Bayesian Optimization for Balancing Metrics in Recommender Systems

IJCAI Tutorial 2020





Yunbo Ouyang
Sr. Software Engineer

Brendan Gavin
Software Engineer

Cyrus DiCiccio
Sr. Software Engineer

Viral Gupta
Tech Lead Comms AI

Kinjal Basu
Staff Software Engineer

Overall Agenda

1. Introduction to Bayesian Optimization
2. Reformulating a Recommender System
3. Practical Considerations
4. Infrastructure
5. Future Direction

Introduction to Bayesian Optimization

Yunbo Ouyang
Sr. Software Engineer

Brendan Gavin
Software Engineer

Introduction to Bayesian Optimization

1. Problem Setup
2. Gaussian Processes
3. Optimization with GP's
4. Neural Architecture Search
5. Demo

Why Bayesian Optimization?

E.g. in machine learning: optimizing hyperparameters

$$f(v; w, x)$$

f(): F1, precision, recall, accuracy, etc

Why Bayesian Optimization?

E.g. in machine learning: optimizing hyperparameters

$$\underset{\text{parameters}}{f(\overset{\text{feature vector}}{v}; w, \overset{\text{hyperparameters}}{x})}$$

f(): F1, precision, recall, accuracy, etc

parameters: neural network weights, decision tree features to split on

hyperparameters: learning rate, classification threshold

Why Bayesian Optimization?

E.g. in machine learning: optimizing hyperparameters

feature vector

hyperparameters

$$f(v; w, x)$$

parameters

parameters: fast/efficient to optimize

hyperparameters: slow/resource intensive to optimize

Why Bayesian Optimization?

E.g. in machine learning: optimizing hyperparameters

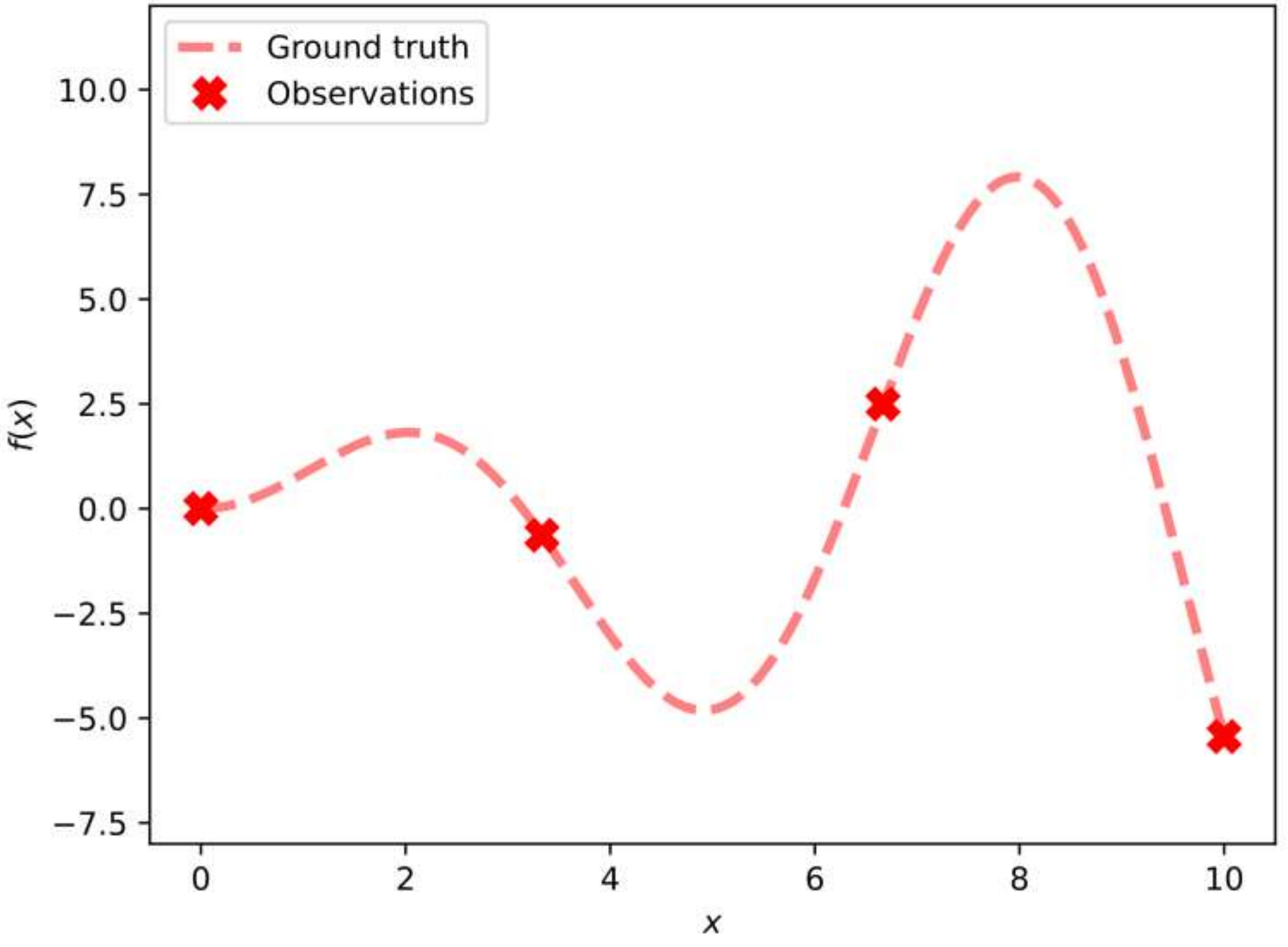
Want to optimize $f(x)$ with a method that:

- minimizes function evaluations
- can optimize black box $f(x)$
- quantifies uncertainty in solution

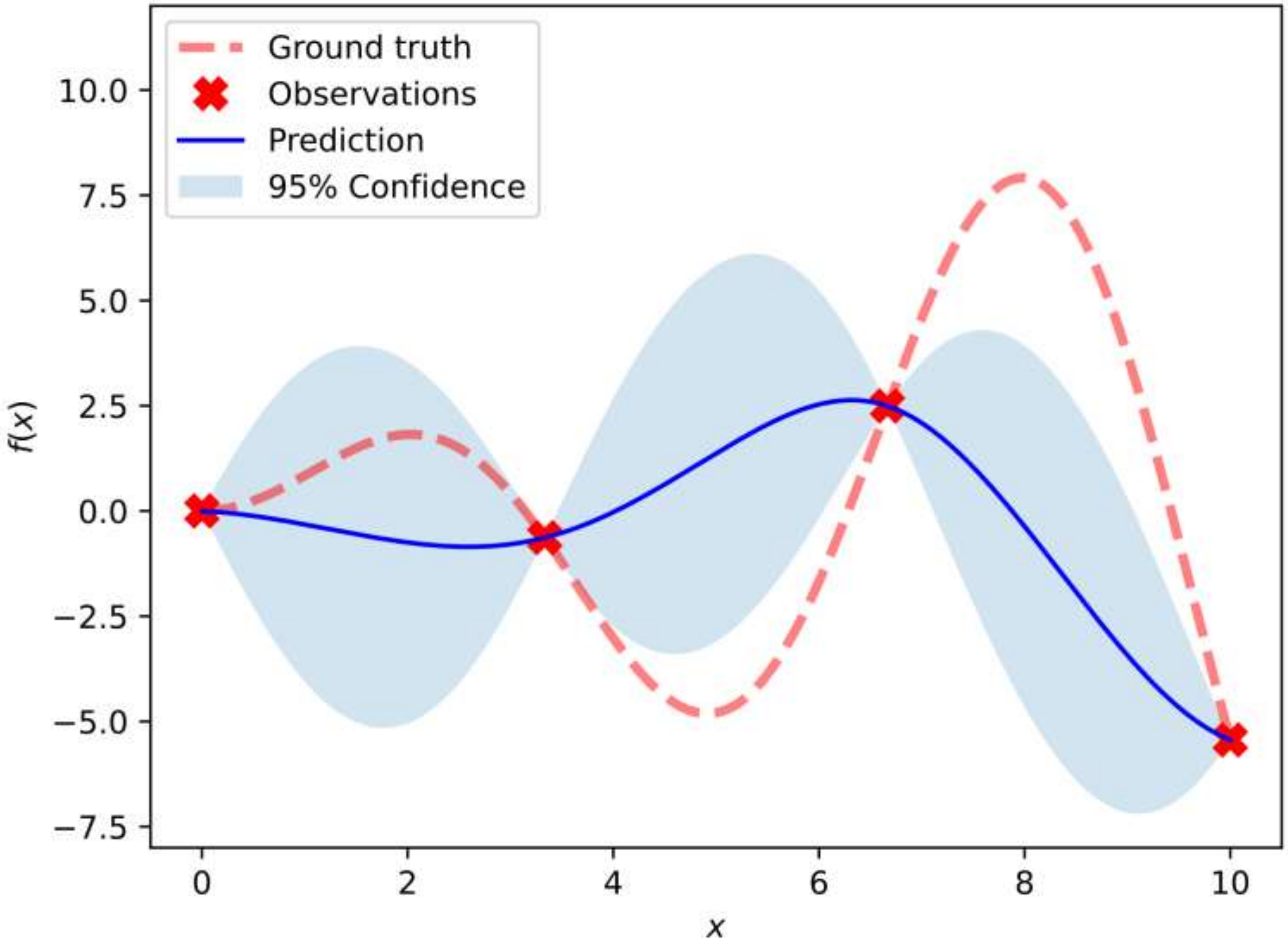
Bayesian optimization

- model $f(x)$ as a Gaussian random process
- fit model to data
- use uncertainty quantification to inform search for optimal x

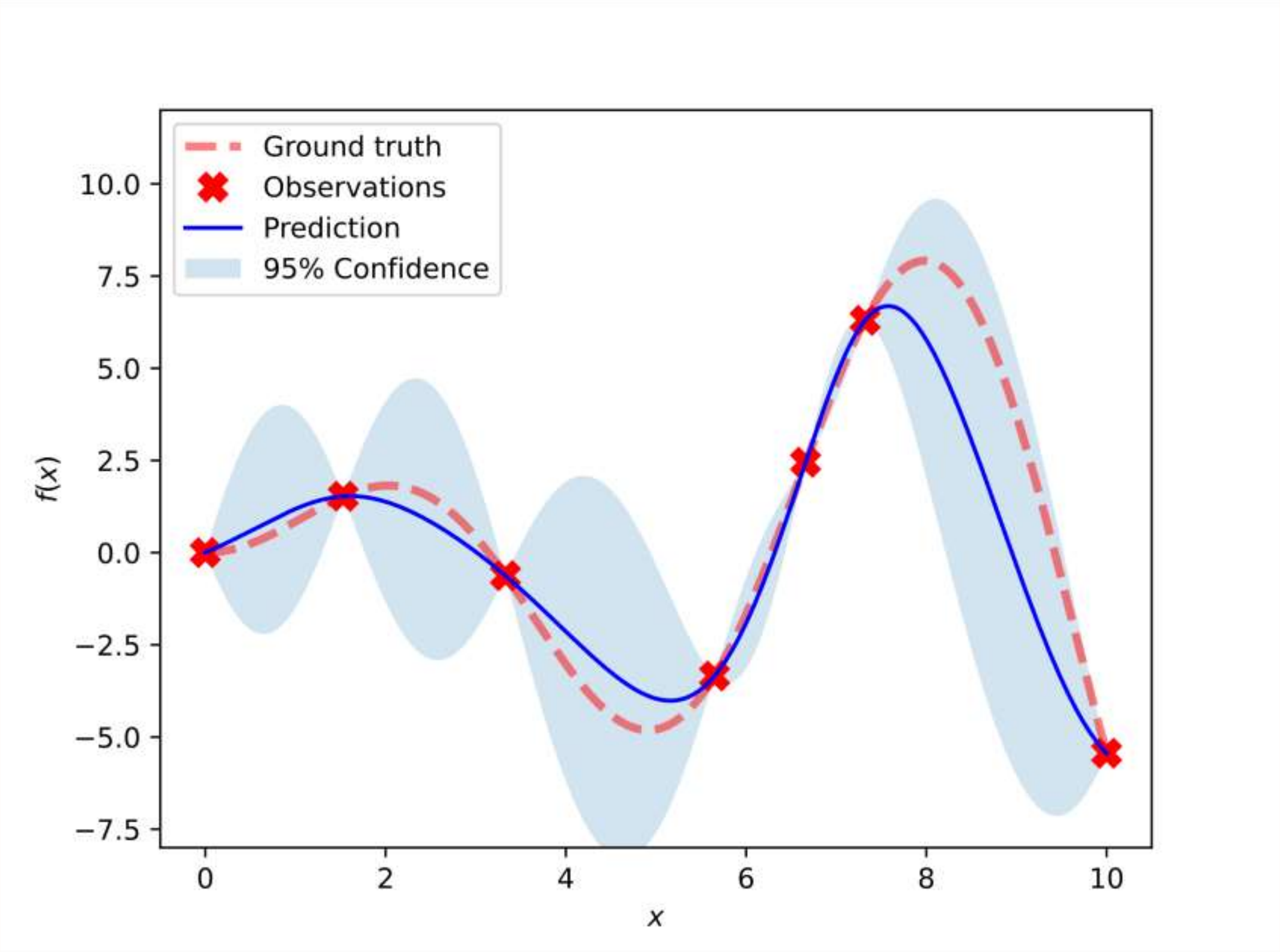
Bayesian optimization



Bayesian optimization



Bayesian optimization



Gaussian Processes

Gaussian Processes: distributions for sampling functions

Gaussian random variable

$$f \sim N(\mu, \sigma)$$

Gaussian Processes: distributions for sampling functions

Gaussian random variable

$$f \sim N(\mu, \sigma)$$

Gaussian multivariate RV

$$\begin{bmatrix} f_1 \\ f_2 \\ \vdots \end{bmatrix} \sim N\left(\begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \end{bmatrix}, \begin{bmatrix} K_{11} & K_{12} & \dots \\ K_{21} & K_{22} & \dots \\ \vdots & \vdots & \ddots \end{bmatrix}\right)$$

$$K_{ij} = cov(f_i, f_j)$$

Gaussian Processes: distributions for sampling functions

Gaussian random variable

$$f \sim N(\mu, \sigma)$$

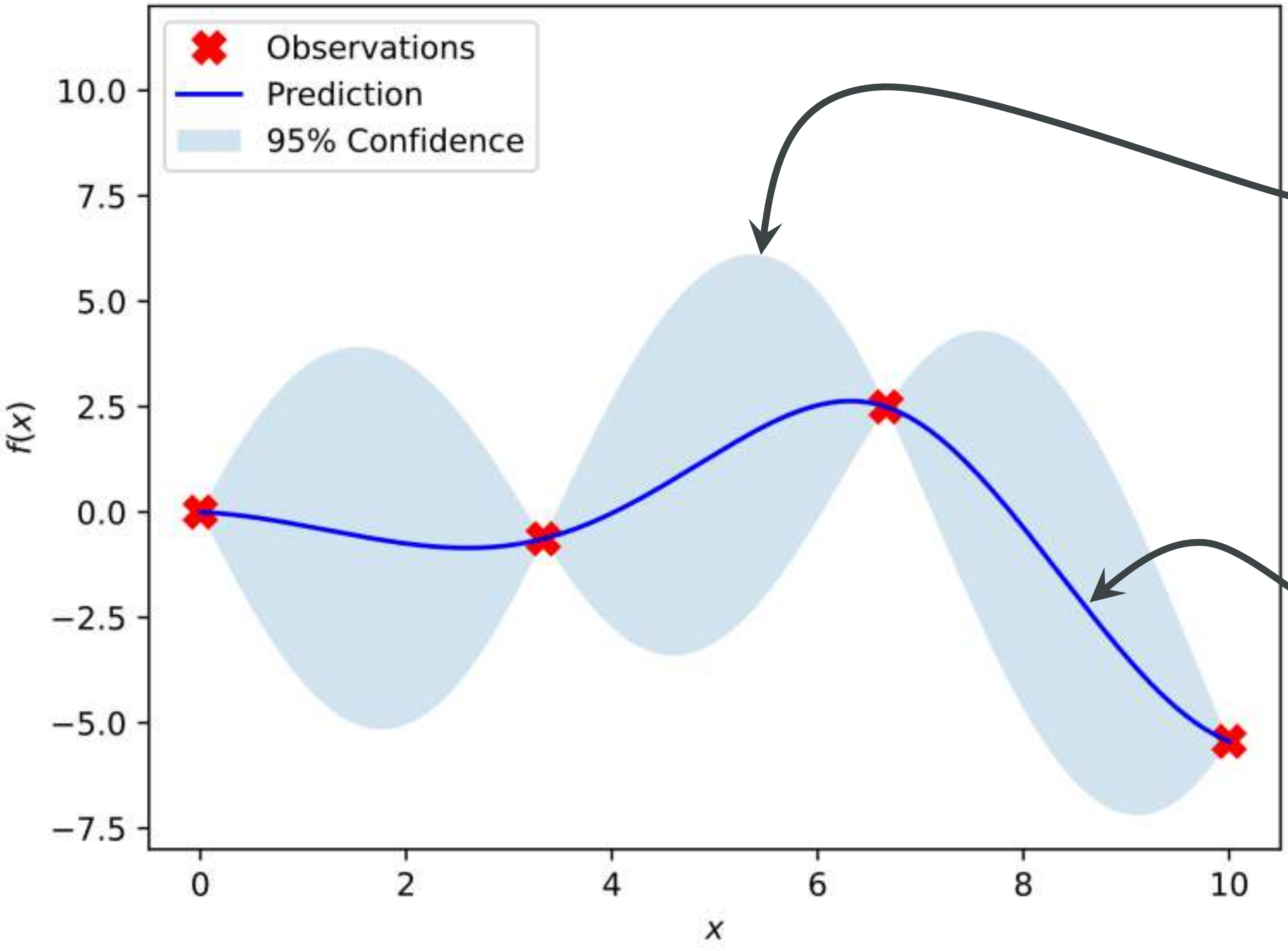
Gaussian multivariate RV

$$\begin{bmatrix} f_1 \\ f_2 \\ \vdots \end{bmatrix} \sim N\left(\begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \end{bmatrix}, \begin{bmatrix} K_{11} & K_{12} & \dots \\ K_{21} & K_{22} & \dots \\ \vdots & \vdots & \ddots \end{bmatrix} \right) \qquad K_{ij} = cov(f_i, f_j)$$

