Dynamic Price Floors
Final Deliverable

05 • 04 • 18

Team


Brian
Levis


Daniel
Grimshaw


Edric
Xiang


Forest
Hu


Ganesh
Jaladanki


Hermish
Mehta


Justin
Lu


Nilay
Khatore

Agenda

01 Introduction
02 Strategies
03 Results
04 Hand-off
05 Q & A

01 Introduction

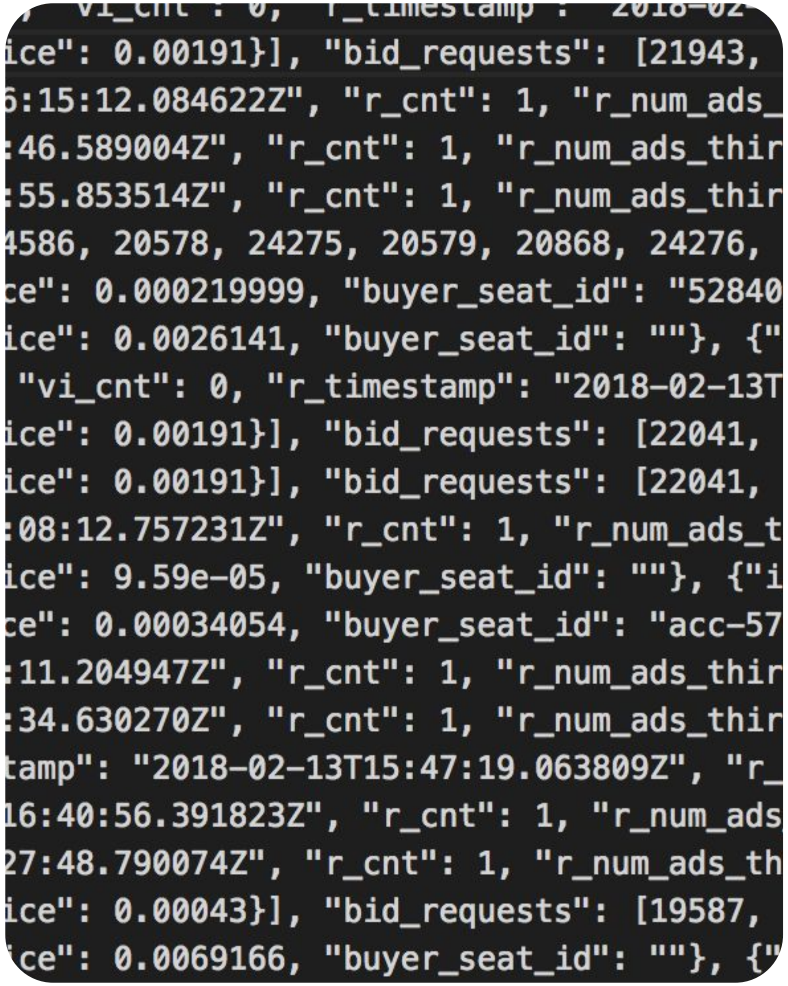
a. Recap
b. Methodology

Recap

Data

We had 5 days of data, containing roughly one billion RTB auctions

We investigated several strategies for setting price floors based off of incoming bids



Recap

Assumptions & Caveats

- We assumed we had access to every possible bid (if there was no price floor)
- We were unable to calculate true revenue or even a baseline revenue, but we were able to compare strategies in a meaningful way
- The vast majority of bid requests were not followed by bids (above the unknown price floor)

