CSE599s, Spring 2014, Online Learning Lecture 20 - 6/5/2014

CTR Predictions and Literature References

Lecturer: Brendan McMahan or Ofer Dekel *Scribe: Javad Hosseini*

In the final lecture of the online learning course, we talk about one of the applications of *Online Learning* and provide some references on what we have learnt so far and future readings. In section 1, we present a talk in "ACM SIGKDD international conference on Knowledge discovery and data mining" on "Click Through Rates". We present some "Literature References" in section 2.

1 Ad Click Prediction: a view from the trenches

Many billions of dollars a year are spent on web advertising. In search engines, after a user enters his/her query, targeted ads are shown in response to the query. Figure 1 shows an example for Google. Web search ads align incentives. This is because search engines only get paid if the user clicks on the ad. This is not a perfect proxy for relevant ads, but it is good-enough to say that clicked ads are relevant. So search engines like to show relevant ads. On the other hand, users also like relevant ads more. In order to satisfy these, the key ingredient is predicting the probability of a click for a specific ad in response to a specific query. The most important features for this task are search phrases and ad text. Figure 2 shows the system overview of a prediction system for click probability. The data including ads, features and labels are streaming and we do not know the dimensionality in advance. There are some challenges in systems for training massive models on massive data with minimum resources. These systems should be able to handle

- billions of unique features (model coefficients)
- billions of predictions per day serving live traffic
- billions of training examples