Gaussian process

$$f(x) \sim N(\mu(x), K(x, x')) \qquad K(x, x') = cov(f(x), f(x'))$$

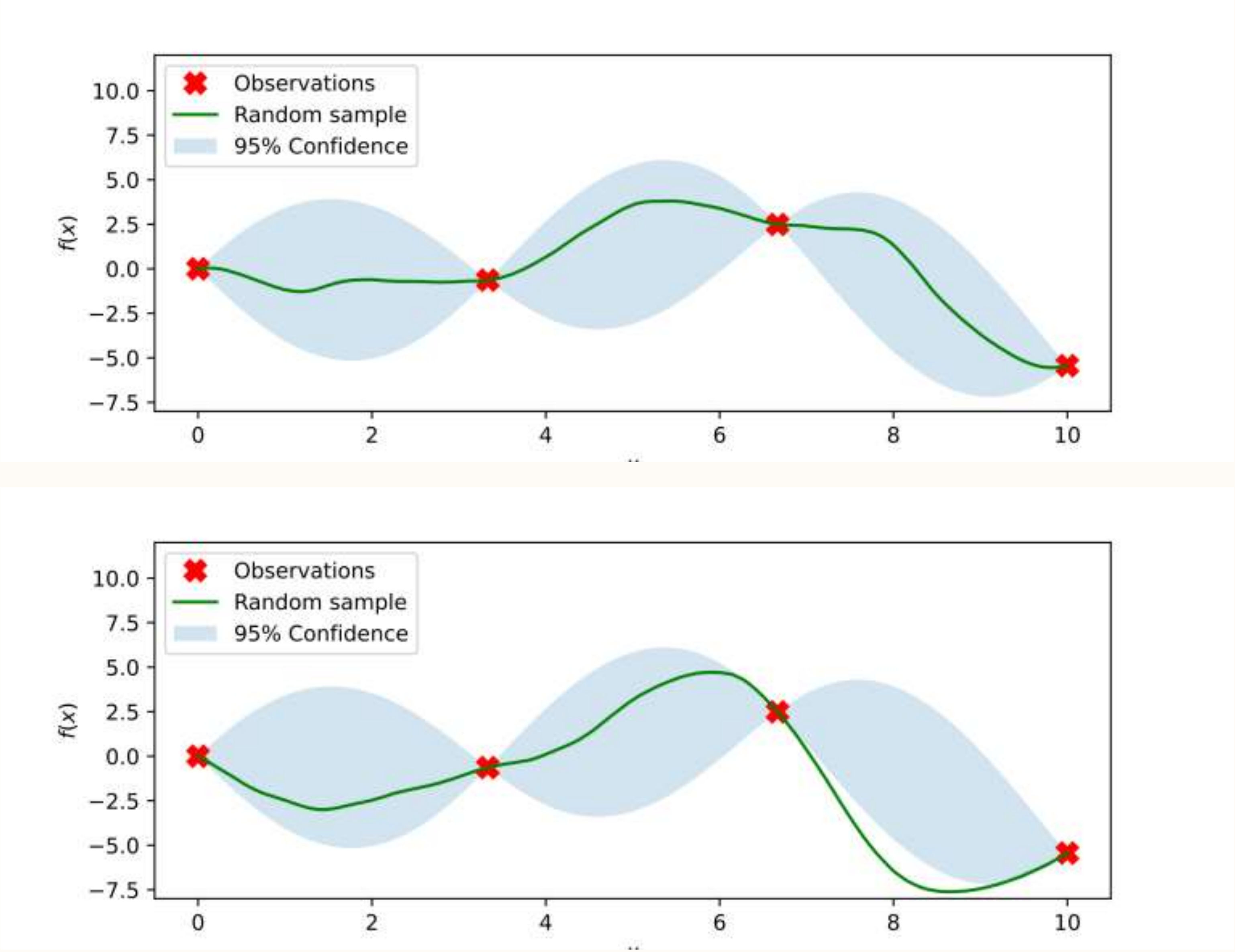
Gaussian Processes: distributions for sampling functions



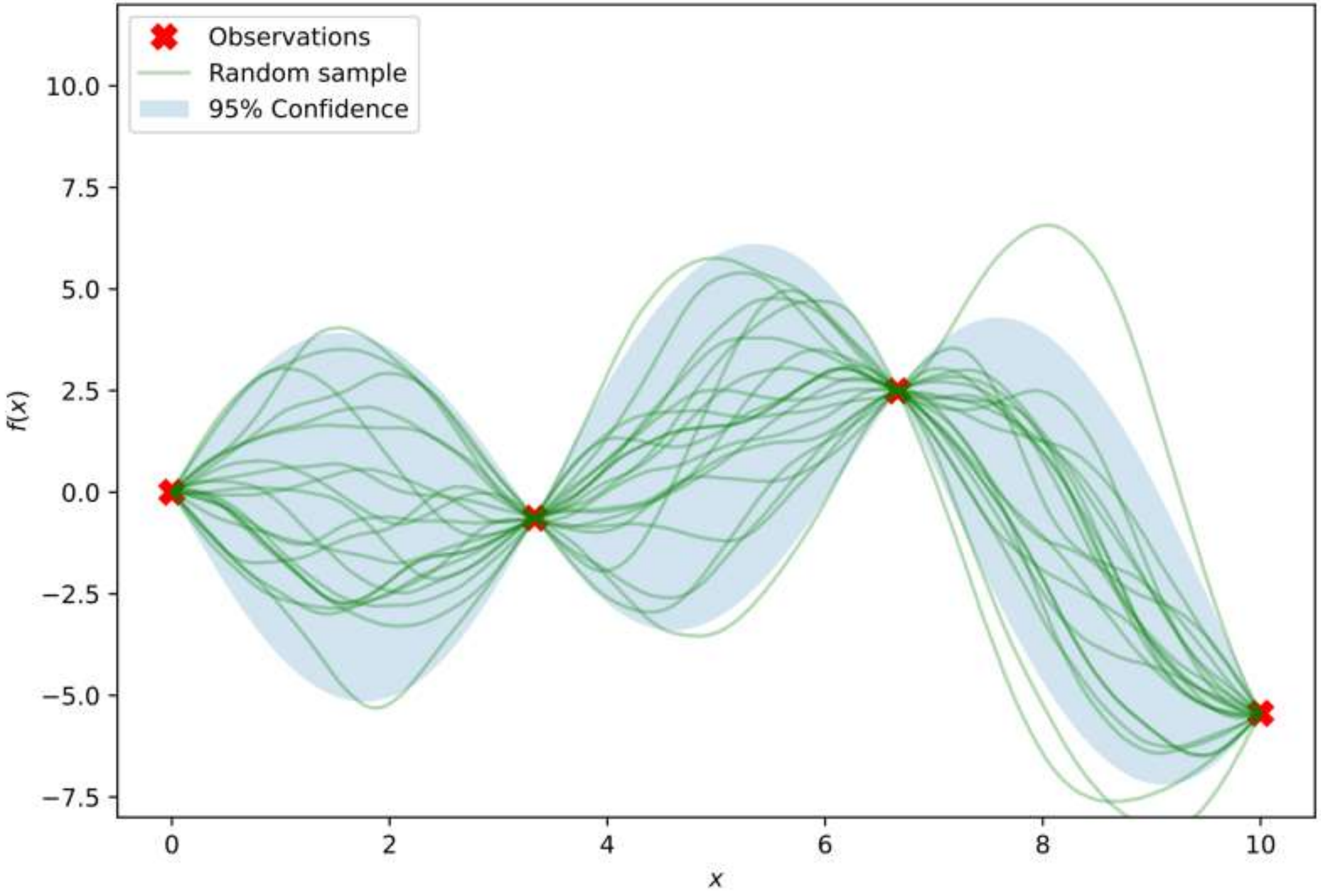
95% confidence from $K(x, x)$

Mean function $\mu(x)$

Gaussian Processes: distributions for sampling functions

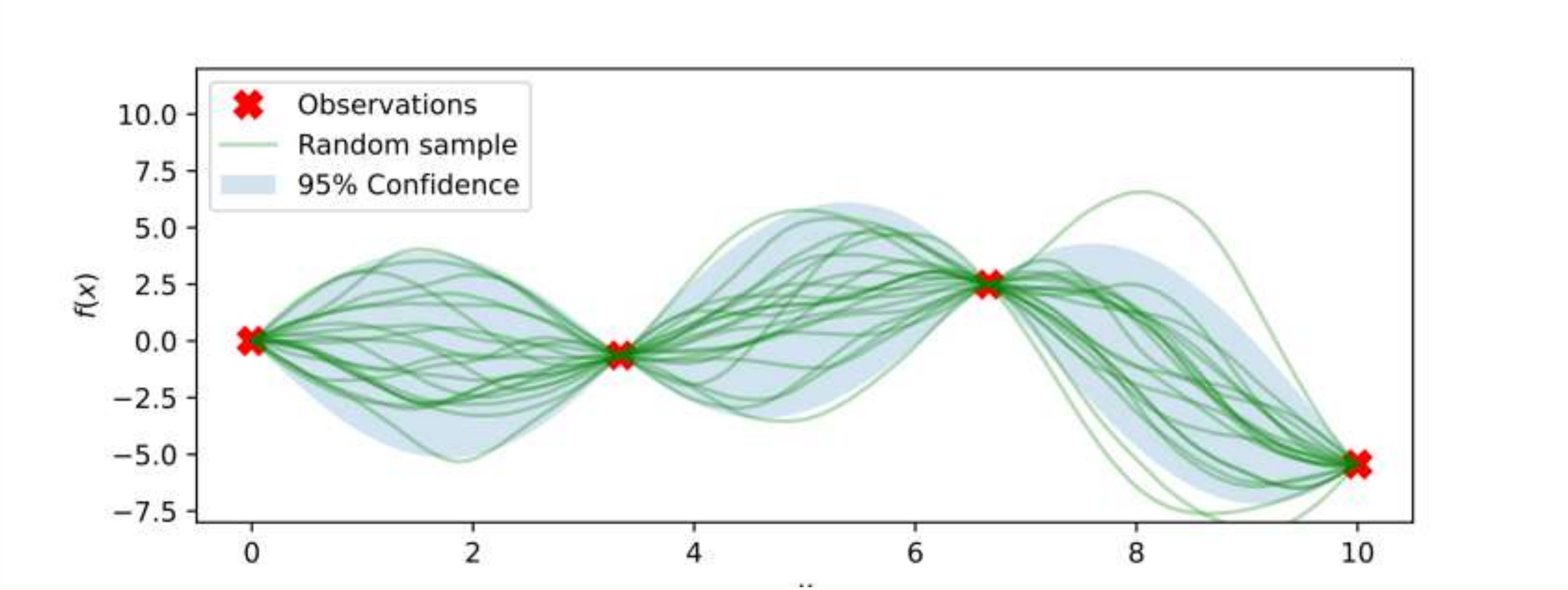


Gaussian Processes: distributions for sampling functions

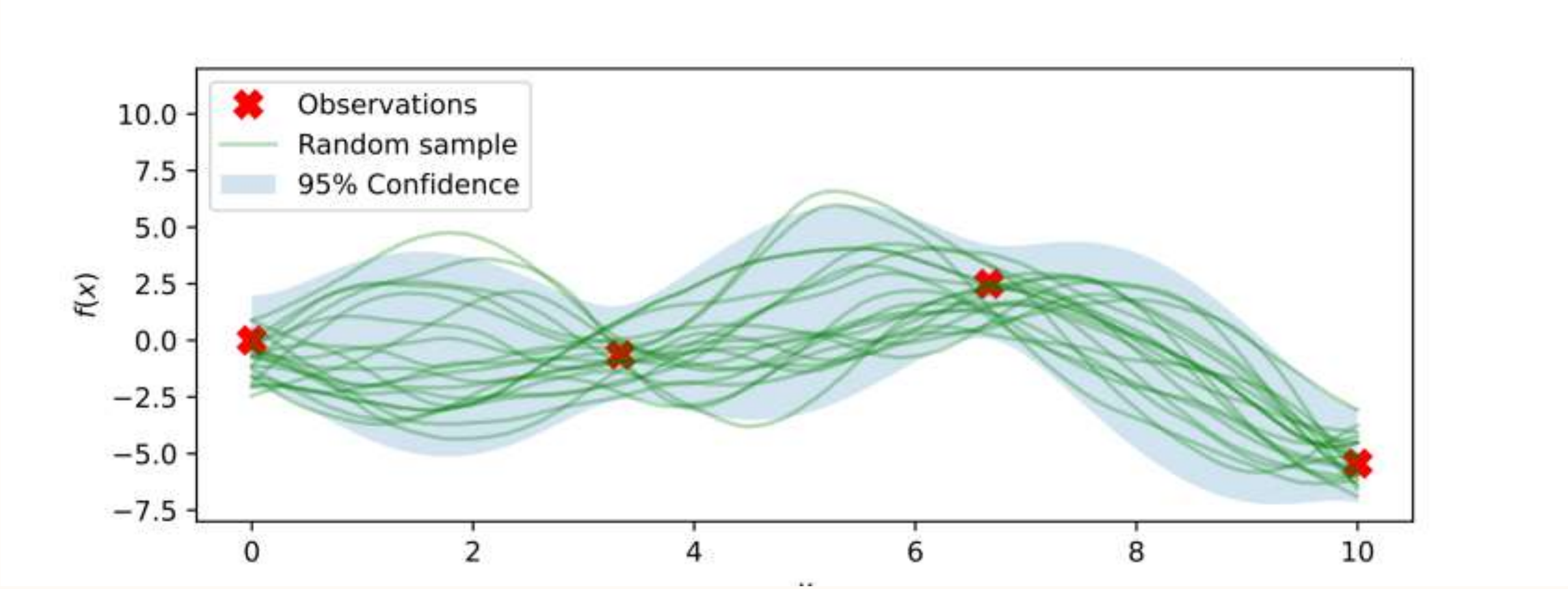


Gaussian Processes: distributions for sampling functions

Noiseless Measurements



Measurements With Noise



Gaussian Processes: distributions for sampling functions

Multitask Gaussian processes

$$\begin{bmatrix} f_1(x_1) \\ f_2(x_2) \\ \vdots \end{bmatrix} \sim N\left(\begin{bmatrix} \mu_1(x_1) \\ \mu_2(x_2) \\ \vdots \end{bmatrix}, \begin{bmatrix} K_{11}(x_1, x_1') & K_{12}(x_1, x_2') & \dots \\ K_{21}(x_2, x_1') & K_{22}(x_2, x_2') & \dots \\ \vdots & \vdots & \ddots \end{bmatrix} \right)$$

Gaussian Processes: distributions for sampling functions

Multitask Gaussian processes

$$\begin{bmatrix} f_1(x_1) \\ f_2(x_2) \\ \vdots \end{bmatrix} \sim N\left(\begin{bmatrix} \mu_1(x_1) \\ \mu_2(x_2) \\ \vdots \end{bmatrix}, \begin{bmatrix} K_{11}(x_1, x_1') & K_{12}(x_1, x_2') & \dots \\ K_{21}(x_2, x_1') & K_{22}(x_2, x_2') & \dots \\ \vdots & \vdots & \ddots \end{bmatrix} \right)$$

ICM approximation: $K_{ij}(x_i, x_j') \approx b_{ij} K(x_i, x_j')$

Optimization with Gaussian Processes

Optimization with Gaussian processes

The basic recipe:

1. Choose kernel and mean functions

$$\mu(x) \qquad K(x, x')$$

2. Sample function $f(x)$ at several points x

3. Use maximum likelihood to fit GP parameters

4. Are we done? If not, select new points to sample from; GOTO step 3

Choosing mean and kernel functions

Mean function: usually $\mu(x) = 0$

Choosing mean and kernel functions

Mean function: usually $\mu(x) = 0$

$$\mu_{new} = E[f_{new}|f_{old}] = K_{new/old}K_{old/old}^{-1}f_{old}$$

Choosing mean and kernel functions

Mean function: usually $\mu(x) = 0$

Kernel function: usually $K(x, x') = K(||x - x'||)$

Choosing mean and kernel functions

Mean function: usually $\mu(x) = 0$

Kernel function: usually $K(x, x') = K(||x - x'||)$

RBF Kernel

$$K(x, x') = \alpha \exp\left(-\frac{||x - x'||^2}{2\sigma^2}\right)$$

Choosing mean and kernel functions

Mean function: usually $\mu(x) = 0$

Kernel function: usually $K(x, x') = K(||x - x'||)$

RBF Kernel

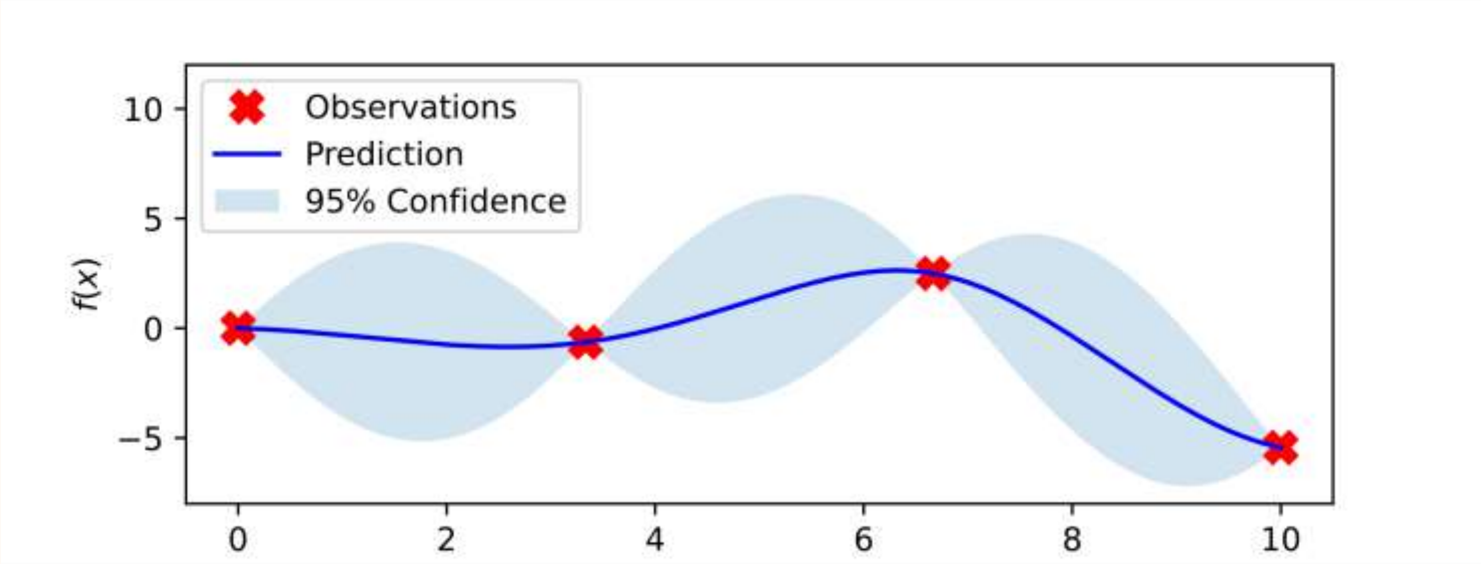
$$K(x, x') = \alpha \exp\left(-\frac{||x - x'||^2}{2\sigma^2}\right)$$

Matern Kernel

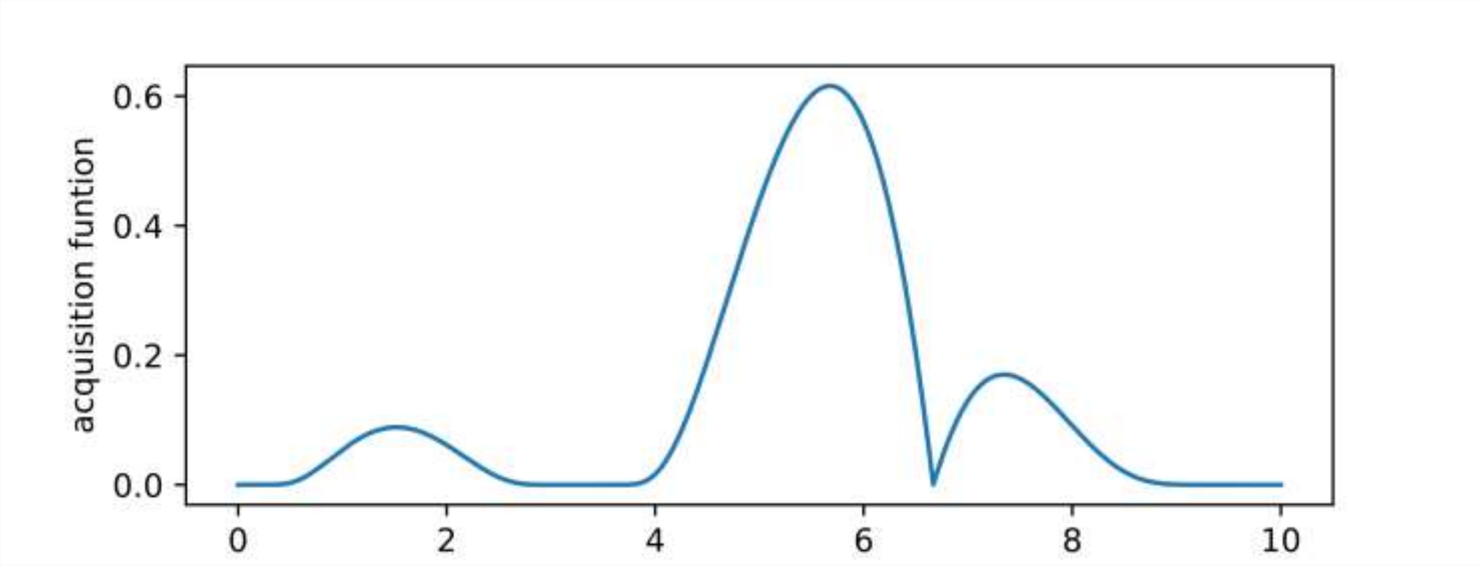
$$K(x, x') = \alpha \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu}\frac{||x - x'||}{\sigma}\right)^{\nu} K_\nu\left(\sqrt{2\nu}\frac{||x - x'||}{\sigma}\right)$$

Which values of x to sample?

