean response taken as the prediction for regression tasks. In the example in Figure 1, note that instead of picking a categorical outcome, we could indeed predict the *probability* of "fraud" (and therefore, also of "no fraud"). The corresponding prediction at a leaf would simply be the fraction of "fraud" responses contained therein.

The literature on decision trees is too large to summarize here; the excellent survey by Murthy [27] describes several algorithmic variants and applications.

3. ON-LINE ADVERTISING

Over the last few years, computational advertising has emerged as a field in its own right [9], with large numbers of advertisers, users, publishers, and networks transacting at a high frequency in an automated fashion. Due to the sheer volume of the traffic, the monetary implications of on-line advertising are substantial. On-line advertising can be split into two broad categories: **display** and **search** advertising. Display advertising can itself be divided into two channels: **guaranteed** delivery (GD) and **non-guaranteed** delivery (NGD). Below we describe these advertising models and the predictions tasks that arise therein.

Display Advertising: Guaranteed Delivery. Under GD advertising, advertisers negotiate deals with publishers to target specific user segments with ad campaigns, usually for a predetermined period. For example, a contract might dictate that advertiser $\mathcal{A}$ will pay publisher $\mathcal{P}$ a certain amount if ads from $\mathcal{A}$'s campaign are shown to male users in the age group 30-40 in Germany in the period May 15–June 15, *guaranteeing* at least 2,000,000 impressions (ad views). Publishers tend to have fairly accurate projections of the traffic on their websites from different segments, and so it becomes a planning problem for them to decide how most profitably to apportion future impressions among competing advertisers [4].

Before committing to a contract, $\mathcal{A}$ would naturally want to estimate the value it would get from the campaign under consideration. Specifically $\mathcal{A}$ would want to know to what extent the campaign would engage the user segment being considered. User clicks serve as a proxy for user engagement, and so an estimate of the click-through rate (CTR) for the campaign, among the specified user segment, would be a useful input to $\mathcal{A}$'s strategy. The projected CTR could also be used by $\mathcal{P}$ to attract advertisers such as $\mathcal{A}$ to purchase GD campaigns.

It is quite common that there will not be sufficient historical data exactly associating the chosen user segment with $\mathcal{A}$'s intended campaign; rather, the corresponding CTR-estimate would have to be derived based on statistics from $\mathcal{A}$' previous campaigns on similar segments. Thus, the case arises for learning a mapping between (1) keys comprising user, publisher, and advertiser attributes, and (2) CTR.

Display Advertising: Non-guaranteed Delivery. In the NGD model, advertisers place bids in an auction for showing their ads on publisher pages; the auction is conducted through an intermediary entity called the ad exchange. An auction occurs every time a user visits a participating publisher's web page on which there are slots for NGD ads (see Figure 2). McAfee [24] provides a compelling overview of the challenges involved in designing ad exchanges. Our specific interest here is in a supervised learning problem that the exchange needs to solve.

Advertisers place bids in the exchange using different "pricing types" to display their ads. In particular, an advertiser might choose to pay a cost per mille (CPM), cost per click (CPC), or cost per action (CPA). Under CPM, the advertiser agrees to pay its bid (divided by 1,000) if the publisher gives it an impression; under CPC, the advertiser only needs to pay if the impression results in a click; under CPA, a payment is due only if an action (such as a purchase or a subscription) occurs.

How must the winner of each auction be selected so as to maximize the publisher's revenue? The publisher's overall expected revenue is maximized if in each auction, the ad exchange picks an ad that maximizes *expected* payment. Under the CPM pricing type, the advertiser's expected payment is the same as its bid (divided by 1,000), but under CPC, the expected payment is the product of the bid and the *probability* of a click. Likewise, under CPA, the expected payment is the product of the bid and the probability of an action. Thus, in order to act optimally, the exchange needs to "normalize" bids under different pricing types by multiplying them with the corresponding probabilities, before they can be compared and a winner picked. Herein lies the problem of accurately predicting the probability of a click or an action, based on recorded interactions among users, publishers, and advertisers. Figure 2 provides an illustration.

It is common practice to learn separate prediction models for different pricing types. For example, one could learn independent models for predicting the click-through rate (CTR: the probability of a click given an impression) and the post-click conversion rate (PCC: the probability of a conversion (or action) given a click) [1, 34]. Multiplying the predictions of these models on a key would yield the probability of an action given an impression.

Search Advertising. "Sponsored search" [15] is similar to NGD display advertising in that advertisers bid to show their ads to the user. However, unlike in display, bids are placed on *search keywords*, which convey a strong signal of

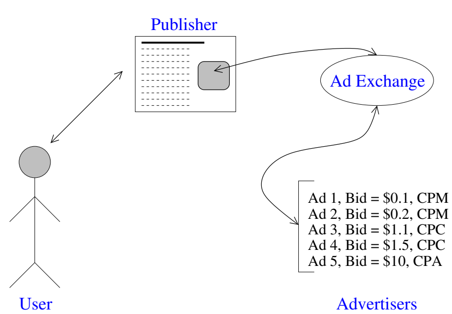

Figure 2: Schematic view of the interaction between the entities participating in NGD advertising. If the probabilities of clicks on Ad 3 and Ad 4 are 2% and 1% respectively, their expected payments are $0.022 and $0.015, respectively.

the user's *intent*. Thus, when a user searches on a publisher page using a certain set of keywords, advertisers who have bid on these and related keywords become eligible to show their ads. Payment is typically "pay per click" (PPC), which is analogous to the CPC pricing type in display advertising. Naturally, the most profitable strategy for the publisher is to rank the ads based on their expected revenue, here equal to the product of the bid and the probability of a click. Thus, search advertising, too, calls for CTR-prediction.

In summary, probability-estimation problems lie at the heart of virtually every channel of on-line advertising. User, publisher, and advertiser features constitute the key; historical logs yield empirical estimates of the corresponding probabilities. Given the volume of advertising traffic, even small improvements in prediction accuracy can result in significant revenue gains.

