University of California

Los Angeles

Modern Models for Learning Large-Scale Highly Skewed Online Advertising Data

A thesis submitted in partial satisfaction

of the requirements for the degree

Master of Science in Statistics

by

Qiang Zhang

2015

Abstract of the Thesis

Modern Models for Learning Large-Scale Highly Skewed Online Advertising Data

by

Qiang Zhang

Master of Science in Statistics

University of California, Los Angeles, 2015

Professor Ying Nian Wu, Chair

Click through rate (CTR) and conversation rate estimation are two core prediction tasks in online advertising. However, four major challenges emerged as data scientists trying to analyze the advertising data - sheer volume, the amount of data available for mining is massive; complex structure, there is no easy way to tell what factors drive a user to click an ad or make a conversion and how the factors interacted with one another; high cardinality for categorical variables, features like device id usually have tons of possible values which will lead to very sparse data; severe skewness in response variable with the majority of the users not clicking the ad. In this paper, I will make a comprehensive summary of the state-of-art machine learning models (decision tree based, regularized logistic regression, online learning, and factorization machine) that are often used in the industry to solve the problem. Insights and practical tricks are then provided based on a wide range of experiments conducted on multiple data sets with different characteristics.

The thesis of Qiang Zhang is approved.

Nicolas Christou

Hongquan Xu

Ying Nian Wu, Committee Chair

University of California, Los Angeles

2015

To my mother . . .

who—among so many other things—

saw to it that I learned to touch-type

while I was still in elementary school

Table of Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Online advertising, also called online marketing, is a form of marketing and advertising which uses the Internet to deliver marketing messages to consumers. Display, search engine, social media marketing, email and recently mobile are the most common channels for online advertising. It involves both a publisher (usually the website owner) and an advertiser (who wants to sell a product). The advertiser provides the advertisement for the publisher to integrate into its online content to get impressions.

Various payment options are provided by publishers to suit the needs of different advertiser. Traditionally most popular payment option is called cost per impression (CPI) or cost per mille (CPM), in which advertiser pays each time an ad is displayed. It would be an appropriate choice for the advertiser if its goal is to get a message delivered to the target audience, for example to build brand awareness. For most advertisers, however, paying for impression are not cost effective as more impressions do not necessarily generate more revenue. Accordingly, performance based payment options emerge and become more and more popular in the industry. Among them, cost per click (CPC) and cost per action (CPA) are the two most popular options. Under the CPC framework, advertisers only have to pay when a customer clicked on their ads, whereas CPA gives advertisers even more assurance by allowing them to pay only if the customers take a predefined action on their website for example, a registration, a form submission, or even a purchase and as a result is often considered as the optimal way to buy online

advertising for advertisers.

An online bidding system that incorporates CPC and CPA paying options needs to calculate the expected price for impression when deciding which ad to display, which depends on the probability of the impression being clicked or leading to a specified action. Estimating these probabilities accurately is crucial for an efficient marketplace [1].

Machine learning algorithms, especially classification and probability estimation models have long been used to predict the probabilities a certain user clicking on a candidate ad and also the propensity of a conversion. Among them, decision tree, logistic regression, random forest, gradient boosted machine are used most often. However, unlike other simple classification problems, estimating CTR and conversion rate from a large-scale digital advertising data is not an easy task as four major challenges are usually faced by modelers who tries to mine the data and apply machine learning algorithms, and thus requires more customized models to solve the problem.

In Chapter 2, the major challenges in encountered modeling digital advertising data are illustrated. Chapter 3 introduces main stream and state-of-art models used to predict CTR and conversion rate in the industry. Chapter 4 presents the experiment setups including the data sets along with the software implementations and parameter tuning methods that are used in the experiments. A wide range of experiments are conducted in Chapter 5, based on which insights and suggestions in mining online advertising data will be provided. Conclusion and future work and are covered in Chapter 6.

CHAPTER 2

Challenges

2.1 Enormous Size

Given the massive global growth of the Internet and the ever-increasing development of the technology, new data becomes available at an astonishing rate. In 2011, EMC pointed out that the worlds information is doubling every two years. According to the U.S. Chamber of Commerce Foundation, 90 percent of the world's data has been produced in the last two years by 2013. At Facebook, they have over 750 million daily active users and over 1 million active advertisers [2]. Consequently, the sheer volume of data itself makes predicting clicks and conversions very challenging tasks, and scalability becomes the key.

2.2 Complex Problem

The factors that influence a user to click on a banner ad or make a conversion at a website may come from various aspects and vary person to person, site to site and time to time. And what makes the problem even harder is that all different factors not solely affect users behavior in their own way but instead may interact with each other. Therefore, to accurately predict clicks and conversions, the models used must be able to capture the complex structures in the data, which can either be higher order interaction effects between features or other historical information hidden in the data.

2.3 High Cardinality

In machine learning, categorical features and numerical features are usually treated differently, and for digital advertising data, categorical variables tend to dominate. In predicting conversion rate, for instance, features like the users device, operating system, browser, most of the demographic features such as city, country, language, and the website or the mobile app where the user is directed from are all categorical. However, what makes the problem annoying is not the categorical property, but the high cardinality that comes from those features. For example, the site name is always the case as an ad is always displayed in a wide range of sites to reach as much potential customers. High cardinality also naturally happens for identification variables, like device id, or user id for registered users. Essentially, high cardinality in categorical variables leads to a very sparse input feature space.

2.4 Severe Skewness

For most advertising channels and platforms, the average CTR or conversion rate is very low. According to Sizmek's annual Benchmarks Report, in North America, the average click through rate (CTR) for standard banners is 0.08% in 2013. In terms of machine learning, this indicates an imbalanced learning problem in which the class distribution is highly skewed. As most classification learning algorithms assume or expect balanced class distributions or equal misclassification costs, the imbalanced learning problem is concerned with the performance in the presence of underrepresented data and severe class distribution skewness.