Start out with random points



Later, use acquisition function



Acquisition functions

Balance exploration and exploitation

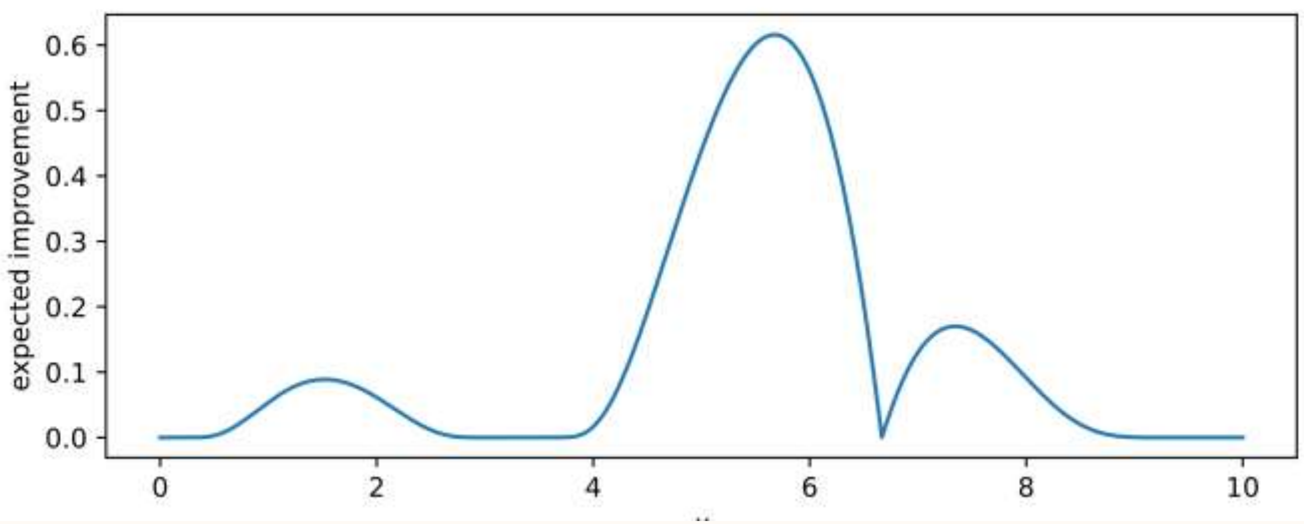
Expected improvement: $a(x) = E[max(0, f(x) - f^*)]$

Upper confidence bound (UCB): $a(x) = \mu(x) + \sqrt{\beta K(x, x)}$

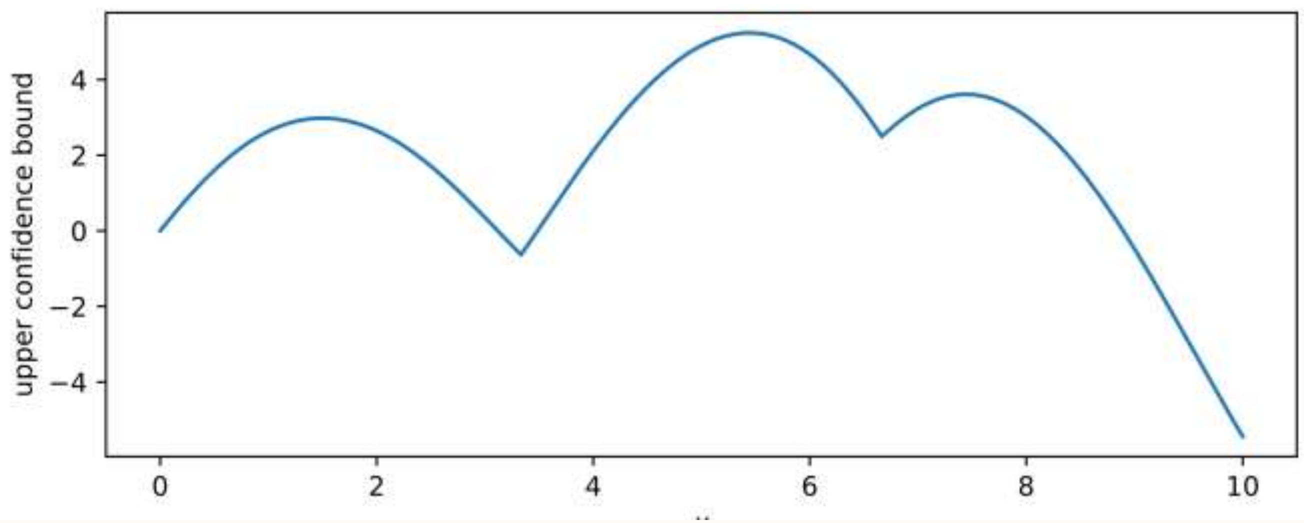
Thompson sampling: $a(x) = P(f(x) > f(x') \ \ \forall x')$

Acquisition functions

Expected Improvement



Upper confidence bound



Gaussian Process Regression: use maximum likelihood to fit GP parameters

Sample function and treat problem as discrete Gaussian RV

$$f(x) \sim N(\mu(x), K(x, x')) \longrightarrow \begin{bmatrix} f_1 \\ f_2 \\ \vdots \end{bmatrix} \sim N\left(\begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \end{bmatrix}, \begin{bmatrix} K_{11} & K_{12} & \dots \\ K_{21} & K_{22} & \dots \\ \vdots & \vdots & \ddots \end{bmatrix} \right)$$

$$K_{ij} = \alpha \exp\left(-\frac{||x_i - x_j||^2}{2\sigma^2} \right)$$

Gaussian Process Regression: use maximum likelihood to fit GP parameters

Use maximum likelihood to fit kernel parameters

$$\log p(f) \propto -f^T K^{-1} f - \log\left(\det K\right)$$

$$\frac{\partial}{\partial \theta_i} \log p(f) \propto f^T K^{-1} \frac{\partial K}{\partial \theta_i} K^{-1} f - tr\left(K^{-1} \frac{\partial K}{\partial \theta_i}\right)$$

Optimization with Gaussian processes

The basic recipe:

1. Choose kernel and mean functions

$$\mu(x) \qquad K(x, x')$$

2. Sample function $f(x)$ at several points x

3. Use maximum likelihood to fit GP parameters

4. Are we done? If not, select new points to sample from; GOTO step 3

Additional Resources

Gaussian process book:
http://www.gaussianprocess.org/

- Especially chapters 2 and 5

Peter Frazier’s tutorial on Bayesian optimization: arXiv:1807.02811

Eytan Bakshy’s publications: https://eytan.github.io/

Neural Architecture Search (NAS)

Components

1. Search Space
2. Search Strategy
3. Performance Estimation Strategy

NAS Overlapped with BO

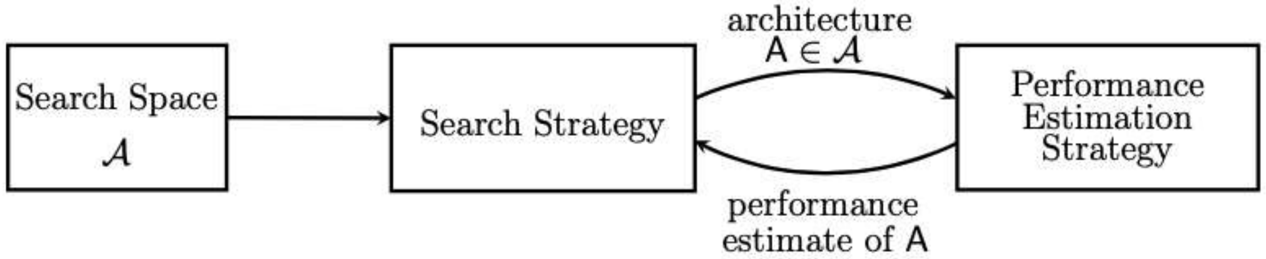
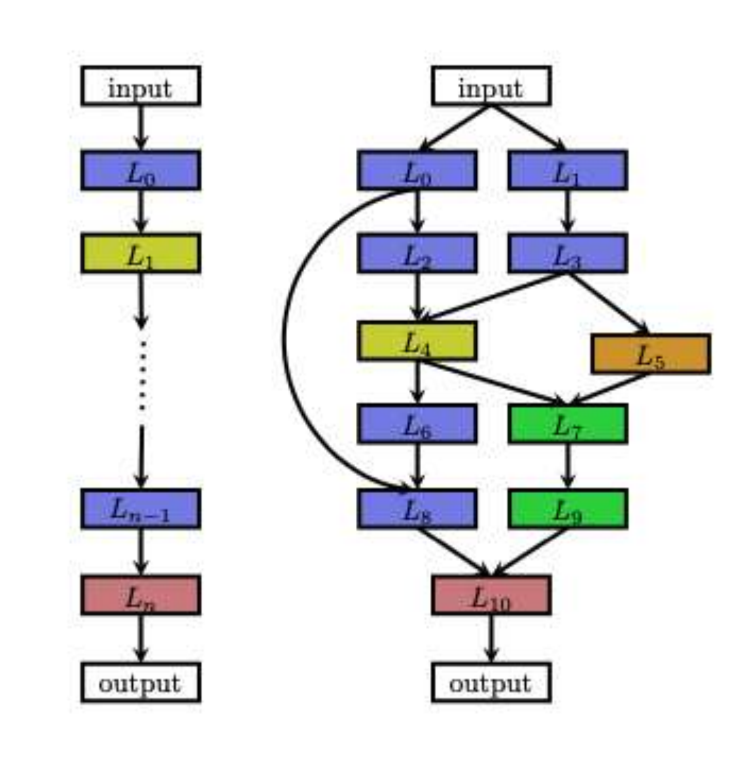


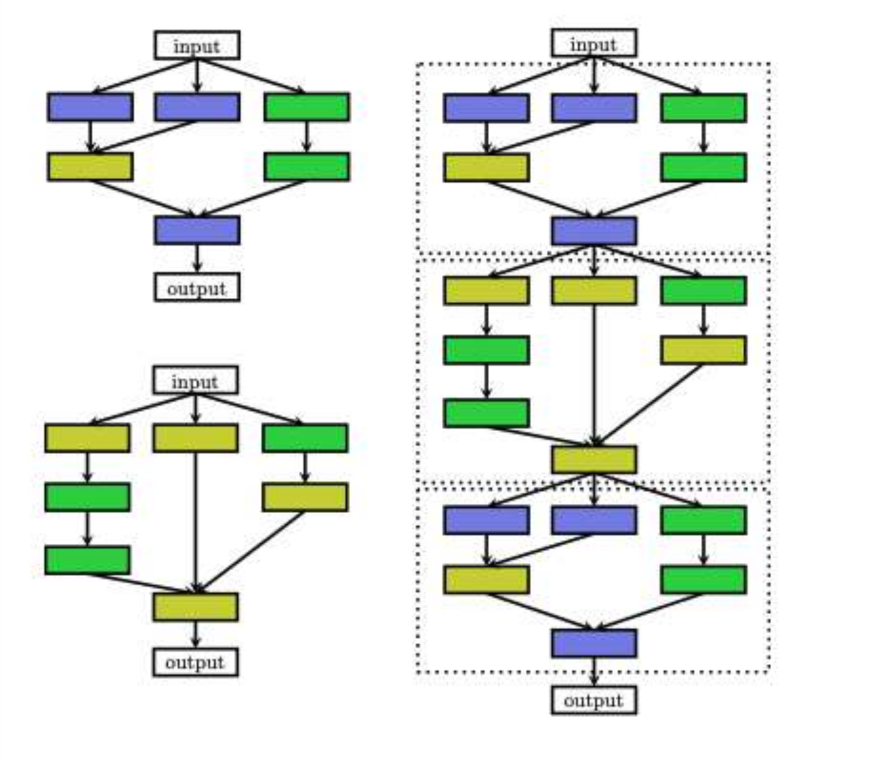
Figure 1: Abstract illustration of Neural Architecture Search methods. A search strategy selects an architecture A from a predefined search space $\mathcal{A}$. The architecture is passed to a performance estimation strategy, which returns the estimated performance of A to the search strategy.

- Search Space
  - Both BO and NAS need to define a proper search space
- Search Strategy
  - BO is one important search strategy
- Performance Estimation Strategy
  - Standard training and evaluation is expensive. Performance estimation is required

Search Space

The choice of the search space largely determines the difficulty of the optimization problem





Search Strategy

- Existing methods: Random Search, Bayesian Optimization, Evolutionary Methods, Reinforcement Learning, Gradient-based Methods
- BO has not been widely applied to NAS yet since Gaussian Process focuses more on low-dimensional continuous optimization problems

Performance Estimation Strategy

- Training each architecture to be evaluated from scratch frequently yields computational demands in the order of thousands of GPU days for NAS

Speed-up method	How are speed-ups achieved?	References
Lower fidelity estimates	Training time reduced by training for fewer epochs, on subset of data, downscaled models, downscaled data, ...	Li et al. (2017), Zoph et al. (2018), Zela et al. (2018), Falkner et al. (2018), Real et al. (2019), Runge et al. (2019)
Learning Curve Extrapolation	Training time reduced as performance can be extrapolated after just a few epochs of training.	Swersky et al. (2014), Domhan et al. (2015), Klein et al. (2017a), Baker et al. (2017b)
Weight Inheritance/ Network Morphisms	Instead of training models from scratch, they are warm-started by inheriting weights of, e.g., a parent model.	Real et al. (2017), Elsken et al. (2017), Elsken et al. (2019), Cai et al. (2018a,b)
One-Shot Models/ Weight Sharing	Only the one-shot model needs to be trained; its weights are then shared across different architectures that are just subgraphs of the one-shot model.	Saxena and Verbeek (2016), Pham et al. (2018), Bender et al. (2018), Liu et al. (2019b), Cai et al. (2019), Xie et al. (2019)

Table 1: Overview of different methods for speeding up performance estimation in NAS.

References

- Neural Architecture Search Survey: https://arxiv.org/pdf/1808.05377.pdf
- AutoML online book: http://www.automl.org/book/
- A blog for Neural Architecture Search: https://lilianweng.github.io/lil-log/2020/08/06/neural-architecture-search.html

Bayesian Optimization Demo

Bayesian Optimization Framework Overview

- GPyOpt: A Bayesian Optimization library based on GPy. Not actively developed and maintained as of now.
- Ray-Tune: Have multiple hyperparameter tuning frameworks integrated, including BayesianOptimization and BoTorch.
- BoTorch: A flexible and scalable Bayesian Optimization library based on GPyTorch and PyTorch.

Advantages of BoTorch

- BoTorch provides **a modular and easily extensible interface** for composing Bayesian optimization primitives, including probabilistic models, acquisition functions, and optimizers.
- BoTorch harnesses the power of PyTorch, including auto-differentiation, native support for highly parallelized modern hardware (e.g. GPUs) using device-agnostic code, and a dynamic computation graph.

BoTorch Models: SingleTaskGP

- A single-task exact GP model. Use this model when you have independent output(s) and all outputs use the same training data

Example

```
>>> train_X = torch.rand(20, 2)
>>> train_Y = torch.sin(train_X).sum(dim=1, keepdim=True)
>>> model = SingleTaskGP(train_X, train_Y)
```

BoTorch Models: FixedNoiseGP

- A single-task exact GP model using fixed noise levels.

Example

```
>>> train_X = torch.rand(20, 2)
>>> train_Y = torch.sin(train_X).sum(dim=1, keepdim=True)
>>> train_Yvar = torch.full_like(train_Y, 0.2)
>>> model = FixedNoiseGP(train_X, train_Y, train_Yvar)
```

BoTorch Models: MultiTaskGP

- Multi-task exact GP that uses a simple ICM (Intrinsic Coregionalization Model) kernel.

Example

```
>>> X1, X2 = torch.rand(10, 2), torch.rand(20, 2)
>>> i1, i2 = torch.zeros(10, 1), torch.ones(20, 1)
>>> train_X = torch.cat([
>>>     torch.cat([X1, i1], -1), torch.cat([X2, i2], -1),
>>> ])
>>> train_Y = torch.cat(f1(X1), f2(X2)).unsqueeze(-1)
>>> model = MultiTaskGP(train_X, train_Y, task_feature=-1)
```

BoTorch Model Fitting: fit_gpytorch_model

Parameters:

- **mll** (MarginalLogLikelihood) – MarginalLogLikelihood to be maximized
- **optimizer** (Callable) – The optimizer function
- **kwargs** (Any) – Arguments passed along to the optimizer function

Example

```
>>> gp = SingleTaskGP(train_X, train_Y)
>>> mll = ExactMarginalLogLikelihood(gp.likelihood, gp)
>>> fit_gpytorch_model(mll)
```

BoTorch Acquisition Functions: ExpectedImprovement

- Single-outcome Expected Improvement
- EI(x) = E(max(y - best_f, 0)), y ~ f(x)

Example

```
>>> model = SingleTaskGP(train_X, train_Y)
>>> EI = ExpectedImprovement(model, best_f=0.2)
>>> ei = EI(test_X)
```

BoTorch Acquisition Functions: UpperConfidenceBound

- Single-outcome Upper Confidence Bound (UCB).
- UCB(x) = mu(x) + sqrt(beta) * sigma(x)

Example

```
>>> model = SingleTaskGP(train_X, train_Y)
>>> UCB = UpperConfidenceBound(model, beta=0.2)
>>> ucb = UCB(test_X)
```

BoTorch Optimizers: optimize_acqf

- Optimize the acquisition function
- Parameters
  - **acq_function** (AcquisitionFunction) – An AcquisitionFunction.
  - **bounds** (Tensor) – A *2 x d* tensor of lower and upper bounds for each column of *X*.
  - **q** (int) – The number of candidates.
  - **num_restarts** (int) – The number of starting points for multistart acquisition function optimization.
  - **raw_samples** (int) – The number of samples for initialization.

