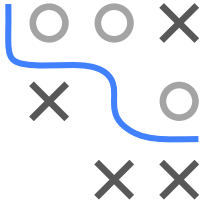
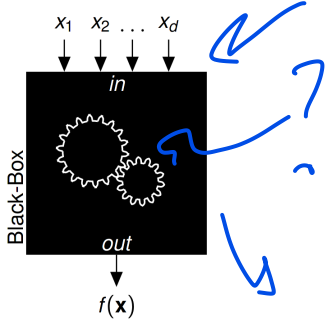
Optimization in Machine Learning



Bayesian Optimization
Black Box Optimization



Learning goals

- Definition and properties
- Examples
- Naive approaches

STANDARD VS. BLACK-BOX OPTIMIZATION

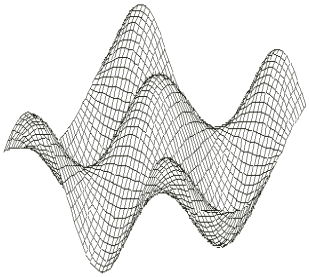
Optimization: Find

$$\min_{\mathbf{x} \in \mathcal{S}} f(\mathbf{x})$$

with objective function

$$f : \mathcal{S} \to \mathbb{R},$$

where $\mathcal{S}$ is usually box constrained.



If we are lucky ...

- ... we have an analytic description of $f : \mathcal{S} \to \mathbb{R}$
- ... we can calculate gradients and use gradient-based methods (e.g. gradient descent) for optimization

STANDARD VS. BLACK-BOX OPTIMIZATION

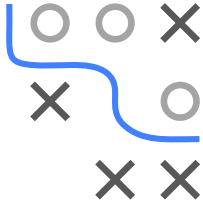
Optimization: Find

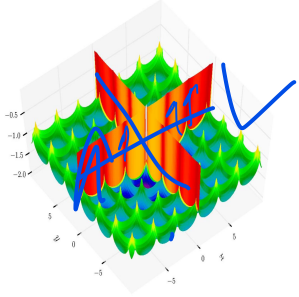
$$\min_{\mathbf{x}\in\mathcal{S}} f(\mathbf{x})$$

with objective function

$$f: \mathcal{S} \to \mathbb{R},$$

where $\mathcal{S}$ is usually box constrained.

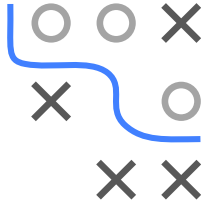




Optimization gets harder ...

- ... if we cannot calculate gradients (because f is not differentiable or f is not known to us)
- ... but as long as evaluations of f are cheap, we can use standard derivative-free optimization methods (e.g. Nelder-Mead, simulated annealing, EAs)

STANDARD VS. BLACK-BOX OPTIMIZATION



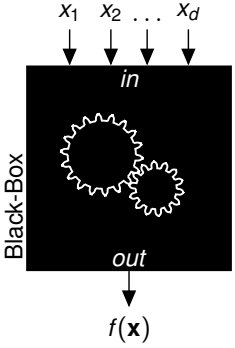
Optimization: Find

$$\min_{\mathbf{x}\in\mathcal{S}} f(\mathbf{x})$$

with objective function

$$f: \mathcal{S} \to \mathbb{R},$$

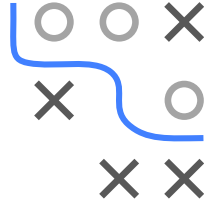
where $\mathcal{S}$ is usually box constrained.



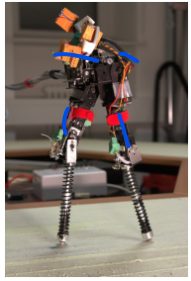
Optimization gets **really hard** if ...

- ... there is no analytic description of $f: \mathcal{S} \to \mathbb{R}$ (**black box**)
- ... evaluations of f for given values of **x** are **time consuming**

EXAMPLES FOR BAYESIAN OPTIMIZATION

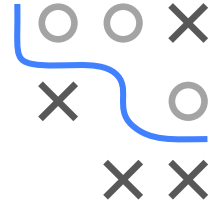


1. Robot Gait Optimization: The robot's gait is controlled by a **parameterized controller**



- **Goal:** Find parameters s.t. average velocity (directional speed) of the robot is maximized
- Parameters of the gait control e.g. joints of ankles and knees
- *Calandra et al. (2014). An Experimental Evaluation of Bayesian Optimization on Bipedal Locomotion*

EXAMPLES FOR BAYESIAN OPTIMIZATION



❷ Optimization of a cookie recipe

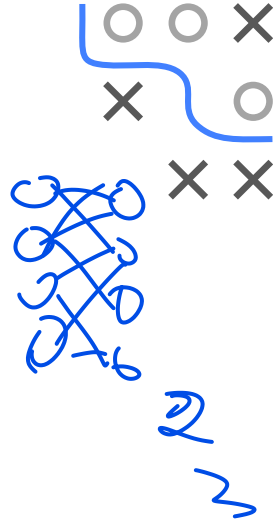
https://www.bettycrocker.com

Ingredient	Salt (tsp)†	Total Sugar (g)	Brown Sugar (%)	Vanilla (tsp)†	Chip Quantity (g)	Chip Type
Min	0	150	0	0.25	114	{Dark, Milk,
Max	0.5	500	1	1	228	White}

- **Goal:** Find “optimal” composition and amounts of ingredients
- **Evaluation:** Cookies are baked according to the recipe, tested and rated by volunteers
- *Kochanski et al. (2017). Bayesian Optimization for a Better Dessert*

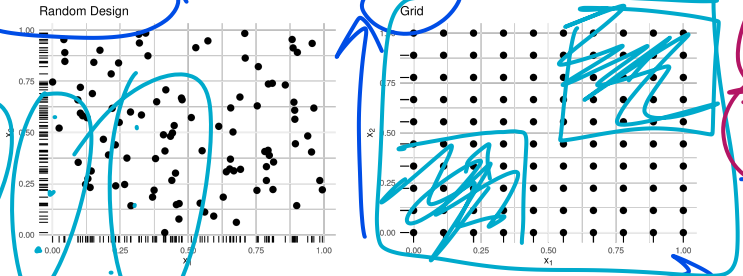
NAIVE APPROACHES

1. Empirical knowledge / manual tuning
   - Select parameters based on “expert” knowledge
   - **Advantages:** Can lead to fairly good outcomes for known problems
   - **Disadvantages:** Very (!) inefficient, poor reproducibility, chosen solution can also be far away from a global optimum



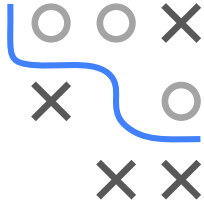
NAIVE APPROACHES

2. Random search / Grid search
   - Random search: Evaluate uniformly sampled inputs
   - Grid search: Exhaustive search of a predefined grid of inputs
   - **Advantages:** Easy, intuitive, parallelization is trivial
   - **Disadvantages:** Inefficient, search large irrelevant areas

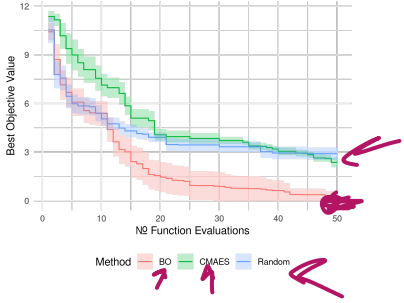


Rug plots of RS vs. GS.

NAIVE APPROACHES



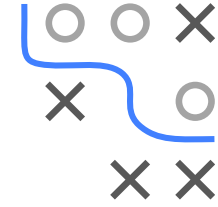
3. Traditional black-box optimization
   - Traditional approaches that do not require derivatives
   - E.g. Nelder-Mead, simulated annealing, EAs
   - **Advantages:** Truly iterative, focuses on relevant regions
   - **Disadvantages:** Still inefficient; usually lots of evaluations are needed to produce good outcomes

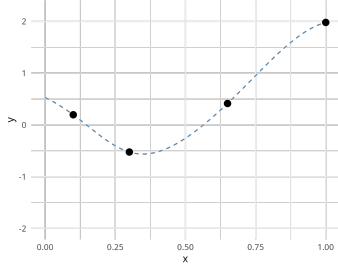


BO vs. CMAES vs. RS on 2D Ackley.

