Optimizing Ad Placement in Mobile Apps for Maximum Revenue

Team 11:
Waseem Khan
Zeinab Soumahoro
Madi Zhaksylyk
Uzma Rauf

Do you enjoy seeing ads in the mobile apps you use?

Problem Context

- Mobile App Market Dynamics
- Revenue Through Ads
- Trade-off between maximizing revenue and maintaining user engagement

Data Sources

- Real Historical revenue data for each ad type

  Source: Google Admob account report for the specific application.

- User engagement metrics from Google Analytics

Challenges and Assumptions

Challenges

1. hard to quantify the quality of user experience.

2. hard to isolate the effect of an add on user experience.

Assumptions:

1. Installs and uninstalls as a good proxy of user experience.

2. no other factor affecting the quality of user experience, except for the placement of ads.

Problem Statement

How to maximize revenue while ensuring quality user experience in mobile ap?

Proposed Solution

- dynamic programming solution addresses this by adjusting ad placements based on ongoing user engagement
- metrics and revenue data, ensuring a harmonious balance

Proposed Mathematical Formulation

Maximize the total revenue over the campaign period while considering user engagement

$$\text{Maximize } Z = \sum_{t=1}^{T} \sum_{i=1}^{N} R_i \times I_{it}$$

N: Number of ad types (e.g., Banner, Interstitial, App Open)

T: Total number of days for the ad campaign.

Iit : Number of impressions for ad type i on day t.

Ri : Average revenue per impression for ad type i

Constraints

1. Impression Adjustment Constraint

Ensures the number of impressions for each ad type is adjusted based on the disengagement score and does not exceed the base number of impressions significantly.

$$I_{it} \leq B_i \times (1 - D_i) \quad \forall i, \forall t$$

2. User Engagement Constraint

Ensures that cumulative disengagement score for each ad type should be updated based on the number of impressions and the disengagement decrease rate.

$$D_{i,t+1} = D_{it} + (D_i \times I_{it}) - (\lambda \times D_{it}) \quad \forall i, \forall t$$

3. Revenue Threshold Constraint

Ensures that the revenue for each ad type exceeds the minimum threshold.