CHAPTER 3

Models

In this section, four families of machine learning models that are potential to address the above challenges are introduced. The first big family are decision tree based models such as single decision tree, bagging, random forest and gradient boosting machines. Then logistic regression, especially regularized logistic regression is introduced, followed by the online learning algorithms including a specific algorithm of online regularized logistic regression (FTRL-Proximal). Last, a new model class called factorization machine is presented.

3.1 Decision Tree Based Models

3.1.1 Single Decision Tree

As a non-parametric supervised learning method, decision tree tries to construct a set of decision rules from the data to predict the target variable. It can be used either for classification or regression. Though decision trees are accused to be inaccurate [3] and unstable [4] and not robust to imbalanced class distributions, the flexibility in handling both categorical and numerical data and its ability to capture higher order interaction effects make them a good base learner in ensemble models. Two families of tree based ensemble models are mainly employed in large scale classification problems bagging (random forest) and boosting trees (AdaBoost and gradient tree boosting). They differ as to how to build the trees and how the results of single decision trees are combined, and each has its own

advantages.

3.1.2 Bagging

The first family of tree-based ensemble model, known as bootstrap aggregating or bagging, builds multiple trees independently on bootstrap samples of original training set and then combines the results of single trees by taking the average. These methods are used as a way to reduce the variance of a single decision tree by introducing randomization into its construction procedure and is able to give substantial improvement of performance over unstable learners [5].

3.1.3 Random Forest

When individual trees are similar, which unfortunately happens very often for very large data sets and balanced class distribution, bagging may not be able to give a boost. Random forest, as a special version of bagging, provide an improvement over bagged trees by way of a small tweak that decorrelates the tree. When growing the trees, not all features are considered at each split. Instead, only a random subset of features are used in the tree splitting. Such randomness introduced in the tree building process always give dissimilar trees which help further reduce the variance of the estimate.

Usually, both bagging and random forest work best with complex if not fully developed trees. For balanced data, we can keep expanding each single tree without too much concern as the averaging step will cancel out the extremes; for imbalanced data, however, a second thought is necessary when considering fully growing a single tree as the probability estimate from a few or even a single sample may not be able to truly represent the probability in a imbalanced data set. A recommended practice is to specify the minimum leaf size, say 500, for each decision tree so that the probability estimates from 500 observations are more

reliable and robust.

As all the trees in random forest (also bagging) are independent from one another, it is possible to take advantage of parallelization in the modeling process as a way to reduce run time and improve efficiency. In Python, for example, the *scikit-learn* package provides such functionality in the random forest modules to build trees in parallel on multiple cores on a single machine. If a cluster of machines are available, one can even further reduce the run time.

Whats more, random forest are more robust to the noise in the data compared to Adaboost [6] which will be metioned in next subsection.

3.1.4 AdaBoost

AdaBoost (short for Adaptive Boosting), introduced by Freund and Schapire [7] in 1995 , is a different category of ensemble decision tree models as the trees are built in a sequential order and the final predictor is a weighted sum of each individual tree. The motivation is to combine several week learner to produce a powerful committee [3]. The key idea of AdaBoost is applying weights to each training sample and the weights are updated at each boosting iteration based on the prediction accuracy of that specific sample in the previous iteration. To be specific, in the first iteration, all samples are assigned equal weight $1/n$ where n is the total number of observation, so the first step simply trains a simple learner on the raw data. For successive iterations, those instances that are incorrectly predicted by the boosted model generated at previous step receive an increased weight while those correctly predicted ones have their weights decreased. As iterations proceed, instances that are difficult to predict will receive ever-increasing weights so that the subsequent learners will focus on those observations. The predictions from all of them are then combined through a weighted sum to produce

the final prediction:

$$F(x) = sign\left(\sum_{j=1}^{M} \alpha_j F_j(x)\right) \tag{3.1}$$

where $\alpha_1, \alpha_2, \cdots, \alpha_M$, are computed based on the accuracy of each respective tree $F_j(x)$ so that more accurate trees will have a higher influence on the final model.

Unlike random forest where trees are built independently, AdaBoost constructs trees in a sequential order and consequently they cannot be parallelized. In spite of this, the efficiency of AdaBoost algorithm is still comparable to random forest as in most situations, the trees in a boosting algorithm are very shallow if not decision stamps, which means the effort we spend on building each single tree is not as much as we spend on building trees in random forest. Another point worth mentioning is that the depth of a tree defines the level of variable interactions that can be captured by the boosting model. In general, a tree of depth h can capture interactions of order h at most. This ensures that the boosting model is still able to capture the complex structure of the data by modeling the interaction effects.

3.1.5 Gradient Tree Boosting

Though in many applications AdaBoost can improve the accuracy of decision trees dramatically, its robustness against overlapping class distributions (a very common characteristic of most click through rate, conversion rate data) and especially mislabeling of the training data become its major drawback [8]. A gradient boosting model is a generalization of boosting to arbitrary differentiable loss function that attempts to mitigate these problems. In the application of CTR or conversion rate prediction, gradient tree boosting, a specific gradient boosting model with a logit loss is often chosen and often yields promising results.

To understand gradient tree boosting, we can simply learn the general form of gradient boosting machine with an arbitrary loss function.