Optimization in Machine Learning

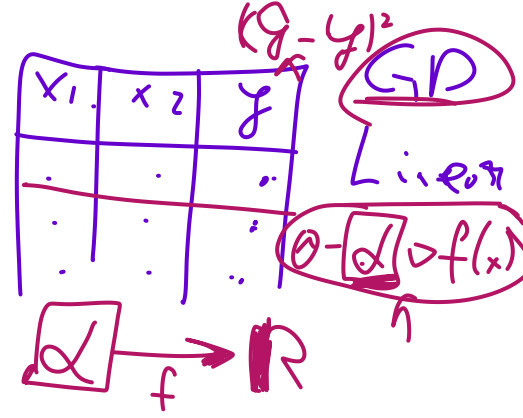
Bayesian Optimization
Basic BO Loop and Surrogate Modelling





Learning goals

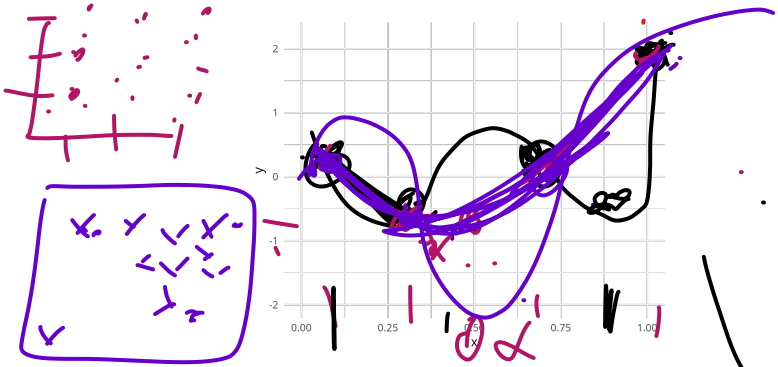
- Initial design
- Surrogate modeling
- Basic loop

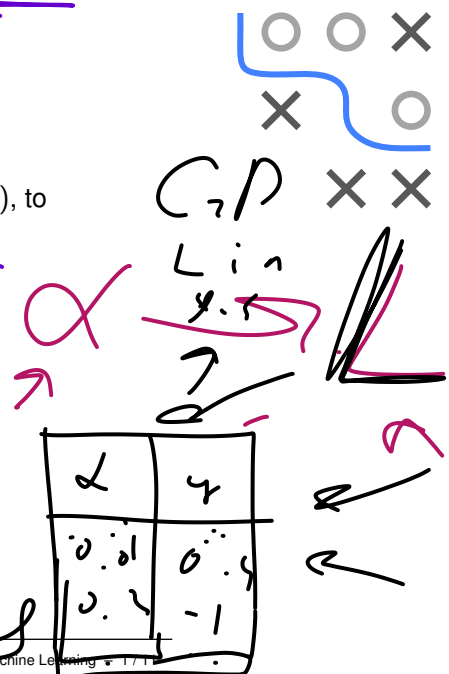


OPTIMIZATION VIA SURROGATE MODELING

Starting point:

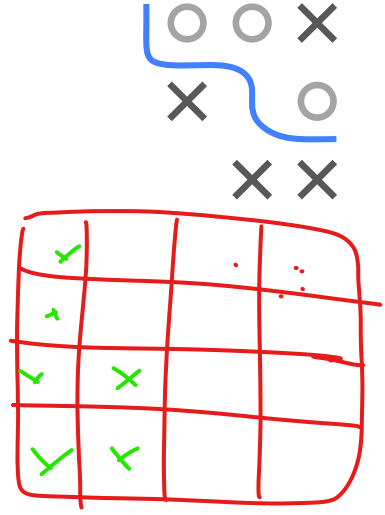
- We do not know the objective function $f : \mathcal{S} \to \mathbb{R}$
- But we can evaluate f for a few different inputs $\mathbf{x} \in \mathcal{S}$
- For now we assume that those evaluations are noise-free
- **Idea:** Use the data $\mathcal{D}^{[t]} = \{(\mathbf{x}^{[i]}, y^{[i]})\}_{i=1,\dots,t}$, $y^{[i]} := f(\mathbf{x}^{[i]})$, to derive properties about the unknown function f





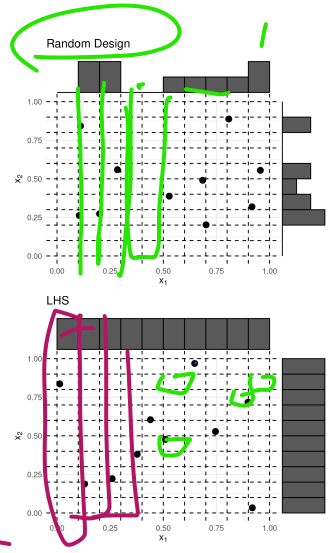
INITIAL DESIGN

- Should cover / explore input space sufficiently:
  - Random design
  - Latin hypercube sampling
  - Sobol sampling
- Type of design usually has not the largest effect
- A more important choice is the **size** of the initial design
  - Should neither be too small (bad initial fit) nor too large (spending too much budget without doing “intelligent” optimization)
  - Rule of thumb: $4d$



LATIN HYPERCUBE SAMPLING

- LHS partitions the search space into bins of equal probability
- Goal is to attain a more even distribution of sample points than random sampling
- Allow at most one sample per bin; exactly one sample per row and column

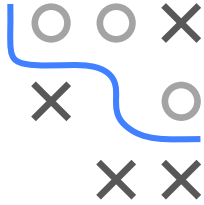


Marginal histograms RS vs. LHS

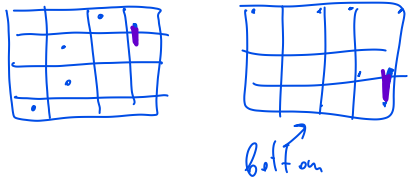
LATIN HYPERCUBE SAMPLING

Actual sampling of points, e.g., constructed via **Maximin**:

- The minimum distance between any two points in $\mathcal{D}$ is $2q = \min_{\mathbf{x}\in\mathcal{D},\mathbf{x}'\in\mathcal{D}} \rho(\mathbf{x},\mathbf{x}')$ (ρ any metric, e.g., Euclidean distance)
- q is the packing radius - the radius of the largest ball that can be placed around every design point such that no two balls overlap
- Goal: Find $\mathcal{D}$ that maximizes $2q$: $\max_{\mathcal{D}} \min_{\mathbf{x}\in\mathcal{D},\mathbf{x}'\in\mathcal{D}} \rho(\mathbf{x},\mathbf{x}')$
- Ensures that the design points in $\mathcal{D}$ are as far apart from each other as possible

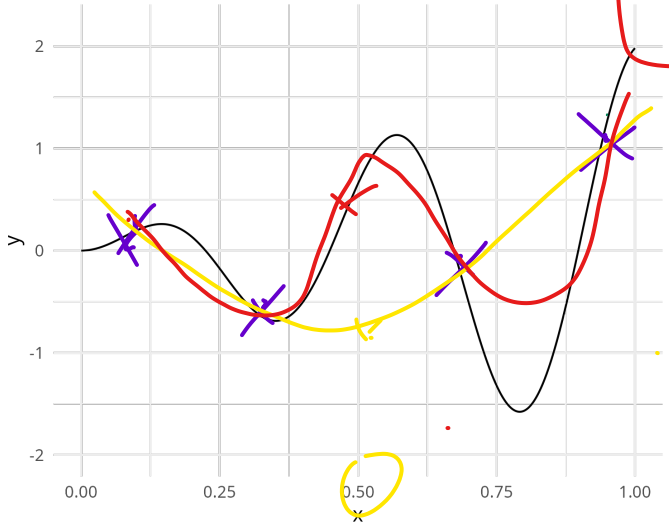


maximize the minimum distance between pairs of points



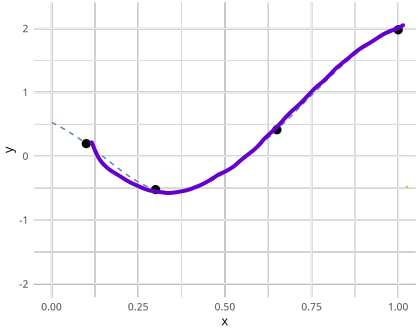
SURROGATE MODELING

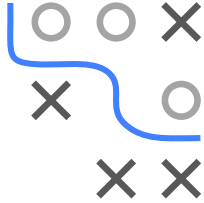
Running example = minimize this “black-box”:



SURROGATE MODELING

1. **Fit** a **regression model** $\hat{f} : \mathcal{D}^{[t]} \to \mathbb{R}$ (blue) to extract maximum information from the design points (black) and learn properties of f

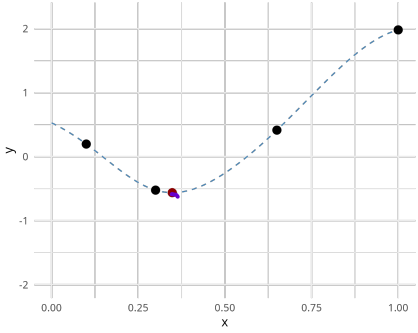


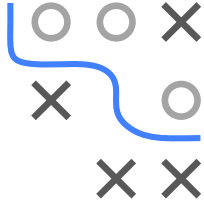


As we can eval f without noise, we fit an interpolator

SURROGATE MODELING

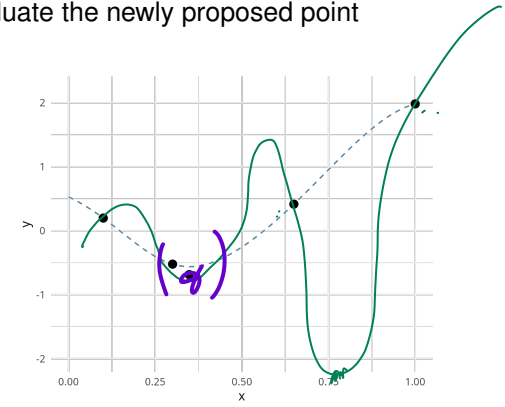
❷ Instead of the expensive f, we optimize the cheap surrogate $\hat{f}$ (blue) to **propose** a new point (red) for evaluation

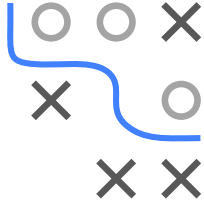




SURROGATE MODELING

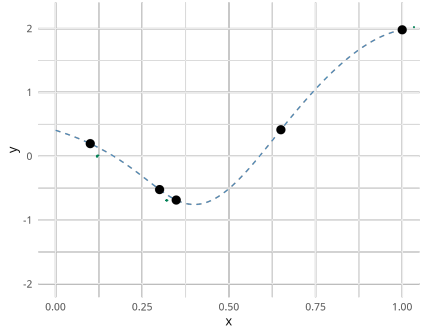
❸ We finally evaluate the newly proposed point