$$R_i \times I_{it} \geq M \quad \forall i, \forall t$$

Implementation using Python

1. Prepare data

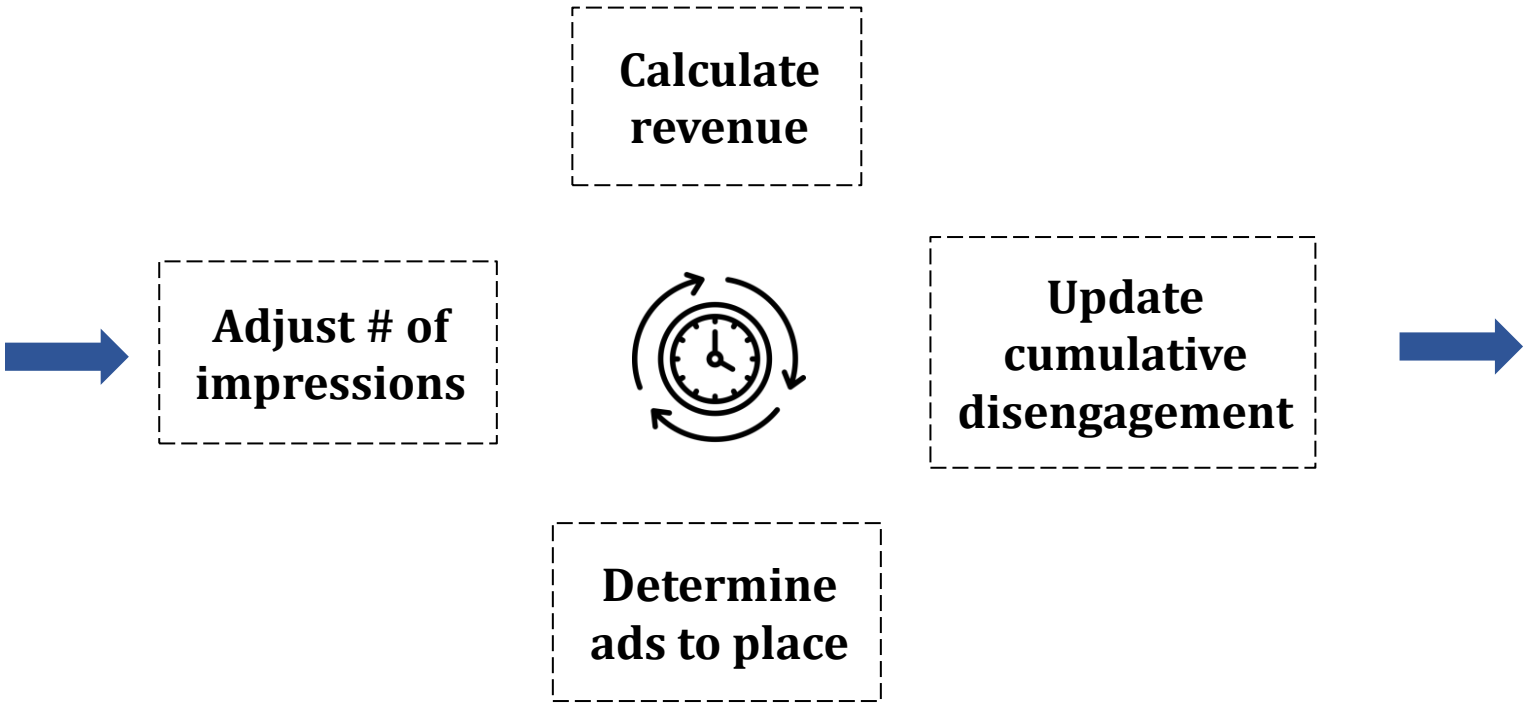
	format	earnings	impressions	clicks	cpm	new_installs	new_uninstalls	year	month	day	day_of_week
90	App open	10.63	1676	150	6.34	381	442	2023	11	14	1
20	Banner	7.18	2550	87	2.82	556	380	2023	10	17	1
35	Interstitial	3.85	604	35	6.38	495	480	2023	10	24	1
1	Banner	5.24	2148	77	2.44	405	344	2023	10	2	0
54	App open	5.48	892	117	6.14	529	410	2023	11	2	3
31	Interstitial	3.80	463	41	8.21	352	376	2023	10	22	6
116	Interstitial	3.11	792	50	3.93	702	437	2023	11	22	2
21	Interstitial	5.87	893	63	6.57	556	380	2023	10	17	1
24	Banner	9.87	2614	113	3.77	536	377	2023	10	19	3
104	Interstitial	1.16	409	30	2.84	361	380	2023	11	18	5

2. Initialize parameters

- base disengagement scores
- base impressions
- disengagement decrease rate
- min revenue threshold

Implementation using Python

3. Iterative optimization (over t days)



4. Output strategy

- Ads to place on day t
- For each ad type:
  - Net revenue
  - Impressions

Integration of Dynamic Programming

- OptiGuide is effective with Gurobi-style optimization problems, especially linear and integer optimization
- Since our solution involves dynamic programming, we needed to tailor OptiGuide source code for our needs:
  - Ad placement decisions over a temporal sequence
  - Custom input handling (dynamic inputs)
  - Algorithm customization

```python
# %% Constant strings to match code lines in the source code for AdPlacementOptimizer
DATA_CODE_STR = "# OPTIGUIDE OVERRIDE INPUT VALUES CODE GOES HERE"
ALGORITHM_CODE_STR = "# OPTIGUIDE ALGORITHM CODE GOES HERE"


# %%
class OptiGuideAgent(AssistantAgent):
    """Modified OptiGuide Agent for AdPlacementOptimizer."""

    def __init__(self, name, source_code, doc_str="", example_qa="", debug_times=3, **kwargs):
        """
        Initialize the agent with the source code of AdPlacementOptimizer.
        """
        assert source_code.find(DATA_CODE_STR) >= 0, "DATA_CODE_STR not found."
        assert source_code.find(ALGORITHM_CODE_STR) >= 0, "ALGORITHM_CODE_STR not found."
        super().__init__(name, **kwargs)
        self._source_code = source_code
        self._doc_str = doc_str
        self._example_qa = example_qa
        self._origin_execution_result = _run_with_exec(source_code)
        self._writer = AssistantAgent("writer", llm_config=self.llm_config)
        self._safeguard = AssistantAgent("safeguard", llm_config=self.llm_config)
        self._debug_times_left = self.debug_times = debug_times
        self._success = False
```