ice": 0.00191}], "bid_requests": [21943,
5:15:12.084622Z", "r_cnt": 1, "r_num_ads_
:46.589004Z", "r_cnt": 1, "r_num_ads_thir
:55.853514Z", "r_cnt": 1, "r_num_ads_thir
4586, 20578, 24275, 20579, 20868, 24276,
ce": 0.000219999, "buyer_seat_id": "52840
ice": 0.0026141, "buyer_seat_id": ""}, {"
"vi_cnt": 0, "r_timestamp": "2018-02-13T
ice": 0.00191}], "bid_requests": [22041,
ice": 0.00191}], "bid_requests": [22041,
:08:12.757231Z", "r_cnt": 1, "r_num_ads_t
ice": 9.59e-05, "buyer_seat_id": ""}, {"i
ce": 0.00034054, "buyer_seat_id": "acc-57
:11.204947Z", "r_cnt": 1, "r_num_ads_thir
:34.630270Z", "r_cnt": 1, "r_num_ads_thir
tamp": "2018-02-13T15:47:19.063809Z", "r_
16:40:56.391823Z", "r_cnt": 1, "r_num_ads
27:48.790074Z", "r_cnt": 1, "r_num_ads_th
ice": 0.00043}], "bid_requests": [19587,
ce": 0.0069166, "buyer_seat_id": ""}, {"

Simulator

- We tuned each strategy on 2/11-13, and tested on 2/14-15
- Each strategy must return a price floor based on a set of input features
- The bids *above this price floor* are then revealed to the strategy

```
@abstractmethod
def calculate_price_floor(self, input_features):
    pass

@abstractmethod
def process_line(self, line, input_features, bids):
    pass
```

```
INPUT_FEATURES = {
    'geo_dma_code', 'geo_timezone', 'ua_name', '
    'ua_device', 'r_num_ads_third_party', 'creat
    'r_num_ads_requested', 'rate_metric', 'geo_c
    'r_timestamp', 'ua_os', 'ua_device_type', 'ua
}
```

02

Pricing Strategies

a. OneShot
b. Weighted Running Average
c. Optimization
d. Vowpal Wabbit
e. Random Forest
f. Running Average

OneShot

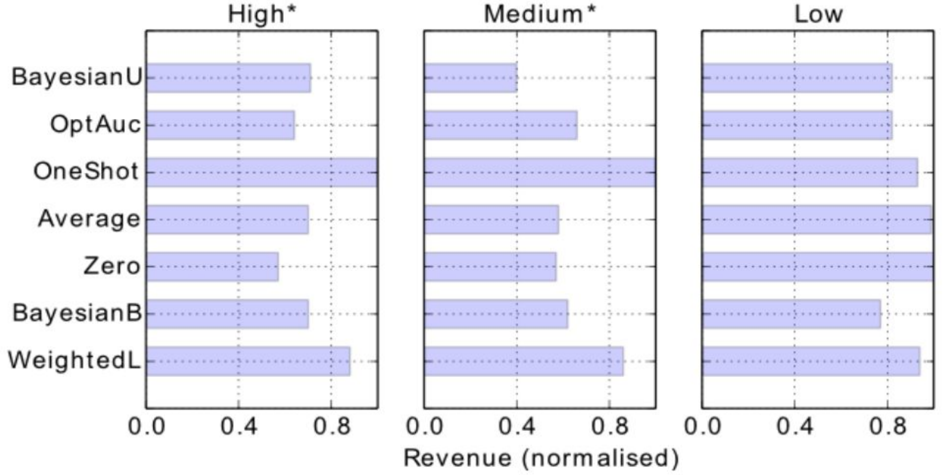
Summary

Dynamically adjust price floor based on previous bid

λ_h: Price floor too high
λ_e: Price floor between max and second bid (ideal)
λ_l: Price floor too low

Responsively evolves over time to produce results

$$\begin{cases} \alpha(t+1) = (1-\epsilon^t\lambda_h)a(t) & \text{if } \alpha(t) > b_1(t) \\ \alpha(t+1) = (1+\epsilon^t\lambda_e)a(t) & \text{if } b_1(t) \geq \alpha(t) \geq b_2(t) \\ \alpha(t+1) = (1+\epsilon^t\lambda_l)\alpha(t) & \text{if } b_2(t) > \alpha(t) \end{cases}$$