3.1 Data Characteristics

For the purposes of this paper, we adopt the generic notation of "successes" and "tries" to denote clicks and impressions, respectively, in CTR-prediction; and conversions and clicks, respectively, in PCC-prediction. For simplicity, we refer to the ratio of successes and tries as "CTR". To prepare a data set for CTR-prediction, we aggregate all the events sharing the same key, with which we then associate the total number of successes, the total number of tries, and the corresponding CTR.

Table 1 describes the three data sets we use in our experiments: one each from the GD, NGD, and Search domains. The GD and NGD data sets are prepared from traffic logged at Yahoo: to maintain confidentiality, we do not disclose the actual features used, and only report scaled versions of the observed successes and CTRs. The Search data set is a public one from the 2012 KDD Cup challenge (Track 2) [29].[1] Our data sets illustrate the key challenges to be surmounted in tackling prediction-problems in on-line advertising.

Large size. The foremost characteristic of data sets collected on the Internet is their large size. Production systems routinely have to process data sets with billions of tries. Our GD and NGD data sets, recorded over the period of a few days for specific advertisers and publishers, themselves have tens of millions of tries. Grouping event-level data based on keys brings down the effective number of lines, which is still a few millions in our NGD data set.

High-arity categorical features. In on-line advertising, features are derived from the three main participating entities involved: users, publishers, and advertisers. User features, for example, could pertain to users' age, gender, geographical location, preferences, purchasing habits, and such. Publisher features could include the publisher index, page URL, content category (such as News, Sports, and Entertainment), and so on. Likewise, advertiser features could also include various categories, along with indices identifying advertisers, campaigns, and ads. Observe that most of these features are naturally categorical. Moreover, unique

[1]The KDD Cup data set is available at: `http://www.kddcup2012.org/c/kddcup2012-track2`. We use the following features from this data set: `url`, `ad_id`, `advertiser_id`, `depth`, `position`, `gender`, `age`. We ignore query-related features that require further text-processing to become useful.

Table 1: Summary of data sets used for our experiments. For purposes of confidentiality, the successes and CTR in the GD and NGD data sets have been scaled. All the features used are categorical.

	GD	**NGD**	**Search**
Tries	148,641,519	21,899,406	152,288,386
Successes	46,423	87,254	6,298,022
CTR	0.0003	0.004	0.041
Features	8	11	7
Feature arities	600, 21, 21, 17, 10, 2, 2, 2	9574, 8225, 940, 874, 313, 231, 182, 61, 37, 16, 4	579903, 26313, 15155, 6, 3, 3, 3
Keys	21,399	4,055,962	8,830,065

indices for publishers, advertisers, and campaigns can take thousands of values, as seen in our NGD data set. An additional source of information in search advertising is the search query itself, which usually gives rise to real-valued features after processing (for example, by taking dot products of bag-of-words representations). We do not consider this class of features in this paper, but naturally it would be useful to combine them (such as through a mixture model) for obtaining more accurate predictions.

Sparse training instances. It is commonly seen with high-arity categorical features that a large number of feature-values occur infrequently in the data, but taken together, these feature-values constitute a significant number of tries. Consider the feature taking 9,574 values in our NGD data set. Of these feature-values, 9,564 *each* occur less than 1% of the time, but *together* they constitute roughly 50% of the tries. Hence, a learning algorithm cannot afford to ignore feature-values that occur infrequently.

Unknown feature-values and cold starts. Both due to technical and business reasons, often it happens that some features corresponding to an event cannot be logged. Thus, for many features, "missing" becomes a feature-value, often with a significant fraction of the tries. "Cold starts" refer to situations in which feature-values not seen during training present themselves on-line, while testing. The continual birth of new ads and web pages leads to a regular stream of cold starts in on-line advertising.

Imbalanced classes. Clicks and conversions are rare events: the number of successes in any CTR-prediction task is usually a few orders of magnitude lower than the number of tries. Imbalanced classes, which are challenging even in the classification setting, become especially so when the objective is probability-estimation [42].

The challenges enumerated above give a unique flavor to supervised learning problems in on-line advertising. Large data sizes are common to most applications on the web. Domains such as on-line shopping and recommendation systems, where entities such as users, publishers, advertisers, and products are categorized and indexed, also share the characteristic of having categorical features.

3.2 Why Decision Trees?

Decision trees are particularly well-suited for the CTR-prediction tasks described above, since they naturally get built using categorical features. Methods such as logistic regression, support vector machines, and neural networks are inherently set up to work on real-valued features: to employ them for these tasks, we would first have to translate our categorical features into useful real-valued (for example, boolean) features.

Indeed Rosales *et al.* [34] present a successful application of logistic regression to CTR-prediction, wherein categorical features are first converted into boolean features of the form "Does feature F take value v?" The number of such "unary" boolean features is the arity of F. Since logistic regression combines these features *linearly* (before passing the result through a sigmoid), it becomes necessary to have more discriminative boolean features. Rosales *et al.* therefore also construct "quadratic" boolean features of the form "Does feature F_1 take value v_1 and feature F_2 take value v_2?" The fundamental tradeoff in this scheme is that we can obtain increasingly discriminative boolean features by adding more terms to the conjunction—but the size of the resulting set of boolean features grows exponentially!

Hashing is a common technique for learning a smaller number of coefficients than the number of real-valued features used in logistic regression [39]. While it is true that a minimal amount of collision is usually harmless, we find for our data sets and features, when using a hash table with 2^{26} entries, that the collision ratio can be as high as 100, significantly reducing the prediction-accuracy.

Agarwal *et al.* [2] also present an architecture based on logistic regression for CTR-prediction. Indeed their approach has benefits such as faster computational speed (if initialized favorably) and the ability to train separate parts of the model at different time scales. Yet, the success of their scheme crucially depends on a refined understanding and analysis of the domain: how different features interact, which ones suffer cold starts, and so on. Other work in the on-line advertising setting [1, 22, 25] also explicitly models known dependencies among features (such as hierarchies) to improve the accuracy of prediction.