OptiGuide: Question 1

What is the best ad placement plan for the next 5 days?

```
writer (to Ad Optimization):

num_days = 5  # Number of days for the ad campaign
ad_placement_optimizer = AdPlacementOptimizer(num_days, ad_types, revenue_per_impression, disengagement_scores, base_impression
s, disengagement_decrease_rate, min_revenue)
ad_placement_optimizer.print_optimal_strategy()
```

Day	Ad Type	Net Revenue	Impressions
Day 1	Banner	$5.10	2954.0
	Interstitial	$1.00	547.0
	OpenAd	$4.98	1301.0
Day 2	Banner	$4.59	2658.0
	OpenAd	$4.48	1170.0
Day 3	Banner	$4.13	2392.0
	OpenAd	$4.03	1053.0
Day 4	Banner	$3.72	2152.0
	OpenAd	$3.62	947.0
Day 5	Banner	$3.34	1936.0
	OpenAd	$3.26	852.0

Total Revenue over 5 days: $42.25

OptiGuide Question 2:

Give me the ad placement strategy for the next 4 days if I add a new ad type with an average revenue of 0.002 per impression?

```python
num_days = 4  # Number of days for the ad campaign
ad_types.append("New Ad")  # Adding the new ad type to the list
revenue_per_impression.append(0.003)  # Average revenue per impression for the new ad type
disengagement_scores.append(0.001)
base_impressions.append(1000)

optimizer = AdPlacementOptimizer(num_days, ad_types, revenue_per_impression, disengagement_scores, base_impressions, disengagem
ent_decrease_rate, min_revenue)

optimizer.print_optimal_strategy()
```

Day	Ad Type	Net Revenue	Impressions
Day 1	Banner	$5.10	2954.0
	Interstitial	$1.00	547.0
	OpenAd	$4.98	1301.0
	New Ad	$1.80	900.0
Day 2	Banner	$4.59	2658.0
	OpenAd	$4.48	1170.0
	New Ad	$1.62	810.0
Day 3	Banner	$4.13	2392.0
	OpenAd	$4.03	1053.0
	New Ad	$1.46	729.0
Day 4	Banner	$3.72	2152.0
	OpenAd	$3.62	947.0
	New Ad	$1.31	656.0

Total Revenue over 4 days: $41.84

OptiGuide Question 3:

What is the best ad placement strategy for the next 5 days if the minimum revenue threshold is set to 3 instead of 1?

```
--------------------------------------------------------------------------------
writer (to Ad Optimization):

num_days = 5  # Number of days for the ad campaign
min_revenue = 3.0  # Updated minimum revenue threshold for showing an ad

# Creating an instance of AdPlacementOptimizer with specified parameters
optimizer = AdPlacementOptimizer(num_days, ad_types, revenue_per_impression, disengagement_scores, base_impressions, disengagem
ent_decrease_rate, min_revenue)

optimizer.print_optimal_strategy()  # Executing the ad placement strategy to get the optimal ad placement strategy and total re
venue

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

Day	Ad Type	Net Revenue	Impressions
Day 1	Banner	$5.10	2954.0
	OpenAd	$4.98	1301.0
Day 2	Banner	$4.59	2658.0
	OpenAd	$4.48	1170.0
Day 3	Banner	$4.13	2392.0
	OpenAd	$4.03	1053.0
Day 4	Banner	$3.72	2152.0
	OpenAd	$3.62	947.0
Day 5	Banner	$3.34	1936.0
	OpenAd	$3.26	852.0

Total Revenue over 5 days: $41.25

OptiGuide Question 4:

What is the best ad placement strategy for the next 10 days if the initial disengagement score for interstitial ad type was 0.006 instead of 0.003?