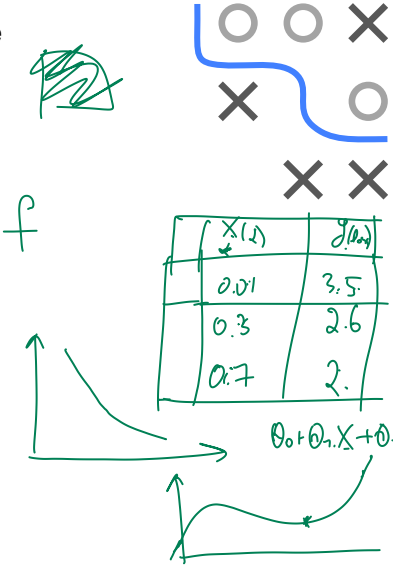




SURROGATE MODELING

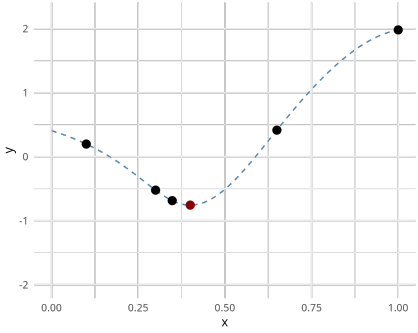
- After evaluation of the new point, we **adjust** the model on the expanded dataset via (slower) refitting or a (cheaper) online update

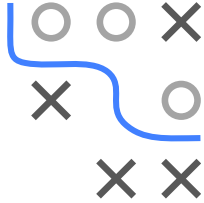




SURROGATE MODELING

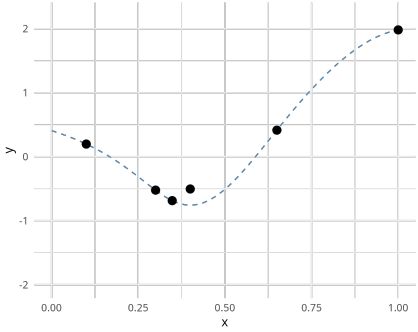
- We again obtain a new candidate point (red) by optimizing the cheap surrogate model function (blue) ...

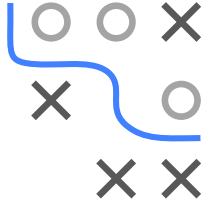




SURROGATE MODELING

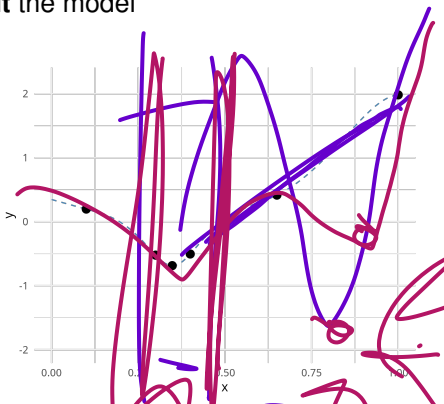
- ... and evaluate that candidate

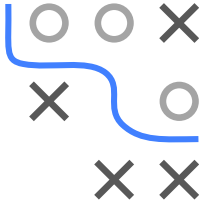




SURROGATE MODELING

- We repeat: (i) **fit** the model



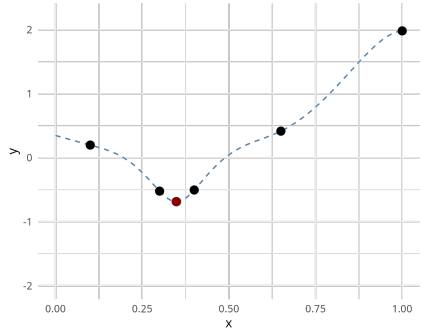


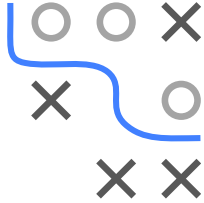
exploit,

explore

SURROGATE MODELING

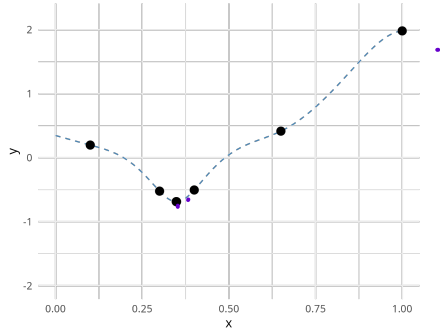
- (ii) **propose** a new point

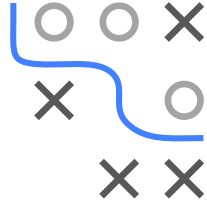




SURROGATE MODELING

- (iii) **evaluate** that point



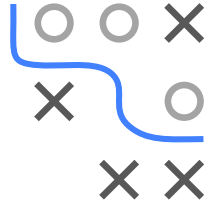


- We observe that the algorithm converged

BASIC LOOP

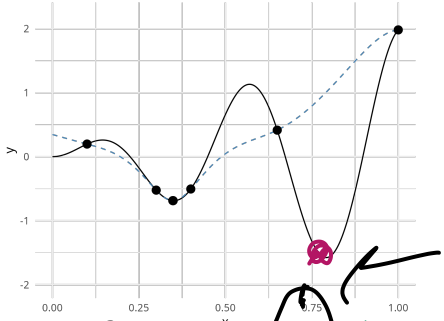
The basic loop of our sequential optimization procedure is:

1. Fit surrogate model $\hat{f}$ on previous evaluations $\mathcal{D}^{[t]} = \{(\mathbf{x}^{[i]}, y^{[i]})\}_{i=1,\ldots,t}$
2. Optimize the surrogate model $\hat{f}$ to obtain a new point $\mathbf{x}^{[t+1]} := \arg\min_{\mathbf{x} \in \mathcal{S}} \hat{f}(\mathbf{x})$
3. Evaluate $\mathbf{x}^{[t+1]}$ and update data $\mathcal{D}^{[t+1]} = \mathcal{D}^{[t]} \cup \{(\mathbf{x}^{[t+1]}, f(\mathbf{x}^{[t+1]}))\}$

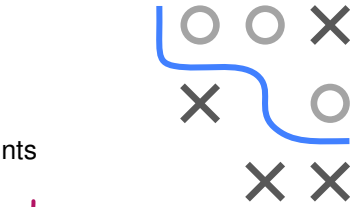


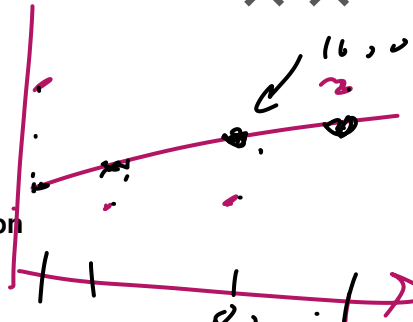
EXPLORATION VS. EXPLOITATION

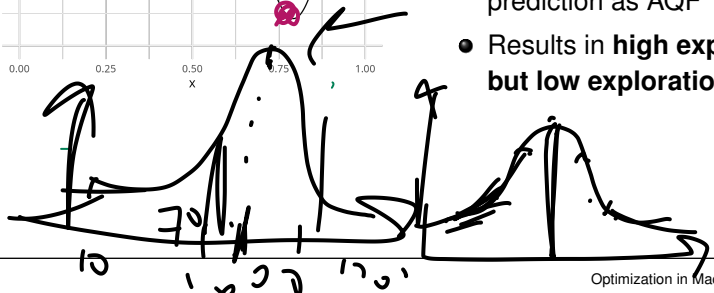
We see: We ran into a local minimum. We did not “explore” the most crucial areas and **missed** the global minimum.



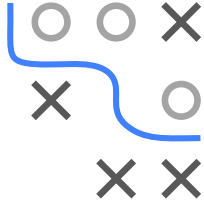
- Better ways to propose points based on our model exist, so-called **acquisition functions**
- Optimizing SM directly corresponds to raw / mean prediction as AQF
- Results in **high exploitation but low exploration**



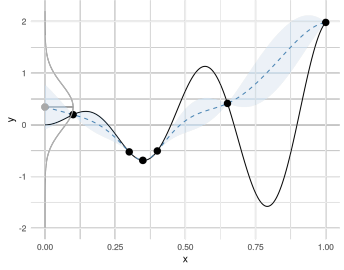




Optimization in Machine Learning



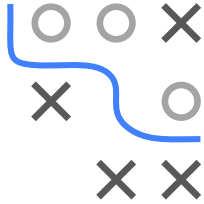
Bayesian Optimization
Posterior Uncertainty and Acquisition Functions I



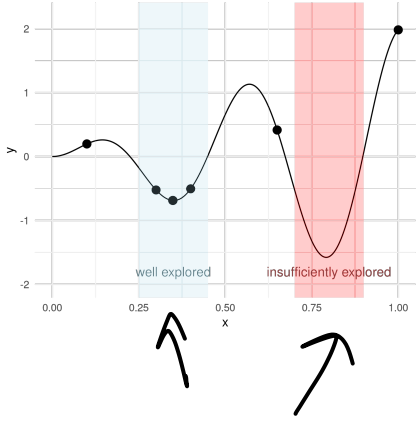
Learning goals

- Bayesian surrogate modeling
- Acquisition functions
- Lower confidence bound

BAYESIAN SURROGATE MODELING

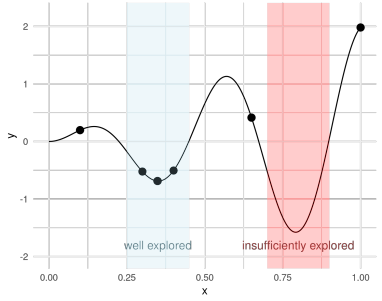


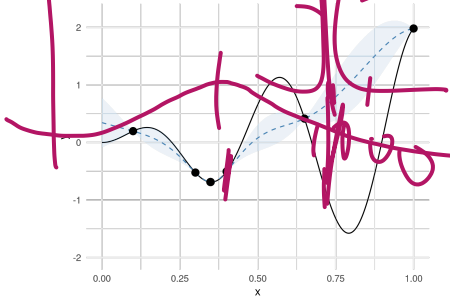
Goal:
Find trade-off between **exploration** (areas we have not visited yet) and **exploitation** (search around good design points)