In this paper, we adopt the philosophical perspective that in the future, new features are bound to become periodically available in the on-line advertising world; also, correlations between features and CTRs will change with time. Rather than invest the manual effort to keep pace with this evolving environment, it makes sense to trust the algorithm with representation discovery. In this respect, the approach of constructing a manageable number of highly-conjunctive features (as leaves in a decision tree) is much more scalable than working with a large number of shallow features, or even a small number of handcrafted ones. Of course, decision trees, too, could benefit from manual intervention in feature-construction whenever possible.

Decision trees are an added boon when dealing with cold starts: if we reach a node and find no child corresponding to a feature-value, we can simply use the empirical CTR at that node as our prediction. Logistic regression has no such natural fall-back, and calls for specialized handling [2]. The well-documented tendency of logistic regression to systematically under-predict probabilities of rare events [26, 38, 42] is also a weakness in the case of CTR-prediction.

Ensemble methods, which combine predictions from a set of base predictors, usually work quite well in practice because they achieve a lower variance than individual predictors. Indeed random forests [7] outperform decision trees on a number of problems, and so it is natural to consider how they might fare for CTR-prediction. Unfortunately, when the target predictions are small probabilities, ensemble methods are prone to over-predict [28]. Since individual predictions cannot fall below 0, they cannot cancel out large over-prediction errors. Calibration of the probabilities [22, 28] can remedy this phenomenon to some extent. Gradient boosted decision trees [40] are yet another ensemble method; unfortunately boosting is a sequential process, which is not practically feasible when each tree takes a few hours to build.

We conclude that decision trees are an ideal choice for CTR-prediction tasks in on-line advertising. The main challenges they face in this relatively-new domain are the high arity of categorical features, as well as the size of the data, which necessitates large trees. In the next section, we describe our approach for addressing these challenges.

4. TREE-BUILDING WITH CLUSTERING AND CROSS-VALIDATION

In our aggregated training data, each record R is of the form $\langle \mathbf{X}, \mathbf{s}, \mathbf{t} \rangle$, where key $\mathbf{X}$ is a vector of feature-values, s is the corresponding number of successes, and t the corresponding number of tries, $t \geq s \geq 0$, $t > 0$. Our objective is to learn a predictor that given a key $\bar{\mathbf{X}}$, makes a prediction $\bar{p} \in [0, 1]$. We find it convenient to denote the key, successes, and tries corresponding to record R as $\mathbf{X}(R)$, $s(R)$, and $t(R)$, respectively.

We initialize tree-building with a root node that contains the entire training data set. Since decision trees are constructed recursively, let us consider a general node N, and let D be the set of records that have reached N. Our options are to either

1. **split** N into child nodes, partition D into corresponding sets associated with each child node, and undertake tree-building from each child node, or

2. **stop**, make N a leaf, assign the output response $N.\bar{p}$.

We describe these steps in sections 4.1 and 4.2, before summarizing the full procedure in Section 4.3.

4.1 Splitting Strategy

A splitting strategy is defined by two choices: (1) the set of splits Θ considered at N, and (2) the "goodness" function g to evaluate each split $\theta \in \Theta$. Once these choices are fixed, the natural approach is to split N based on

$$\theta^* = \operatorname*{argmax}_{\theta \in \Theta} g(\theta).$$

The main contributions of our paper are choices of Θ and g specifically intended for large decision trees with high-arity categorical features.

4.1.1 *Clustered Splits*

In canonical decision tree-building [32], exactly one split is considered for each feature that occurs in D. For feature f that takes values $v_1, v_2, \ldots, v_m$ in the records present in D, the split is of the form $v_1|v_2|\ldots|v_m$. Recall that m can be very large in our application, and that a significant number of feature-values each have only a small number of tries. Consequently a split of the form $v_1|v_2|\ldots|v_m$ will result in a large number of child nodes wherein subsequent stopping and splitting decisions will become suboptimal, owing to the lack of statistically-significant data.

We propose considering a larger set of splits Θ, with multiple "clustered" splits corresponding to each feature f. For $j \in \{1, 2, \ldots, m\}$, let D_j be the set of all records in D in which f takes the value v_j. Let S_j be the total number of successes in D_j, and T_j the total number of tries in D_j. Thus, $\frac{S_j}{T_j}$ is the *empirical* CTR of the records in D for which feature f takes value v_j. Without loss of generality, let

$$\frac{S_1}{T_1} \leq \frac{S_2}{T_2} \leq \cdots \leq \frac{S_m}{T_m}.$$

Rather than consider the full m-arity split $v_1|v_2|\ldots|v_m$, we cluster feature-values with similar CTRs to enforce a lower-arity split. For example, if we consider 2-clusterings, we would get $m-1$ splits, each with arity 2:

$$\begin{aligned} &v_1|v_2 \vee v_3 \vee \cdots \vee v_m, \\ &v_1 \vee v_2|v_3 \vee v_4 \vee \cdots \vee v_m, \\ &\vdots \\ &v_1 \vee v_2 \vee \cdots \vee v_{k-1}|v_m. \end{aligned}$$