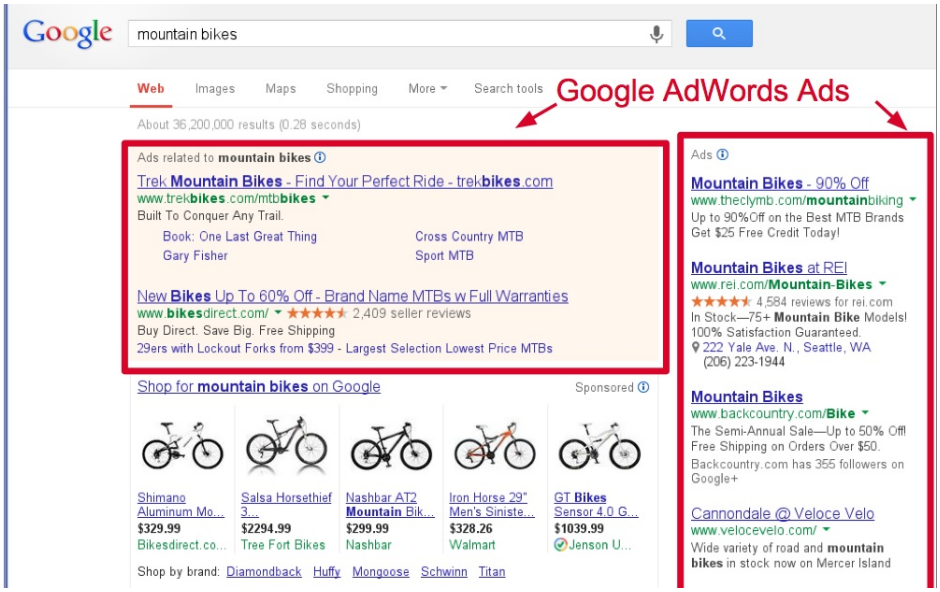

Figure 1: Google Adwords ads

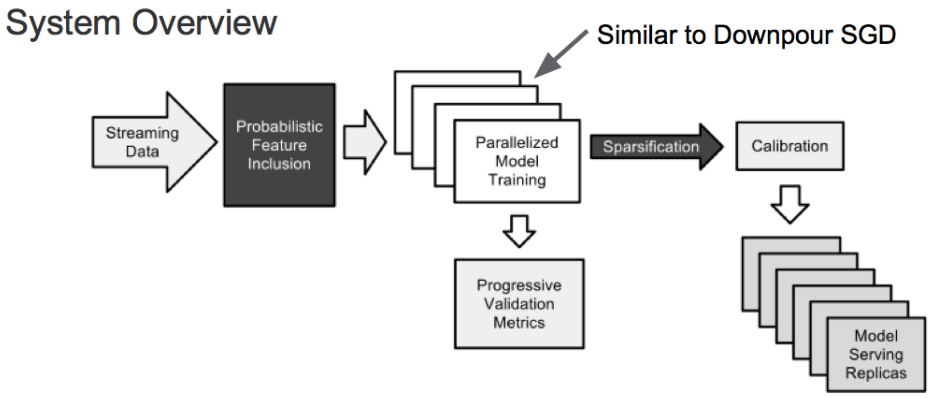

Figure 2: System Overview

1.1 FTRL-Proximal Online Learning Algorithms

The Follow The Regularized Leader Proximal (FTRL-Proximal) algorithm is similar to stochastic gradient descent, but has much sparser models. It is an online algorithm, therefore can be served for the click prediction task. It is simple, rich and has rich theory. If no regularization is used, this algorithm will be equivalent to Online (Stochastic) Gradient Descent [3]. FTRL-Proximal implements regularization in a way similar to RDA [5], but gives better accuracy in our experiments. The key is re-expressing gradient descent implicitly as:

$$w_{t+1} = \underset{w}{\operatorname{argmin}}(g_{1:t}.w + \frac{1}{2}\sum_{s=1}^{t}\sigma_s \left\|w - w_s\right\|_2^2 + \lambda_1 \left\|w\right\|_1). \tag{1}$$

This algorithm can be implemented as easily as gradient descent with a few tricks.

1.2 Per-Coordinate Learning Rates

Instead of having a constant learning rate in stochastic gradient descent, one can think of per-coordinate learning rates. In order to have an intuition why this can be helpful, consider predicting for English queries in the US, which includes 1,000,000 examples. Also, consider learning for Iceland queries in Iceland (100 training examples). If we imagine that the features are disjoint, we will train separate SGD models for each, i.e. generally around $\frac{1}{\#examples}$. On the other hand, if we train a single model by concatenating the feature vectors and interleave training examples, there will be no good learning rate to choose. SGD will either never converge for iceland (with a low learning rate) or will oscillate widely for the US (with a large learning rate). While there may be a middle ground, it will be still sub-optimal.

Theory indicates learning rate

$$\eta_{t,i} = \frac{\alpha}{\beta + \sqrt{\sum_{s=1}^{t} g_{s,i}^2}}. \tag{2}$$

Implementing this requires storing one extra statistics per coordinate. However, we have a trick for reducing this if you are training multiple similar models. Using per-coordinate learning rates will lead to huge accuracy improvement. This improved AUC by 11.2% versus a global learning rate while for the advertisement task a 1% improvement is large [4, 1]

1.3 Techniques for saving memory

Due to L1 regularization used in FTRL-Proximal, there are some features that are not included in the final model. This motivates seeking techniques for saving memory both while training and testing. While training, we may use probabilistic feature inclusion. Grouping similar documents and randomized rounding are other techniques for saving memory while training. While testing, feature reduction because of L1 regularization leads to saving memory. Also, we may still use randomized rounding. These techniques are discussed later in this section.

Figure 3 shows the tradeoff frontier for a small (10^6 examples) dataset between the fraction of selected features by L1 regularization and AUC for three algorithms including FTRL-Proximal. It can be seen that smaller models will have better accuracy. However, if the fraction of selected features becomes too low, the performance will decrease. Each line varies the L1 parameter for different size/accuracy tradeoffs.

In order to evaluate FTRL-Proximal, we compare it to a baseline algorithm. In baseline, only features that occur at least K times are included. We have tuned the L1 parameter to provide a good sparsity/accuracy tradeoff. Then we tuned K to get the same accuracy with the baseline. This resulted in 3 times non-zero features in baseline. We can conclude that L1-regularization selects features well.

