
codebase
software consulting

Dynamic Price Floors

CodeBase

University of California, Berkeley

May 6, 2018

Introduction

Polymorph provides publishers the opportunity to set a price floor to protect valuation of inventory and generate additional revenue. In the case of a single RTB auction, a price floor lifts revenue if it falls between the top two bids, and lowers revenue if it falls above the highest bid.

Under the reasonable assumption that a publisher's optimal (highest-revenue) static price floor does not vary significantly day-to-day, revenue may be optimized by manually tweaking the floor on a per-placement basis. However, dynamically adjusting price floor based on past bid amounts and known information about the auction has the potential to not only reduce maintenance time but also to lift publisher revenue.

Methodology

Please refer to the attached code in `polymorph-dpf.zip`

Preparation

We were provided five days of unified auction logs, from 2/11/18 12:00AM to 2/15/18 11:59PM. Bid lines and direct-sold campaigns were filtered out such that what remained was roughly 1.2 billion RTB auction entries. The 128 files per hour were then consolidated into 32 files per hour for convenience.

Analysis

40% of auctions did not receive any bids. Only about 35% of auctions received more than one bid. This could be reflective of effective price floors, because there is only a single bid response in the event of an effective floor.

Our objective was to implement and evaluate pricing strategies. To simulate the effect of a pricing strategy however, we needed access to a distribution of

incoming bids. We chose to assume that we had access to every possible bid. Even though this did not give us a true distribution, making this assumption allowed us to compare strategies in a meaningful way.

Simulation

We built a simulator to standardize our results and enforce that strategies used only valid information to set floors. The Python class `Simulator` in `simulator/simulator.py` was extended by strategies, which implemented the abstract methods `calculate_price_floor(input_features)` and `process_line(line, input_features, bids)`. We trained and tuned our strategies on the first three days of data, and tested on the last two days. Here is an example strategy which sets the price floor as 0.45 times the average of the last 25 bids.

```
class SimpleRunningAverage(Simulator):
"""
Sets the global reserve price as a weight times the average of
the last 25 bids.
"""
def __init__(*args, history_len=25, weight=0.45, **kwargs):
    super().__init__(**kwargs)
    self.most_recent_bids = deque([0], maxlen=history_len)
    self.weight = weight

def calculate_price_floor(self, input_features):
    return self.weight * sum(self.most_recent_bids)
                       / self.most_recent_bids.maxlen

def process_line(self, line, input_features, bids):
    if len(bids) > 0:
        self.most_recent_bids.append(bids[0])
```

The fields available to strategies as inputs are found in `simulator/utils.py`. For more examples, see `simulator_comparator.py`.

Pricing Strategies

OneShot

Overview

When considering the highest two bidders and a price floor, there are three possible cases. OneShot[2] adaptively updates the price floor for the next auction differently in each of the three cases. MultiShot takes this idea further

by clustering different sites together by revenue and running different OneShot instances on each cluster.

Detailed Description

In the first case, the price floor is above both bids. In this case, we decrease the current price floor by multiplying by $(1 - \epsilon^t * \lambda_h)$ (because this value is guaranteed to be less than one), where ϵ is the time decay factor between 0 and 1, and λ_h is a hyperparameter. The smaller the time decay factor, the smaller the impact future auctions will have on the price floor. In the second case, the price floor is between first and second bids. In this case, we increase the price floor multiplicatively by $(1 + \epsilon_t * \lambda_e)$, where λ_e is a hyperparameter. In the last case, the price floor is below both bids. In this case, we increase the price floor multiplicatively by $(1 + \epsilon_t * \lambda_l)$, where λ_l is a hyperparameter. The reason why there are different hyperparameters for the second and third cases is to allow for us to adjust the price floor differently in those cases; in the third case, we want to increase the price floor more than in the second case in order to quickly optimize for a value that would lead to the ideal auction case. Lastly, when outputting a price floor, we take the price floor only if it is less than the price floor ceiling that is set.

An issue with using a singular instance of OneShot is a strong fluctuation in price floors over time. This is caused by the variation in different sites and the ideal price points for them. We tackled this issue by running multiple instances of OneShot. In MultiShot, we cluster sites by their revenue; in other words, sites with higher revenue will use a different instance of OneShot than sites with lower revenue. By doing so, strong fluctuations in price floor do not occur, and more revenue can be generated.