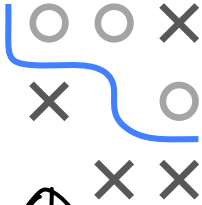


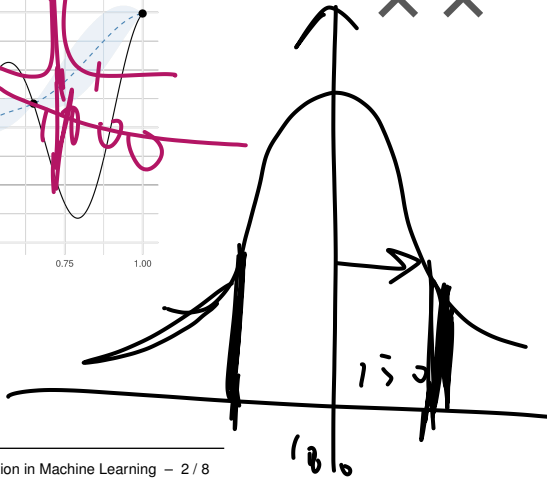
BAYESIAN SURROGATE MODELING

- **Idea**: Use a **Bayesian approach** to build SM that yields estimates for the posterior mean $\hat{f}(\mathbf{x})$ and the posterior variance $\hat{s}^2(\mathbf{x})$
- $\hat{s}^2(\mathbf{x})$ expresses "confidence"/"certainty" in prediction



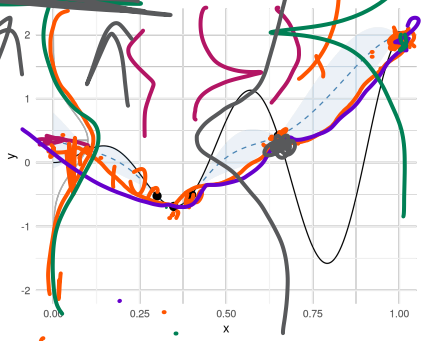






BAYESIAN SURROGATE MODELING

- Denote by $Y \mid \mathbf{x}, \mathcal{D}^{[t]}$ the (conditional) RV associated with the posterior predictive distribution of a new point $\mathbf{x}$ under a SM; will abbreviate it as $Y(\mathbf{x})$
- Most prominent choice for a SM is a **Gaussian process**, here $Y(\mathbf{x}) \sim \mathcal{N}\left(\hat{f}(\mathbf{x}), \hat{s}^2(\mathbf{x})\right)$

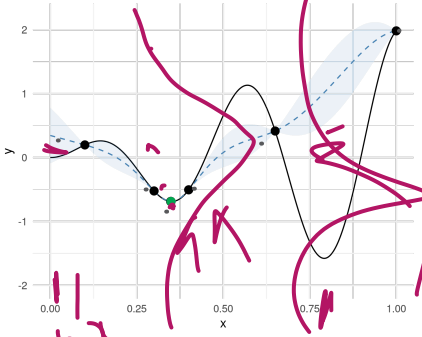


For now we assume an interpolating SM; $\hat{f}(\mathbf{x}) = f(\mathbf{x})$ and $\hat{s}(\mathbf{x}) = 0$ for training points

ACQUISITION FUNCTIONS

To sequentially propose new points based on the SM, we make use of so-called acquisition functions $a : \mathcal{S} \to \mathbb{R}$

Let $f_{\min} := \min \left\{ f(\mathbf{x}^{[1]}), \ldots, f(\mathbf{x}^{[t]}) \right\}$ denote the best observed value so far (visualized in green - we will need this later!)



In the examples before we simply used the posterior mean $a(\mathbf{x}) = \hat{f}(\mathbf{x})$ as acquisition function - ignoring uncertainty

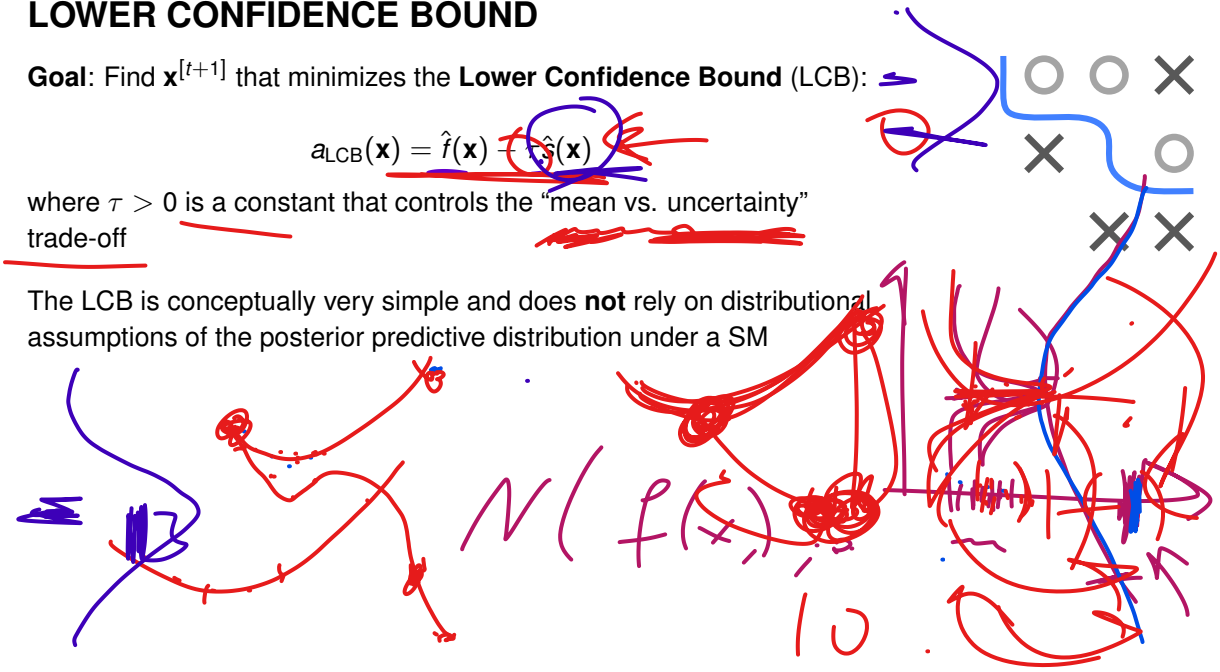
LOWER CONFIDENCE BOUND

Goal: Find $\mathbf{x}^{[t+1]}$ that minimizes the **Lower Confidence Bound** (LCB):

$$a_{\text{LCB}}(\mathbf{x}) = \hat{f}(\mathbf{x}) - \tau \hat{s}(\mathbf{x})$$

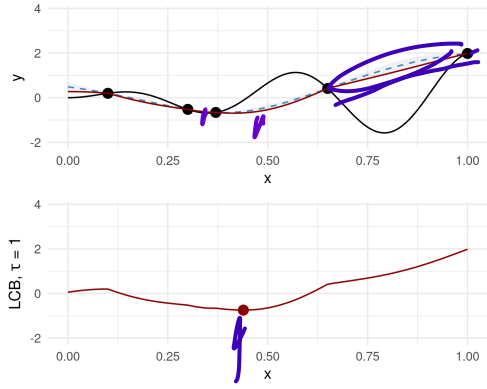
where $\tau > 0$ is a constant that controls the "mean vs. uncertainty" trade-off

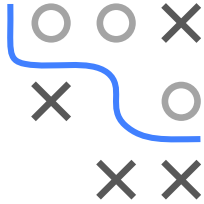
The LCB is conceptually very simple and does **not** rely on distributional assumptions of the posterior predictive distribution under a SM



LOWER CONFIDENCE BOUND

$\tau = 1$

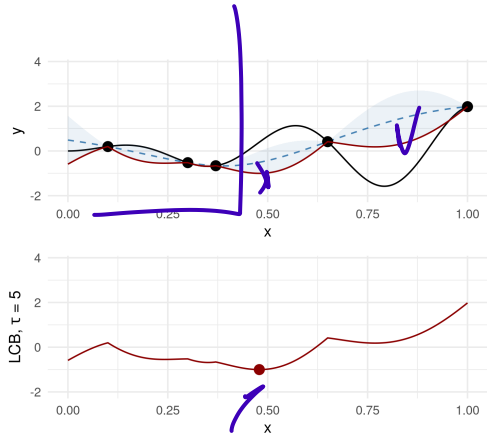


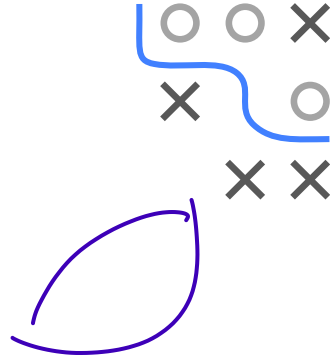


Top: Design points and SM showing $\hat{f}(\mathbf{x})$ (blue) and $\hat{f}(\mathbf{x}) - \tau\hat{s}(\mathbf{x})$ (red)
Bottom: the red point depicts $\arg\min_{\mathbf{x}\in\mathcal{S}} a_{\text{LCB}}(\mathbf{x})$