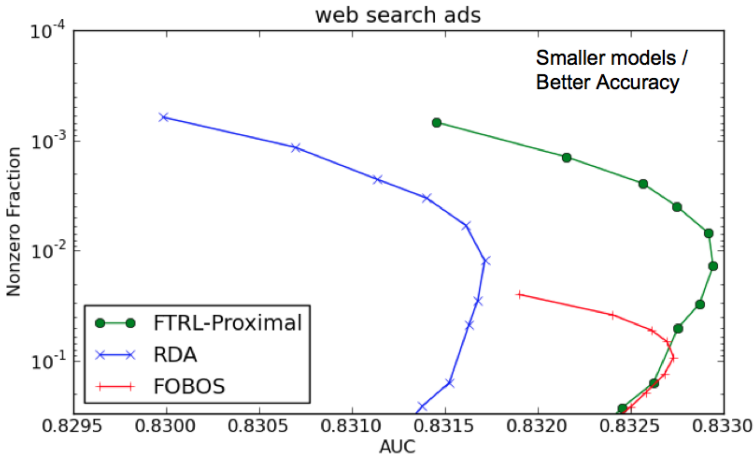

Figure 3: Tradeoff between and Performance

1.3.1 Probabilistic Feature Inclusion

Long-tailed distributions produce many features that only occur a handful of times. In some of our models, half of the unique features occur only once in a training set of billion examples. There generally are not useful for prediction. If we want to use this intuition to reduce dimensionality, it cannot be done directly. Since if we read all the stream of data and then want to prune low-accuracy features, we have already used a lot of memory. We want to distinguish rare features from common features without tracking any state. This can be done by probabilistic techniques. Two probabilistic features that can be used are Poisson inclusion and Bloom-filter inclusion. In Poisson inclusion, when a feature not in the model is seen, it is added to the model with probability p. Intuitively, with low p, if a feature is seen a lot of times, it will be included. Otherwise, chance of inclusion of a feature is low. In Bloom-filter inclusion, a rolling set of counting Bloom-filters is used to count occurrences. As a result, a feature will be added after K occurrences, but sometimes features with lower than K occurrences will also be added. In Table 1, the amount of saved RAM and also detriment in Auc is shown when these techniques are used. while the decrease in performance is low, we see around 50% decrease in used RAM for different methods.

Method	RAM Saved	AucLoss Detriment
Bloom ($n = 2$)	66%	0.008%
Bloom ($n = 1$)	55%	0.003%
Poisson ($p = 0.03$)	60%	0.020%
Poisson ($p = 0.1$)	40%	0.006%

Table 1: Feature Inclusion Tradeoff

1.3.2 Storing coefficients with fewer bits: Randomized Rounding

The learning rate tells us how accurately we might know the coefficient. If training on one example might change the coefficient by Δ, one does not know the value with more precision than Δ. Therefore, we do not need to waste memory storing a full-precision floating point value with 32 or 64 bits. In practice, storing a 16-bit fixed point representation is easy and works well. However, we need to be careful about accumulating errors. While an update may not change the value of a coefficient, lots of small updates will be important. We can use randomized rounding trick [2]. For training with randomized rounding, usual update is computed at full-precision, then the result is randomly projected to an adjacent value expressible in the fixed-point encoding. It should be noted that rounding should be unbiased, i.e. E[rounded coefficient] = unrounded coefficient. In our experiments, there was no accuracy loss with a 16-bit fixed-point representation, compared to 32 or 64 bit floating point values, which means 50%-75% less memory. Figure 4 shows this technique.

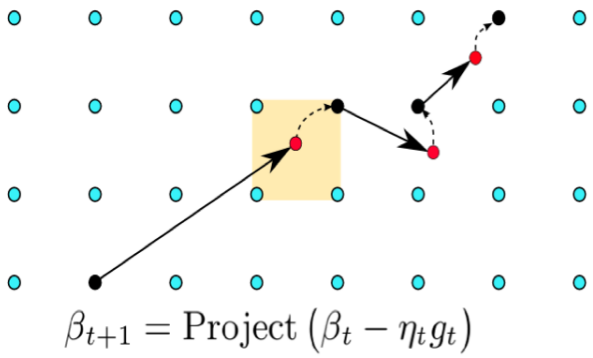

Figure 4: Randomized Rounding

1.3.3 Training many similar models Grouped

Training similar models causes loss of memory. As an example, assume we train four similar models separately with coefficients stored in a hash table. As it is shown in Figure5, for each feature, we may use 10 bytes for its key, its gradient-squared sum (used to compute learning rate) and the coefficient. The reason that we want to have similar models is that we may want to try different feature variations, learning rates, regularization strengths, etc.