Hyperparameter Tuning

We isolated hyperparameter tuning into two phases: first, we considered a singular OneShot instance. We tested various values λ_h, λ_e, and λ_l, which respectively represent the algorithm's weighted responses to overshooting, landing in between the first and second largest bid, and undershooting. Optimization factors included both maximum revenue and minimum overshoot rate, which we believed were the two the most important metrics. The second phase involved optimizing the number of MultiShot buckets that the data would be grouped into. We tuned all of these parameters by taking a manual gradient descent approach.

Our results are summarized below.

Hyperparameter	Max revenue	Min overshoot	Mixed
λ_h	0.02	0.60	0.05
λ_e	0.90	0.10	0.60
λ_l	0.88	0.30	0.70
Number of clusters	84	4	80
Revenue	250.6	14.6	153.5
Overshoot	90.6%	0.15%	80.7%

Results

The results show the algorithm's affinity towards picking higher price points to optimize for revenue in lieu of minimizing overshoot. In the simulations that we ran with the MultiShot algorithm, we received the strongest results solely with parameters that encouraged a higher price floor. In contrast, models that attempted to minimize overshoot, at any rate, were often met with dismal revenue results.

However, this model can be considered to be unstable. If deployed in production, the high price floor pushed by the algorithm would encourage higher bids by bidders, which would then push up the price floor generated by the algorithm even further. As a result, we recommend additional experiments on MultiShot with custom parameters, potentially with a time-decay factor to optimize at a certain price point. Following are the results from final testing.

```
----------------------------------------
MultiShot with 84 Buckets
----------------------------------------
Total Revenue: 339306.1161881846
Auction Count: 518420532
Auction Count (non-null): 300064887
Price Floor Engaged (non-null): 5.27%
Price Floor Too High (non-null): 94.71%
Average Revenue: 0.0006544997646585969
Average Revenue (not-null): 0.0011307758117936157
Average Bid Count: 0.954231081650138
Average Bid Count (non-null): 1.648620036638942
Average Bid Amount: 0.0028150392288430067
Average Bid Amount (non-null): 0.004863528516142781
Average Price Floor: 0.029849202372346
Time taken: 8962.81 seconds (0.017 milliseconds per auction)
```

Weighted Running Average

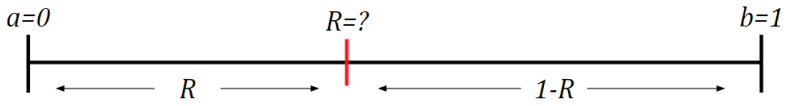
Overview

The running average approach is based on a derivation of the optimal price floor to maximize revenue in a second-price auction[1]. The derivation assumes a simplified ideal model of a second-price auction and finds that the optimal price floor is the average of the first and second highest bids. Our implementation of running average adapts this idea to real-world auction data by introducing weights to shift the average higher or lower, as well as a method of simulating lost bid information below an existing price floor. We also explore potential price trends in auction features by creating strategies that record separate running averages for each value of a certain feature, for example, separate averages per website.

[1] http://www.cs.ubc.ca/ cs532l/gt2/slides/11-6.pdf

Optimal Price Floor Derivation

We are given two optimal bidders who participate in a second-price auction with price floor R. We assume that the bid prices for each bidder, b_1 and b_2, are uniformly distributed on some range $[a, b]$. For clarity, we will normalize this range to $[0, 1]$.



The probability of R engaging, overshooting, and undershooting as well as the expected revenues from each case can be calculated as follows.

$$\begin{aligned}
&\text{Overshoot: } Pr(R > b_1, b_2) = R^2, && \text{revenue} = 0\\
&\text{Engage: } Pr(b_1 > R > b_2 \text{ or } b_2 > R > b_1) = 2(1-R)(R), && \text{revenue} = R\\
&\text{Undershoot: } Pr(R < b_1, b_2) = (1-R)^2, && \text{revenue} = \tfrac{1+2R}{3}
\end{aligned}$$

The optimal R can be derived by maximizing R in the total expected revenue.

$$\begin{aligned}
&\text{Expected revenue} = 2(1-R)(R) \cdot R + (1-R)^2 \cdot \tfrac{1+2R}{3} = \tfrac{1+3R^2-4R^3}{3}\\
&\text{Maximize: } \tfrac{d}{dx}(\text{Expected revenue}) = 2R - 4R^2 = 0
\end{aligned}$$

Thus, the optimal price floor that maximizes revenue is $R = 0.5$. However, the actual bid distribution is not uniform, and skews more toward the lower end with a few very high outliers. Thus, we can shift the running average by testing values of $R < 0.5$.