Bayesian Optimization Demo Scripts

1. Fit a Gaussian Process model to data

```
import torch
from botorch.models import SingleTaskGP
from botorch.fit import fit_gpytorch_model
from gpytorch.mlls import ExactMarginalLogLikelihood

train_X = torch.rand(10, 2)
Y = 1 - (train_X - 0.5).norm(dim=-1, keepdim=True)  # explicit output dimension
Y += 0.1 * torch.rand_like(Y)
train_Y = (Y - Y.mean()) / Y.std()

gp = SingleTaskGP(train_X, train_Y)
mll = ExactMarginalLogLikelihood(gp.likelihood, gp)
fit_gpytorch_model(mll)
```

Bayesian Optimization Demo Scripts -- Continued

2. Construct an acquisition function

```
from botorch.acquisition import UpperConfidenceBound

UCB = UpperConfidenceBound(gp, beta=0.1)
```

3. Optimize the acquisition function

```
from botorch.optim import optimize_acqf

bounds = torch.stack([torch.zeros(2), torch.ones(2)])
candidate, acq_value = optimize_acqf(
    UCB, bounds=bounds, q=1, num_restarts=5, raw_samples=20,
)
```

Reformulating a Recommendation System Problem

Viral Gupta

Tech Lead Comms AI

Yunbo Ouyang

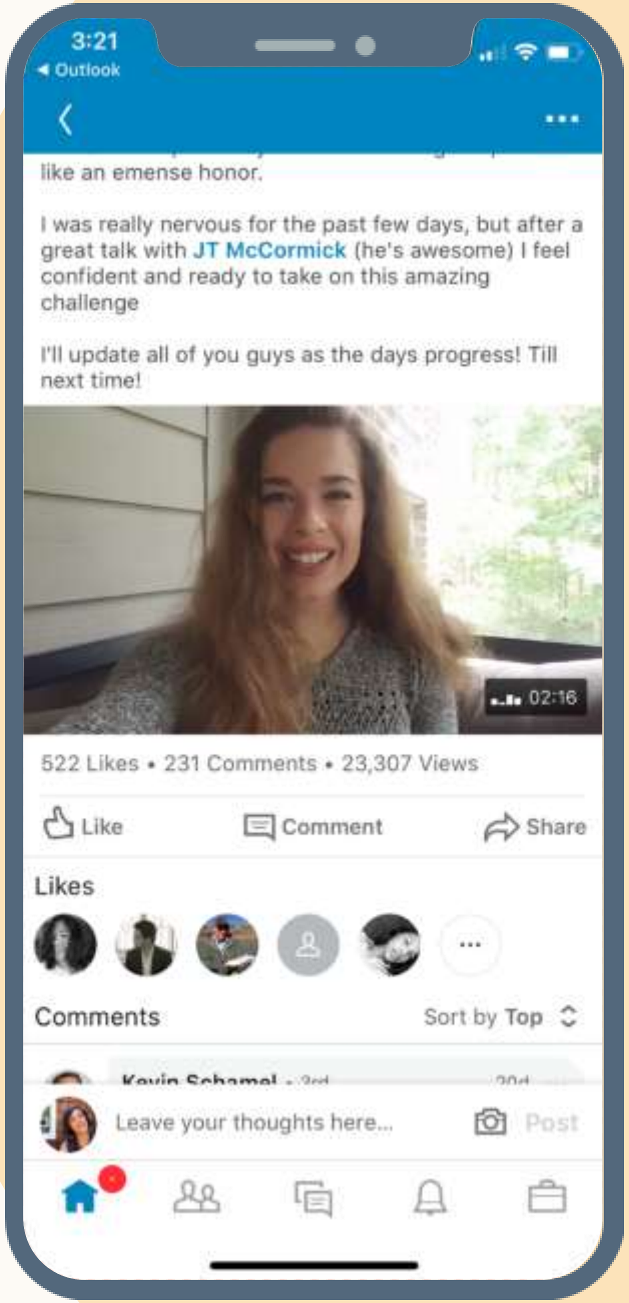
Sr. Software Engineer

Agenda

1. Real World Problems
2. Examples
3. Solving the Problem

LinkedIn Feed

Mission: Enable Members to build an active professional community that advances their career.

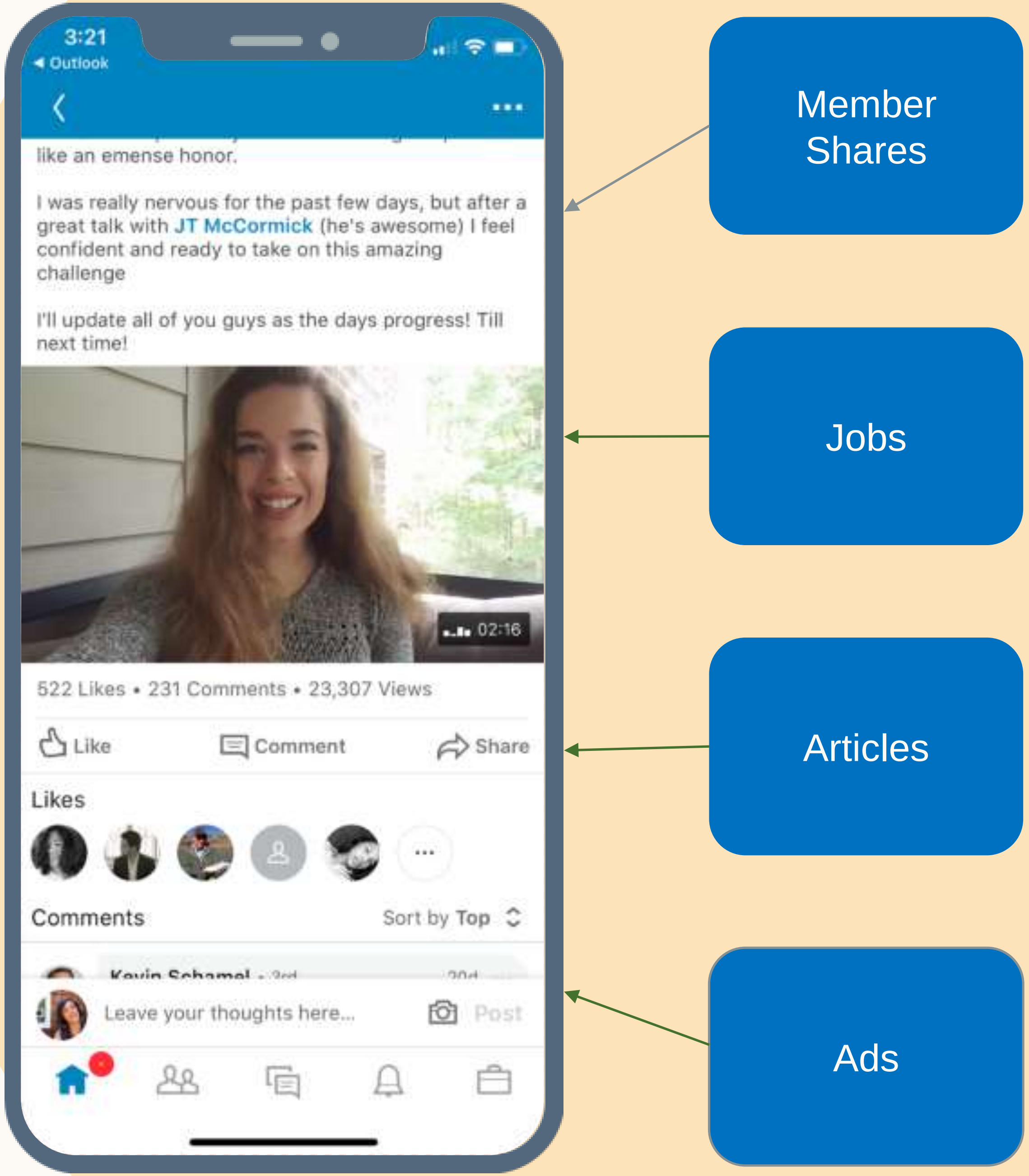


LinkedIn Feed

Mission: Enable Members to build an active professional community that advances their career.

Heterogenous List:

- Shares from a member's connections
- Recommendations such as jobs, articles, courses, etc.
- Sponsored content or ads



Important Metrics

Viral Actions (VA)

Members liked, shared or commented on an item

Job Applies (JA)

Members applied for a job

Engaged Feed Sessions (EFS)

Sessions where a member engaged with anything on feed.

Ranking Function

- m – Member, u - Item

$$S(m,u) \coloneqq P_{VA}(m,u) + \; x_{EFS}\, P_{EFS}(m,u) + x_{JA}\, P_{JA}(m,u)$$

- The weight vector $x = (x_{EFS}, x_{JA})$ controls the balance between the three business metrics: EFS, VA and JA.
- A Sample Business Strategy is

$$\begin{array}{ll} Maximize. & VA(x) \\ s.t. & EFS(x) > c_{EFS} \\ & JA(x) > c_{JA} \end{array}$$

Modeling The Metrics

- $Y_{i,j}^{k}(x) \in \{0,1\}$ denotes if the i-th member during the j-th session which was served by parameter x, did action k or not. Here k = VA, EFS or JA.

- We model this data as follows

$$\sum_i \sum_j Y_{i,j}(x) \sim \text{Gaussian}\,(\,f_k(x), \sigma^2)$$

- Assume a Gaussian process prior on each of the latent function f_k .

$$f_k(x) \sim \mathrm{N}\,(\,0, K_{RBF}(x, x))$$

Reformulation

We approximate each of the metrics as:

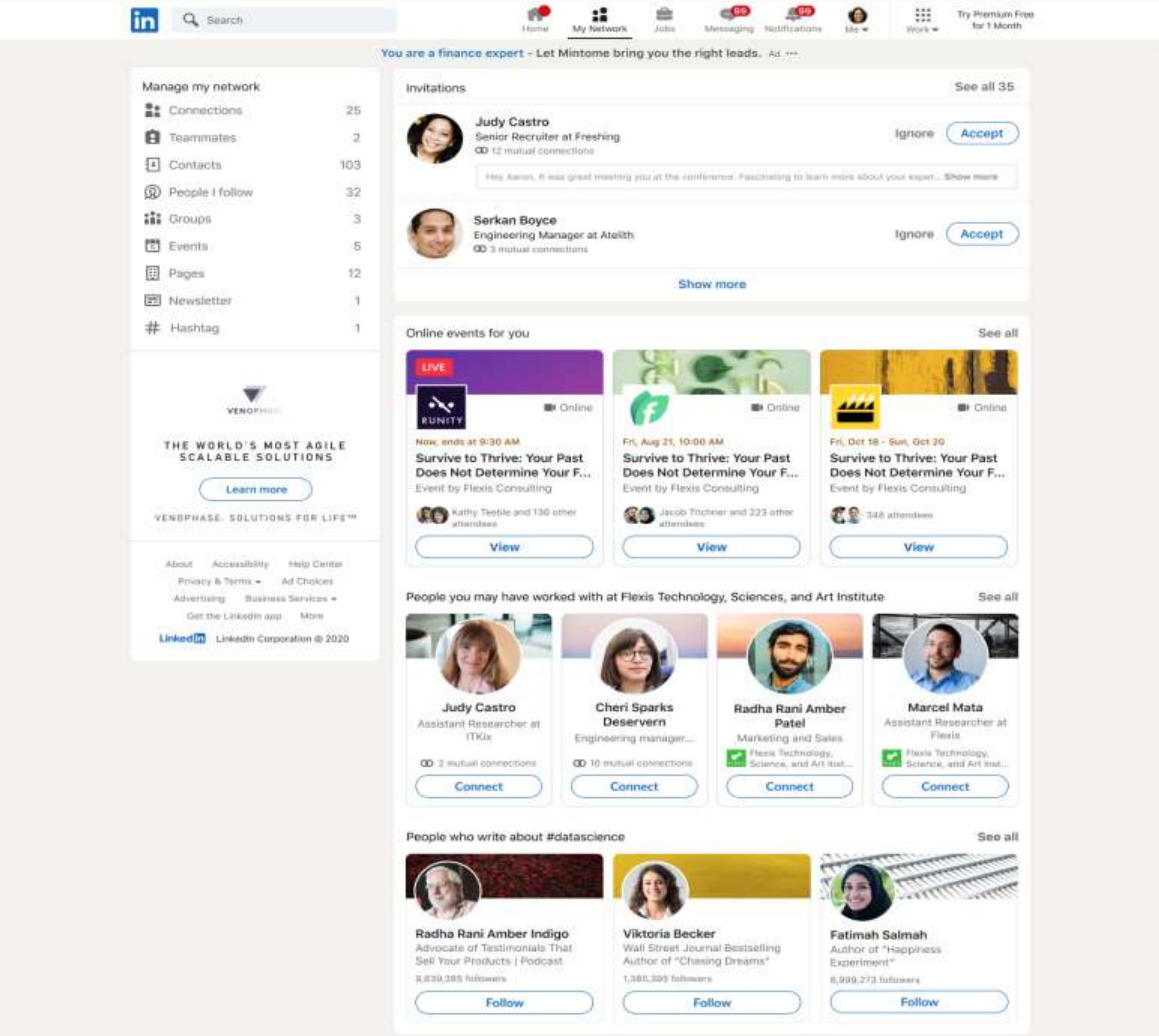
$$\begin{aligned} VA(x) &= f_{VA}(x) \\ EFS(x) &= f_{EFS}(x) \\ JA(x) &= f_{JA}(x) \end{aligned}$$

The original optimization problem can be written through this parametrization.

$$\begin{aligned} &Maximize. \quad VA(x) \\ &s.t.\ EFS(x) > c_{EFS} \\ &\quad\ JA(x) > c_{JA} \end{aligned} \quad \Rightarrow \quad \begin{aligned} &Maximize \quad f_{VA}(x) \\ &s.t. \quad f_{EFS}(x) > c_{EFS} \\ &\qquad f_{JA}(x) > c_{JA} \end{aligned} \quad \Rightarrow \quad \begin{aligned} &Maximize \ f(x) \\ &x \in X \end{aligned}$$

Benefit: The last problem can now be solved using techniques from the literature of Bayesian Optimization.

PYMK Recommendation



PYMK Recommendation

- Main driver of network growth at LinkedIn.
- Besides network growth, Engagement is another Important piece.

Business Metrics

- Accept Rate: the fraction of accepted invitations out of all invitations sent from PYMK.
- Invite Rate: the fraction of impressions for which members send invitations, out of all impressions delivered.
- Member Engagement Metric: the fraction of invitations sent that helps member engagement.

PYMK Scoring Function

- m – Member, c - candidate

$$S(m, c) \coloneqq P_{InviteAccept}(m, c) * (1 + \alpha\, P_{Engage}(m, c))$$

- *pInviteAccept* is a score predicting a combination of accept score and invite score.
- *pEngage* is a score predicting member engagement score.
- α is the hyper-parameter we need to tune

Reformulation

We approximate each of the metrics as:

$$
\begin{aligned}
Invite(x) &= f_{invite}(x)\\
Accept(x) &= f_{accept}(x)\\
Engage(x) &= f_{engage}(x)
\end{aligned}
$$

The original optimization problem can be written through this parametrization.

$$
\begin{aligned}
&Maximize. \quad Invite(x)\\
&s.t.\ Accept(x) > c_{accept}\\
&\quad Engage(x) > c_{engage}
\end{aligned}
\quad \Rightarrow \quad
\begin{aligned}
&Maximize \quad f_{accept}(x)\\
&s.t. \quad f_{invite}(x) > c_{invite}\\
&\quad f_{engage}(x) > c_{engage}
\end{aligned}
\quad \Rightarrow \quad
\begin{aligned}
&Maximize \ f(x)\\
&x \in X
\end{aligned}
$$

Benefit: The last problem can now be solved using techniques from the literature of Bayesian Optimization.

Modeling The Metrics

- $Y_{i,j}^{k}(x) \in \{0,1\}$ denotes if the i-th member during the j-th session which was served by parameter x, did action k or not. Here k = invite, accept, engage.

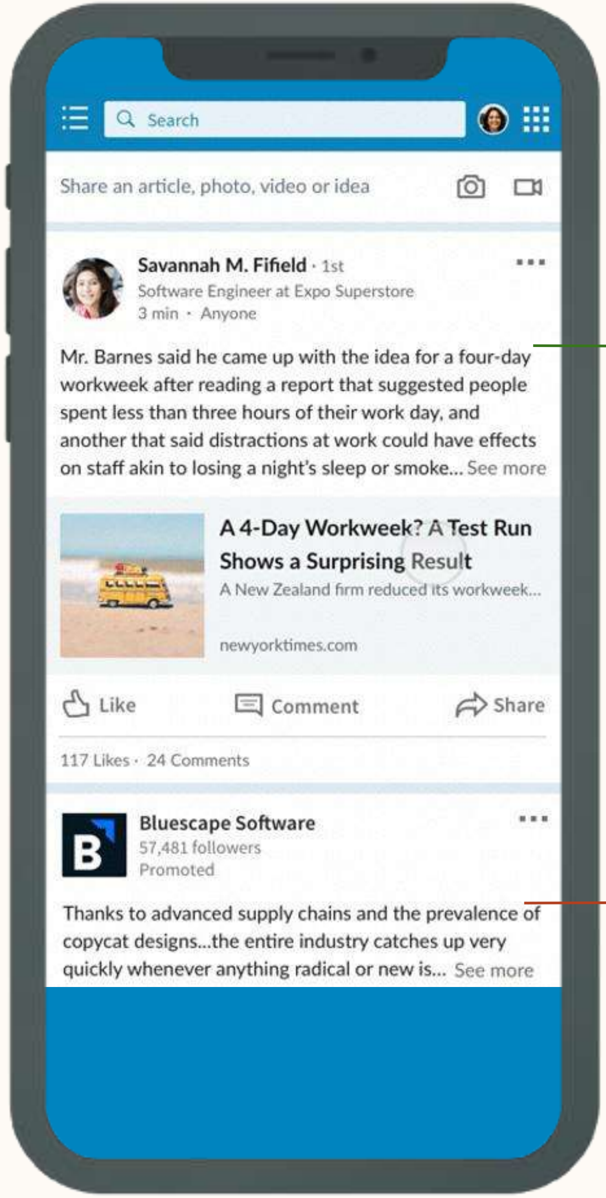
- We model this data as follows

$$\sum_i \sum_j Y_{i,j}(x) \sim \text{Gaussian}\,(\, f(x), \sigma^2)$$

- Assume a Gaussian process prior on each of the latent function f_k .