```
--------------------------------------------------------------------------------
writer (to Ad Optimization):

num_days = 10  # Number of days for the ad campaign
ad_types = ["Banner", "Interstitial", "OpenAd"]  # Types of ads
# Average revenue per impression for each ad type
revenue_per_impression = [banner_revenue_avg_per_imp, interstitial_revenue_avg_per_imp, app_open_revenue_avg_per_imp]
disengagement_scores = [0.001, 0.006, 0.002]  # Disengagement scores for each ad type
# Average base impressions for each ad type
base_impressions = [banner_impressions_avg, interstitial_impressions_avg, app_open_impressions_avg]
disengagement_decrease_rate = 0.1  # Rate at which disengagement decreases over time
min_revenue = 1.0  # Minimum revenue threshold for showing an ad

optimizer = AdPlacementOptimizer(num_days, ad_types, revenue_per_impression, disengagement_scores, base_impressions, disengagem
ent_decrease_rate, min_revenue)
optimizer.print_optimal_strategy()
```

Day	Ad Type	Net Revenue	Impressions
Day 1	Banner	$5.10	2954.0
	OpenAd	$4.98	1301.0
Day 2	Banner	$4.59	2658.0
	OpenAd	$4.48	1170.0
Day 3	Banner	$4.13	2392.0
	OpenAd	$4.03	1053.0
Day 4	Banner	$3.72	2152.0
	OpenAd	$3.62	947.0
Day 5	Banner	$3.34	1936.0
	OpenAd	$3.26	852.0
Day 6	Banner	$3.01	1742.0
	OpenAd	$2.93	766.0
Day 7	Banner	$2.71	1567.0
	OpenAd	$2.64	689.0
Day 8	Banner	$2.44	1410.0
	OpenAd	$2.37	620.0
Day 9	Banner	$2.19	1269.0
	OpenAd	$2.13	558.0
Day 10	Banner	$1.97	1142.0
	OpenAd	$1.92	502.0

Total Revenue over 10 days: $65.55

OptiGuide Question 5:

What would the ad placement for next 7 days look like if we take out interstitial ad?

```
--------------------------------------------------------------------------------
writer (to Ad Optimization):

```python
# Adjust the number of days and remove the "Interstitial" ad type from the ad_types list

num_days = 7  # Number of days for the ad campaign
ad_types.remove("Interstitial")  # Remove the "Interstitial" ad type from the list

# Continue with the rest of the code...
```
This code snippet will adjust the number of days for the ad campaign to 7 and remove the "Interstitial" ad type from the list o
f ad types. You can continue with the rest of the code to get the optimal ad placement strategy for the next 7 days.
```

Day	Ad Type	Net Revenue	Impressions
Day 1	Banner	$5.10	2954.0
	OpenAd	$1.00	547.0
Day 2	Banner	$4.59	2658.0
Day 3	Banner	$4.13	2392.0
Day 4	Banner	$3.72	2152.0
Day 5	Banner	$3.34	1936.0
Day 6	Banner	$3.01	1742.0
Day 7	Banner	$2.71	1567.0

Total Revenue over 7 days: $27.60

Conclusion and Future Work

- Our dynamic programming approach to ad placement in Android apps is designed to optimize revenue while enhancing user experience. By adopting this model, app developers and marketers can expect a more balanced and effective ad strategy, leading to increased profitability and user satisfaction

- OptiGuide:
  - Designed for Gurobi-style (linear, non-linear, integer optimization problems)
  - Need to make it more generic for a set of optimization problems like dynamic programming
  - Need to ask specific questions

Thank you