Simulating Lost Bids

One more remaining issue is that when a price floor is in place, we lose the bids that would normally occur below the price floor. This causes a running average approach to only take into account bids above the current average, resulting in an average that can only strictly increase to unreasonable levels. To mitigate this, we use the optimal price floor derivation and its assumption of a uniform bid distribution to simulate the lost second-highest bid. Given a uniform distribution of bids on the normalized range $[0, 1]$:

$$\begin{aligned}
&\text{Expected high bid} \approx 0.66\\
&\text{Expected low bid} \approx 0.33\\
&\text{Ratio} = 0.66/0.33 = 0.5\\
&\text{Simulated low bid} = 0.5 \cdot \text{high bid}
\end{aligned}$$

We will call the simulated bid ratio/multiplier s. Due to actual bid distributions skewing toward the lower end, values of $s < 0.5$ may be more optimal. We simulate the second bid in case of a 0- or 1-bidder auction.

1. 0-bidder auction: We must assume that our price floor has overshot all participating bidders. There do exist many auctions with no bidders even without a price floor, but the two situations cannot be differentiated while a price floor is in effect. We treat the price floor R as the "winner" of the auction (first bidder), and generate a fake second bid using $s \cdot R$.

2. 1-bidder auction: We generate a second bid in the same way using the single top bidder, who is guaranteed to be higher than our price floor. An alternative method worth exploring would be simply using our price floor as the second bidder. This makes sense because the winner of a second-price auction pays the second bidder's value, and in this case the winner pays the price floor. This alternative method would be more insulated against extremely high first bids, but may have other tradeoffs depending on the true gap between the first and second bidders.

Implementation

Several variations of the running average approach were implemented.

Weighted Running Average: We calculate a universal running average to set our price floor. It takes in two parameters: an integer window and a fraction weight. Only auctions with two or more bidders are used; auctions with zero or one bidders are ignored. When such an auction is encountered, it takes the bid prices of the top two bidders and incorporates them into two separated running averages, lowAvg and highAvg, which record the second-highest and highest bid price respectively. The number of auctions in the past that these two averages record is determined by the window parameter. When a price floor is requested, the lowAvg and highAvg are weighted and summed together such that the price floor = weight * lowAvg + (1-weight) * highAvg. Thus, a higher weight parameter will shift the price floor closer to the average second-highest bid.

Weighted Running Average with Simulated Second Bidder: We modify the above strategy to account for losing bid information below the price floor in a real-life scenario. We introduce a new parameter, sim_bid, representing a multiplier used to generate a fake second bid as described in the previous section. We use a flexible parameter rather than a fixed value because the ratio of 0.5 we derived earlier does not always generate the best results due to the non-uniform distribution of real bids. Unlike our original Weighted Running Average, we now take into account 0- and 1-bidder auctions by simulating the missing bids.

Running Averages per Feature: The Weighted Running Average approach was modified to keep separate running averages for each value of a selected feature, such as site_id. Since advertising on some websites may be more valuable than others, keeping separate running averages would capture more of these price trends than a single universal average. This strategy was tested on the auction features site_id and pub_network_id.

Results

The window, weight, and sim_bid parameters were tuned on the Weighted Running Average with Simulated Second Bidder strategy. We found that the window parameter had a negligible effect on resulting revenue and price floor engagement. On the contrary, the weight and sim_bid parameters had a significant effect. Values that shifted the average upward netted increased revenue at the expense of increased overshoot rates, and vice versa for values that shifted

the average downward. We found a middle ground between revenue and engagement with the values window=500, weight=0.6, and sim_bid=0.3.