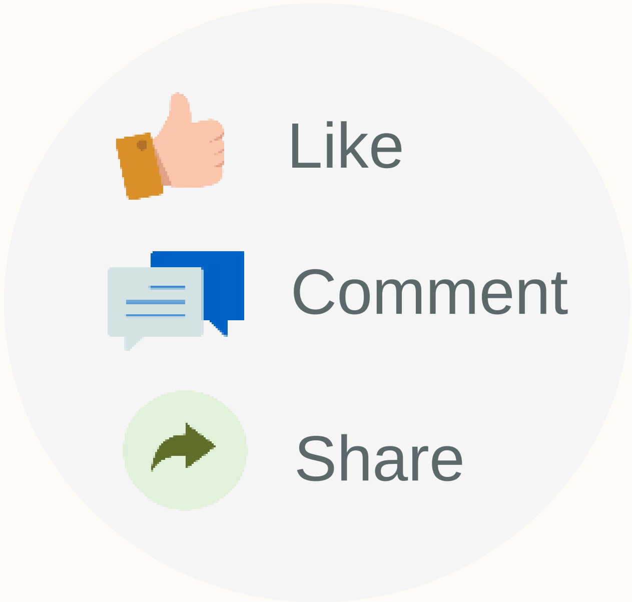
$$f_k(x) \sim \mathrm{N}\,(\,0, K_{RBF}(x,x))$$

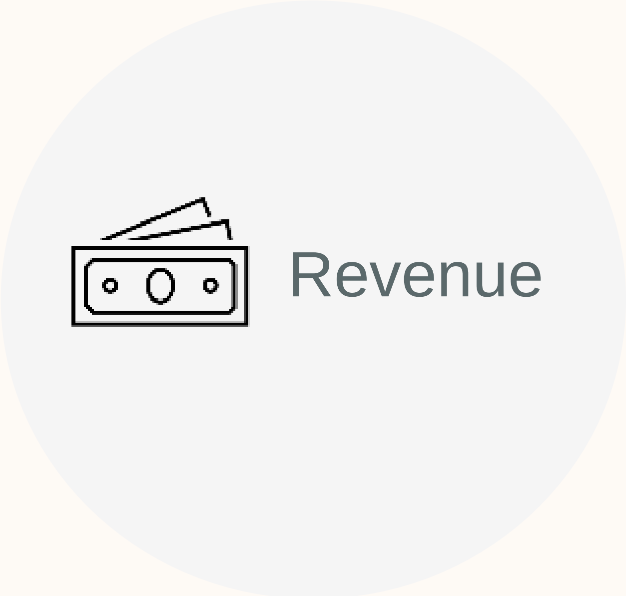
LinkedIn Feed is mixture of OU and SU



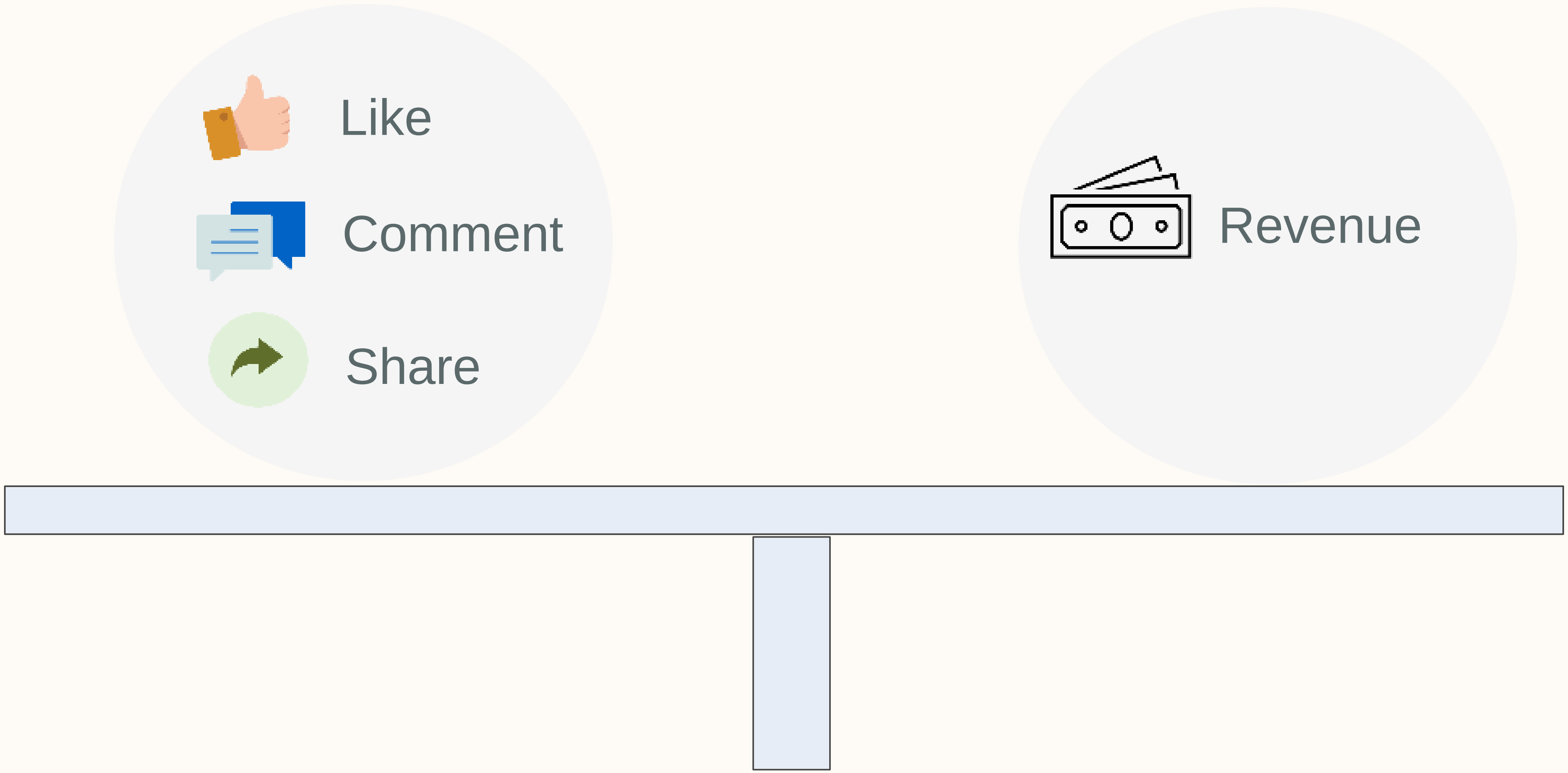
Organic Update

Sponsored Update





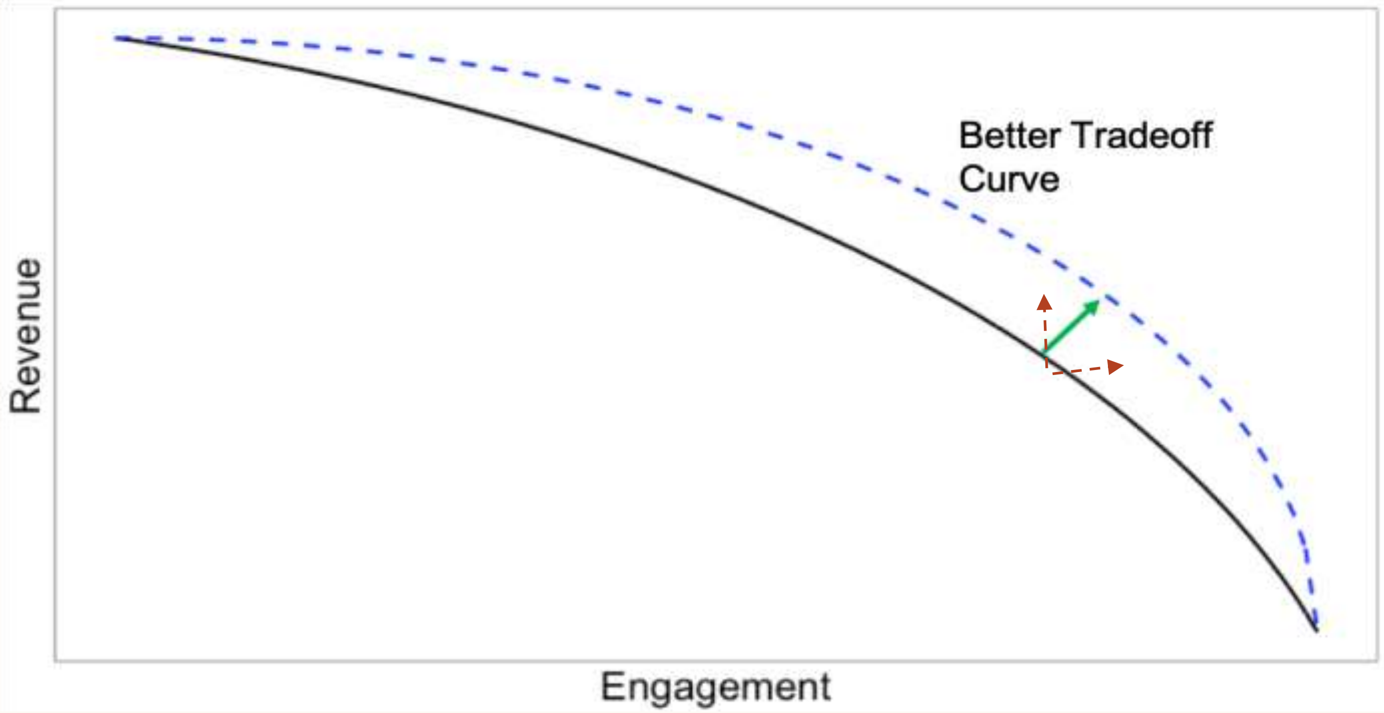
Need to find a desired balance



Revenue Engagement Tradeoff (RENT)

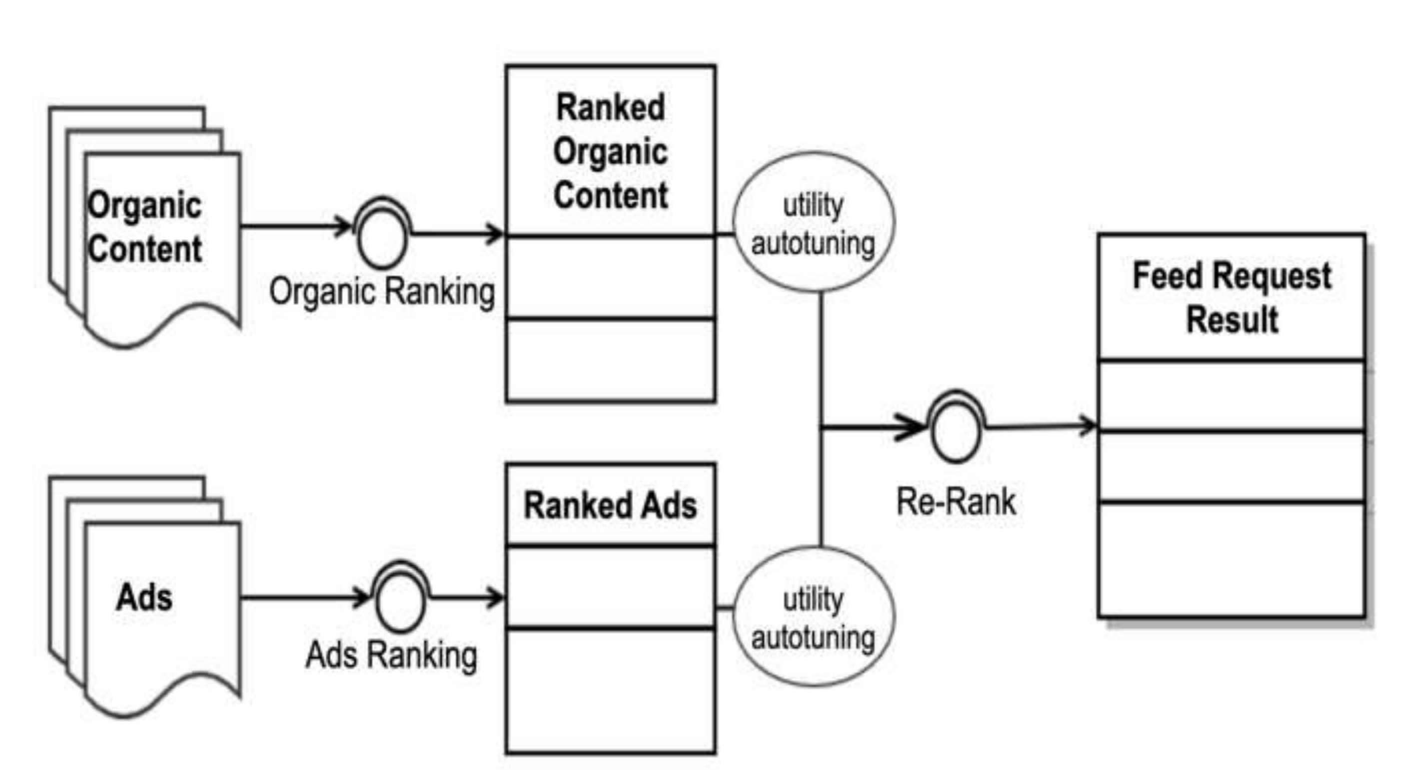
- RENT is a mechanism that blends sponsored updates (SU) and organic updates (OU) to optimize revenue and engagement tradeoff

- Two important while conflicting objectives are considered



Fast blending system with a light-weight infrastructure

- High velocity on model dev
- Ranking invariance
- Calibration with autotuning



Ads use-case

- Compare ads models M1 and M2 by making sure each have equal impressions.
- This is controlled by LCF factor that affects the score and hence the ranking on the feed.
- Increasing LCF gives higher score and hence increase rank which increases number of impressions

$$Minimize. \quad |SU_{impressions}(LCF) - SU_{impressions}(\text{control})|$$

Solution

1. Mathematical Abstraction
2. Bayesian Optimization
3. Thompson Sampling

Mathematical Abstraction – Constrained Optimization

$$\max_x y(x)$$
$$s.t. \, y_1(x) \geq c_1, \cdots, y_k(x) \geq c_k \cdots, y_K(x) \geq c_K$$

Goal

- When launching a new model online, we need to tune hyperparameters to optimize the major metric while making sure the guardrail metrics do not drop

Notation

- x: The hyperparameter to tune
- $y(x)$: The major business metric to optimize
- $y_k(x)$: The guardrail metrics
- c_k: The constraint value

Model Each Metric via Gaussian Process

Examples

- $y(x)$: Member viral actions in Feed; invitation rate in PYMK
- $y_k(x)$: Job application metrics in Feed; acceptance rate in PYMK

Use Gaussian Process to model online metrics.

Different distributions can be used ($n(x)$ is a normalizer if we use Binomial or Poisson distribution):

- $y(x) \sim Gaussian(f(x), \sigma^2), f(x) \sim Gaussian\left(0, K(x, x)\right)$
- $y(x) \sim Binomial(n(x), Sigmoid(f(x))), f(x) \sim Gaussian\left(0, K(x, x)\right)$
- $y(x) \sim Poisson\left(e^{n(x)f(x)}\right), f(x) \sim Gaussian\left(0, K(x, x)\right)$

Combine All Metrics – Method of Lagrange Multipliers

Equivalent to maximize the following

$$L(x) = y(x) + \lambda_1 \sigma_\alpha(y_1(x) - c_1) + \cdots \lambda_K \sigma_\alpha(y_K(x) - c_K)$$

$$\sigma_\alpha(z) = (1 + e^{-\alpha z})^{-1}$$

- σ_α is a smoothing Sigmoid function which take the value close to 1 for positive z and close to 0 for negative z.
- $\lambda_k \sigma_\alpha(y_k(x) - c_k)$ penalizes constraint violation.
- λ_k is a positive multiplier, which is tunable in different applications
- The optimal value of $L(x)$ optimizes $y(x)$ while satisfying all constraints.

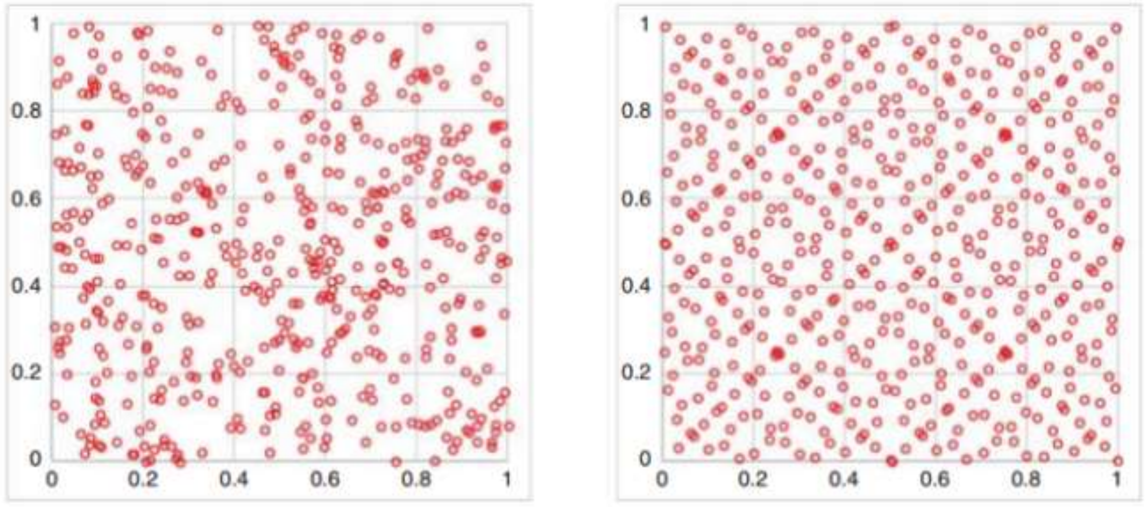
Langrange Multipliers + Gaussian Process

Posterior samples of $y(x)$ and $y_k(x)$ could be easily obtained via Gaussian Process Regression. So could $L(x)$.

$$L(x) = y(x) + \lambda_1 \sigma_\alpha(y_1(x) - c_1) + \cdots \lambda_K \sigma_\alpha(y_K(x) - c_K)$$

Collecting Metrics

- Metrics $y(x)$ and $y_k(x)$ are collected on equal spaced Sobol points $x_1, \cdots, x_n$.
- Sobol points offer a lower discrepancy than the same number of random numbers.



left: random, right: Sobol — Modern Computational Finance (Antoine Savine, Wiley, 2018), page 205

Reweighting Points

- Initially recommender systems are served equally by $x_1, \cdots, x_n$.
- After collecting metrics and apply Gaussian Process Regression, we generate samples from $L(x)$.
- We need to assign each point x_i a probability value p_i to split traffic.