LOWER CONFIDENCE BOUND

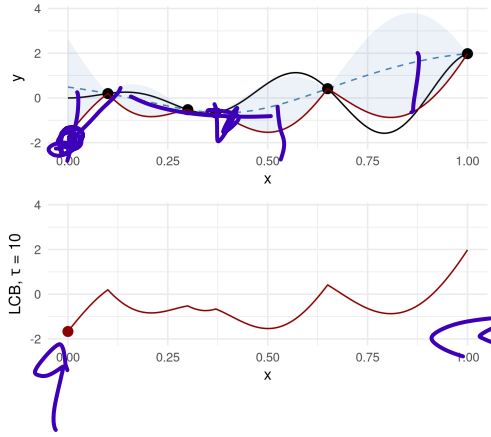
$\tau = 5$

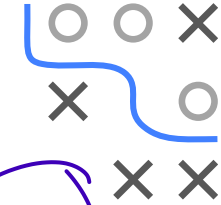




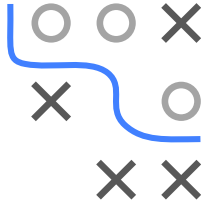
LOWER CONFIDENCE BOUND

$\tau = 10$

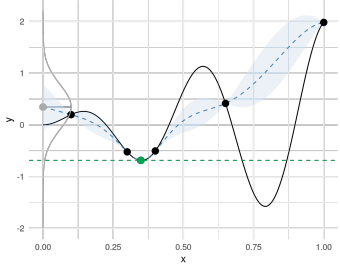




Optimization in Machine Learning



Bayesian Optimization
Posterior Uncertainty and Acquisition Functions II



Learning goals

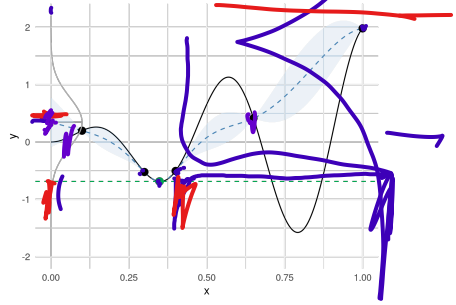
- Probability of improvement
- Expected improvement

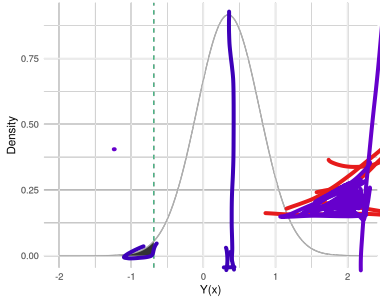
PROBABILITY OF IMPROVEMENT

Goal: Find $\mathbf{x}^{[t+1]}$ that maximizes the **Probability of Improvement** (PI):

$$a_{\text{PI}}(\mathbf{x}) = \mathbb{P}(Y(\mathbf{x}) \lesssim f_{\min}) = \Phi\left(\frac{f_{\min} - \hat{f}(\mathbf{x})}{\hat{s}(\mathbf{x})}\right)$$

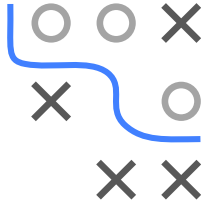
where $\Phi(\cdot)$ is the standard normal cdf (assuming Gaussian posterior)





Left: The green vertical line represents $f_{\min}$. **Right:** $a_{\text{PI}}(\mathbf{x})$ is given by the black area.

PROBABILITY OF IMPROVEMENT



Goal: Find $\mathbf{x}^{[t+1]}$ that maximizes the **Probability of Improvement** (PI):

$$a_{\text{PI}}(\mathbf{x}) = \mathbb{P}(Y(\mathbf{x}) < f_{\min}) = \Phi\left(\frac{f_{\min} - \hat{f}(\mathbf{x})}{\hat{s}(\mathbf{x})}\right)$$

where $\Phi(\cdot)$ is the standard normal cdf (assuming Gaussian posterior)

Note: $a_{\text{PI}}(\mathbf{x}) = 0$ for design points $\mathbf{x}$, since

- $\hat{s}(\mathbf{x}) = 0$,
- $\hat{f}(\mathbf{x}) = f(\mathbf{x}) \geq f_{\min} \quad \Leftrightarrow \quad f_{\min} - \hat{f}(\mathbf{x}) \leq 0.$

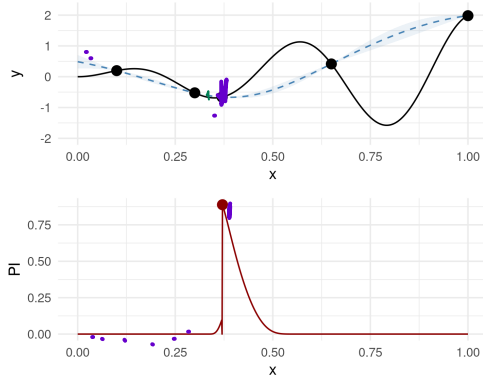
Therefore:

$$\Phi\left(\frac{f_{\min} - \hat{f}(\mathbf{x})}{\hat{s}(\mathbf{x})}\right) = \Phi\left(-\infty\right) = 0$$

PROBABILITY OF IMPROVEMENT

The PI does not take the size of the improvement into account
Often it will propose points close to the current $f_{\min}$

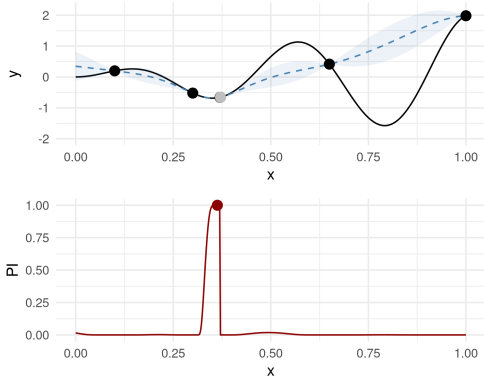
We use the PI (red line) to propose the next point ...

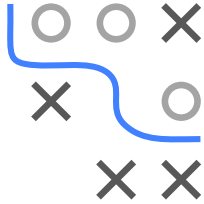


The red point depicts $\arg\max_{\mathbf{x} \in \mathcal{S}} a_{\text{PI}}(\mathbf{x})$

PROBABILITY OF IMPROVEMENT

... evaluate that point, refit the SM and propose the next point

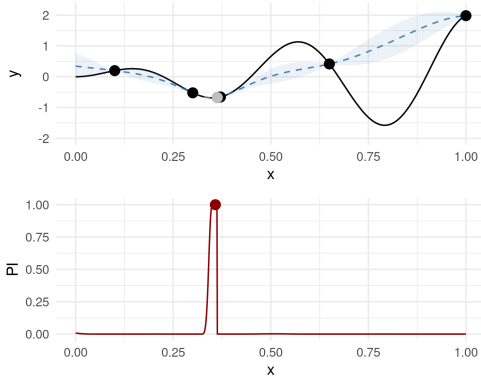


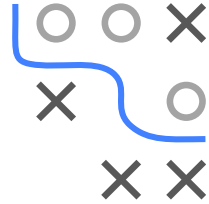


(grey point = prev point from last iter)

PROBABILITY OF IMPROVEMENT

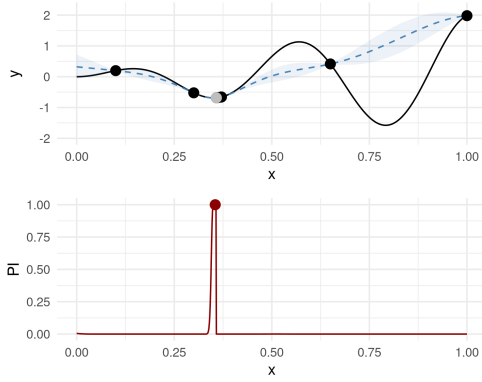
...

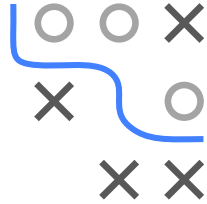




PROBABILITY OF IMPROVEMENT

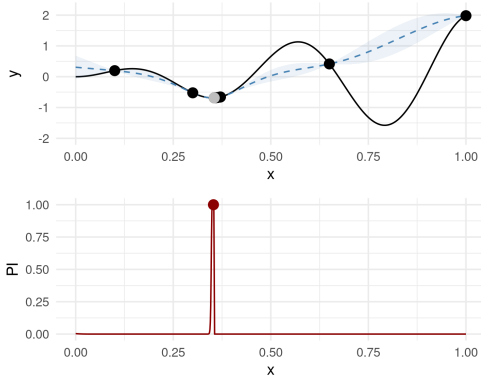
In our example, using the PI results in spending plenty of time optimizing the local optimum ...

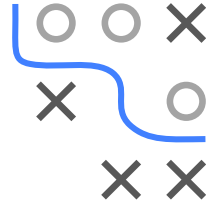




PROBABILITY OF IMPROVEMENT

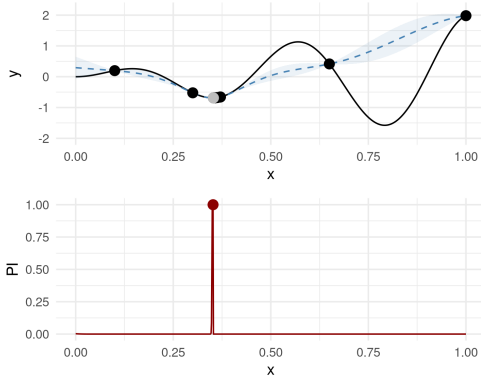
...