Below are the results for a Running Average per Feature strategy instance with separate averages for each unique site_id. The simulator was run on two days worth of data (Day 14 and 15) and parameters window=500, weight=0.6, and sim_bid=0.3. Note that a simulation run with no price floor ($0) had a Price Floor Engaged of 57%, meaning our 30% engagement is actually still capturing half of the auctions from before. Our Average Revenue (not-null) per auction of $0.0014 represents a 7x improvement over a simulation with no price floor run on the same two days of data. By tweaking the weight and sim_bid parameters, we can achieve even better engagement rates while still retaining a net increase in revenue.

```
------------------------------------------
GT site_id
------------------------------------------
Total Revenue: 424276.93237573805
Auction Count: 518420532
Auction Count (non-null): 300064887
Price Floor Engaged (non-null): 30.63%
Price Floor Too High (non-null): 64.70%
Average Revenue: 0.0008184030264753057
Average Revenue (not-null): 0.0014139506178733204
Average Bid Count: 0.954231081650138
Average Bid Count (non-null): 1.648620036638942
Average Bid Amount: 0.0028150392288430067
Average Bid Amount (non-null): 0.004863528516142781
Average Price Floor: 0.002432278846190381
Time taken: 1196.38 seconds (0.002 milliseconds per auction)
```

Optimization

Motivation

An abstract interpretation of the game-theoretic approach is that it attempts to reduce the problem of setting an optimal reserve price into an optimization problem. The optimal reserve price, r^*, erupts from maximizing the expected revenue given the distribution of the bidders, formally represented by the equation below

$$r^* = \arg\max_r E(\text{rev}_r(X, Y))$$

$$\text{rev}_r(X, Y) = \begin{cases} 0 & : X < r \\ r & : Y < r \leq X \\ X & : r \leq Y \end{cases}$$

where X and Y are the highest and second highest bids respectively. In the simplest model, we represented two bidders as uniformly distributed random variables over the same interval

$$X_1 \sim \text{uniform}(0, 1)$$
$$X_2 \sim \text{uniform}(0, 1)$$

We can now explicitly compute the expectation of the revenue function, in terms of our price floor.

$$\begin{aligned} E(\text{rev}_r(X, Y)) &= r^2 \cdot 0 + 2r(1 - r) \cdot r + (1 - r)^2 \cdot \frac{1 + 2r}{3} \\ &= \frac{1 + 3r^2 - 4r^3}{3} \end{aligned}$$

As proved above, this function of r is maximized at exactly 0.5, which gives us our optimal price floor [1]. However, while this result inspires the heuristic which lead to the weighted running average strategy, we should expect it to be limited by the unrealistic assumption of uniform bidding.

A more flexible strategy which relaxes this assumption, then, would involve inferring the distribution of the highest and second highest bids and setting the price floor to be the value which once again maximizes the expected revenue. Although this once again will not lead to an exact strategy, we will develop a heuristic based on this model which yields successful results.

Overview

Rather than assuming the bidders are uniformly, or even independently, placed, we attempt to construct an empirical joint distribution based on previous auctions. In particular, assume the last n auctions results have the following highest and second highest bids

$$A_n = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$$

where (x_i, y_i) represents the highest and second highest bid of the ith auction respectively. As n grows large, we should expect this empirical distribution to uniformly converge to underlying distribution of bids; we therefore use some fixed n to approximate this. In this setting, each point in A_n is assumed to be equally likely, hence applying the definition of the optimal price floor,

$$r^* \approx \arg\max_r \sum_{t=1}^{n} \text{rev}_r(x_t, y_t)$$

Notice that since the points (x_i, y_i) are fixed, this is simply a function of one variable r, given A_n is fixed which we can denote as f.

$$f(r; A_n) = \sum_{t=1}^{n} \text{rev}_r(x_t, y_t)$$

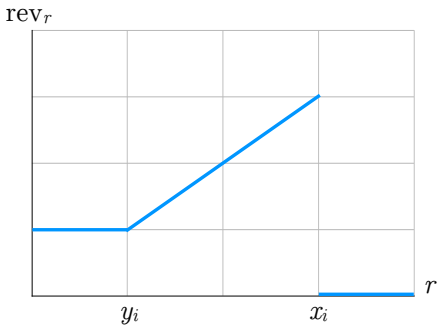
Finding the optimal reserve price is optimal to finding the global minimum of this function. We now present two approaches to finding this maximum,