Let's denote the training set as $\{(x_1, y_1), \cdots, (x_n, y_n)\}$, we want to find a model $G^*(x)$ that minimizes the loss function $L(y, F(x))$:

$$F^* = \arg\max_F \mathrm{E}_{x,y}\left[L(y, F(x))\right] \tag{3.2}$$

As In practice, only training data is available, we usually find $\hat{F}(x)$ that minimizes the average loss function over the training set.

$$\hat{F}(x) = \arg\max_F \sum_{i=1}^{n} L(y_i, F(x_i)) \tag{3.3}$$

Gradient boosting method fits $\hat{F}(x)$ as an additive expansion (weighted sum) of a set of base learners $h_i(x)$ from a certain class $\mathscr{H}$. For gradient tree boosting, $h_i(x)$ is constrained to be a single decision tree.

$$F(x) = \sum_{j=1}^{M} \beta_j h_j(x) \tag{3.4}$$

To find $\hat{F}(x)$, the algorithms usually starts with a constant function $F_0(x)$, and incrementally expands it in a greedy manner:

$$F_m(x) = F_{m-1}(x) + \arg\max_{f \in \mathscr{H}} \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i)) \tag{3.5}$$

However, choosing the best F for an arbitrary loss function at each iteration is not very easy. Friedman, J. H. proposed to solve a much easier problem instead to find F [9]. The idea is apply the concept of steepest descent where in each iteration, we simply find an F that most closely approximates the negative gradient of the loss function L, which is further derived as a so-called pseudo-residual. Having chosen F, the multiplier γ can be selected using line search. In this way we can find a weighted sum of base classifiers (decision trees for gradient tree boosting) as a final model for prediction.

Since gradient tree boosting is also constructed sequentially like AdaBoost, it shares the advantages of AdaBoost in dealing with online advertising data as mentioned above. And on top of that, we also pointed out that gradient tree

boosting is more robust to overlapping class distributions and mislabeling of class membership in the training data.

Now we can see that ensemble of decision trees provide solutions for dealing with the complex structure of the online advertising data and are able to give accurate prediction even when the class distributions is highly skewed. However, a major concern for decision tree based model is its ability to handle high cardinality in categorical variables. On one hand, for a categorical feature with N values there are $2^N - 2$ possible decision rules on the feature, which is too many to consider by a long way.

For software implementations, both R and Python provide packages for decision tree, bagging, random forest, Adaboost and gradient boosting trees. R packages usually takes input data in the data.frame format, which can be thought as a spreadsheet like table with raw data (both numeric and categorical) while the *scikit-learn* package in Python requires categorical features encoded, such as as dummy variables, before being fed to the algorithm. As both languages requires reading data into memory before running the algorithm, for large scale high dimensional data, the memory efficiency becomes a concern. A graduate student at University of Washington introduced an optimized general purpose gradient boosting library under Apache license called xgboost which takes in sparse data format and utilizes parallelization in cross validation to train the classifier efficiently. Xgboost receives wide recognition among machine learning community due to its efficiency and ease in implementation. For the experiments in the paper, I will use all three software for different tasks.

3.2 Regularized Logistic Regression

Logistic regression (LR), as a special case of generalized linear model with a logit link function, is a very different binary or multinomial classifier than decision tree

based models. Like other forms of regression analysis, logistic regression makes use of one or more predictor variables that may be either continuous or categorical data. Unlike ordinary linear regression, however, logistic regression maps the linear combination of the input features into a probability via a logistic function. A detailed introduction to logistic regression can be found on any statistics textbook so I will skip it here.

The biggest advantage of logistic regression over tree based models is its ability to handle high cardinality in categorical features elegantly by assuming independence between input features and estimating their effects (coefficients) simultaneously. What's more, as a optimization problem, implementations of logistic regression are usually faster than those tree based models.

However, standard logistic regression tends to fail in the context of high cardinarlity categorical data (sparse data) as it is prone to overfit the training data with too much irrelevant information presents and thus generalizes poorly. To solve this, regularized logistic regression turns out to be a natural fit. Like any other regularized linear models, regularized LR controls the complexity of the model by explicitly introducing penalty terms in the objective (likelihood) function which we want to optimize.

There are different penalty terms that can be applied, with different properties. The most commonly used ones are L2 and L1 norm of the weights or a combination of both, with which the procedures associated are known as ridge regression, LASSO and elastic net respectively. Ridge regression shrinks the regression coefficients by imposing a penalty on their size such that unimportant variables will end up with weights very close to zero while the coefficients for informative variables are shrunk by only a very small portion [3]. LASSO, which stands for Least Absolute Shrinkage and Selection Operator [10], not only shrinks the regression coefficients (in a slightly different way though), but performs variable selection at the same time, which means the model will end up with some

of the parameters estimates being exactly zero. And the elastic net, again, is a combination of both. The objective functions for the three regularized logistic regressions are listed below.

$$\boldsymbol{\beta}^{LASSO} = \underset{\beta_0,\boldsymbol{\beta}}{max} \left\{ \sum_{i=1}^{n} \left[y_i \left(\beta_0 + \boldsymbol{\beta}^T \boldsymbol{x}_i \right) - \log \left(1 + e^{\beta_0 + \boldsymbol{\beta}^T \boldsymbol{x}_i} \right) \right] - \lambda \sum_{j=1}^{p} |\beta_j| \right\} \quad (3.6a)$$

$$\boldsymbol{\beta}^{ridge} = \underset{\beta_0,\boldsymbol{\beta}}{max} \left\{ \sum_{i=1}^{n} \left[y_i \left(\beta_0 + \boldsymbol{\beta}^T \boldsymbol{x}_i \right) - \log \left(1 + e^{\beta_0 + \boldsymbol{\beta}^T \boldsymbol{x}_i} \right) \right] - \lambda \sum_{j=1}^{p} |\beta_j|^2 \right\} \quad (3.6b)$$

$$\begin{aligned} \boldsymbol{\beta}^{elastic} = \underset{\beta_0,\boldsymbol{\beta}}{max} \Bigg\{ & \sum_{i=1}^{n} \left[y_i \left(\beta_0 + \boldsymbol{\beta}^T \boldsymbol{x}_i \right) - \log \left(1 + e^{\beta_0 + \boldsymbol{\beta}^T \boldsymbol{x}_i} \right) \right] - \lambda_1 \sum_{j=1}^{p} |\beta_j| \\ & - \lambda_2 \sum_{j=1}^{p} |\beta_j|^2 \Bigg\} \end{aligned} \quad (3.6c)$$