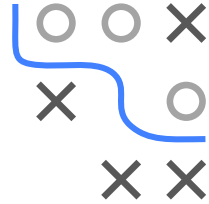




PROBABILITY OF IMPROVEMENT

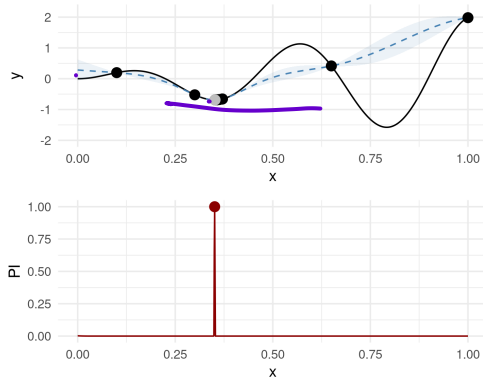
...

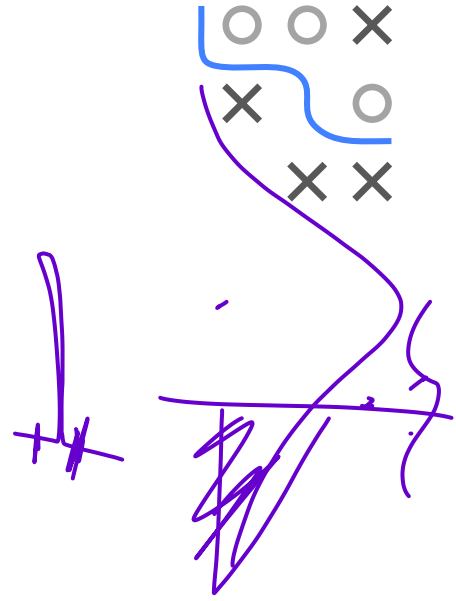




PROBABILITY OF IMPROVEMENT

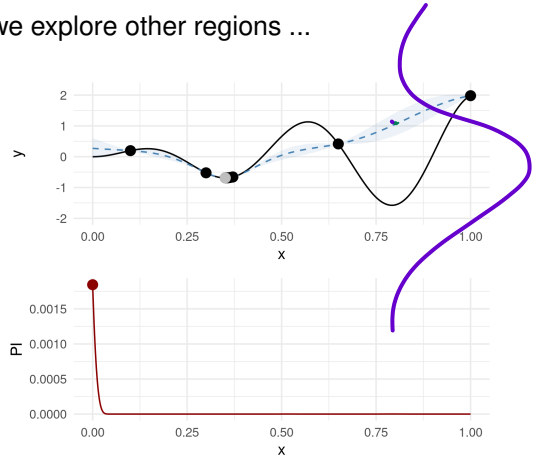
...

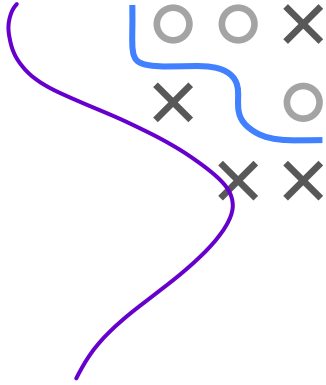




PROBABILITY OF IMPROVEMENT

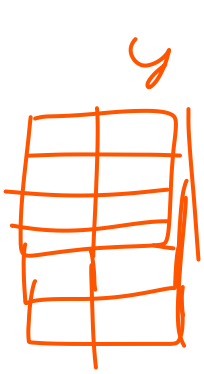
... eventually, we explore other regions ...

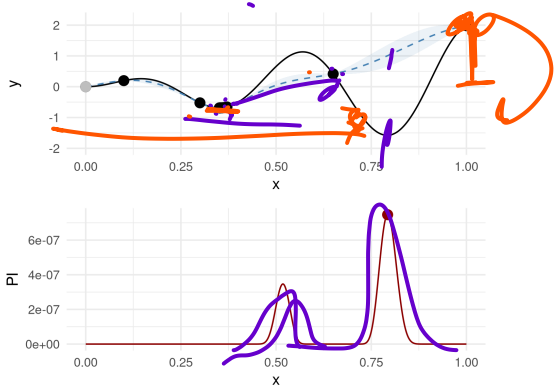


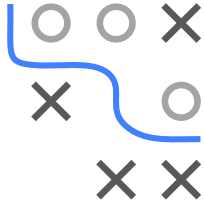


PROBABILITY OF IMPROVEMENT

...



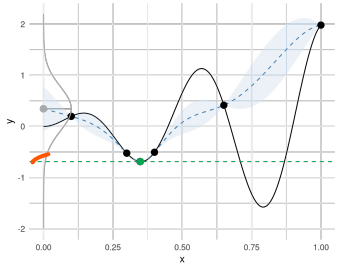


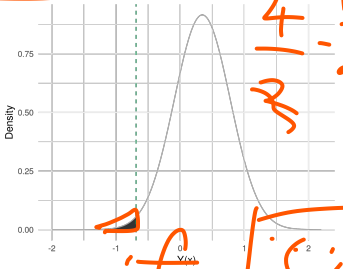


EXPECTED IMPROVEMENT

Goal: Propose $\mathbf{x}^{[t+1]}$ that maximizes the **Expected Improvement** (EI):

$$a_{\text{EI}}(\mathbf{x}) \quad = \quad \mathbb{E}(\max\{f_{\min} - Y(\mathbf{x}), 0\})$$



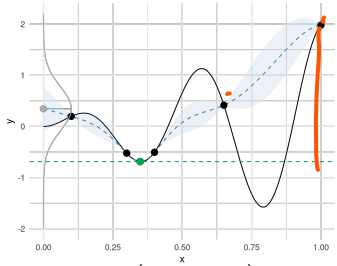


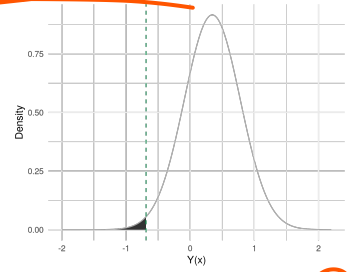
- We now take the expectation in the tail, instead of the prob as in PI.
- Improvement is always assumed ≥ 0.

EXPECTED IMPROVEMENT

Goal: Propose $\mathbf{x}^{[t+1]}$ that maximizes the **Expected Improvement** (EI):

$$a_{\text{EI}}(\mathbf{x}) \quad = \quad \mathbb{E}(\max\{f_{\min} - Y(\mathbf{x}), 0\})$$



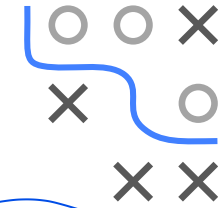


If $Y(\mathbf{x}) \sim \mathcal{N}\left(\hat{f}(\mathbf{x}), \hat{s}^2(\mathbf{x})\right)$, we can express the EI in closed-form as:

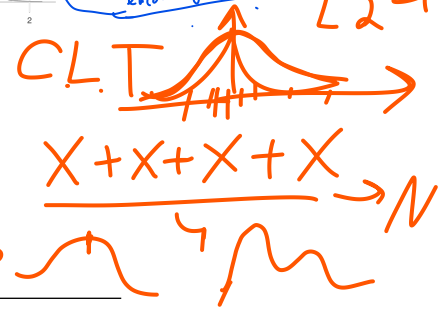
$$a_{\text{EI}}(\mathbf{x}) = (f_{\min} - \hat{f}(\mathbf{x}))\Phi\Big(\frac{f_{\min} - \hat{f}(\mathbf{x})}{\hat{s}(\mathbf{x})}\Big) + \hat{s}(\mathbf{x})\phi\Big(\frac{f_{\min} - \hat{f}(\mathbf{x})}{\hat{s}(\mathbf{x})}\Big),$$

- $a_{\text{EI}}(\mathbf{x}) = 0$ at design points $\mathbf{x}$:

$$a_{\text{EI}}(\mathbf{x}) = (f_{\min} - \hat{f}(\mathbf{x}))\underbrace{\Phi\Big(\frac{f_{\min} - \hat{f}(\mathbf{x})}{\hat{s}(\mathbf{x})}\Big)}_{=0,\ \text{see PI}} + \underbrace{\hat{s}(\mathbf{x})}_{=0}\phi\Big(\frac{f_{\min} - \hat{f}(\mathbf{x})}{\hat{s}(\mathbf{x})}\Big)$$

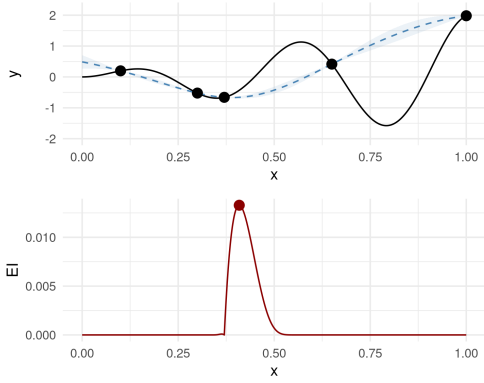


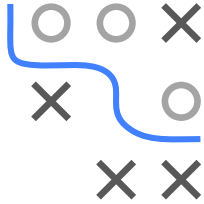
Assumes that uncertainty does not harm as which is unrealistic but encourages exploration



EXPECTED IMPROVEMENT

We use the EI (red line) to propose the next point ...