Implementation: Linear Heuristic

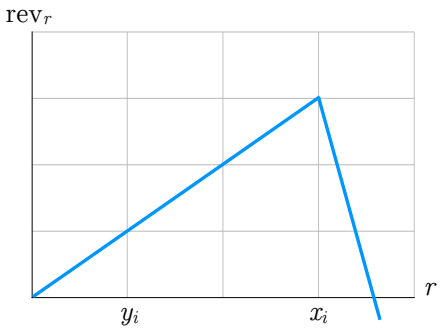
One reason for the difficulty of this problem is that the objective function we are trying to maximize is discontinuous,

$$f(r; A_n) = \text{rev}_r(x_1, y_1) + \text{rev}_r(x_2, y_2) + \cdots + \text{rev}_r(x_n, y_n)$$

where each of the $\text{rev}_r(x_i, y_i)$ hides the underlying discontinuity. Graphically, we can understand each of these terms for some fixed bids as



Although this function is piece-wise linear, it clearly is not continuous; therefore, it is unlikely the sum of these revenue functions defined for individual points will be continuous as well. While the linearity of the objective function suggests linear programming, the discontinuity means it can only be represented as a mixed integer program, which is intractable.



Therefore, we instead approximate this function using the minimum of two linear functions, as shown above. For this particular set of bids we could express the approximate revenue as the solution to the linear program

$$\begin{aligned} &\max\ z_i \\ &\quad z_i \leq r \\ &\quad z_i \leq -\delta(r - x_i) + x_i \\ &r \geq 0 \end{aligned}$$

where δ is the arbitrary slope we set to specify the penalty for overshooting. The overall revenue will be the sum of all z_is, which we maximize with respect to choice of r. The final linear program solves for the best possible revenue which maximizes this function.

Using our insight form the weighted average function, we know that was import to multiply this by some discount factor, γ.

Implementation: Brute Force

There is also a clever brute force solution to this maximization problem, which involves cleverly checking the search space. First notice that that

$$f(r; A_n) = \sum_{t=1}^{n} \text{rev}_r(x_t, y_t)$$

can have at most n discontinuities at the points in $\{x_1, x_2, \ldots, x_n\}$. First notice that for a non-trivial number of bids, the optimal revenue must at most x_n, lest the overall revenue be 0. Consider a candidate optimal price floor which lies between these points.

$$r \in (x_i, x_{i+1})$$
$$0 \leq i \leq n-1 \text{ where } x_0 := 0$$

Over this interval, the revenue function, along with each of the terms it sums must be differentiable. We can then compute the derivative at the point $(r, f(r))$.

$$\begin{aligned}
\tfrac{\mathrm{d}}{\mathrm{d}r} f(r; A_n) &= \tfrac{\mathrm{d}}{\mathrm{d}r} \sum_{t=1}^{n} \mathrm{rev}_r(x_t, y_t) \\
&= \sum_{t=1}^{n} \tfrac{\mathrm{d}}{\mathrm{d}r} \mathrm{rev}_r(x_t, y_t) \\
&\geq 0
\end{aligned}$$

Notice the derivative is zero if and only if r lies along constant regions of all constituent functions. Therefore taking an infinitesimal step to the right will keep the revenue the same. Otherwise, if positive, then moving to the right will surely increase the objective function. Hence, we have shown that there must be an optimum which lies at the boundary of these intervals.

Therefore our brute force approach aims to maximize revenue along each of these upper bid points, and similar to past strategies, return this multiplied by a fixed discount value, γ.

Vowpal Wabbit

Overview

Because profit is maximized when price floors are as close to the highest bid as possible (without going over), the idea behind this approach was to predict the highest bid price b_1 and find scalar multipliers that maximize revenue. In other words, we wanted to find the best predictive model m and multiplier value λ such that the value λp_m is maximized (where p_m is the prediction from m), yet $\lambda p_m \leq b_1$. To accomplish this, we decided to use a machine learning model trained on several features of bidders, such as geographical identifiers, campaign types, and various ids. We decided to use Vowpal Wabbit, a fast learning framework, to train models for this purpose.