A efficient algorithms for L1-regularized regression called OW-LQN, which scales to millions of rows is proposed by by Galen Andrew, Jiafeng Guo [11]. But due to the lack of an handy implementation, only L2-regularized LR is experimented in this paper. A scalable implementation of L2-regularized logistic regression can be found in LIBLINEAR [12], which is a popular open source machine learning library developed at the National Taiwan University written in C++. It takes input data in a sparse format, which is memory friendly, and optimizes the objective function using a coordinate descent algorithm [10].

Both OW-LQN and LIBLINEAR are scalable to millions of rows of data, which is enough for many small to mid size online marketing analytics tasks. But as we mentioned in Chapter 2 Section 3 that for some large-scale online advertising, especially click through rate prediction, the volume of data to mine is huge, even billions or records a day. This would pose a big challenge for many machine learning algorithms, let alone the above two. Such big volume of data, plus the high dimensionality (sparsity) nature, will be very expensive to mine and analyze as a whole. As a result, online learning or stochastic gradient descent algorithms seems a better fit under such circumstances.

3.3 Online Learning Algorithms

3.3.1 Stochastic Gradient Descent

Consider the problem of minimizing an objective function that has the following form:

$$F(\boldsymbol{\beta}) = \sum_{i=1}^{n} F_i(\boldsymbol{\beta}), \tag{3.7}$$

where $\boldsymbol{\beta}$ is the parameter to be estimated and each F_i is only associated with the i-th observation in the training data set. For instance, in terms of optimizing logistic regression models, the F_i has the form:

$$F_i(\beta_0, \boldsymbol{\beta}) = y_i \left(\beta_0 + \boldsymbol{\beta}^T \boldsymbol{x}_i\right) - \log\left(1 + e^{\beta_0 + \boldsymbol{\beta}^T \boldsymbol{x}_i}\right) \tag{3.8}$$

To minimize $F(\boldsymbol{\beta})$, a standard gradient descent method would have an update function as follows:

$$\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \lambda \nabla F(\boldsymbol{\beta}) = \boldsymbol{\beta} - \lambda \sum_{i=1}^{n} \nabla F_i(\boldsymbol{\beta}), \tag{3.9}$$

where λ is a step size (or learning rate in machine learning terms).

In many cases, each $F_i(\boldsymbol{\beta})$ has a simple form and the number of observations is not too big which enables inexpensive evaluation of sum function and sum gradient. In other cases, however, evaluating the sum-gradient may require expensive computation when the set it enormous and has no simple formulas. To avoid expensive computation at every step, stochastic gradient descent uses the gradient at a single instance to approximate the true gradient of $F(\boldsymbol{\beta})$. So the update function of $\boldsymbol{\beta}$ can be written as:

$$\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \lambda \nabla F_i(\boldsymbol{\beta}), \tag{3.10}$$

The algorithm goes through the training data and performs the update for each instance until it converges. It may take more than one pass (some times referred to as epoch) of data before convergence and it is usually suggested to shuffle the

training instances before each pass to prevent any periodic trend in the data. A decaying step size (learning rate) is typically used to ensure convergence.

To get back to the problems we want to solve in this paper, we can use a online (stochastic) version of regularized logistic regression to deal with the enormous and sparse data sets. An implementation of stochastic gradient descent can be found in the *scikit-learn* package in Python which supports both hinge loss for support vector machine and log loss for logistic regression along with L1 and L2 regularization.

3.3.2 Follow-The-Regularized-Leader (FTRL-Proximal)

Another algorithm for online regularized logistic regression is called follow-the-regularized-leader (FTRL or FTRL-Proximal). It implements the same idea as stochastic gradient descent which is to approximate the gradient of the loss function (with regularization) using the gradient at a single instance but uses a per-coordinate learning rate with a few memory saving tricks. H. Brendan McMahan states that FTRL-Proximal algorithm is able to produce sparser models than stochastic gradient descent and has excellent convergence properties too [13]. The algorithm is only lines of code and the pseudo-code can be found in the paper.

3.4 Factorization Machine

Recall the two major families of models that have been introduced. Decision tree based models are able to capture complex structures in the data i.e. the higher order interaction effects but does not work well with sparse data; while logistic regression related approaches are able to handle sparsity, but higher order interactions has to be manually selected and crafted if we want to capture more complex structures of the data.

In 2010, Steffen Rendle introduced a new model class call Factorization Ma-

chine that combines the advantages of Support Vector Machines (SVM) with factorization models [14]. FM models all interaction between variables using factorized parameters and thus is able to estimate interactions even in problems with huge sparsity.

The model equiration for a factorization machine of degree $d = 2$ is defined as:

$$y(x) = \beta_0 + \sum_{i=1}^{p} \beta_i x_i + \sum_{i=1}^{p} \sum_{j=i+1}^{p} \langle \boldsymbol{v}_i, \boldsymbol{v}_j \rangle x_i x_j \tag{3.11}$$

where y is the link function which, for classification, can be viewed as the logit function. β_0 is the global bias, β_i is the main effect of the i-th feature, and

$$\boldsymbol{V} = \begin{pmatrix} \boldsymbol{v}_1 \\ \boldsymbol{v}_2 \\ \vdots \\ \boldsymbol{v}_p \end{pmatrix}, \qquad \langle \boldsymbol{v}_i, \boldsymbol{v}_j \rangle = \sum_{j=1}^{k} v_{i,f} \cdot v_{j,f} \stackrel{\triangle}{=} w_{ij} \tag{3.12}$$

where w_{ij} is the interaction effect between variable i and j. $\boldsymbol{v}_i$ is a vector describes the i-th variable with k factors. $k \in \mathbb{N}_0^+$ is a hyperparameter that defines the dimensionality of the factorization.