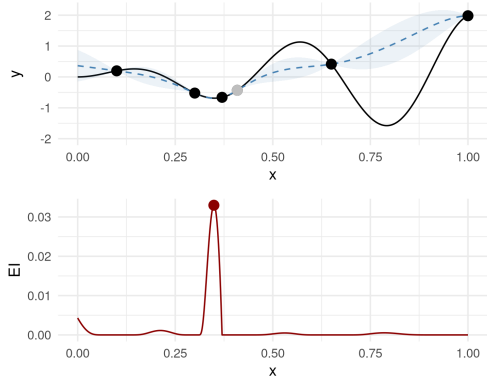


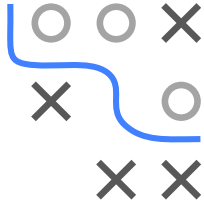


The red point depicts $\arg\max_{\mathbf{x} \in \mathcal{S}} a_{\text{EI}}(\mathbf{x})$

EXPECTED IMPROVEMENT

... evaluate that point, refit the SM and propose the next point

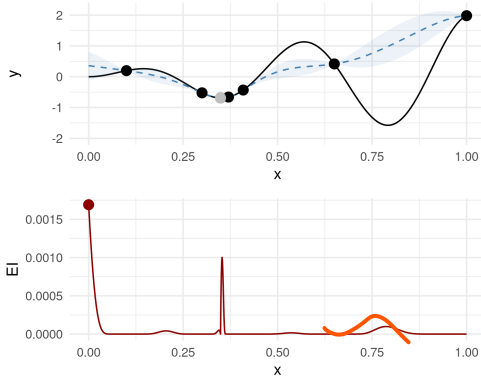


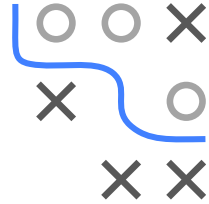


(grey point = prev point from last iter)

EXPECTED IMPROVEMENT

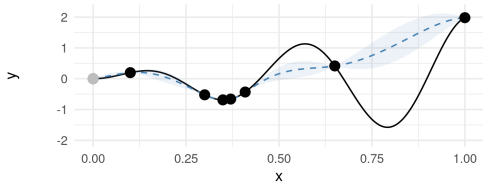
...

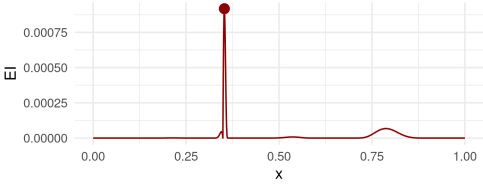


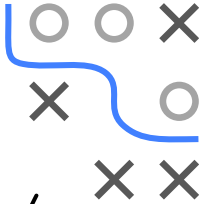


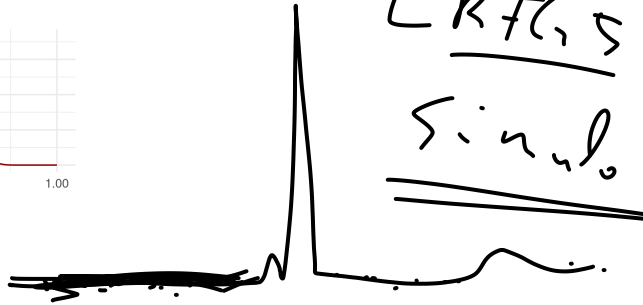
EXPECTED IMPROVEMENT

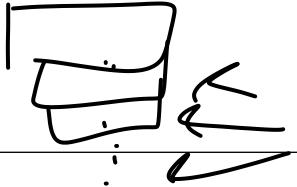
...





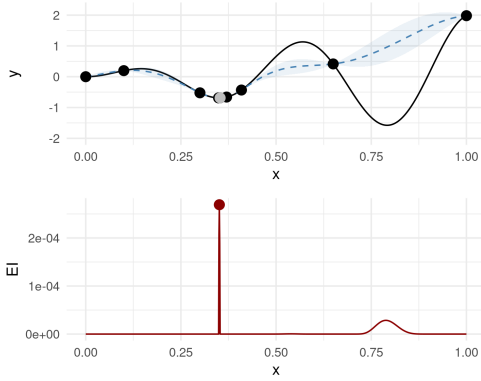


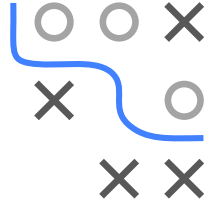




EXPECTED IMPROVEMENT

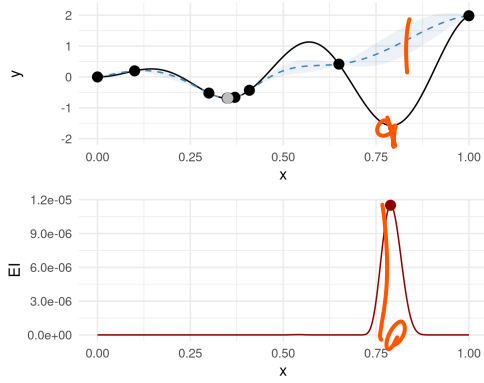
...





EXPECTED IMPROVEMENT

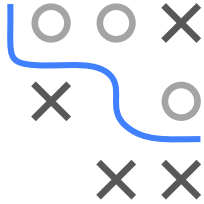
The EI is capable of exploration and quickly proposes promising points in areas we have not visited yet



Here, also a result of well-calibrated uncertainty $\hat{s}(\mathbf{x})$ of our GP.

DISCUSSION

- Under some mild conditions: BO with a GP as SM and EI is a **global optimizer**, i.e., convergence to the **global** (!) optimum is guaranteed given unlimited budget (Bull 2011)
- Cannot be proven for the PI or the vanilla LCB
- LCB can be proven to converge in a similar manner if the mean-variance trade-off parameter is chosen adaptively and "correctly" (Srinivas et al. 2010)
- In practice, both LCB and EI work quite well

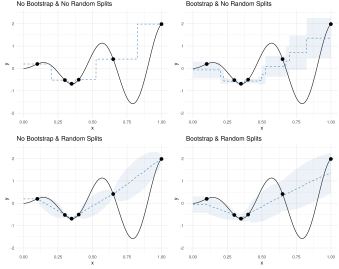


Other ACQFs:

- Entropy based: Entropy search, predictive entropy search, max value entropy search
- Knowledge Gradient
- Thompson Sampling
- ...

Optimization in Machine Learning

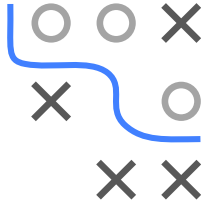
Bayesian Optimization
Important Surrogate Models



Learning goals

- Search space / input data peculiarities in black box problems
- Gaussian process
- Random forest

SURROGATE MODELS



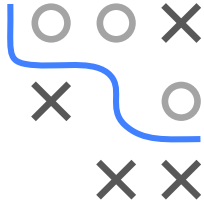
Desiderata:

- Regression model (there are also classification approaches)
- Non-linear local model
- Accurate predictions (especially for small sample sizes)
- Often: uncertainty estimates
- Robust, works often well without human modeler intervention

Depending on the application:

- Can handle different types of inputs (numerical and categorical)
- Can handle dependencies (i.e., hierarchical input)

GAUSSIAN PROCESS

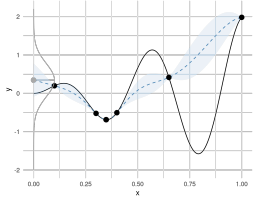


Posterior predictive distribution for test point $\mathbf{x} \in \mathcal{S}$:

$$Y(\mathbf{x}) \mid \mathbf{x}, \mathcal{D}^{[t]} \sim \mathcal{N}\left(\hat{f}(\mathbf{x}), \hat{s}^2(\mathbf{x})\right)$$

with

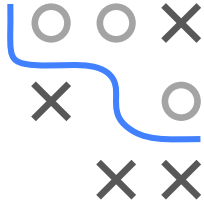
$$\begin{aligned} \hat{f}(\mathbf{x}) &= k(\mathbf{x})^\top \boldsymbol{K}^{-1} \boldsymbol{y} \\ \hat{s}^2(\mathbf{x}) &= k(\mathbf{x}, \mathbf{x}) - k(\mathbf{x})^\top \boldsymbol{K}^{-1} k(\mathbf{x}) \end{aligned}$$



Kernel method, based on kernel / Gram matrix $\boldsymbol{K} := \left(k(\mathbf{x}^{[i]}, \mathbf{x}^{[j]})\right)_{i,j}$

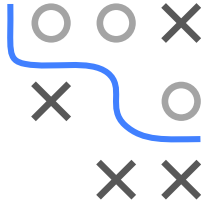
GAUSSIAN PROCESS

Example kernel functions:



- Radial basis function kernel (also known as Gauss kernel):
  $k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{d(\mathbf{x},\mathbf{x}')^2}{2l^2}\right)$
  - l length scale; $d(\cdot,\cdot)$ Euclidean distance
  - infinitely differentiable - very “smooth”
- Matérn kernels:
  $k(\mathbf{x}, \mathbf{x}') = \frac{1}{\Gamma(\nu)2^{\nu-1}}\left(\frac{\sqrt{2\nu}}{l}d(\mathbf{x},\mathbf{x}')\right)^{\nu} K_\nu\left(\frac{\sqrt{2\nu}}{l}d(\mathbf{x},\mathbf{x}')\right)$
  - l length scale; $d(\cdot,\cdot)$ Euclidean distance; $K_\nu(\cdot)$ modified Bessel function; $\Gamma(\cdot)$ Gamma function
  - for $\nu = 3/2$ once differentiable, for $\nu = 5/2$ twice differentiable
  - Popular choice as a kernel function when using a GP as SM