For a fixed value of $k \in \{2, 3, \ldots, m\}$, the number of k-clusterings that respect the indexing based on CTRs is $\binom{m-1}{k-1}$. One is apt to wonder why we do not consider clusterings that do not respect the CTR-based indexing, for example

$$v_1 \vee v_3|v_2 \vee v_5 \vee v_6|v_4 \vee v_7 \vee v_8 \vee \cdots \vee v_m$$

as a 3-clustering. The reason owes to our choice of "sum squared error" (SSE) as a metric for evaluating clusterings. Concretely, a k-clustering

$$\mathcal{C} : \{1, 2, \ldots, m\} \to \{1, 2, \ldots, k\}$$

assigns each feature-value to a cluster (with at least one feature-value in each cluster). The SSE of $\mathcal{C}$ is given by

$$SSE(\mathcal{C}) \stackrel{\text{def}}{=} \sum_{i=1}^{m} t_i \left(\frac{S_i}{T_i} - \mu_{\mathcal{C}(i)} \right)^2,$$

where the mean μ_l of cluster $l \in \{1, 2, \ldots, k\}$ is given by

$$\mu_l = \frac{\sum_{i=1}^{m} \mathbf{1}[\mathcal{C}(i) = l]\, S_i}{\sum_{i=1}^{m} \mathbf{1}[\mathcal{C}(i) = l]\, T_i},$$

where $\mathbf{1}[\cdot]$ denotes the indicator function. It is easy to show that among k-clusterings that minimize SSE, there exists one that respects the CTR-index. Even so, surely it would not be computationally feasible to evaluate all $\binom{m-1}{k-1}$ possible k-clusterings that respect the CTR-indexing in order to find an optimal one! Observe that our clustering is based on CTRs, and therefore in 1-dimensional space. For the 1-dimensional case, Wang and Song [37] present a dynamic programming algorithm that can indeed find an optimal k-clustering in $O(m^2 k)$ time. We show in our experiments that the k-means++ heuristic [3], which runs in $O(mk)$ time to find an approximate solution, indeed suffices in practice, and can be run instead of dynamic programming when m is large.

SSE is indeed the most natural and commonly-used metric for evaluating clusterings [16, see Chapter 14]. Additionally, this metric enables the use of an efficient procedure for finding an optimal k-clustering of the m values taken by f.

Recall that our intent is to split N into a few dense clusters, rather than into m sparse ones. However, observe that as we increase k, the SSE of the optimal k-clustering is monotonically non-increasing: the optimal number of clusters to minimize SSE is indeed m, the arity of f! The phenomenon we are up against is *overfitting*: while a larger number of clusters will fit the *training* data better, they are likely to generalize poorly to *test* data.

We devise a goodness function that explicitly takes this aspect into account while evaluating a clustering. SSE still forms the basis of the goodness function, but a cross-validation-based approach is adopted to achieve better generalization. Indeed this method shows that a smaller number of clusters can generalize better.

4.1.2 Goodness of a Split

Consider two separate data sets, D^{train} and D^{test}, each having records R of the form $\langle \mathbf{X}, s, t\rangle$[2]. Now consider a split θ, which partitions the space of keys into q sets $\pi_1, \pi_2, \ldots, \pi_q$. For every set π_u, $u \in \{1, 2, \ldots, q\}$, consider all the records from D^{train} whose keys belong to π_u: let S_u^{train} denote the total number of successes in these records, and T_u^{train} the total number of tries in these records. We can now associate a prediction p_u^{train} with π_u, as an estimate of the CTR of records whose keys fall in π_u:

$$p^{train}(\pi_u) = \frac{S_u^{train}}{T_u^{train}}.$$

Note that π_u is a set determined solely by the split θ, and $p^{train}(\pi_u)$ is estimated solely from D^{train}. Now, for every record R, let $\pi(R)$ denote the set among $\pi_1, \pi_2, \ldots, \pi_q$ to which $\mathbf{X}(R)$ belongs. The error that results from predicting the CTR of R based on the split θ and training data D^{train} is then $\left|\frac{s(R)}{t(R)} - p^{train}(\pi(R))\right|$. In aggregate, the SSE induced by the split θ, when trained on D^{train}, but tested on D^{test}, is given by:

$$SSE(D^{train}, D^{test}, \theta) = \sum_{R \in D^{test}} t(R) \left(\frac{s(R)}{t(R)} - p^{train}(\pi(R))\right)^2.$$

It is easy to observe that among the k-clusterings of a feature f with arity m, the full-arity split will minimize $SSE(D^{train}, D^{train}, \cdot)$. However, the full-arity split will not generalize well to records in D^{test}. Our key idea is that splits corresponding to the optimal k-clustering on D^{train} for values of k smaller than m are likely to generalize better to records in D^{test}.

To concretely define the goodness of a split, we separate D, the data at the node N we are considering to split, into five folds, with each fold containing roughly the same number of successes and tries (each event line is put into a fold selected uniformly at random). We combine some four folds at a time to derive a training set, and take the remaining fold to be the test set. For every split θ, we define $g(\theta)$ to be the negative of the average value of SSE over the five resulting train-test configurations.

[2]As we see subsequently, D^{train} and D^{test} will both be drawn from the *training* data for tree-building. D^{test} is not to be confused with the data we use to test our algorithm: of course the algorithm itself does not have access to that data!

A couple of technical issues arise in the cross-validation scheme described above, owing to the features being categorical. For nodes with small numbers of tries, it is quite possible that certain feature-values will not occur in some of the folds. Thus, a split on a training-fold might not contain a feature-value that occurs in a test record. If so, we take the average CTR of D^{train} to be the prediction for the test record.[3]

A second issue to consider is that for a feature f and a fixed value of k, the optimal k-clusterings found in the five train-folds might be different. Consequently we cannot associate a cross-validated SSE (and therefore, goodness) with a split. The natural alternative is to associate a cross-validated SSE with every value of k. Precisely, the cross-validated SSE of k is the average SSE among the five train-test folds. The SSE for each fold is the corresponding *test* error on the optimal k-clustering of the corresponding *training* set.

Cross-validation is typically used *outside* a learning algorithm to tune its parameters; our novel use of this technique *within* each recursive step of decision tree-building invests the learning method with greater autonomy. The plot in Figure 3 validates our choice of cross-validated SSE in defining the goodness of splits. In this plot, we consider a feature with arity 600 in our GD data set, measuring the cross-validated SSE for all values of k between 1 and 600. The understandably high error for very small values of k obscures the "u-shaped" pattern in the curve, which we amplify in the inset. Observe that the optimal goodness is achieved by a value of $k = 41$. Owing to the randomness involved in creating folds for cross-validation, this number could vary from run to run. However, our experiments show that such variations are minor.