The most innovative idea of FM is to break the independence in estimating interaction effects, which means the data that is used to estimate one interaction effect can also help estimating other related interactions. This will give more reliable interaction effect estimates compared to those directly and independently estimated interaction effects from the sparse data.

An open source implementation of factorization machine called libFM is available.

CHAPTER 4

Experiments

4.1 Data

In this paper, I experimented with 2 unique data sets, both of which are online advertising data. One is click level data sets with more than 40 million rows from a previous data science competitions on Kaggle - Avazu Click-Through Rate Prediction (later referred to as CTR). Inherently click level data should be high imbalanced, with majority of the impressions not being clicked. But the real average CTR is a business secret and the sponsor of the competition is not willing to disclose. So the data I experimented with is only a slightly imbalanced sample of the original data set. Features include anonymous information like the ip address of the user, device id, site category and a bunch of encrypted categorical variables with meaning not disclosed. The data set and the detailed descriptions of the features can be found on the competition website on Kaggle.

The second data set is a conversion level data that I acquired from a company that I once worked for (late referred to as Conversion). Though not all conversion level data sets are necessarily highly skewed, most of them are and this is one of those. As can be expected, conversion level data tend to be smaller but more specific compared with click level data. For this data set, in addition to the device and demographic information, we also knowledge of how long a use spent on the website and other very specific information of the users that we believe are directly associated with the conversion.

Data set	# obs in training	# obs in test	# of features	% minority
CTR	40,428,967	4,577,464	23	16.7
Conversion	305,172	84,093	24	0.9
CTR_sub	4,000,000	1,000,000	23	16.4

Table 4.1: Summary of the Data Sets

Data set	Cardinality (training)				
	V1	V2	V3	V4	V5
CTR	6,729,486	2,686,408	8,552,	8,251	7,745
Conversion	102	56	33	24	20
CTR_sub	1,056,051	356,977	6,061	4,114	3,351

Table 4.2: Cardinality of Top 5 Categorical Features

To get a comprehensive comparison of different models' performance in learning data in various sizes, I sampled the first 5,000,000 observations from the Avazu click level data set as a third data set (later referred to as CTR_sub) to experiment with.

All three data sets are split into training and test set based on a time related variable. For CTR data, the training set contains 10 days traffic while the test is the 11th day. For the Conversion data, the training set contains records throughout 2013 while in the test set are the records during the first quarter of 2014. For CTR_sub data, the first 4 million records are used as training and the remaining 1 million as test.

A basic summary of the three data sets is shown in Table 4.1. Table 4.2 lists the top 5 categorical features with the largest cardinality in each data set.

4.2 Performance Measurement

Though classification models are used in learning and predicting CTR and conversion rate, but the predicted class label is usually not what of interest. Instead a probability score is preferred as it allows us to rank the customers and also can be used to calculate revenue related metrics that are essential to any business. As a result, any label based performance metrics are improper here, such as overall classification accuracy, true positive (recall, sensitivity), true negative (specificity) and etc. In this paper, Receiver Operating Characteristic (ROC or ROC curve) and Area Under the Curve (AUC) are used as error metrics to measure the performance of different models for the conversion data as they are proven to be an ideal metric for the domains with skewed class distribution and unequal misclassification costs [15]. For the Avazu CTR data set, in addition to ROC and AUC, I also used the evaluation metric required for the competition the log loss. The formula of log loss is shown below. From the definition we can find log loss penalizes most on extreme predictions.

$$L(y, \hat{y}) = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)] \tag{4.1}$$

4.3 Software Implementation

One of the purposes of this paper is to introduce as many potential models and implementations of the models that can be employed to mine and analyze online advertising data, so I tried to use different software implementations of the modeling techniques that are introduced in previous sections.

In general I used R, Python, xgboost, LIBLINEAR, libFM, and FTRL-Proximal. And Specifically:

- *rpart* package in R for decision tree, bagging of decision tree

- *scikit-learn* package in Python for decision tree, bagging, random forest, Adaboost, gradient boosting machine
- xgboost for gradient boosting machine
- LIBLINEAR for L2-regularized logistic regressions
- libFM for factorization machine
- python code based on FTRL-Proximal algorithm

4.4 Parameter Tuning

Most of the models used in the paper require extensive parameter tuning and choosing the right parameters is the determining factor of the models performance, so I will devote this section to discussing the parameter settings within each model along with some intermediate results from the experiments.

4.4.1 Decision Tree Based Models

When implementing any decision tree based algorithm, the complexity parameters of the tree structure is usually the first thing to consider. They include the depth of the tree (*depth*), the minimum number of observations in a node to split (*min_samples_split*) on and the minimum number of observations in a leaf node (*min_samples_node*). For some decision tree algorithm, such as CART, the minimum information gain or increase in gini index (*cp*) whichever metrics is used has to be met before splitting the node is another complex parameter. Those complex parameters are in fact correlated with one another, and in practice, one usually only need to specify one or two of them to control the complexity of the tree structure.