Improvements and Experiments

- Tuned hyperparameters for low overshoot rate
- Adaptive price floor ceiling
- Overshooting adjustments (λ_h)

```python
def update(self, bids, price_floor):
    first, second = self.max2(bids)
    revenue = self.calculate_revenue_helper(first, second, price_floor)
    diff = self.calculate_differential(first, second, price_floor)
    if len(bids) >= self.oneshot_min_n:
        self.oneshot(first, second, diff)
    else:
        if len(self.revenues) == 5:
            self.revenues.pop(0)
        self.revenues.append(revenue)
    return revenue
```

Results / Numbers

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%

Takeaways

Trade off between high revenue and low overshoot rate

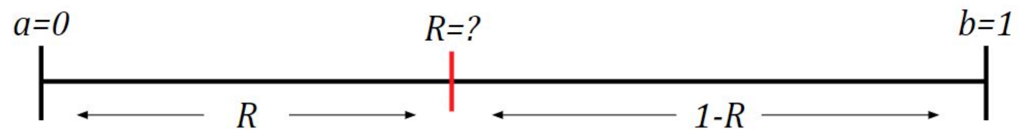
Biased towards revenue in lieu of overshoot

Multishot method is extremely promising for quick results

Weighted Running Average

Summary and Recap

- Derived optimal price floor for two uniformly distributed bidders
- Found optimal R=0.5, but tried weighing lower bid more to decrease overshoot rate
- Split averages by features, keeping separate averages for site_id, pub_network_id, etc.



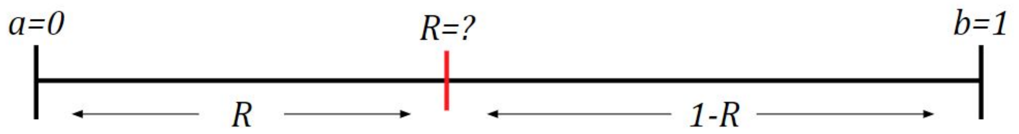
- no sale if both bids below R - happens with probability R^2 and revenue=0
- sale at price R if one bid above reserve and other below - happens with probability $2(1-R)R$ and revenue $= R$
- sale at second highest bid if both bids above reserve - happens with probability $(1-R)^2$ and revenue $= E[\min v_i | \min v_i \geq R] = \frac{1+2R}{3}$

- Expected revenue $= 2(1-R)R^2 + (1-R)^2 \frac{1+2R}{3}$
- Expected revenue $= \frac{1+3R^2-4R^3}{3}$
- Maximizing: $0 = 2R - 4R^2$, or $R = \frac{1}{2}$.

Weighted Running Average

Simulating Lost Bids

- In real-life auction, bids below the price floor are lost
- Lost bid information causes average strategy to strictly increase to unreasonable levels
- Simulate lost bidders by deriving expected ratio between first and second bid using original assumptions



$$\text{Expected high bid} \approx 0.66$$
$$\text{Expected low bid} \approx 0.33$$
$$\text{Ratio} = 0.66/0.33 = 0.5$$

$$\textbf{Simulated low bid} = 0.5 \cdot \text{high bid}$$

Weighted Running Average

Implementation

Parameters

- `window`: number of past auctions to incorporate into average
- `weight`: shift average toward low or high bidder
- `sim_bid`: multiplier for generating simulated second bid using first
- `param_id`: feature used to track separate running averages

```python
class AverageSingleID(sim.Simulator):
    """
    Only works for input features with a single value (e.g. works with site_id,
    pub_network_id, but not bid_requests).
    """

    def __init__(self, window, weight, sim_bid, param_id, **kwargs):
        sim.Simulator.__init__(self, **kwargs)
        self.window = window
        self.weight = weight
        self.simBid = sim_bid
        self.id = param_id
        self.averages = {}
        self.counts = {}

    def calculate_price_floor(self, input_features):
        key = input_features[self.id]
        if key in self.averages:
            return self.averages[key]
        else:
            return DEFAULT_FLOOR

    def process_line(self, line, input_features, bids):
        key = input_features[self.id]
        if key in self.averages:
            reserve = self.averages[key]
        else:
            # If site_id has never been seen before and auction has 0 bids
            reserve = DEFAULT_FLOOR

        low, high = 0, 0
        if len(bids) == 0:
            high = reserve
            low = high * self.simBid
        elif len(bids) == 1:
            high = bids[0]
            low = high * self.simBid
        else:
            high = bids[0]
            low = bids[1]

        weighted_avg = low * self.weight + high * (1 - self.weight)
        if key in self.averages:
            n = min(self.counts[key] + 1, self.window)
            self.averages[key] = self.averages[key] * (n - 1) / n + weighted_avg * 1.0 / n
            self.counts[key] += 1
        else:
            self.averages[key] = weighted_avg
```