We can train many similar models grouped and store data for all the similar models in one hash table. This technique is shown in Figure6. Besides 12 bytes of overhead per coefficient, this technique only uses 2 bytes per coefficient per model. However, we will not have exact per-coordinate learning rates. Instead, we use a good-enough approximation based on the number of observed positive and negative examples. This trick, however, requires using the same learning-rate statistic for all grouped models. The theory-recommended learning rate is

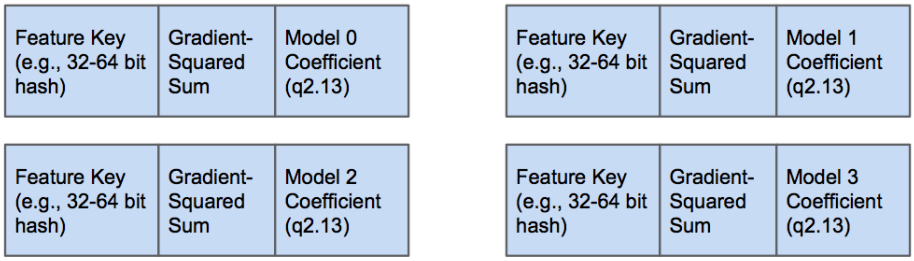

Figure 5: Training many similar models: individually

$$\eta_{t,i} = \frac{\alpha}{\beta + \sqrt{\sum_{s=1}^{t} g_{s,i}^2}}. \tag{3}$$

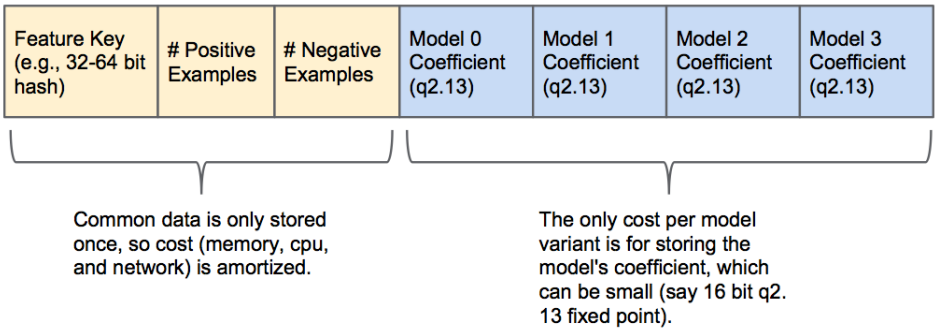

Figure 6: Training many similar models: grouped

If we only use counts, we have

$$\sum g_{t,i}^2 = \sum_{\text{positive events}} (1-p_t)^2 + \sum_{\text{negative events}} (p_t)^2 \approx P(1-\frac{P}{N+P})^2 + N(\frac{P}{N+P})^2 = \frac{PN}{N+P}, \tag{4}$$

where N is the number of positive examples and P is the number of negative examples.

1.4 High-Dimensional Data Visualization for Model Accuracy

The aggregate accuracy is not the only thing that matters. In online learning we also care about progressive validation, i.e. computing prediction on each example before training on it. We also want to look at relative changes in accuracy between models and also be able to quickly spot outliers or problem areas. Figure 7 shows an example of performance monitoring for three models on consecutive data.

1.5 Automated Feature Management System

In Google, we need a system to automatically manage features. There are many raw signals such as words-in-user-query, country-of-origin, etc. Many engineers are working on signals and multiple different learning platforms and teams consume them. There is also a metadata index to manage all these signals. It knows what systems are using what signals, what signals are available to different systems, status and version information/deprecation and also whitelist for production readiness/test status. The automated feature management system is used for automatic testing, alerting, notification and cleanup of unused signals.