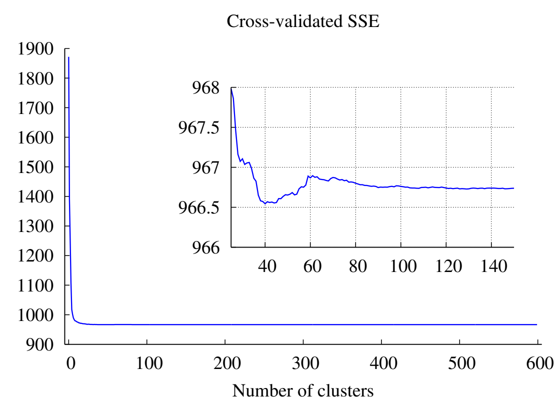

Figure 3: The cross-validated SSE (the negative of goodness) shown for different cluster sizes for a feature with arity 600 in the GD data set. For each cluster size, five train-test pairs are considered. For each pair, the optimal clustering is constructed on the training set, and the SSE of that clustering measured on the test set. The cross-validated SSE for each cluster size is the average of the five test $SSEs$.

[3]Note that cold starts are handled similarly *on-line*: when a test record R reaches a node N that is split on feature f, but the split does not contain the feature-value of f in $\mathbf{X}(R)$, we take the predicted response to be the ratio of successes to tries in N.

4.2 Stopping

We stop growing node N and make it a leaf if any one of three conditions is satisfied.

1. We do not expand N further if the number of successes in the data D reaching N is smaller than a threshold, minSuccesses. A small number of successes implies there is not enough data to justify the decision to split.

2. When we split D into folds, and find that the arity of some feature f becomes 1 in any of the training sets, then a split based on that feature would be trivial. If every feature in D leads to such a trivial split, we do not split N.

3. The conditions listed above are essentially mechanisms to handle edge cases. The main stopping condition we employ draws from the same logic driving our splitting strategy: that the value of k minimizing cross-validated SSE should picked for a split. Consider that we might well find that the cross-validated SSE is minimized by $k = 1$. Such an event is likely to occur when there are few tries in N, and the splits we find overfit the training data in the corresponding folds. A split with $k = 1$ amounts to retaining N as a single cluster.

If we stop and make N a leaf, the ratio of successes to tries in D is set as the corresponding prediction $N.\bar{p}$.

4.3 Summary of Algorithm

In this section, we tie together the splitting strategy detailed in Section 4.1 and the stopping criterion described in Section 4.2. Our tree-building algorithm, denoted **CCDT** (referring to the use of **C**lustering and **C**ross-validation in building **D**ecision **T**rees), initializes a root node with the entire training data set. The algorithm is recursive: at each node N, containing data D, the following steps are followed.

1. If the stopping criteria from Section 4.2 apply, N is made a leaf and a corresponding prediction assigned.

2. If the stopping criteria do not apply, we consider every feature that occurs in the keys of D. For each feature f, we find k^*_f, the cluster-size that yields the lowest cross-validated SSE (the highest goodness), as detailed in sections 4.1.1 and 4.1.2. We then pick a feature f^* such that the $k^*_{f^*}$-clustering of f^* has the highest goodness among all features and cluster-sizes.

3. We find the optimal $k^*_{f^*}$-clustering of f^* on D, and create a child node of N corresponding to each resulting cluster. In turn, we undertake tree-building from each child node.

We implement a distributed version of CCDT using the Map-Reduce framework [10]. Following the approach adopted by Kota and Agarwal [19, see Section 3], our procedure starts at the root node and iteratively extends the tree one level at a time.

5. EXPERIMENTAL EVALUATION

In this section, we evaluate the performance of CCDT on the NGD, GD, and Search data sets summarized in Table 1. The first two data sets are proprietary; we plan to release them in the public domain in the near future. The Search data set is a public one, from the KDD Cup 2012 (Track 2) [29]. For both the GD and the NGD data sets, recall that we have event-level data (clicks, views) which we aggregate over over the corresponding feature set. Thus we obtain successes (s) and tries (t) for every feature vector $\mathbf{X}$. For the GD data set, the training and test sets are obtained by splitting the event-level data randomly in the ratio 85:15. For the NGD data set we have event-level data for 31 consecutive days. The first 30 days' events constitute the training set and the 31^{st} day's events the test set. Recall that in this paper, we only use the 7 categorical features found in the Search data set, leaving out text-based ones. We aggregate the given train and test sets on these 7 features.

5.1 Methods Compared

We compare CCDT with four other algorithms.

DT: Our first comparison is with an existing **D**ecision **T**ree model for predicting rare-event probabilities in a large-scale system [19]. This "DT" variant uses gain ratio as the criterion to select a winning feature at each node split. A node is not split further if it reaches homogeneity or if the number of successes at the node falls below a threshold (minSuccesses). After the tree is built, pruning and Poisson-based smoothing are applied [30]. In addition, DT uses some enhancements to deal with the sparseness of feature-values, such as grouping together children with small numbers of tries based on a heuristic. A full description of this model is provided by Kota and Agarwal [19, see Section 2.4]

AC-DT: In order to observe the effect of clustering, we also compare CCDT with a **D**ecision **T**ree built on features that are **C**lustered ***a*** *priori* (AC-DT). We cluster each feature with the optimal number of clusters using the technique described in Section 4.1, where cross-validated SSE is the evaluation metric. Essentially this step amounts to preprocessing the training data and building the model using the DT variant described above (CCDT does the same processing *recursively*). Table 2 lists the arities of the features in the GD and NGD data sets before and after clustering.

DT+H: We also compare CCDT with a hybrid model recently proposed for CTR-prediction [1]. Using the prediction from **DT** as a base estimate, this hybrid model, DT+H, learns multiplicative corrections that explicitly rely on the **H**ierarchical relationships within advertiser and publisher features [1, see Section 6.2]. Thus, the DT+H model has the advantage of being provided these relationships as input, which is not available to other models in this comparison.