Thompson Sampling

- Use Thompson Sampling to assign probability value p_i to each point x_i
  - Step 1: For all candidate points $x_1, \cdots, x_n$, obtain posterior samples from $L(x)$: $\hat{L}(x_1), \hat{L}(x_2), \cdots, \hat{L}(x_n)$.
  - Step 2: Pick x_i^* such that $\hat{L}(x_i^*)$ is the largest among $\hat{L}(x_1), \hat{L}(x_2), \cdots, \hat{L}(x_n)$
  - Repeat Step 1 and Step 2 1000 times.
  - Step 3: Aggregate $x_1^*, \cdots, x_{1000}^*$. Set p_i as the frequency of x_i in $x_1^*, \cdots, x_{1000}^*$

Intuition of Thompson Sampling

- Thompson Sampling is used to select the optimal hyperparameters via sampling from a distribution $(x_1, p_1), (x_2, p_2), \cdots, (x_n, p_n)$ among all candidates.
- p_i is equal to the probability of each candidate being optimal among all.
- Thompson Sampling is proved to converge to the optimal value in various settings.
- Thompson Sampling is widely used in online A/B test, online advertising and other applications.

Exploration versus Exploitation

- Exploration: explore sub-optimal but high potential regions in the search space
- Exploitation: search around the current optimal candidate
- Thompson Sampling automatically balances exploration and exploitation via a probability distribution

Infrastructure

Viral Gupta
Tech Lead
CommsAI

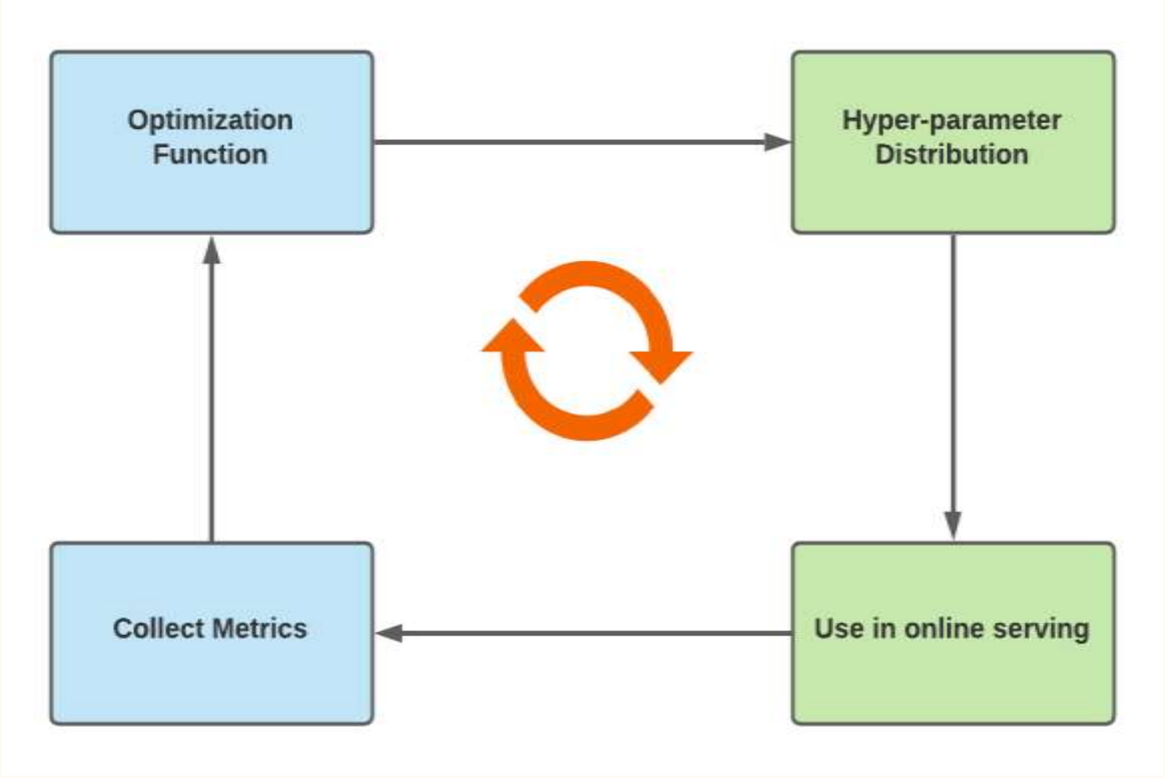
Agenda

1. System Design
2. At-Scale Use
3. Visualizations
4. Results

High Level Steps

We need to run the following steps in a loop

- Optimization Function emits hyper-parameter distribution
- Use the hyper-parameters in online serving
- Collect Metrics
- Call the optimization function



Goals of the Infrastructure

Few realities about the clients of the Optimization package

- Every team within LinkedIn can have different online serving system. eg: Streaming, offline, rest end point.

- Tracking data and metrics for each system can be very different. Eg. Different calculation methodology.

To make a minimum viable product and stay focused on solving the optimization problem well we built the Optimization package as a library.

Flow Architecture

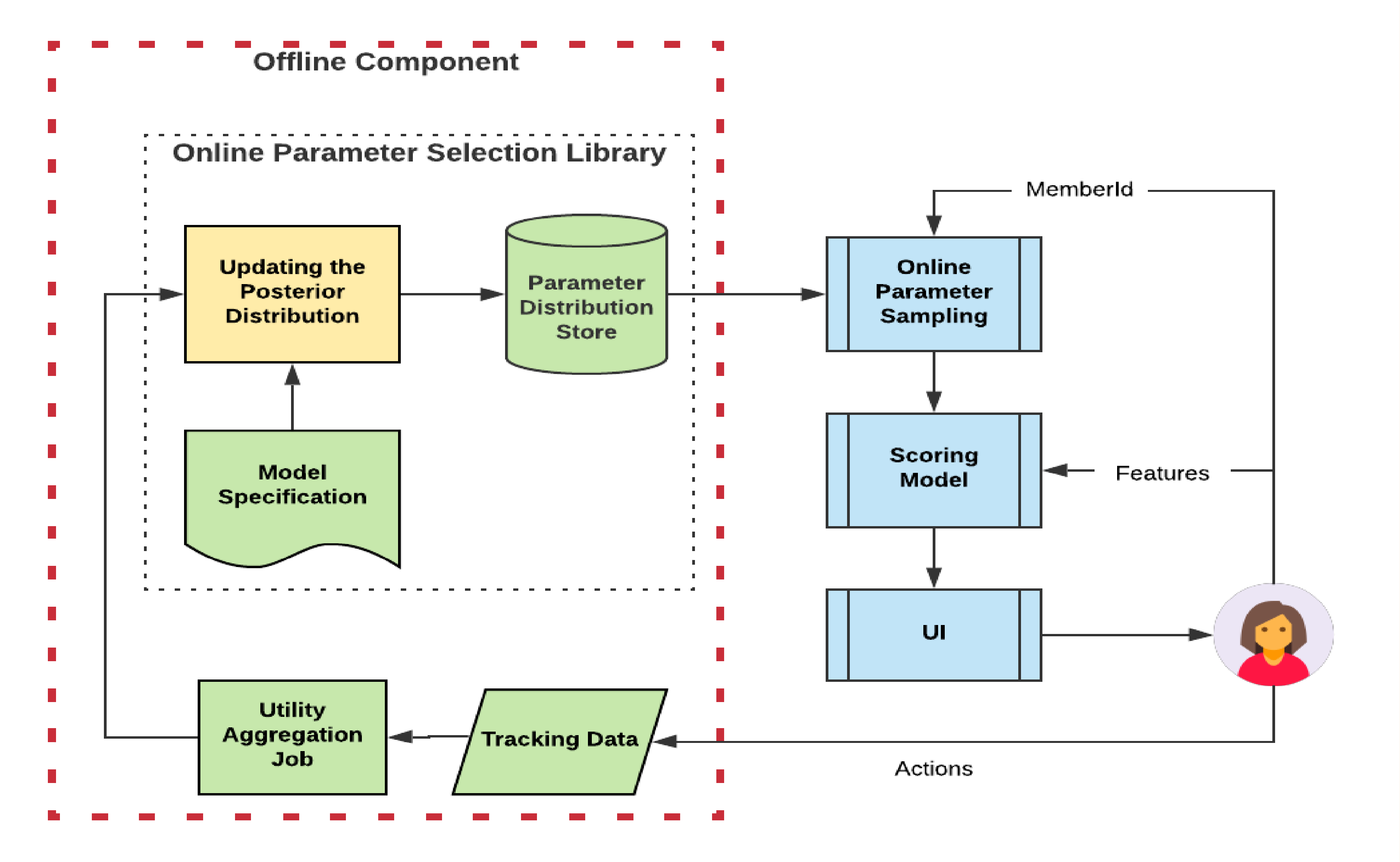
Built the Optimization package as a library. It will expand as a spark flow.

Client will call the Optimization package in its existing spark offline flow.

Client flow will contain the following

- A hadoop/spark flow to collect metrics and aggregate.
- Call the Optimization package which expands into spark flow
- A flow to use the output of the optimizer.

Overall System Architecture



Notifications Example

Send the right volume of notification at most opportune time to improve sessions and clicks keeping send volume same.

$$\begin{aligned} &Maximize. \quad Sessions(tr) \\ &\quad s.t. \quad Clicks(tr) > c_{Clicks} \\ &\quad\quad Send\,Volume(tr) < c_{Send\,Volume} \end{aligned}$$

We have the following ML models

$P_{Click}(m,i): Probability\ of\ a\ click\ on\ item\ i\ member\ m$

$P_{Visit}(m,i): Probability\ of\ a\ visit$

Notifications Scoring Function

- m – Member, i - Item

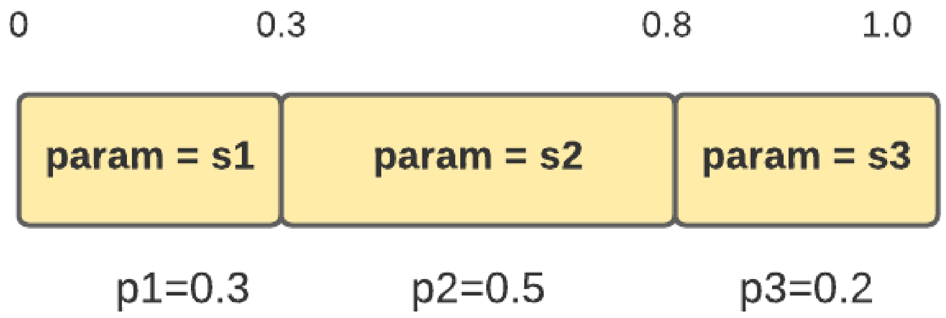
$$S(m,i) \coloneqq P_{Click}(m,i) + \; x_\alpha \, P_{Visit}(m,i) > x_{th}$$

- The weight vector $x = (x_\alpha, x_{th})$ controls the balance between the business metrics: Sessions, Clicks, Send Volume.
- A Sample Business Strategy is

$$\begin{aligned} &Maximize. \quad Sessions(x) \\ &s.t. \quad Clicks(x) > c_{Clicks} \\ &\quad Send\ Volume(x) < \; c_{Send\ Volume} \end{aligned}$$

Notifications Optimization

Optimizer outputs a probability distribution over parameters.



Parameter Set	Threshold	Alpha
s1	0.1	2.0
s2	0.3	5.0
s3	0.4	8.0

Using Optimizer output

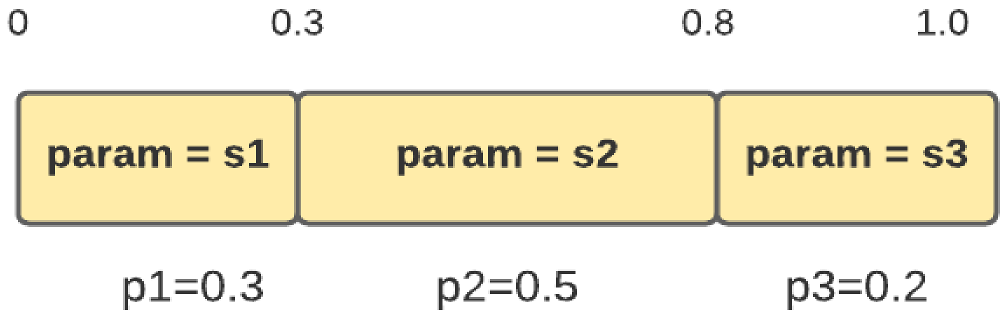
The next step is to just map all members in the treatment to parameter Set using the CDF.

```
def assignParameterToTreatmentMembers(members, parameterCdf):
    memberAssignment = []
    for m_i in members:
        memberHash = hash(m_i)   ## memberHash is in (0,1)
        parameter = binarySearch(membeHash, parameterCdf)
        memberAssignment.append((m_i, parameter))
    return memberAssignment
```

Member Assignment to hyper-parameters

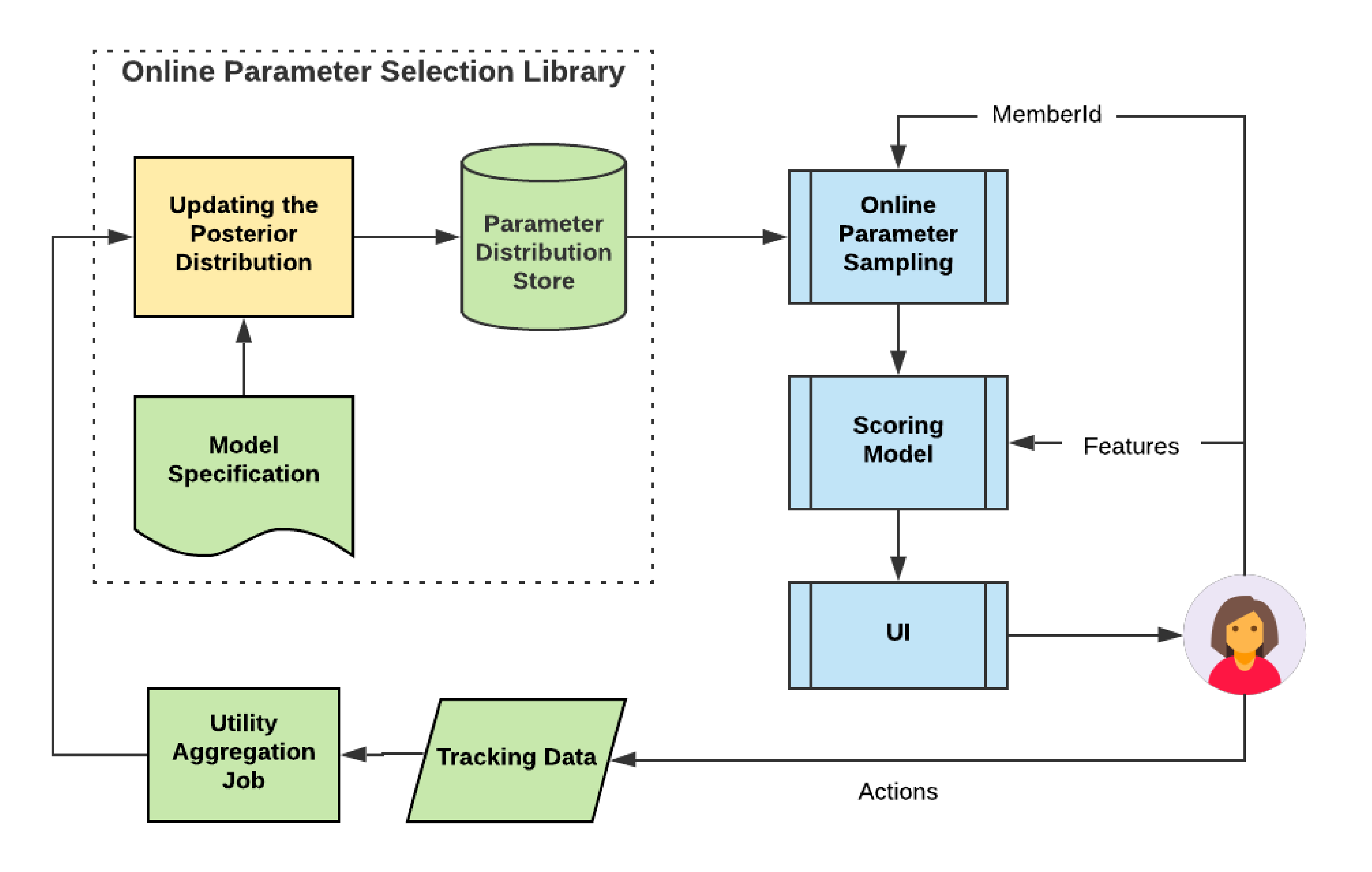
Optimizer outputs a probability distribution over parameters.

Member	memerHash	parameter assigned
"Foo"	0.2	s1
"Bar"	0.7	s2



Parameter Set	Threshold	Alpha
s1	0.1	2.0
s2	0.3	5.0
s3	0.4	8.0

Overall System Architecture



Tracking Data Aggregation

Tracking Data from user activity logs are available on HDFS.

The raw data gets aggregated by the client flows.

userId	parameterSet	Impressions	Clicks	Sessions
Foo	s1	10	4	2
Bar	s2	25	12	3
Alice	s3	45	5	1
Bob	s1	12	2	2
John	s2	78	23	8
Liz	s1	87	27	6
John1	s2	7	2	1

parameterSet	Impressions	Clicks	Sessions
s1	109	33	10
s2	110	37	12
s3	45	5	1

Library API: Problem Specification

- Problem Specification
  - Treatment models and a control model
  - Parameters and search range
  - Objective metric and constraint metrics
  - The number of exploration iterations
- Exploration – Exploitation Setup
  - The algorithm starts with a few exploration iterations that outputs uniform serving probability.
  - Exploration stage is the stage in the algorithm which explores each point equally while exploitation stage is the stage to reassign probabilities to different points.

```
{
  "treatmentModels":["treatmentModel-1"],
  "controlModel":"controlModel-1",
  "exploreNumIterations":"6",
  "params":[
        {
          "fieldName":"threshold",
          "parameterInfo":{
            "searchRange":{
              "low":"0.17",
              "high":"0.24"
            },
            "dataType":"Float"
          }
        }
     ]
  },
```

Library API: Problem Specification Continued

```
    "Constraints":[
      {
        "utilityName":"SendsByGenerated",
        "ColumnNames":[
          "sentCount",
          "generatedCount"
        ],
        "distribution":"gaussian",
        "upperBound":{
          "multiplier":"Inf"
        },
        "lowerBound":{
          "multiplier":"1.0"
        }
      }
    ]
  }
}
```

```
    "Objective":{
      "objectiveType":"max",
      "objectiveParts":[
        {
          "utilityName":"ClickRate",
          "ColumnNames":[
            "clickCount",
            "impressedCount"
          ],
          "distribution":"gaussian"
        }
      ]
    },
```

Offline System

The heart of the product

Tracking

- All member activities are tracked with the parameter of interest.
- ETL into HDFS for easy consumption

Utility Evaluation

- Using the tracking data we generate $(x, f_k(x))$ for each function k.
- The data is kept in appropriate schema that is problem agnostic.