For single decision tree, bagging and random forest where complex tree struc-

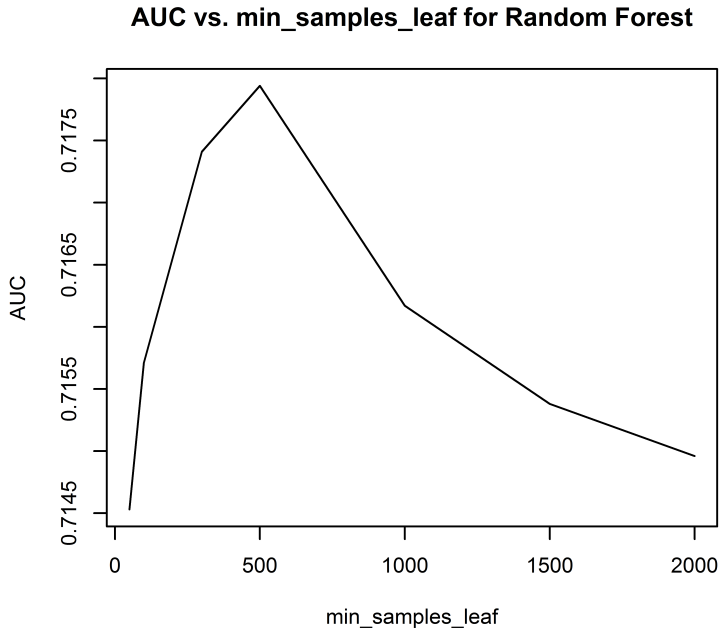

Figure 4.1: AUC against *min_samples_leaf* in RF on Conversion Data

tures are desired, only specifying *min_samples_node* is enough if one is not using a CART like algorithm where *cp* is also better to be tuned. The principle of choosing *min_samples_node* is to consider how many observations are needed for a reliable estimate of the probability. For highly imbalanced problem, this number should be specified bigger. For the Conversion data, 500 turns out to be a fair choice. Figure 4.1 shows the performance of best random forest models different different choices of *min_samples_node* for the Conversion data. Note that each random forest consists of 500 trees.

In boosting tree models where shallow tree structures are needed, *depth* should be properly specified along with *min_samples_node*. *depth* defines the highest order of interaction effects allowed in the tree structure, usually 6 to 10 is enough for a complex problem. The learning rate and the number of trees in boosting models parameter need to be tuned too. A practical tip is to target a certain number of trees, say 500, and tune the learning rate only.

4.4.2 Regularized Logistic Regression

The LIBLINEAR implementation of regularized logistic regression only supports L2-regularized logistic regression, and there is only one tuning parameter (denoted as c) in the model controls the size of the shrinkage. The larger the c, the more regularization.

4.4.3 FTRL-Proximal

There are four tuning parameters in the algorithm learning rate, calibration factor, L1 and L2 regularization. A grid search is recommended.

4.4.4 Factorization Machine

Two main turning parameters for libFM are the dimension which specifies the order of factorization, namely the k in 3.12 and the number of iterations to run. Other parameters are associated with the optimization methods you choose. Currently stochastic gradient descent (SGD) and alternating least squares (ALS) optimization as well as Bayesian inference using Markov Chain Monte Carlo (MCMC) are supported. A very detailed manual can be found on libFM website.

CHAPTER 5

Results

Table 5.1 compares the performance of different models on the Conversion data. All models listed in the table are selected based on cross validation using the parameter tuning methods mentioned in Chapter 4 section 4. For Conversion set, tree based models outperform the linear models by a big margin. The most likely reason for the distinction in performance between tree based models and logistic regression models is that the tree based models naturally capture the higher order interaction effects among features which either regularized logistic regression or FTRL is not able to learn without explicitly adding higher order interaction effects in the model. To prove the idea, I manually added a few interaction terms in FTRL model based on my understanding of the problem and observe a considerable boost in performance (0.71035 AUC).

The limited sample size might account for the mediocre performance of factorization machine as FM usually needs a very large sample size to learn the factorized interaction effects of the data.

Among tree based models, random forest and gradient tree boosting have very close performance but yield substantial superior results than single decision tree. For learning highly skewed data, single decision tree either end up with an over-grown tree with many very small leaves (unstable prediction) or a coarse tree structure prone to miss out subtle details or relations, while random forest grows complex trees aiming to capture as much information as possible and then takes average of hundreds of those over-grown trees to cancel out the extremes

Model	AUC	Log loss	Mean pred	Pred bias
Decision tree	0.70082	0.03827	0.00934	0.00274
Random forest	0.71795	0.03805	0.00902	**0.00242**
Gradient tree boosting (scikit-learn)	**0.71816**	**0.03798**	0.00907	0.00247
Gradient tree boosting (xgboost)	0.71552	0.03815	0.00954	0.00294
LIBLINEAR	0.70548	0.03839	0.00951	0.00291
FTRL	0.70540	0.03910	0.01145	0.00485
libFM	0.70531	0.03887	0.01079	0.00419

Table 5.1: Comparison of Models on Conversion Data Set

and stabilize the prediction. For gradient tree boosting, even each individual tree is shallow, they are built in a sequential order and thus is also able to capture information from different aspect in the data.

A visual display of the model performance between different tree-based models using ROC curve is showed by Figure 5.1 followed by a zoomed version (Figure 5.2) for better visibility.

Figure 5.3 compares the model performance among the best model in each model family introduced in this paper.