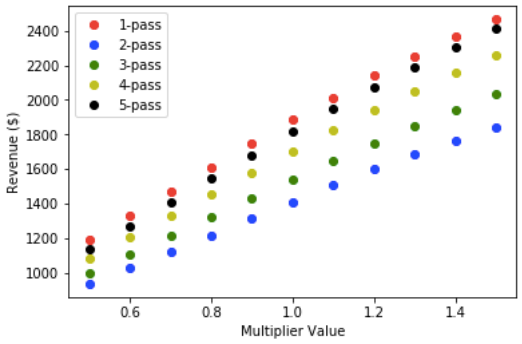
Process

Training models with Vowpal Wabbit required formatting data into trainable examples by stripping the relevant features from bids, then feeding them to the framework. At first, we trained a simple model with one pass through the data using only the features `campaign_id`, `site_id`, and `zone_id`. To simulate revenue with our model, we ran Vowpal Wabbit as a daemon process, which

allowed us to feed bidder features for an auction and retrieve a prediction for the highest bid price in real time. Initial results generated far more revenue than a static price floor of 0.0001 cents. As a result, we decided to experiment with using many more features as Boolean values, as well as trying models with up to 5 passes through the data.

Results

We trained models with 1-5 passes through data from February 11-13 and first tested multipliers from 0.5 to 1.5 with each model, obtaining the following results on the first 15 files of February 14th data:



Because the total revenue kept increasing, we decided to test on multipliers from 2 to 4 in increments of 0.5. We used the 1-pass model, as it gave the best results in the preliminary testing. The following revenues are based on the first 32 files of data on February 14th.

Multiplier	Revenue ($)
2.0	11,183.74
2.5	11,694.98
3.0	11,789.79
3.5	11,246.78
4.0	10,333.79

Below are the results from final testing.

```
-----------------------------------------
VW (Multiplier: 3.0)
-----------------------------------------
Total Revenue: 223703.68625066648
Auction Count: 518420532
Auction Count (non-null): 300064887
Price Floor Engaged (non-null): 43.89%
Price Floor Too High (non-null): 25.92%
Average Revenue: 0.0004315100819553699
Average Revenue (not-null): 0.0007455177061442263
Average Bid Count: 0.954231081650138
Average Bid Count (non-null): 1.648620036638942
Average Bid Amount: 0.0028150392288430067
```

```
Average Bid Amount (non-null): 0.004863528516142781
Average Price Floor: 0.008926980506974137
Time taken: 83064.05 seconds (0.160 milliseconds per auction)
```

Random Forest

Overview

Random forests are a simple ensemble of decision trees. We tested both a regression implementation and a classification implementation.

Regression

The first random forest we implemented posed the problem at hand as a regression problem. Given a bid, calculate a real number to act as the optimal price floor. The labeling for this tree was taken directly from auction theory; assuming the highest two bids to be uniformly distributed and setting the floor exactly halfway between them.

Classification

More commonly random forests are used to solve classification problems, and, as such, this is a much more researched area. In order to transform this inherently continuous problem into a discrete set of bins, k-means was applied to a single day's data on an hour by hour basis to produce a set of potential bins. Each auction was then labeled with the closest bin below it. Three different classification forests were trained, based on the minimum bins, average bins, and maximum bins among the hours.

Results

Each of the classification trees were unable to find any correlation among the data and underfitted accordingly. All of them would pick the same bin for each auction, minimum and average picking the smallest bin, and maximum picking the largest bin. This is likely due to the unevenness of the bins and lack of ability to train on a significant subset of the data. Regression produced a much more interesting result. When trained and evaluated over a small set of data, it consistently guessed above the optimal price floor about 82% of the time. Optimally, it should guess above the optimal floor about 50% of the time, but all of the over guesses were near to the optimal floor. Due to the slow speed of prediction, none of the forests were run through the standard simulator.

Running Average

Overview

A running average based approach generated the most revenue on our subset of the true bid distribution. The price floor was set by multiplying the average of recent top bids by a weight.