Weighted Running Average

Results

window=500, weight=0.6, sim_bid=0.3

```
---------------------------------------------
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
```

Feature: site_id
Average Revenue: **$0.0014**
Engage / Too High: **30%** /
64%

```
---------------------------------------------
GT pub_network_id
---------------------------------------------
Total Revenue: 227656.77194918026
Auction Count: 518420532
Auction Count (non-null): 300064887
Price Floor Engaged (non-null): 25.24%
Price Floor Too High (non-null): 69.67%
Average Revenue: 0.00043913533106204264
Average Revenue (not-null): 0.000758691809045889
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: 0.0028786333704231702
```

Feature: pub_network_id
Average Revenue: **$0.0007**
Engage / Too High: **25%** /
69%

Optimization

Underlying Model

X, Y first and second highest bid distributions

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

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

set reserve price to maximize expected revenue

Optimization

Example

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

assume uniformly distributed bidders

$$X = \max(X_1, X_2)$$
$$Y = \min(X_1, X_2)$$

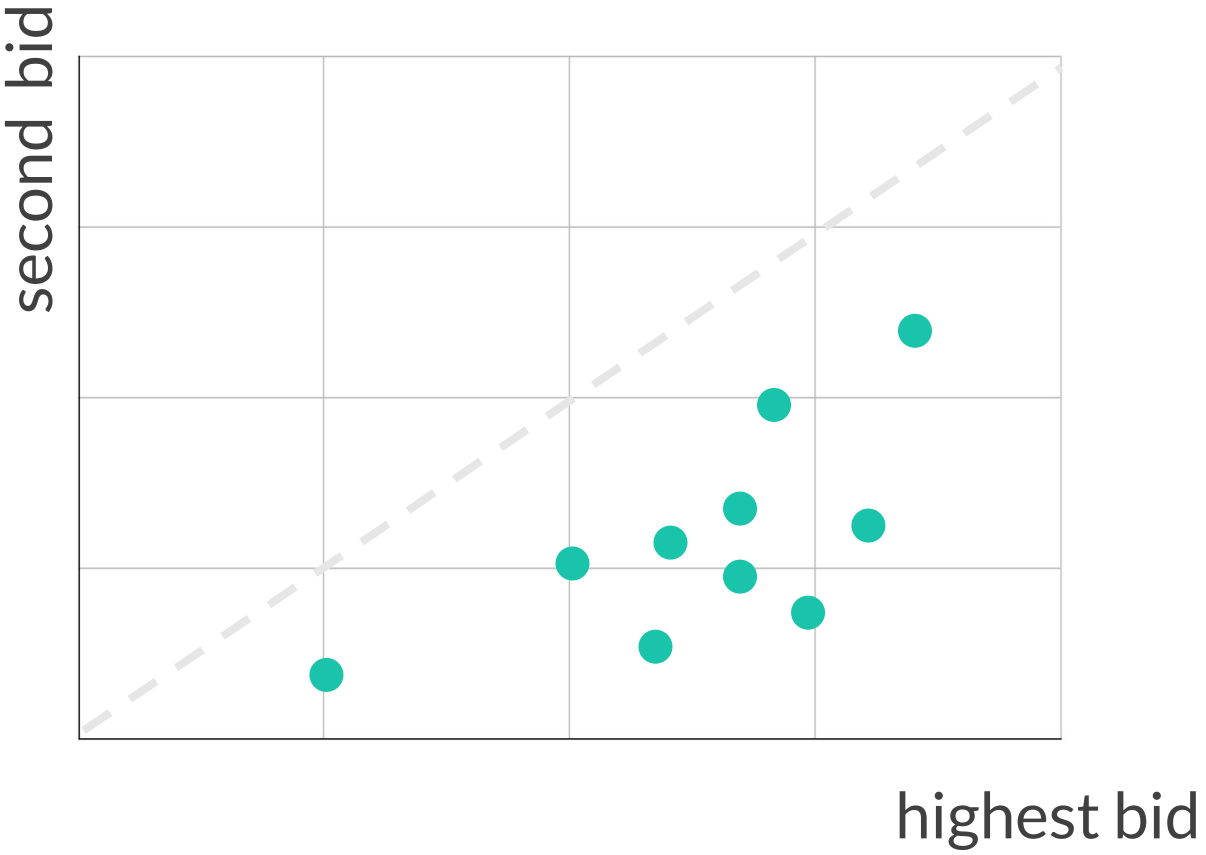
$$E(\text{rev}_r(X,Y)) = r^2(0) + 2r(1-r)(r) + (1-r)^2\frac{1+2r}{3}$$
$$= \frac{1+3r^2-4r^3}{3}$$