Table 5.2 compares the performance of different models on the CTR_sub data. As can be seen from Table 4.1 and Table 4.2 the CT_sub data is very large in size (4 million rows) and high dimensional (5,431,038 with one-hot encoding and 357,008 after combining rare levels). Taking Python *scikit-learn* for example, any data has to be converted to *numpy* array before being fed to any machine learning algorithms. Each *numpy* float64 number takes 8 bytes, this design matrix after one hot encoding would consume $8 \times 5431038 \times 4000000/10^9 = 173$ Terabytes and 11.4 Terabytes after combining rare levels and will not fit into memory on any

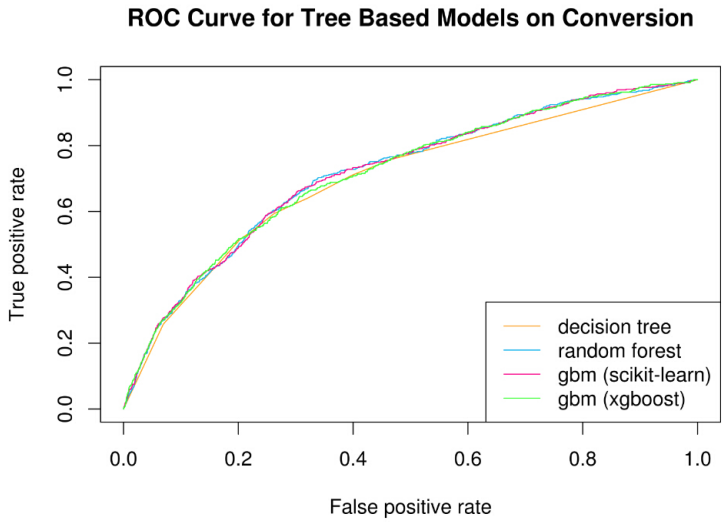

Figure 5.1: ROC Curve for Tree Based Models on Conversion Data

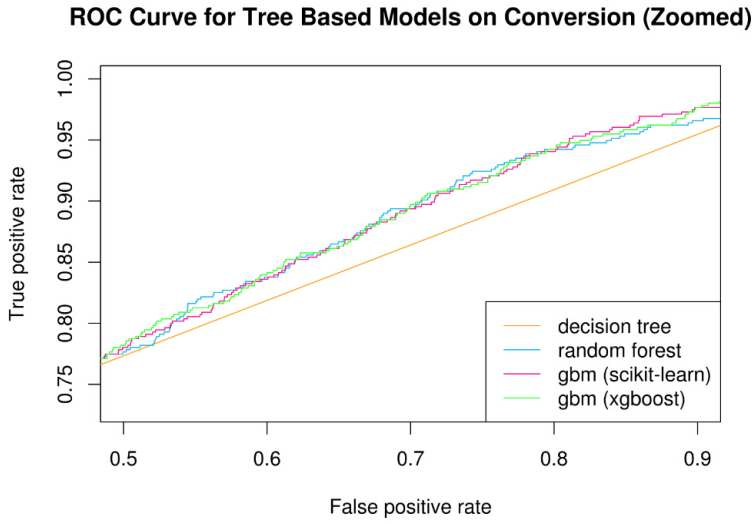

Figure 5.2: ROC Curve for Tree Based Models on Conversion Data (Zoomed)

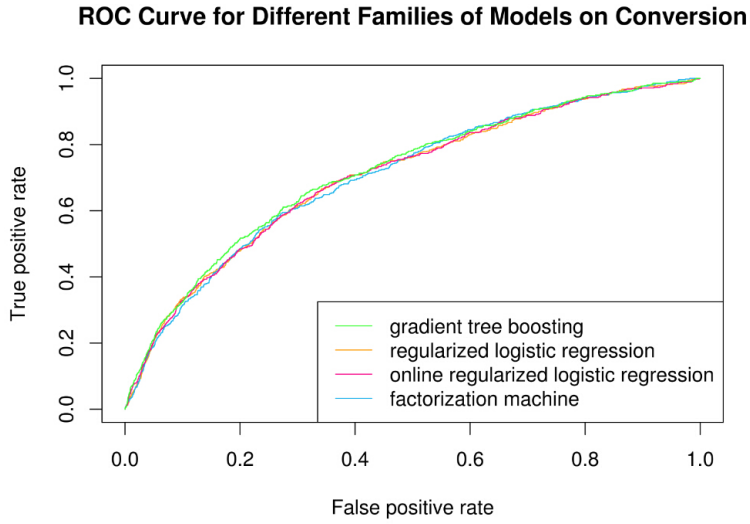

Figure 5.3: ROC Curve for Different Families of Models on Conversion Data

work station. So only linear models are built and compared for CTR_sub data.

With this amount of data, factorization machine is able to model the factorized interaction effects pretty well and shows superiority to (online) regularized logistic regression models which are unable to learn the interaction effects from the raw data at all. As a trade-off, factorization machine (libFM) is more time consuming compared to LIBLINEAR and FTRL-Proximal in training.

For the original CTR data which contains more than 40 million of records with high cardinality categorical features, even LIBLINEAR and libFM fail to scale. So for CTR data, there isn't any model comparison in performance, but in feasibility. Only online learning algorithms which analyze the data one row at a time seem promising in such context. It's worth mentioning that with some complex data preprocessing, such as convert high cardinality categorical features to numerical features based on certain rules will make other models feasible again but it's beyond the scope and interest of this paper. The best performing FTRL model on the CTR data has a log loss as low as 0.39410.

A useful trick to further improve the efficiency of FTRL (or any online learning

Model	AUC	Log loss	Mean pred	Pred bias
LIBLINEAR	0.74663	0.40270	0.17076	**0.00292**
FTRL	0.74512	0.40292	0.17045	0.00323
libFM	**0.75031**	**0.40008**	0.17923	0.00556

Table 5.2: Model Comparison on CTR_sub Data

algorithm in general) is to use hashing trick instead of one hot encoding. To apply one hot encoding (dummy encoding), one has to request one pass of data to learn the mapping from original feature:value pair to the index in design matrix and the size is determined by the number of rows and also the total cardinality of the categorical variables. Hashing trick (also known as feature hashing), on the other hand, is faster and more space efficient since only a hash function is used to define the mapping from original feature:value pair to the index in the design matrix and the dimension of the design matrix is predefined [16]. Though hash collision is inevitable, in practice such collisions rarely affect classification results [17] and the advantages in efficiency make it the preferred encoding method in large scale learning.