Table 2: Arities before/after clustering in AC-DT.

GD		NGD	
Original	**Clustered**	**Original**	**Clustered**
600	41	9651	352
21	12	8268	163
21	5	941	119
17	13	879	285
10	10	313	121
2	2	231	50
2	2	182	46
2	2	61	25
		37	11
		16	15
		4	4

LR: **L**ogistic **R**egression has been successfully used in the past in CTR-prediction tasks [2, 34]. Although it is desirable to use more higher order boolean features, we only use unary and quadratic features in our experiments (see Section 3.2), which themselves lead to collision ratios exceeding one on our data sets when using a hash table with 16 million entries. We train the model using the Vowpal Wabbit [20] software. To address the problem of class-imbalance [42], we scale down negative examples by a factor k and apply a corresponding correction during prediction [17].

The only parameter in CCDT, DT, ACDT, and DT+H, minSuccesses, is tuned separately for each model and data set. Both parameters in LR, the regularization parameter λ and the scaling factor k, are also tuned separately for each data set to minimize RMSE (defined next).

5.2 Evaluation Metrics

Multiple metrics can be used as indicators of prediction quality. To get a full picture of the performance of CCDT, we compute four relevant metrics: Log-Likelihood (LL), Root Mean Squared Error (RMSE), Area Under the ROC Curve (AUC), and the Brier Score (BS).

Whereas LL and RMSE provide aggregate measures of how well the predictions *fit* the empirical probabilities, AUC, another popular metric, measures the relative *ordering* among predictions, not their actual values [23]. BS [8] is akin to the mean squared error, but on unaggregated (event-level) data. It can further be "stratified" [36] to obtain separate metrics for the positive and negative classes. BS^+ is an especially relevant metric in this setting as the positive class is rare. AUC is measured using Fawcett's procedure [12]; the other metrics are defined below. Recall that for each record $\langle \mathbf{X_i}, \mathbf{s_i}, \mathbf{t_i} \rangle$, the predicted probability is $\bar{p_i}$.

$$LL \stackrel{\text{def}}{=} \frac{\sum_i s_i \log \bar{p_i} + (t_i - s_i) \log (1 - \bar{p_i})}{\sum_i t_i}.$$

$$RMSE \stackrel{\text{def}}{=} \sqrt{\frac{\sum_i t_i \left(\bar{p_i} - \frac{s_i}{t_i}\right)^2}{\sum_i t_i}}.$$

$$BS \stackrel{\text{def}}{=} \frac{\sum_i s_i (1 - \bar{p_i})^2 + (t_i - s_i) \bar{p_i}^2}{\sum_i t_i};$$

$$BS^+ \stackrel{\text{def}}{=} \frac{\sum_i s_i (1 - \bar{p_i})^2}{\sum_i s_i}; \quad BS^- \stackrel{\text{def}}{=} \frac{\sum_i (t_i - s_i) \bar{p_i}^2}{\sum_i (t_i - s_i)}.$$

5.3 Results

Tables 3 and 4 list the errors observed for each of the models on the NGD and GD data sets, respectively (Tables 5 and 6 present the percentage lifts over DT).[4] Since the GD data set does not have any hierarchical information, we do not run DT+H on this data set. We use DT as our baseline and make all comparisons with respect to it.

[4]Note that since our data is drawn from Bernoulli trials, for a given t_i, the empirical CTR s_i/t_i is likely more noisy when s_i is small. In our NGD data set, indeed we notice that s_i/(number of keys) is very small, in addition to the mean CTR itself being small, which leads to unreliable estimates of empirical CTR. To circumvent this problem, we aggregate the keys, along with predicted and empirical probabilities, over a smaller set of features. We find no need to do so in the other data sets.

Recall that our optimal clustering algorithm using dynamic programming takes $O(m^2k)$ time at each node; we also test a version of CCDT, denoted CCDT-K, that uses the popular kmeans++ routine for clustering in $O(mk)$ time. We observe from Table 6 that this approximation does not result in a significant loss of accuracy. On the GD data set, CCDT-K takes less than 4 hours to complete on our cluster, whereas CCDT takes over 12 hours.

On both the GD and NGD data sets, it is seen that ACDT indeed improves upon DT on all the metrics as hypothesized. Thus, clustering each feature, even once as a preprocessing step (on the entire training data), is beneficial. On both data sets, we observe that CCDT improves all the metrics further by applying the same principle at each node. Moreover, CCDT has significantly better LL and RMSE. With respect to the AUC metric, a marginal improvement is seen on the NGD data set, while a significant improvement is seen in the GD data set. On both data sets, CCDT performs better than LR on all the metrics listed, affirming our hypothesis from Section 3.2. Observe that on both data sets, the decision tree-based algorithms consistently outperform LR on the BS^+ metric, which measures the accuracy of predictions on the rarer class.

On the Search data set, we compare CCDT-K and LR with the DT baseline; Table 7 shows the lifts in metrics with respect to the baseline. On all the metrics, CCDT outperforms LR. On this data set, we do not see a clear winner across all metrics when comparing CCDT and DT.

As a part of further analysis, we consider the trees built by our algorithms. We find that CCDT and its variants not only perform better on the metrics listed, but they also build relatively compact trees. Figure 4 allows us to visualize the distribution of tries across nodes, in trees built by CCDT and DT on the NGD data set. Most starkly apparent is that the size of the tree learned by CCDT (19,380 nodes) is much smaller than that of DT (157,269 nodes). Naturally, CCDT bears the overhead of maintaining cluster indices in each node; even so, the overall memory footprint of the CCDT tree is only 5.5MB, compared to the 13MB taken up by the DT model. Thus, even with a more compact representation, CCDT is able to learn a more accurate predictor.

In summary, our experiments affirm the intuition that clustering and cross-validation both contribute to building more compact yet predictive decision trees. CCDT emerges a clear winner when compared with the other models.