[function to maximize over *r*]

Optimization

Problem

- what are bid distributions like?
- more data, less structure principle



Solution

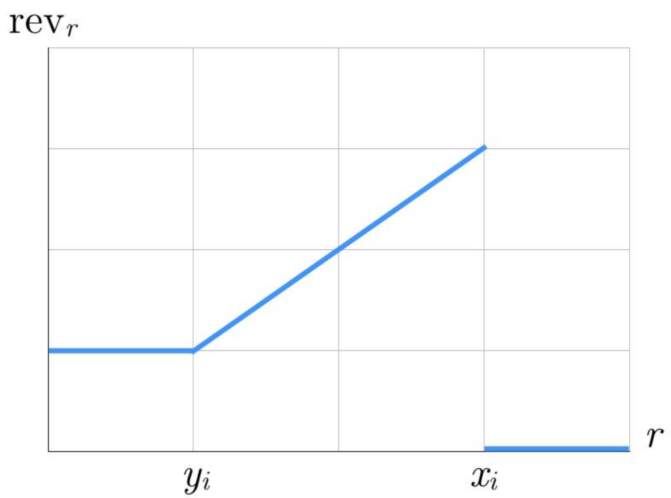
$$A_n = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$$

empirical joint distribution

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

finding the maximum of this function $f(r; A_n)$

Linear Heuristic

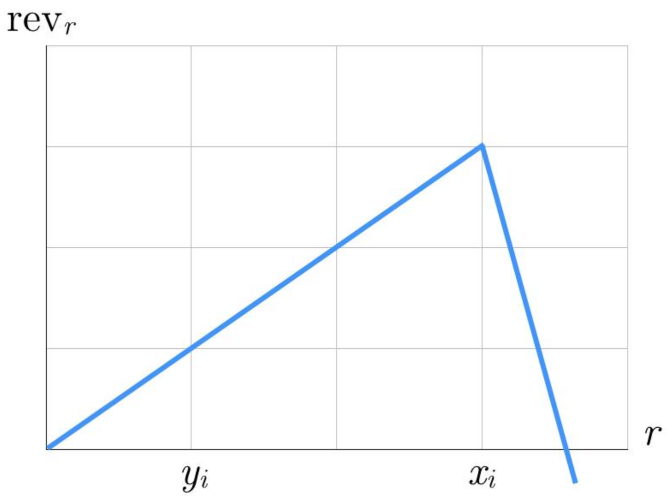


revenue function

$$\max \sum z_i$$
$$z_i \leq r$$
$$z_i \leq -\delta(r - x_i) + x_i$$
$$r \geq 0$$

return γ · r^*

algorithm [γ , □, n]