CHAPTER 6

Conclusion and Future Work

In this paper, we first presented the challenges we have in analyzing online advertising data, namely enormous size, complex structure, high cardinality and severe skewness. Then four families of state-of-art classification (probability estimation) models were introduced to approach the problem. A wide range of empirical experiments were conducted using the models introduced from various sortware packages and implementations and the best models from an extensive cross validation were compared in detail. We found for mid-size tasks where the data (design matrix) fits into memory and is manageable, tree based models, especially random forest and gradient tree boosting tend to be a superior choice as they are able to learn the complex structures of the problem yet give robust predictions. Regularized logistic regression with manually added and seleted interaction terms and feature transformations may also give comparable results but require considerable trial and error. Online learning methods are not necessary in such context and factorization machine usually fail to take advantage of the limited dta. For large-scale task (millions or instances), most tree based models fail to scale if high cardinality also exists. Factorization machine proves to be superior to regularized logistic regression models as the data set is large enough for FM to learn the complex structure. If the data size keeps growing, both FM and LIBLINEAR becomes infeasible and only online learning methods seems promising.

One key factor in improving model performance in machine learning but is a not a focus in this paper is feature engineering. Feature engineering is the

Model	AUC		Log loss		Pred bias	
	w/	w/o	w/	w/o	w/	w/o
LIBLINEAR	0.75032	0.74663	0.40108	0.40270	0.00728	0.00292
FTRL	0.74991	0.74512	0.40172	0.40292	0.00323	0.00323
libFM	**0.75472**	0.75031	**0.39687**	0.40008	**0.00022**	0.00556

Table 6.1: Comparison of Models with and without Simple Feature Engineering on CTR_sub Data

process of transforming raw data into features that better represent the underlying problem to the predictive models, resulting in improved model accuracy on unseen data. In fact, adding interaction terms to logistic regression can be understood as feature engineering in a broad sense. But in practice feature engineering can go far more that that. It is a deep while broad topic and requires profound business understanding of the problem, considerable experience with data and programming skills to efficiently engineer new features. To illustrate the idea, I add a new feature to the CTR_sub data set trying to capture some historical information of the users - the number of impressions that a user have had from this specific web site or application. By simply adding this one feature, the model performance improves quite a bit. See Table 6.1 for details.

So far, all the discussion in this paper is based on single models or at least single family of models. In fact, another way to further improve the predictive performance of models is to use multiple models at the same time. We can either make an ensemble of different families of models that capture different sources of information of data to come up with a comprehensive model by taking a simple average or a weighted average of predictions from different models, or use stacking of models where the output (prediction) of one model is used as input for another model [18].

References

[1] Chapelle, O., Manavoglu, E., & Rosales, R. (2014). Simple and scalable response prediction for display advertising. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 5(4), 61.

[2] He, X., Pan, J., Jin, O., Xu, T., Liu, B., Xu, T., ... & Candela, J. Q. (2014, August). Practical Lessons from Predicting Clicks on Ads at Facebook. *In Proceedings of 20th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (pp. 1-9). ACM.

[3] Hastie, T., Tibshirani, R., Friedman, J., Hastie, T., Friedman, J., & Tibshirani, R. (2009). *The elements of statistical learning* (Vol. 2, No. 1). New York: springer.

[4] Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1993). Classification and Regression Trees, Wadsworth International Group, Belmont, CA, 1984. *Case Description Feature Subset Correct Missed FA Misclass, 1*, 1-3.

[5] Breiman, L. (1996). Bagging predictors. *Machine learning, 24*(2), 123-140.

[6] Breiman, L. (2001). Random forests. *Machine learning, 45*(1), 5-32.

[7] Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences, 55*(1), 119-139.

[8] Long, P. M., & Servedio, R. A. (2010). Random classification noise defeats all convex potential boosters. *Machine Learning, 78*(3), 287-304.

[9] Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 1189-1232.

[10] Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 267-288.

[11] Andrew, G., & Gao, J. (2007, June). Scalable training of L 1-regularized log-linear models. *In Proceedings of the 24th international conference on Machine learning* (pp. 33-40). ACM.

[12] Fan, R. E., Chang, K. W., Hsieh, C. J., Wang, X. R., & Lin, C. J. (2008). LIBLINEAR: A library for large linear classification. *The Journal of Machine Learning Research, 9*, 1871-1874.

[13] McMahan, H. B., Holt, G., Sculley, D., Young, M., Ebner, D., Grady, J., ... & Kubica, J. (2013, August). Ad click prediction: a view from the trenches. *In Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 1222-1230). ACM.

[14] Rendle, S. (2010, December). Factorization machines. *In Data Mining (ICDM), 2010 IEEE 10th International Conference on* (pp. 995-1000). IEEE.

[15] Fawcett, T. (2003). Roc graphs: Notes and practical considerations for data mining researchers.(2003). *Copyright Hewlett-Packard Company.*

[16] Weinberger, K., Dasgupta, A., Langford, J., Smola, A., & Attenberg, J. (2009, June). Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning* (pp. 1113-1120). ACM.

[17] Attenberg, J., Weinberger, K., Dasgupta, A., Smola, A., & Zinkevich, M. (2009, July). Collaborative Email-Spam Filtering with the Hashing Trick. CEAS.

[18] Kotsiantis, S. B., Zaharakis, I., & Pintelas, P. (2007). Supervised machine learning: A review of classification techniques.