6. DISCUSSION AND FUTURE WORK

The central ideas investigated in this paper—clustering and cross-validation—are both well-studied topics in machine learning. The novelty of our contribution is in applying them appropriately for building large decision trees with high-arity categorical features. Historically, efforts in "scaling up" [31] mostly focused on faster algorithms to process large amounts of data. With the paradigm shifting to parallel computing on the cloud, our emphasis is on deriving more *accurate* models. The cross-validation step in CCDT is trivially parallelizable; the use of kmeans++ for clustering involves an $O(mk)$ operation, rather than the $O(m)$ of canonical tree-building. Given the significant gains in prediction accuracy, we consider this additional computational overhead well-justified.

Our approach of undertaking cross-validation at each node is similar to an idea proposed by Blockeel and Struyf [6].

Table 3: Results from NGD data set.

	LL	RMSE	AUC	BS	BS$^+$	BS$^-$
DT	-0.069333	0.006286	0.941650	0.002004	0.187834	**0.000270**
DT+H	-0.063597	0.005400	0.957200	0.001963	0.182713	0.000276
LR	-0.063791	0.005416	0.956800	0.001964	0.182603	0.000278
AC-DT	-0.063203	0.005720	0.957520	0.001977	0.181778	0.000299
CCDT	**-0.062816**	**0.005055**	**0.957890**	**0.001948**	**0.179261**	0.000294

Table 4: Results from GD data set.

	LL	RMSE	AUC	BS	BS$^+$	BS$^-$
DT	-0.018030	0.000808	0.716500	0.0003120	0.247605	0.00000148
LR	-0.018411	0.000880	0.717421	0.0003124	0.248277	**0.00000112**
AC-DT	-0.017098	0.000649	0.808800	0.0003110	**0.245664**	0.00000229
CCDT-K	**-0.016940**	0.000571	0.807700	0.0003107	0.245771	0.00000247
CCDT	-0.017077	**0.000566**	**0.810280**	**0.0003106**	0.245727	0.00000251

Table 5: % lifts in NGD data set over DT.

	LL	RMSE	AUC	BS
DT+H	-8.27%	-14.08%	1.65%	-2.06%
LR	-7.99%	-13.84%	1.61%	-2.03%
AC-DT	-8.84%	-8.99%	1.69%	-1.35%
CCDT	-9.40%	-19.57%	1.72%	-2.78%

Table 6: % lifts in GD data set over DT.

	LL	RMSE	AUC	BS
LR	2.11%	8.91%	0.13%	0.16%
AC-DT	-5.17%	-19.71%	12.88%	-0.30%
CCDT-K	-6.05%	-29.29%	12.73%	-0.42%
CCDT	-5.29%	-29.96%	13.09%	-0.43%

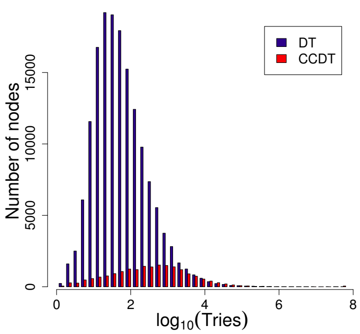

Figure 4: Histograms of nodes having similar number of tries, in the trees learned by CCDT and DT on the NGD data set.

However, whereas their aim is to optimize the computational speed of traditional cross-validation, in our algorithm, cross-validation actually decides whether to split each node, and if so, how. Note that cross-validation is also used in other supervised learning methods, such as neural networks (for adding hidden nodes) [35] and regression [14] (for tuning the regularization coefficient). Our approach of using stratified cross-validation (wherein the folds have roughly-equal numbers of successes and tries) is a reasonable first-cut [18]. When the number of tries is small, it would be worth investigating whether approaches such as bootstrapping [11] (sampling with replacement) would work better.

In the context of decision trees, clustering has been applied to keys that share real-valued features with similar values (that is, in the *input*) [13]. By contrast, the feature-values of categorical features are incomparable, prompting us to cluster them based on their labels (that is, in the *output*). In principle, the ideas presented in this paper can also be applied to supervised learning problems wherein the

Table 7: % lifts in Search data set over DT.

	LL	RMSE	AUC	BS
LR	1.84%	2.72%	-2.68%	0.89%
CCDT-K	-1.62%	1.46%	-0.76%	0.48%

output predictions are categorical or real-valued. We plan to extend our algorithm to such problems in future work. It would also be practical to speed up our Hadoop-based implementation further, using newer frameworks such as Spark [41].

7. CONCLUSION

On-line advertising has grown to become a multi-billion dollar industry, wherein large numbers of users, publishers, and advertisers interact every day through automated channels. In this paper, we consider a key scientific problem that determines the health and profitability of advertising systems: the problem of predicting click-through rates and related probabilities. The input for this supervised learning problem consists of large volumes of data predominantly represented using high-arity categorical features. Solutions need to address these and other challenges such as unknown feature-values, cold starts, sparse training instances, and rare-event probability estimation.

We argue that decision trees are an ideal choice to address problems in on-line advertising, as they naturally handle categorical features, and automatically extract discriminative higher-order conjunctions. We equip decision trees to handle the high arities found in advertising data by employing an efficient procedure to cluster feature-values based on their output responses. We also build cross-validation into the splitting and stopping criteria at each node, rather than using the technique to tune model-building parameters in aggregate. Evaluated on three distinct data sets drawn from the on-line advertising domain, our CCDT algorithm outperforms existing decision tree algorithms as well as logistic regression, significantly reducing the prediction error and the size of the trees constructed. In future work, we plan to adapt CCDT to work on classification and regression tasks, and further optimize its parallelized implementation.

Acknowledgments

The authors thank Nagaraj Kota for sharing the DT and DT+H code, Aravindan Raghuveer for providing the GD data set, and Edo Liberty for offering several useful comments. Early stages of this work also benefited from discussions with Sourangshu Bhattacharya and Sanjay Chawla.

8. 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 *Proc. KDD 2010*, pages 213–222. ACM, 2010.