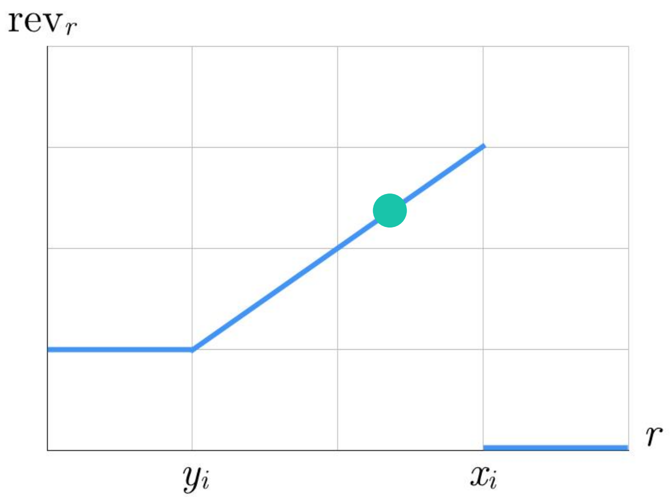


-approximation

Optimization

Brute Force

$$\frac{\mathrm{d}}{\mathrm{d}r} \sum_{i=1}^{n}$$



try all drop points

algorithm[γ, n]: return $\gamma \cdot \arg\max_{x_i} \sum_{t=1}^{n} \mathrm{rev}_r(x_t, y_t)$

Optimization Results

Tuning

linear heuristic[γ, □, *n*]: Fix *n* as 30, find γ and □ separately

brute force[γ, *n*]: Fix *n* as 100, find γ by varying

test on three files, best algorithms:

1. linear heuristic[1, -2, 30]
2. brute force[1, 100]

Optimization Results

Results

	Period	Price Floor Engaged (non-null auctions)	Price Floor Too High (non-null auctions)	Average Revenue (non-null auctions)	Average Price Floor	Latency
linear heuristic	2 minutes	6.25%	93.36%	$3.00 CPM	$48.34 CPM	25.6 ms
brute force	6 minutes	7.79%	92.20%	$3.69 CPM	$47.47 CPM	8.35 ms
weighted average	24 hours	5.69%	94.31%	$2.11 CPM	$39.36 CPM	0.004 ms

Vowpal Wabbit

Recap

- Predict optimal price floor using bidder features
- Strategy: machine learning framework to develop model
- Use multipliers to get as close to highest bid as possible

```
0.001910 | ua_device_iPhone site_id_11257 geo_country_
0.000020 | ua_device_iPhone site_id_11257 geo_country_
0.048544 | ua_device_Microsoft_RM-1092 site_id_17615 g
0.002515 | ua_device_Other site_id_17495 geo_country_c
0.000314 | ua_device_Other site_id_10361 geo_country_c
0.000101 | ua_device_Other site_id_31978 geo_country_c
0.000404 | ua_device_Other site_id_2512 geo_country_co
0.000278 | ua_device_Other site_id_22719 geo_country_c
0.000248 | ua_device_iPhone site_id_1978 geo_country_c
0.001456 | ua_device_Other site_id_3963 geo_country_co
```

Vowpal Wabbit

Implementation

Used Vowpal Wabbit: fast ML framework

Parameters

- 1-pass model: `campaign_id, site_id, zone_id`
- 1-5 pass models: hashed features
- Trained on Feb 11-13

`price_floor = multiplier *` b_1 `prediction`

VOWPAL WABBIT

```
prepared_line = sim.utils.prepare_line(line)
bids, input_features = prepared_line['bids'], prepared_line['input_features']
price_floor = self.calculate_price_floor(input_features, prepared_line, line_num,
if price_floor == None:
    break
price_floor *= multiplier
self.stats.process_line(bids, input_features, price_floor)
```