Modifications

We implemented several variations on an average, which can be found in `running_average/running_average.py`. In the following results, we compare the results of a global average versus a per-`site_id` average. Additionally, we explore the result of adding a zero to our average in the event of an empty bid response. Both the addition of the weight, and the practice of counting null auctions as zero-valued bids prevent price floors from reaching values that are too high.

Results

While the per-site_id running average with introduced zeros resulted in the greatest revenue, the per-`site_id` average resulted in moderately lower revenue but a much more promising price-floor engagement rate.

When null auctions were counted as zero-bids, the optimal weight was very close to 1. When null auctions were ignored, the optimal weight was very close to 0.5.

```
----------------------------------------
Global Running Average
----------------------------------------
Total Revenue: 485298.0741042547
Price Floor Engaged (non-null): 7.73%
Price Floor Too High (non-null): 92.27%
Average Revenue (not-null): 0.0016173104389393441
Average Price Floor: 0.02089341090786549
Time taken: 1598.51 seconds (0.003 milliseconds per auction)
----------------------------------------
Global Running Average with Zeros
----------------------------------------
Total Revenue: 633231.2406333139
Price Floor Engaged (non-null): 5.69%
Price Floor Too High (non-null): 94.31%
Average Revenue (not-null): 0.0021103143622176425
Average Price Floor: 0.039357196287724196
Time taken: 2126.88 seconds (0.004 milliseconds per auction)
----------------------------------------
Running Average by site_id
----------------------------------------
```

```
Total Revenue: 585769.6258170931
Price Floor Engaged (non-null): 35.06%
Price Floor Too High (non-null): 55.92%
Average Revenue (not-null): 0.0019521431903396717
Average Price Floor: 0.003797489056934269
Time taken: 1838.94 seconds (0.004 milliseconds per auction)
-----------------------------------------
Running Average by site_id with Zeros
-----------------------------------------
Total Revenue: 823332.5974897781
Price Floor Engaged (non-null): 6.11%
Price Floor Too High (non-null): 93.88%
Average Revenue (not-null): 0.0027438485246352004
Average Price Floor: 0.17501760124503202
Time taken: 2194.64 seconds (0.004 milliseconds per auction)
```

non-null *means that the originally empty auctions were ignored when calculating statistics*

Comparison of Results

The following results are from 518 million auctions from 02/13/18 12:00 AM - 2/15/18 12:00AM. 300 million auctions received at least one visible bid. 120 million auctions received at least two visible bids. Latency varies by machine, but all methods have good computational performance. Because most auctions in our faux distribution have fewer than two bids, the revenue from running a simulation with no price floor in place is very low.

Strategy	**Revenue**	**Price Floor Engagement**	**Price Floor Overshoot**	**Latency**
Running Average (per-site, with zeros)	$823,332	6.11%	93.88%	0.004 ms
Running Average (grouped by site)	$585,769	35.06%	55.92%	0.004 ms
OneShot	$339,306	5.27%	94.71%	0.017 ms
Weighted Average (grouped by site)	$424,276	30.63%	64.70%	0.002 ms
Vowpal Wabbit	$223,703	43.89%	25.92%	0.156 ms
No Price Floor	$49,862	61.56%	0.00%	0.001 ms

Conclusion and Recommendations

Final revenue varies moderately between approaches, but all strategies show promise. We recommend turning off static price floors temporarily so as to collect a distribution of bids that is more reflective of the truth. Strategies can then be tuned and simulated with this new distribution to obtain a more confident comparison. Alternatively, a comparison approach such as A/B testing could be used to evaluate the effectiveness of several strategies in production.

Contributors

Brian Levis
Daniel Grimshaw
Edric Xiang
Forest Hu
Nagaganesh Jaladanki
Hermish Mehta
Justin Lu
Nilay Khatore

We would like to thank Polymorph for working with us this semester, and for allowing us to tackle such an open-ended problem. This was an excellent opportunity for us as developers to grow together, and we appreciate your support of our organization.

References

[1] R. J. Garratt. *Auction Theory With Experiments*. Department of Economics, University of California at Santa Barbara, 2011.

[2] B. Chen S. Yuan, J. Wang. An empirical study of reserve price optimisation in real-time bidding. *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1897–1906, 2014.