[2] D. Agarwal, B. Long, J. Traupman, D. Xin, and L. Zhang. LASER: a scalable response prediction platform for online advertising. In *Proc. WSDM 2014*, pages 173–182. ACM, 2014.

[3] D. Arthur and S. Vassilvitskii. k-means++: the advantages of careful seeding. In *Proc. SODA 2007*, pages 1027–1035. SIAM, 2007.

[4] V. Bharadwaj, W. Ma, M. Schwarz, J. Shanmugasundaram, E. Vee, J. Xie, and J. Yang. Pricing guaranteed contracts in online display advertising. In *Proc. CIKM 2010*, pages 399–408. ACM, 2010.

[5] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[6] H. Blockeel and J. Struyf. Efficient algorithms for decision tree cross-validation. *Journal of Machine Learning Research*, 3:621–650, 2002.

[7] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[8] G. W. Brier. Verification of forecasts expressed in terms of probability. *Monthly weather review*, 78(1):1–3, 1950.

[9] A. Broder. Computational advertising. In *Proc. SODA 2008*, page 992. SIAM, 2008.

[10] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[11] B. Efron and R. J. Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall/CRC, 1993.

[12] T. Fawcett. ROC graphs: Notes and practical considerations for researchers. Technical report, HP Laboratories, 2004. Retrieved from: `http://home.comcast.net/~tom.fawcett/public_html/papers/ROC101.pdf`.

[13] U. M. Fayyad and K. B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proc. IJCAI 1993*, pages 1022–1029. Morgan Kaufmann, 1993.

[14] G. H. Golub, M. Heath, and G. Wahba. Generalized cross-validation as a method for choosing a good ridge parameter. *Technometrics*, 21(2):215–223, 1979.

[15] T. Graepel, J. Quiñonero Candela, T. Borchert, and R. Herbrich. Web-scale Bayesian click-through rate prediction for sponsored search advertising in Microsoft's Bing search engine. In *Proc. ICML 2010*, pages 13–20. Omnipress, 2010.

[16] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2$^{\text{nd}}$ edition, 2009.

[17] G. King and L. Zeng. Logistic regression in rare events data. *Political Analysis*, 9:137–163, 2001.

[18] R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proc. IJCAI 1995*, pages 1137–1145. Morgan Kaufmann, 1995.

[19] N. Kota and D. Agarwal. Temporal multi-hierarchy smoothing for estimating rates of rare events. In *Proc. KDD 2011*, pages 1361–1369. ACM, 2011.

[20] J. Langford, L. Li, and A. Strehl. Vowpal Wabbit online learning project. Technical report, 2007. `http://hunch.net/?p=309`.

[21] P. Langley. Machine learning as an experimental science. *Machine Learning*, 3(1):5–8, 1988.

[22] K.-c. Lee, B. Orten, A. Dasdan, and W. Li. Estimating conversion rate in display advertising from past performance data. In *Proc. KDD 2012*, pages 768–776. ACM, 2012.

[23] J. M. Lobo, A. Jiménez-Valverde, and R. Real. AUC: a misleading measure of the performance of predictive distribution models. *Journal of Global Ecology and Biogeography*, 17(2):145–151, 2007.

[24] R. P. McAfee. The design of advertising exchanges. *Review of Industrial Organization*, 39(3):169–185, 2011.

[25] A. K. Menon, K. P. Chitrapura, S. Garg, D. Agarwal, and N. Kota. Response prediction using collaborative filtering with hierarchies and side-information. In *Proc. KDD 2011*, pages 141–149. ACM, 2011.

[26] A. K. Menon, X. Jiang, S. Vembu, C. Elkan, and L. Ohno-Machado. Predicting accurate probabilities with a ranking loss. In *Proc. ICML 2012*, pages 703–710. Omnipress, 2012.

[27] S. K. Murthy. Automatic construction of decision trees from data: A multi-disciplinary survey. *Data Mining and Knowledge Discovery*, 2(4):345–389, 1998.

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

[29] Y. Niu, Y. Wang, G. Sun, A. Yue, B. Dalessandro, C. Perlich, and B. Hamner. The Tencent Dataset and KDD-Cup'12. In *KDD-Cup Workshop*. ACM, 2012.

[30] F. J. Provost and P. Domingos. Tree induction for probability-based ranking. *Machine Learning*, 52(3):199–215, 2003.

[31] F. J. Provost and V. Kolluri. A survey of methods for scaling up inductive algorithms. *Data Mining and Knowledge Discovery*, 3(2):131–169, 1999.

[32] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[33] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1992.

[34] R. Rosales, H. Cheng, and E. Manavoglu. Post-click conversion modeling and analysis for non-guaranteed delivery display advertising. In *Proc. WSDM 2012*, pages 293–302. ACM, 2012.

[35] R. Setiono. Feedforward neural network construction using cross validation. *Neural Computation*, 13(12):2865–2877, 2001.

[36] B. C. Wallace and I. J. Dahabreh. Class probability estimates are unreliable for imbalanced data (and how to fix them). In *Proc. ICDM 2012*, pages 695–704. IEEE, 2012.

[37] H. Wang and M. Song. Ckmeans.1d.dp: Optimal k-means clustering in one dimension by dynamic programming. *The R Journal*, 3(2):29–33, December 2011.

[38] X. Wang and D. K. Dey. Generalized extreme value regression for binary response data: An application to B2B electronic payments system adoption. *Annals of Applied Statistics*, 4(4):2000–2023, 2010.

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

[40] J. Ye, J.-H. Chow, J. Chen, and Z. Zheng. Stochastic gradient boosted distributed decision trees. In *Proc. CIKM 2009*, pages 2061–2064. ACM, 2009.

[41] M. Zaharia, M. Chowdhury, S. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. In *Proceedings of HotCloud 2010*, 2010.

[42] T. Zhang. Statistical behavior and consistency of classification methods based on convex risk minimization. *Annals of Statistics*, 32(1):56–134, 2004.