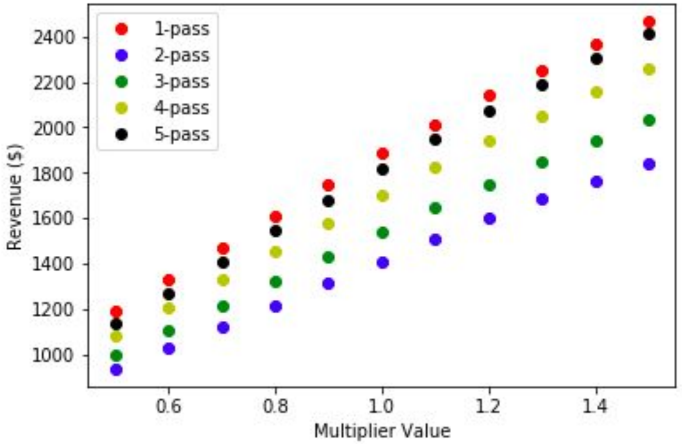
Vowpal Wabbit

Results

Tested on Feb 14, 12 AM
Multipliers ∈ [0.5, 1.5]
1-5 pass models

Higher multipliers ∈ [2.0, 4.0]
1-pass model

Optimum multiplier ≈ 3
Revenue (Feb 14-15): $223,703.77
Engagement: 43.89%



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

Takeaways

Disadvantages:

- Slower than other strategies
- Assumptions made
- Must train offline

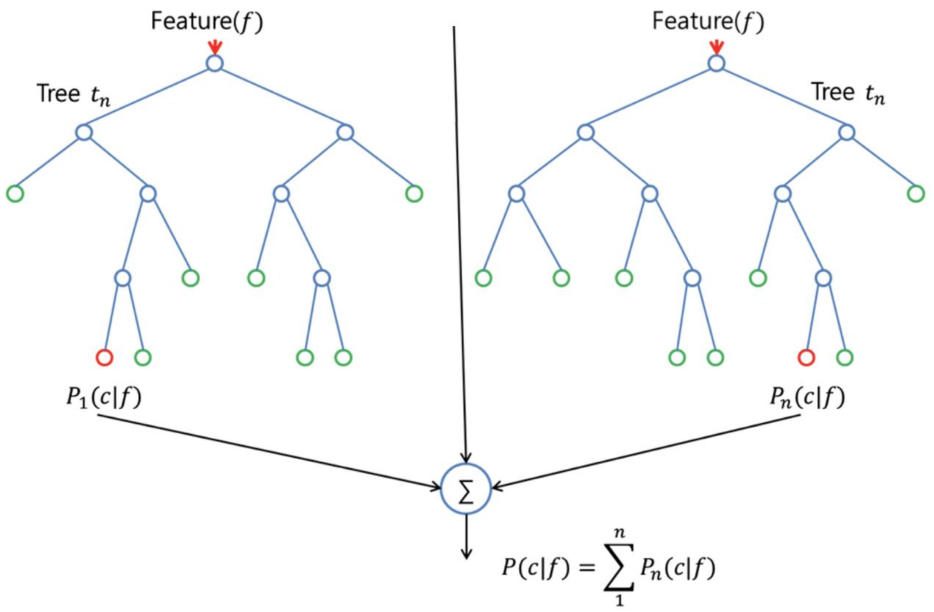
Advantage:

- Great potential with finely tuned model
- High price floor engagement
- Can be trivially parallelized

Random Forests

Classification

- Bids are distributed tri-modally
- k-means bins from 1 day of data
- RF selected one bin for all auctions



Random Forests

Regression

- Labels derived from game theory
- State space is large
- 83% of predictions above optimal

Running Average

Experiments

- Global running average
- Running average for each site_id or publisher_id
- Include auctions with no bids?