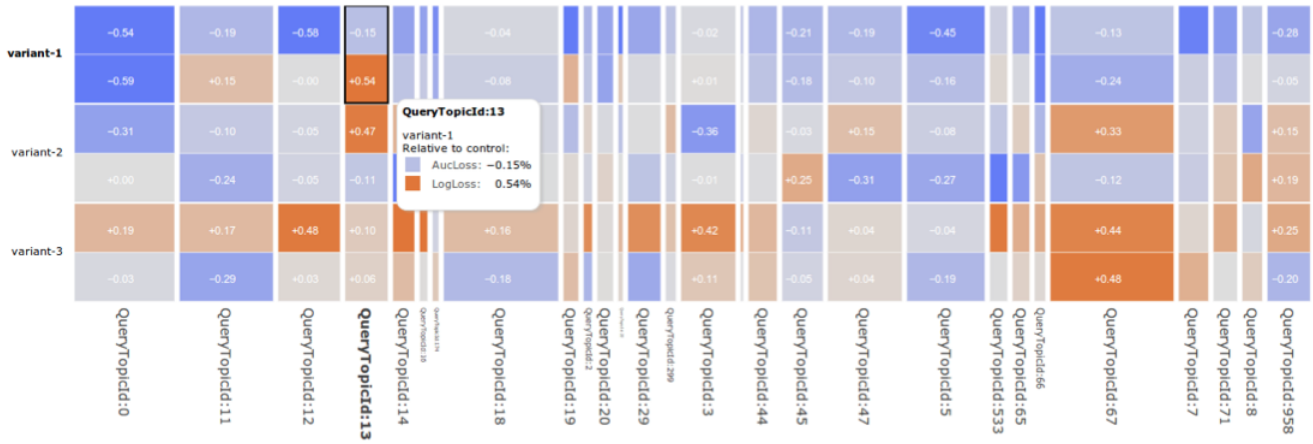

Figure 7: High-Dimensional Data Visualization for Model Accuracy

1.6 Final remarks about the paper

In the paper we had also discussed some other topics:

- Fast approximate training with a single value structure
- Unbiased training on a biased subsample of training data
- Assessing model accuracy with progressive validation
- Cheap confidence estimates
- Approaches for calibrating predictions

There were also some techniques that did not work as well in the paper:

- Aggressive feature hashing
- Randomized feature dropout
- Averaging models trained on different subsets of the features
- Feature vector normalization

2 References and Other Topics

We discussed references and some other topics that can be found here. Presented references include:

1. Online Linear and Convex Optimization
   (a) Projected Gradient Descent View
   (b) Follow-the-Regularized-Leader View
2. Kalai-Vempala
3. Learning with Structure
4. Log(T) Regret for Strongly Convex f

5. "Second-Order" Algorithms (also for Classification in the Mistake Bound Model)
6. "Second-Order" Algorithms for Linear Functions (aka, AdaGrad)
   (a) The per-coordinate gradient descent algorithm
   (b) General feasible sets
7. The Experts Setting / Entropic Regularization
   (a) Experts Setting
   (b) EG vs GD for Squared Error
   (c) Game Theory View
8. Minimax Analysis and Unconstrained Linear Optimization
9. K-Armed Bandits (EXP3) and Contextual Bandits (EXP4)
   (a) Original EXP3 and EXP4 Analysis
   (b) Analysis for Losses (No Mixing Needed)
   (c) Improved EXP4 Analysis
   (d) High-probability bounds for EXP4
10. Stochastic Approaches to the Contextual Bandits Problem
    (a) Stochastic Setting
    (b) Model
11. Bandit Convex Optimization
    (a) General $T^{\frac{3}{4}}$ Regret
    (b) Strongly convex functions $T^{\frac{2}{3}}$ Regret
    (c) Smooth convex functions, $T^{\frac{2}{3}}$ Regret
12. Bandit Linear Optimization
13. Online Submodular Minimization
14. Online Kernel Methods with a Budget of Support Vectors
15. Selective Sampling / Online Active Learning / Label Efficient Learning
16. Other Problems
    (a) Online PCA
    (b) Online One-Class Prediction (e.g., outlier detection)
    (c) Online Ranking

References

[1] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011.

[2] Daniel Golovin, D Sculley, H Brendan McMahan, and Michael Young. Large-scale learning with less ram via randomization. *arXiv preprint arXiv:1303.4664*, 2013.

[3] H Brendan McMahan. Follow-the-regularized-leader and mirror descent: Equivalence theorems and l1 regularization. In *International Conference on Artificial Intelligence and Statistics*, pages 525–533, 2011.

[4] H Brendan McMahan and Matthew Streeter. Adaptive bound optimization for online convex optimization. *arXiv preprint arXiv:1002.4908*, 2010.

[5] Lin Xiao et al. Dual averaging methods for regularized stochastic learning and online optimization. *Journal of Machine Learning Research*, 11(2543-2596):4, 2010.