Bayesian Optimization

- The data and the problem specifications are input to this.
- Using the data, we first estimate each of the posterior distributions of the latent functions.
- Sample from those distributions to get distribution of the parameter x which

The Parameter Store and Online Serving

- The Bayesian Optimization library generates
  - A set of potential candidates for trying in the next round $(x_1, x_2, \ldots, x_n)$
  - A probability of how likely each point is the true maximizer $(p_1, p_2, \ldots, p_n)$ such that $\sum_{i=1}^{n} p_i = 1$

- To serve members with the above distribution, each **`memberId`** is mapped to [0,1] using a hashing function h. For example

$$\sum_{i=1}^{k} p_i < h(user) \leq \sum_{i=1}^{k+1} p_i$$

  Then my feed is served with parameter x_{k+1}

- The parameter store (depending on use-case) can contain
  - `<parameterValue, probability>` i.e. (x_i, p_i) or
  - `<memberId, parameterValue>`

Online System

Serving hundreds of millions of users

Parameter Sampling

- For each member m visiting LinkedIn,
  - Depending on the parameter store, we either evaluate `<m, parameterValue>`
  - Or we directly call the store to retrieve `<m, parameterValue>`

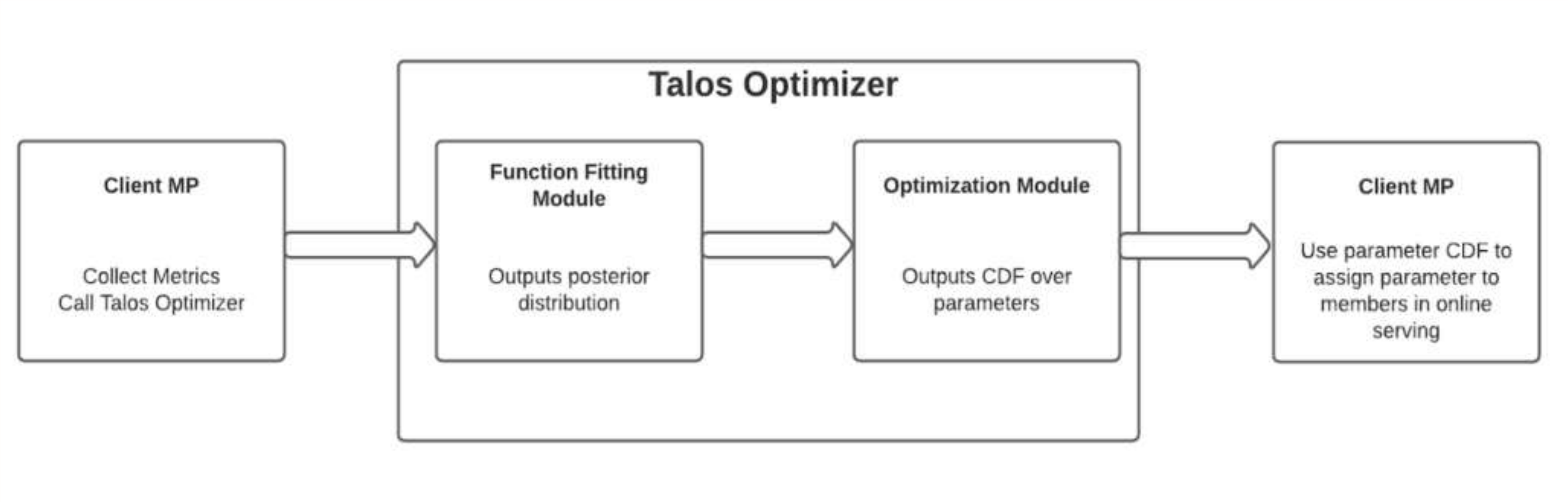
Online Serving

- Depending on the parameter value that is retrieved (say x), the notification item is scored according to the ranking function and served

$$S(m,u) := P_{Click}(m,i) + x_{\alpha}\, P_{Visit}(m,i) > x_{th}$$

Library Design

Library design can be broken into 2 components – Function fitting Module and Optimization module.



Entities

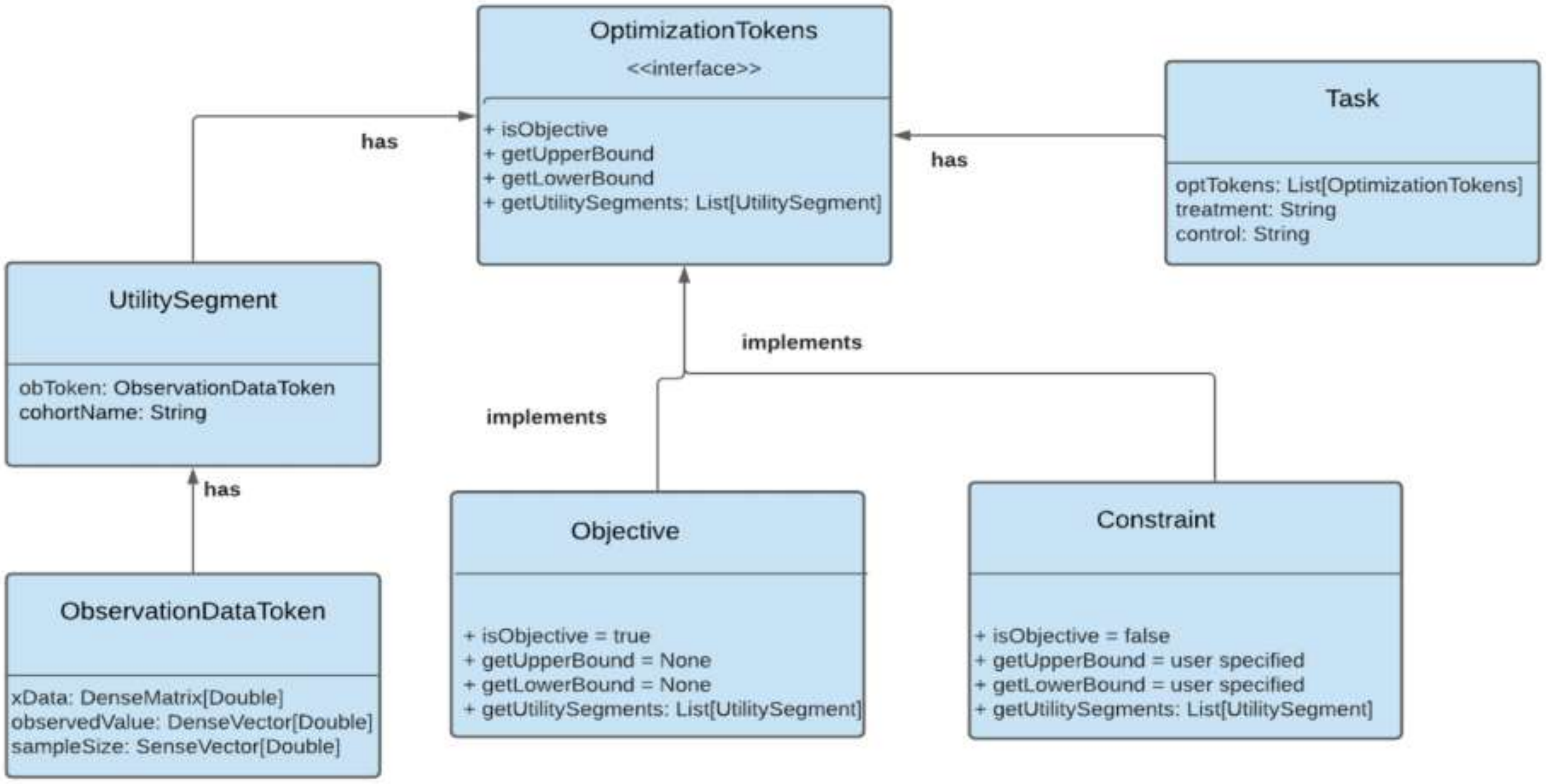
- Task is the single unit of a problem that we want to solve.

- OptimizationToken can be either objective or a constraint. A given task object can have multiple optimizationTokens.

- UtilitySegment encapsulates all the information for modelling a single latent function f_k.

- ObservationDataToken is used to store observed data. The raw metrics y (vector), hyper parameters x (matrix), sampleSize n (vector)

Task Object



Data Structures related to Task Object

Visualization Tools

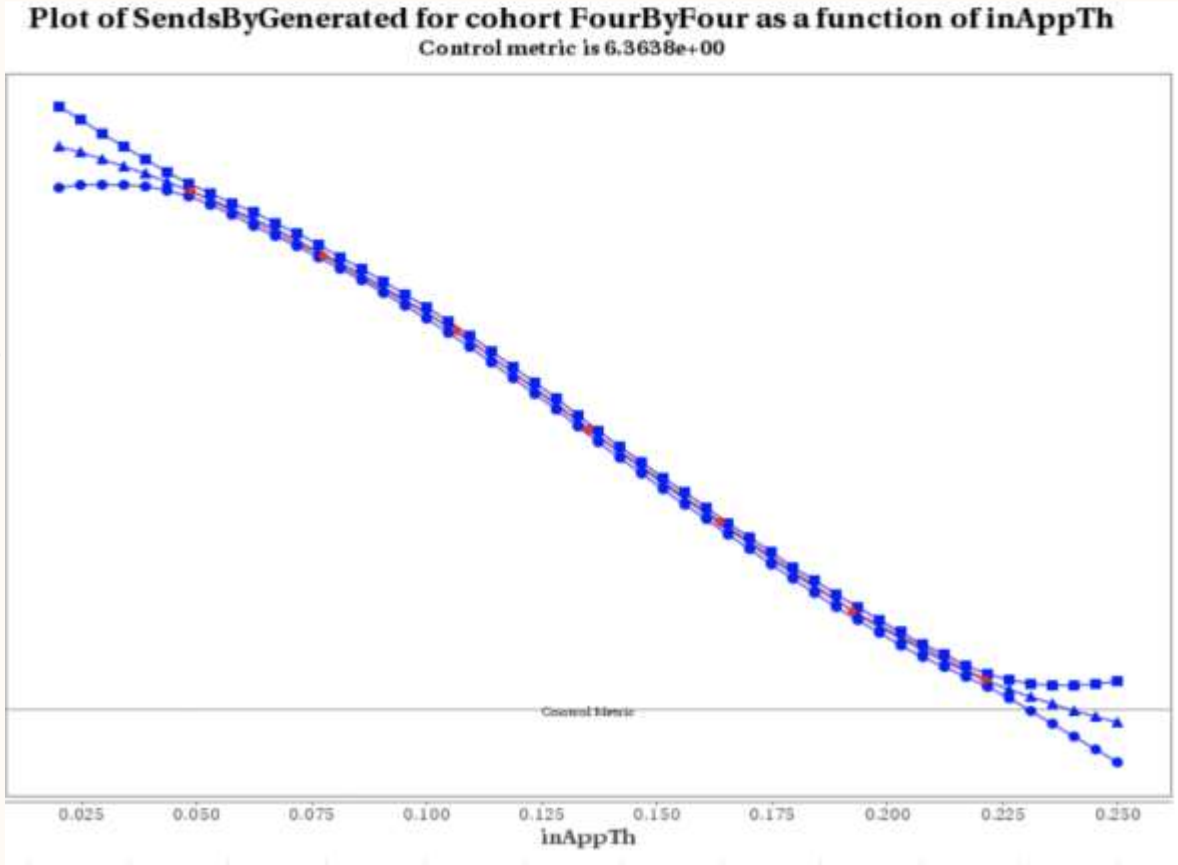
Plots (When tuning 1d parameter)

Shows how Send Volume for a cohort (FourByFour) changes as we change threshold.

The red curve is the observed metrics.

The middle blue line is the mean of the GP estimate.

The left and right blue lines are the GP variance estimates.



Results

Online A/B Testing Results

Table 1: Online A/B results for Online Parameter Selection in LinkedIn Feed Ranking

Metric	Lift (%) vs Control $\boldsymbol{x}_{c_1}$	Lift (%) vs Control $\boldsymbol{x}_{c_2}$
Viral Actions	+3.3%	+1.2%
Engaged Feed Sessions	-0.8%	0%
Job Applies	+12.8%	+6.4%

Operational Efficiency

- Feed model tuning needs 0.5x traffic and improve the tuning speed by 2.5x .
- Ads model tuning improved by 4x.
- Notification model tuning improved by 3x.

Key Takeaways

- Removes the human in the loop: Fully automatic process to find the optimal parameters.
- Drastically improves developer productivity.
- Can scale to multiple competing metrics.
- Very easy onboarding infra for multiple vertical teams. Currently used by Ads, Feed, Notifications, PYMK, etc.

LinkedIn

Extensions of Bayesian Optimization

Cyrus DiCiccio

Sr. Software Engineer

The Simplest Optimization Problem

Our goal is to efficiently solve an optimization problem of the form

$$\max_{x} f(x)$$

Subject to constraints

$$g_i(x) > c_i, \quad i = 1, \ldots, K$$

Basic Bayesian Optimization

Observe data as

$$Y_i = f(x_i) + \varepsilon_i$$

where the error term is assumed to follow a normal dist $\varepsilon_i \sim N(0, \sigma^2)$

and the mean function of interest is assumed to follow a Gaussian process

$$f(\cdot) = GP(\mu(\cdot), K(\cdot,\cdot))$$

with mean mu, typically assumed to be identically zero, and covariance kernel K (common choices include the RBF kernel or the Matern kernel)

Some Natural Extensions

1) Often there is time dependence in A/B experiments.
    - Day over-day fluctuations
    - User behavior may differ on weekdays vs weekends

2) Can we incorporate other sources of information into the optimization?

3) How can we address heterogeneity in users’ affinity towards treatments

Time Dependence

Traditional Bayesian optimization strongly assumes metrics are stable over time. Is this a problem?
It can be when using explore/exploit methods

Examples of time dependence in A/B experiments.

- Day over-day fluctuations
- User behavior may differ on weekdays vs weekends

If an exploit step suggests a new point to sample, but this occurs on a Saturday when users tend to be more active, this may give an overly optimistic sense of the impact

Additional Sources of Information

It is common in the internet industry to iterate through a large volume of models. Past experiment data may be suggestive towards the best hyperparameters for a new model

Many companies have developed “offline replay” systems which simulate the results that would have been seen had users been given a particular treatment.

Such sources of additional information can be leveraged to expedite optimization

"Personalized" Hyperparameter Tuning

It is plausible that different groups of members may have different affinity towards model hyperparameters

For instance, active users of a platform may be very interested in frequent updates

On the other hand, less active users may not appreciate frequent pings

This can be handled in a notification system by allowing a separate hyperparameter for the active and less active users.

Commonalities

- Each of these extensions are easily accommodated through modifications of the Bayesian optimization framework
- They are all used to improve the speed and accuracy of function fits

Time Varying Models

Time Dependence

A/B experiment metrics often see variations in time, such as day over-day fluctuations or differences on weekdays vs weekends or holidays

Bayesian optimizati $Y_i = f(x_i) + \varepsilon_i$ es data is observed as

a more realistic model might be to assume we have a time dependent error ter $Y_{i,t} = f(x_i) + \varepsilon_{i,t} + \varepsilon_t$ urement error:

Reformulating this model

The model incorporating time fluctuations can be expressed as

$$f(x_i) + \varepsilon_{i,t} + \varepsilon_t = f_t(x_i) + \varepsilon_{i,t}$$

That is, we are now trying to optimize for a function that is allowed to vary with time!

Practically, this can be accomplished by simple adjustments to the covariance kernel.

Time Dependent Gaussian Processes

Rather than assuming the function follows a Gaussian process, i.e.

$$f(x) = GP(\mu(x),\ K(x, x'))$$

we can instead assume that the function is a Gaussian process indexed by the hyperparameter and time:

$$f(x, t) = GP(\mu(x, t),\ K((x, t), (x', t')))$$

And Bayesian optimization proceeds exactly as in the time-invariant case!

A Simple Example of a Time Dependent Kernel

A simple example of a kernel for a time dependent Gaussian process is

$$K\left((x,t),(x',t')\right) = \sqrt{\epsilon}^{|t-t'|} \cdot K(x,x').$$

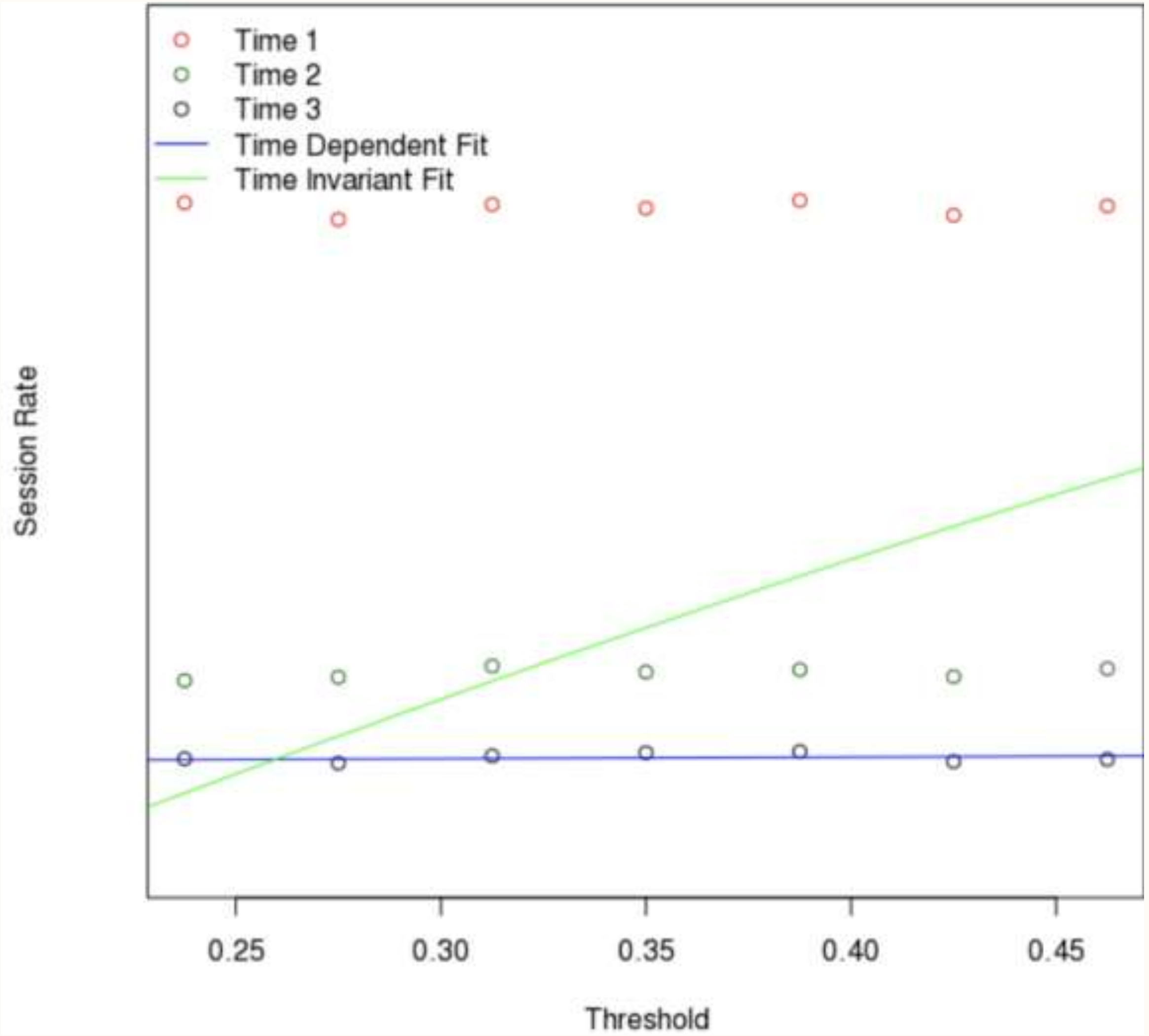
$$\mathrm{cov}(f_t(x), f_{t'}(x')) = \sqrt{\epsilon}^{|t-t'|} \cdot K(x,x')$$

which specifies

$$f_t(x) = \sqrt{\epsilon} f_{t-1}(x) + \sqrt{1-\epsilon} g_t(x)$$

and is equivalent to

An Example



Time 1
Time 2
Time 3
Time Dependent Fit
Time Invariant Fit
Session Rate
0.25
0.30
0.35
0.40
0.45
Threshold

Additional Sources of Information

Additional Sources of Information

It is common in the internet industry to iterate through a large volume of models. Past experiment data may be suggestive towards the best hyperparameters for a new model

Many companies have developed “offline replay” systems which simulate the results that would have been seen had users been given a particular treatment.