Running Average

Results

Auction Count: 518420532
Auction Count (non-null): 300064887
Average Bid Count: 0.954231081650138
Average Bid Count (non-null): 1.648620036638942
Average Bid Amount: 0.0028150392288430067
Average Bid Amount (non-null): 0.004863528516142781

Running Average Without Zeros

Total Revenue: 485298.0741042547
Price Floor Engaged (non-null): 7.73%
Price Floor Too High (non-null): 92.27%
Average Revenue: 0.0009361089003015311
Average Revenue (not-null): 0.0016173104389393441
Average Price Floor: 0.02089341090786549
Time taken: 1598.51 seconds (0.003 milliseconds per auction)

Running Average Including Zero-Bids

Total Revenue: 633231.2406333139
Price Floor Engaged (non-null): 5.69%
Price Floor Too High (non-null): 94.31%
Average Revenue: 0.0012214625030192937
Average Revenue (not-null): 0.0021103143622176425
Average Price Floor: 0.039357196287724196
Time taken: 2126.88 seconds (0.004 milliseconds per auction)

Per-Site Average

Total Revenue: 585769.6258170931
Price Floor Engaged (non-null): 35.06%
Price Floor Too High (non-null): 55.92%
Average Revenue: 0.0011299120880827558
Average Revenue (not-null): 0.0019521431903396717
Average Price Floor: 0.003797489056934269
Time taken: 1838.94 seconds (0.004 milliseconds per auction)

Per-Site Average Including Zero-Bids

Total Revenue: 823332.5974897781
Price Floor Engaged (non-null): 6.11%
Price Floor Too High (non-null): 93.88%
Average Revenue: 0.0015881558438927302
Average Revenue (not-null): 0.0027438485246352004
Average Price Floor: 0.17501760124503202
Time taken: 2194.64 seconds (0.004 milliseconds per auction)

03

Results

a. Comparison
b. Recommendations

Comparison of Best-Performing Strategies

Strategy	Revenue	Price Floor Engaged (non-null auctions)	Price Floor Too High (non-null auctions)	Average Revenue (non-null auctions)	Average Price Floor	Latency
Running Average (per site, with nulls)	$823,332	6.11%	93.88%	$2.74 CPM	$175.02 CPM	0.004 ms
Running Average (grouped by site)	$585,769	35.06%	55.92%	$1.95 CPM	$3.79 CPM	0.004 ms
OneShot	$339,306	5.27%	94.71%	$1.13 CPM	$29.85 CPM	0.017 ms
Weighted Average (grouped by site)	$424,276	30.63%	64.70%	$0.82 CPM	$2.43 CPM	0.002 ms
Vowpal Wabbit	$223,703	43.89%	25.92%	$0.75 CPM	$8.93 CPM	0.156 ms
No Price Floor	$49,862	61.56%	0.00%	$0.17 CPM	$0.00 CPM	0.001 ms

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 results show that these methods have good computational performance.

Recommendations

Implementation Suggestions

- Collect a testing dataset that disables price floors and can produce accurate revenue estimates
- Implement A/B testing or similar to compare approaches
- All strategies are promising, and will behave differently towards a real bid distribution

04

Hand-off

a. Materials
b. Next Steps

Materials

Report

Dynamic Price Floors

CodeBase

University of California, Berkeley

May 4, 2018

Introduction

Polymorph provides publishers the opportunity to set a price floor to protect

Materials

Code

exploratory	Rename directory	2 months ago
gametheory	site_id and pub_network_id results	12 hours ago
linear_optimization	Make tuning even simpler	11 hours ago
oneshot	cleaned up oneshot directory, edited tuning.py	12 minutes ago
randomForest	Add tensorflow version of random forest	2 months ago
randomForestModel	Add RFSimulator	a month ago
results	cleanup	2 hours ago
running_average	move running average to new package	3 hours ago
scripts	update file names	9 days ago
simulator	add per-auction time	2 days ago
vwprediction	Delete polytest	2 days ago

Next Steps

Maintenance

We will be available for two additional weeks to answer questions, fix issues, and help you reproduce our results or implement strategies!

05

Q & A

Q & A