Such sources of additional information can be leveraged to expedite our predictions of the metrics of interest

Joint Model Of Experiment And Additional Information

Assume we observe metrics of interest as

$$Y_i = f(x_i) + \varepsilon_i$$

as well as another "proxy" metric

$$Z_i = g(x_i) + \varepsilon'_i$$

and we believe that g is informative of the function of interest f

We can jointly model the functions f and g to improve our predictions of f

Joint Gaussian Processes

We still assume $$f(\cdot) = GP(\mu(\cdot), K(\cdot,\cdot))$$

and also that $$g(\cdot) = GP(\mu(\cdot), K(\cdot,\cdot))$$

Further assume the Gaussian processes f and g are correlated. Namely $$cov(f(x), g(x')) = \varrho \cdot K(x,x')$$

(Note that the common marginal priors is not a
necessary assumption, but one typically made to simplify

Extensions To Multiple Sources Of Information

Suppose we are interested in modelling C different Gaussian Processes,

$$f_1(x), \ldots, f_C(x)$$

A simple joint model is specified by the Intrinsic Coregionalization Model (ICM). This

$$f_c(\cdot) = GP(\mu(\cdot), K(\cdot,\cdot))$$

and also jointly

$$cov(f_c(x), f_{c'}(x')) = b_{c,c'} \cdot K(x, x')$$

Notification Sessions Modeling

Send a notification if a score exceeds a threshold

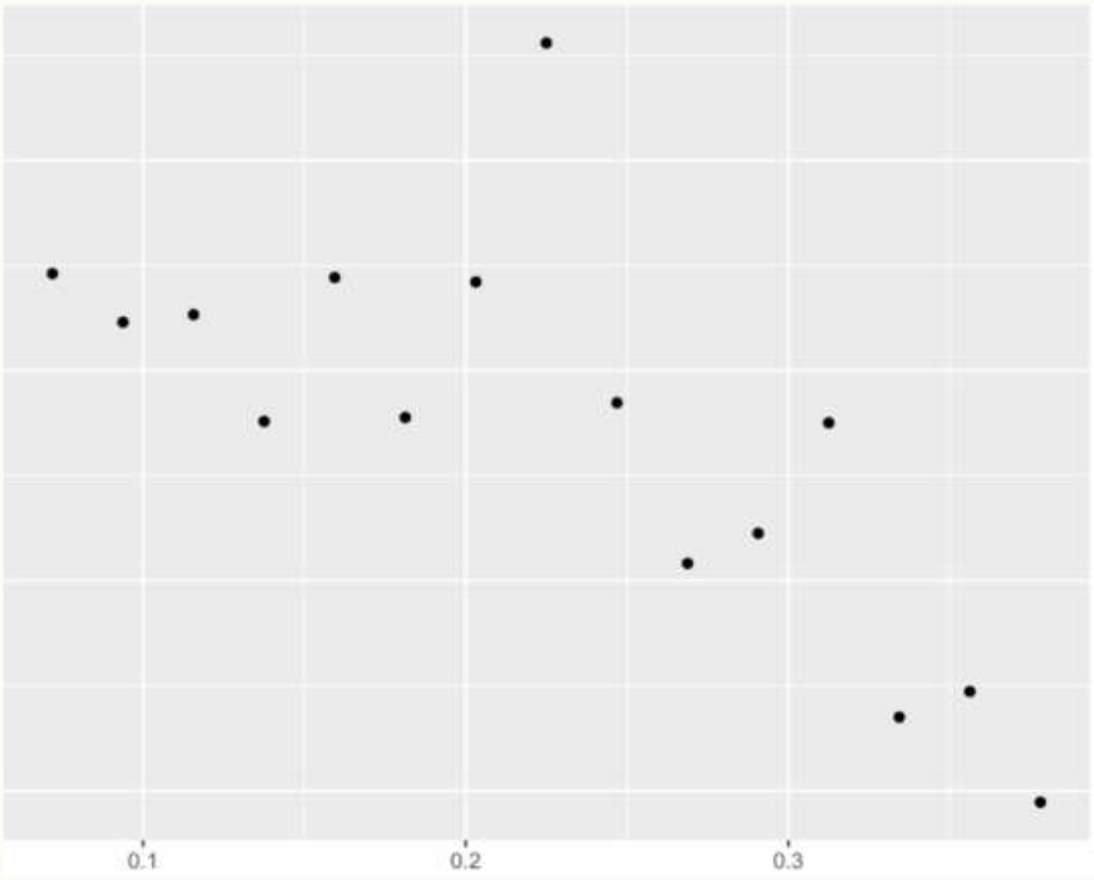
Our goal is to estimate the sessions that would be observed for a given threshold

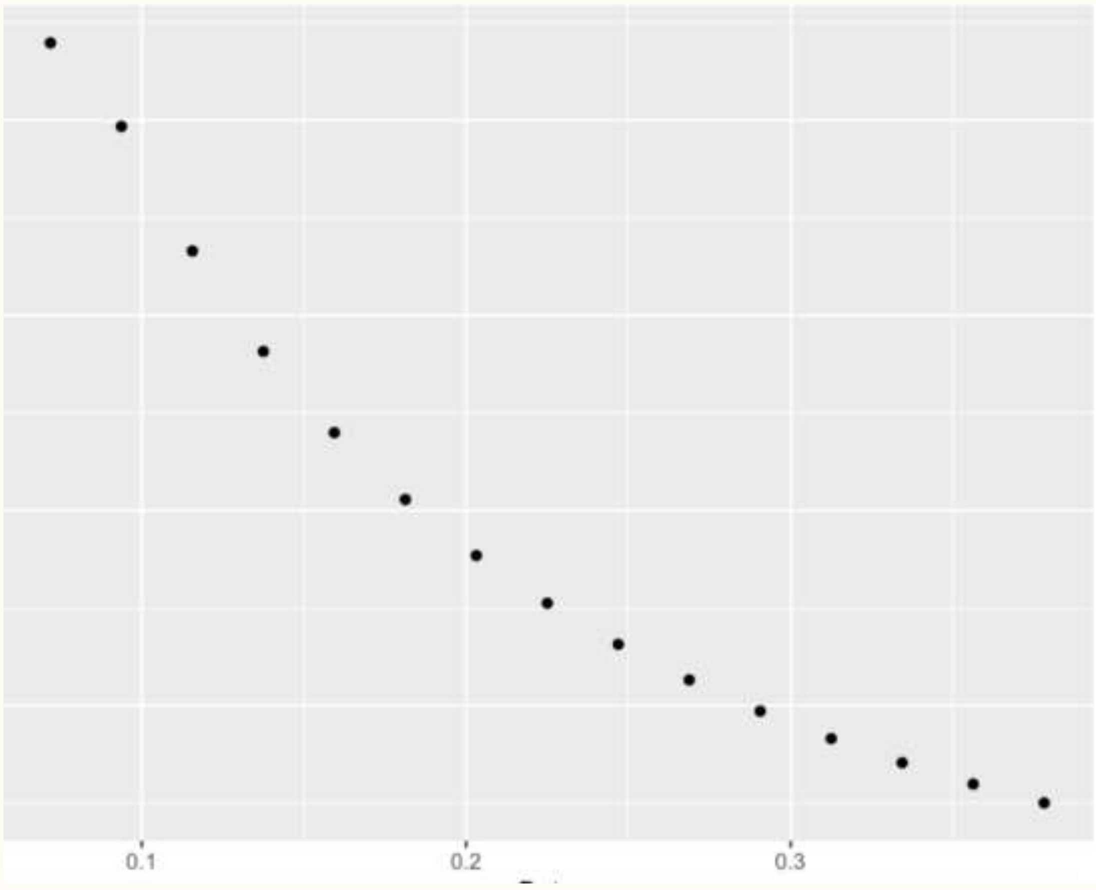
The sessions induced by sending a notification are easily modelled, for instance by a survival model

Based on offline data, we can estimate the induced sessions as

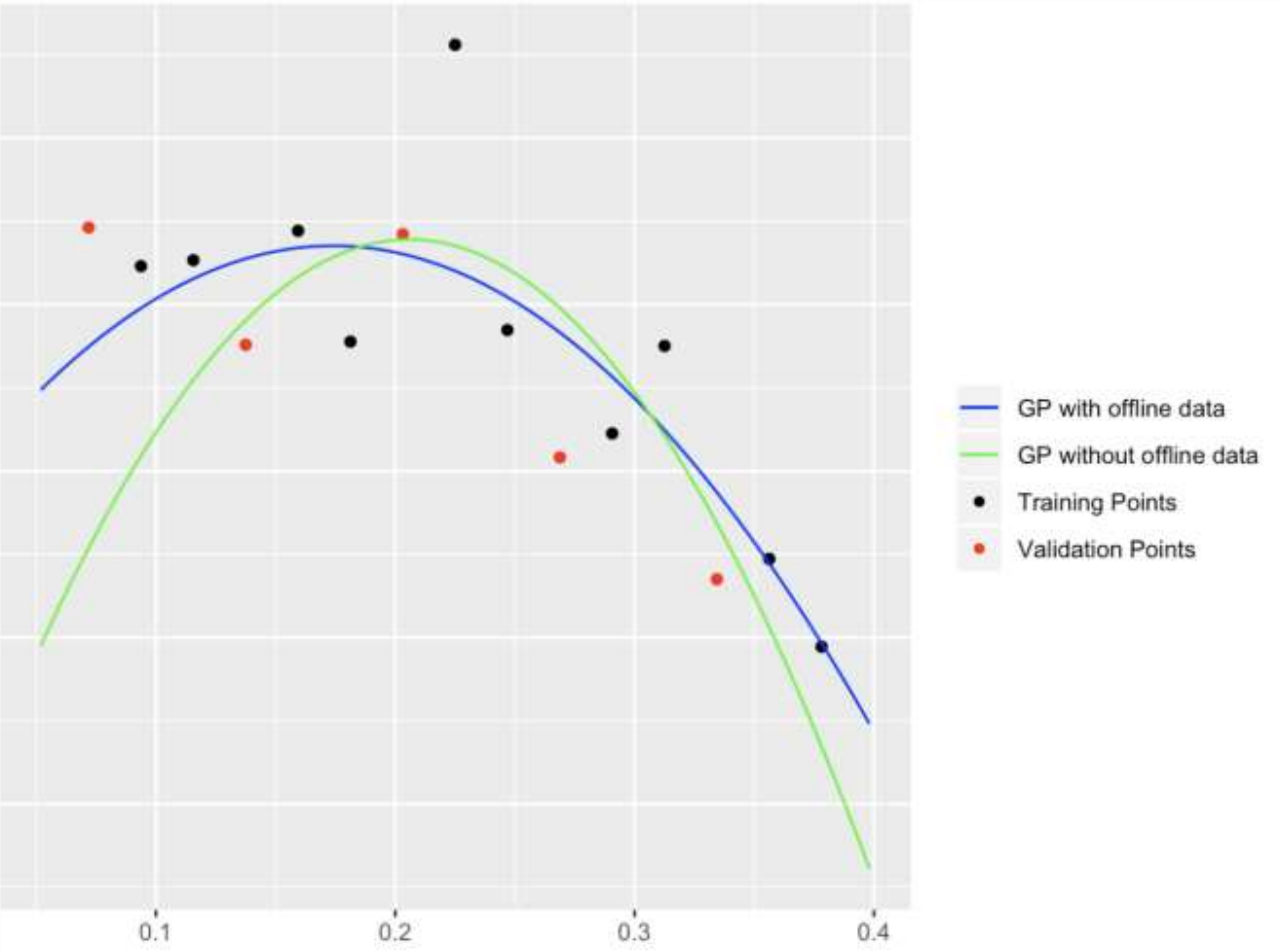
$$\sum_{score>threshold} P(session|sent) + \sum_{score<threshold} P(session|not\ sent)$$

Online vs Offline Sessions





Comparison of GP Fit With Left Out Data



Mean Squared Error:

- Without offline data:
  0.00928

- With offline data:
  0.00325

Relative efficiency: 285%

Group Level Optimization

Group Definitions

Often, segments of users can be given by ad-hoc definitions based on domain specific knowledge

Demographic segments (e.g. students vs full-time employees)
- Usage patterns (e.g. heavy vs light users)

Recently, advances have been made towards identifying heterogeneous cohorts of users which can be leveraged in the cohort definition phase

Causal Discovery of Groups

1. Launch an A/B experiment for a variety of hyperparameter values
2. Using this randomized experiment data, apply an automated method for discovering subgroups of the population which demonstrate heterogeneity in treatment effect
3. An example of such a tool is the "causal tree" methodology proposed by Athey and Imbens (2016)
4. This takes in user features (usage, demographics, etc), and results in a tree whose splits are chosen according to heterogeneity in treatment effect

Optimization by Group

We now want to give each group a hyperparameter value which globally optimizes the metrics of interest.

Suppose we have G groups, and each group g is exposed to hyperparameter x_g

Our optimization problem can be expressed as

$$\max_{x_1, \ldots x_g} f(x_1, \ldots, x_G)$$

which is generally not tractable

Grey Box Optimization

Generally, the metric of interest can be expressed as a (weighted) average of the metric across the cohorts

That is,

$$f(x_1, \ldots, x_G) = f_1(x_1) + \cdots + f_G(x_G)$$

so we estimate the objective metric (and similarly constraint metrics) separately for each cohort as a function of the hyperparameter for that cohort

This estimation requires dramatically less data than estimating a single function of G variables

Grey Box Optimization

Using the decomposition

$$f(x_1, \ldots, x_G) = f_1(x_1) + \cdots + f_G(x_G)$$

Bayesian optimizations (explore/exploit) proceeds exactly as previously seen

Now, the mean and variances of the objective and constraints are simply the sum of the independent GPs for each cohort

A Simple Example

Send a notification if a score exceeds a threshold

Ad-hoc cohorts based on usage frequency:

- Four by four: daily active users
- One by three: weekly active users
- One by one: monthly active users
- Onboarding
- Dormant

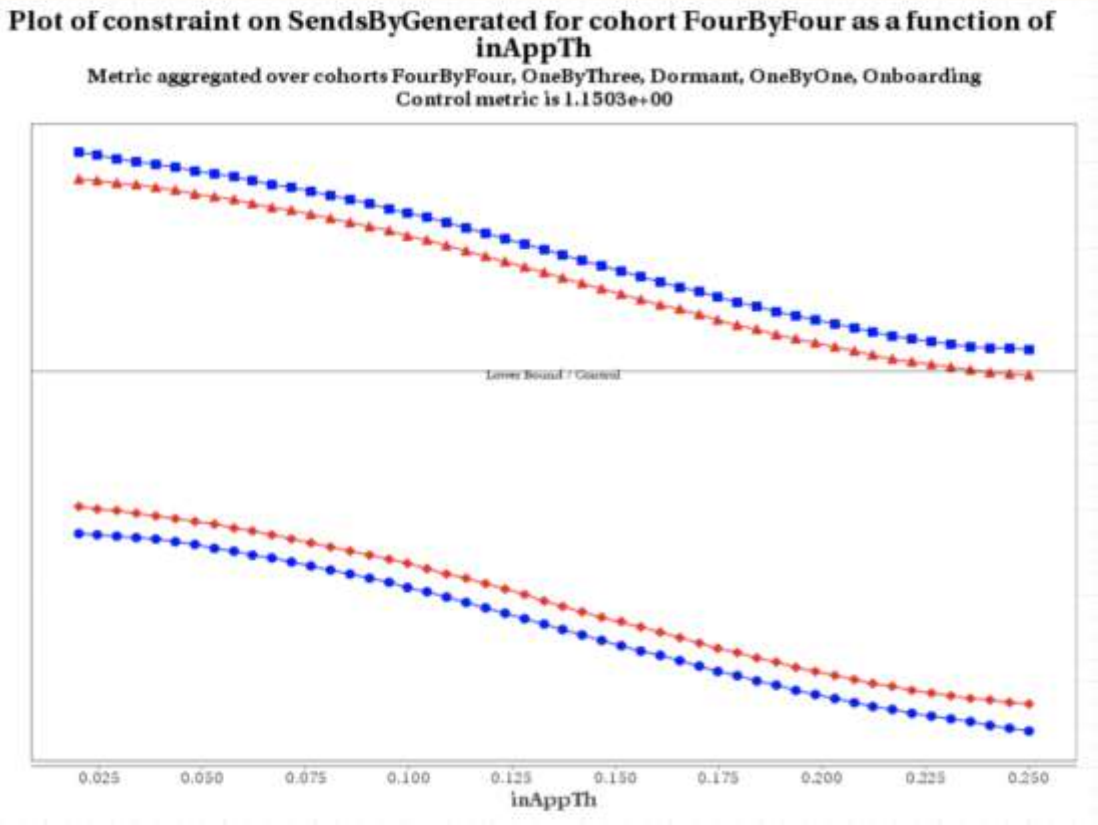
Influence Of One Parameter On Number Of Sent Messages

Marginal plot of Send Volume for FourByFour members

The two red lines are maximum and minimum over all cohorts except FourByFour

$$\max_{th_{1X3},\, th_{1X1},\, th_{dormant},\, th_{on}} f_{sends,4X4}(th_{4X4}) + f_{sends,1X3}(th_{1X3}) + f_{sends,1X1}(th_{1X1}) + f_{sends,dormant}(th_{dormant}) + f_{sends,on}(th_{on})$$

The blue lines are the variance estimates



Summary

- An advantage of Bayesian optimization is that it is easily modified to suit the nuances of the optimization problem at hand

- Some easy extensions
  - Incorporate time dependence seen in many online experiments
  - Incorporate additional sources of information
  - Personalize user experience through group specific optimization

Thank you

Marco Alvarado, Mavis Li, Yafei Wang, Jason Xu, Jinyun Yan, Shuo Yang, Wensheng Sun, Yiping Yuan, Ajith Muralidharan, Matthew Walker, Boyi Chen, Jun Jia, Haichao Wei, Huiji Gao, Zheng Li, Mohsen Jamali, Shipeng Yu, Hema Raghavan, Romer Rosales

LinkedIn