GAUSSIAN PROCESS



Pros:

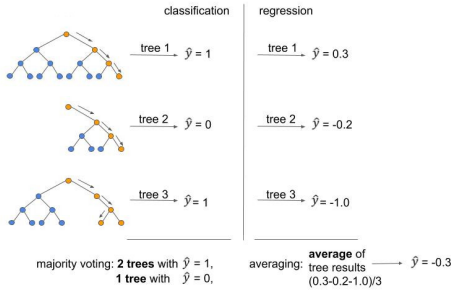
- Smooth, local, powerful estimator, also for small data
- GPs yield well-calibrated uncertainty estimates
- The posterior predictive distribution under a GP is normal

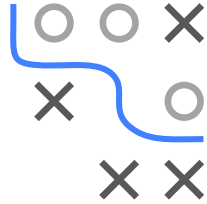
Cons:

- Vanilla GPs scale cubic in the number of data points
- Can natively only handle numeric features
  Mixed inputs / dependencies require special kernels
- GPs aren't that robust; numerical problems can occur
- Can be sensitive to the choice of kernel and hyperparameters

RANDOM FOREST

- Bagging ensemble
- Fit B decision trees on bootstrap samples
- Feature subsampling

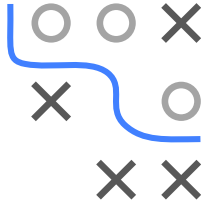




“extratrees” / random splits:

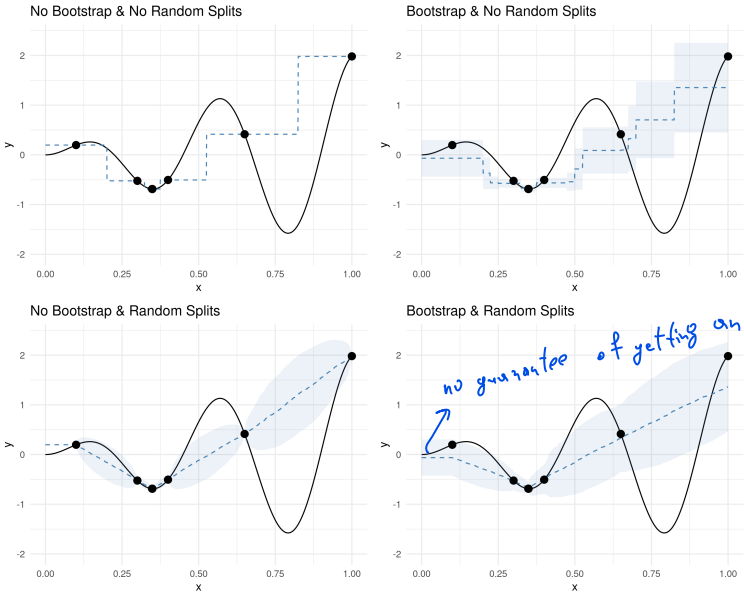
- Choose split location uniformly at random
- Results in a “smoother” mean prediction

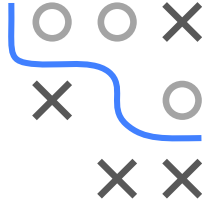
RANDOM FOREST - MEAN AND VARIANCE



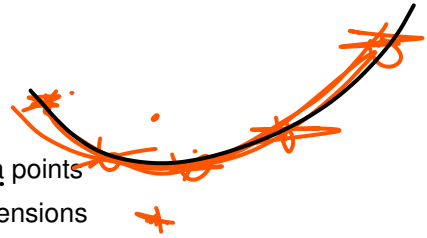
- Let $\hat{f}_b : \mathcal{S} \to \mathbb{R}$ be the mean prediction of a decision tree b (mean of all data points in the same node as observation $\mathbf{x} \in \mathcal{S}$)
- Let $\hat{s}_b^2 : \mathcal{S} \to \mathbb{R}$ be the variance prediction (variance of all data points in the same node as observation $\mathbf{x} \in \mathcal{S}$)
- Mean prediction of forest: $\hat{f} : \mathcal{S} \to \mathbb{R}$, $\mathbf{x} \mapsto \frac{1}{B} \sum_{b=1}^{B} \hat{f}_b(\mathbf{x})$
- Variance prediction of forest: $\hat{s}^2 : \mathcal{S} \to \mathbb{R}$, $\mathbf{x} \mapsto \left(\frac{1}{B} \sum_{b=1}^{B} \hat{s}_b^2(\mathbf{x}) + \hat{f}_b(\mathbf{x})^2 \right) - \hat{f}(\mathbf{x})^2$
  (law of total variance assuming a mixture of B models)
- Alternative variance estimator:
  - (infinitesimal) Jackknife
- Variance prediction derived from randomness of individual trees
  - Bagging / boostrap samples
  - Features sampled at random
  - (randomized split locations in the case of “extratrees”)

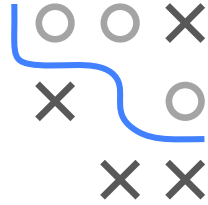
RANDOM FOREST - DIFFERENT CHOICES





RANDOM FOREST



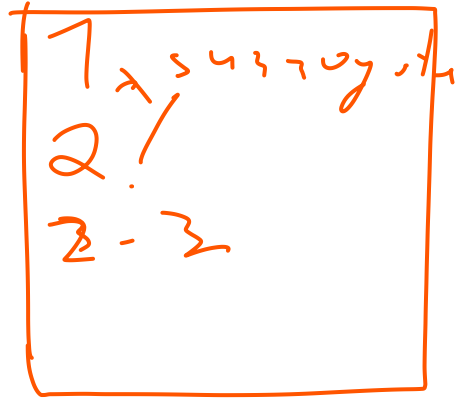


Pros:

- Cheap(er) to train
- Scales well with the number of data points
- Scales well with the number of dimensions
- Can easily handle hierarchical mixed spaces. Either via imputation or directly respecting dependencies in the tree structure
- Robust

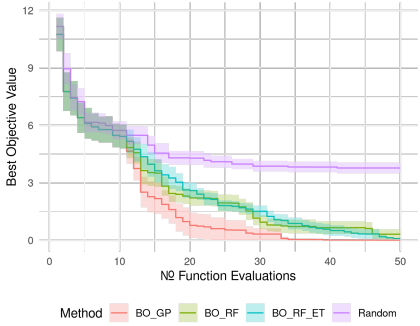
Cons:

- Suboptimal uncertainty estimates
- Not really Bayesian (no real posterior predictive distribution)
- Poor extrapolation



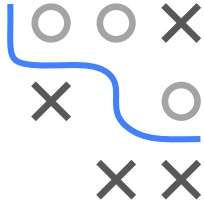
EXAMPLE

Minimize the 2D Ackley Function using BO_GP (GP with Matérn 3/2, EI), BO_RF (standard Random Forest, EI), BO_RF_ET (Random Forest with extratrees, EI) or a random search:



Python package

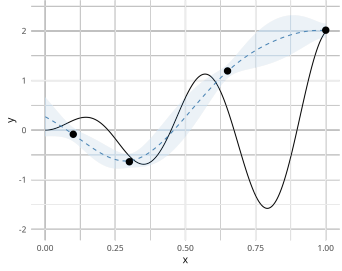
gpytorch



Strong BO_GP performance. BO_RF and BO_RF_ET not too bad either. BO_RF_ET maybe slightly better final performance than BO_RF.

Optimization in Machine Learning

Bayesian Optimization
Noisy Bayesian Optimization



Learning goals

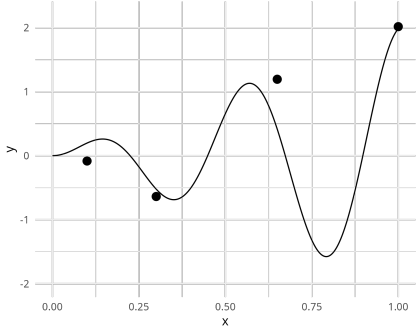
- Noisy surrogate modeling
- Noisy acquisition functions
- Final best point

NOISY EVALUATIONS

In many real-life applications, we cannot access the true function values $f(\mathbf{x})$ but only a **noisy** version thereof

$$f(\mathbf{x}) + \epsilon(\mathbf{x})$$

For the sake of simplicity, we assume $\epsilon(\mathbf{x}) \sim \mathcal{N}\left(0, \sigma_\epsilon^2\right)$ for now

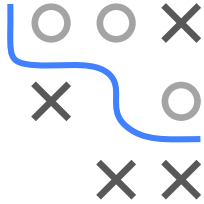


NOISY EVALUATIONS

In many real-life applications, we cannot access the true function values $f(\mathbf{x})$ but only a **noisy** version thereof

$$f(\mathbf{x}) + \epsilon(\mathbf{x})$$

For the sake of simplicity, we assume $\epsilon(\mathbf{x}) \sim \mathcal{N}\left(0, \sigma_\epsilon^2\right)$ for now



Examples:

- HPO (due to non-deterministic learning algorithm and/or resampling technique)
- Oil drilling optimization (an oil sample is only an estimate)
- Robot gait optimization (velocity of a run of a robot is an estimate of true velocity)

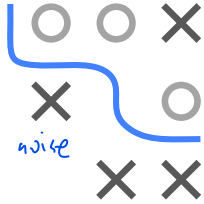
(but we fix seed, right?)
maybe this does it again
eval from same x produces dif. resu lts

fixing seed does not matter

NOISY EVALUATIONS

22.01.25

This raises the following problems:



- **Surrogate modeling:** So far we used an interpolating GP that is based on noise-free observations; as a consequence, the variance is modeled as 0

  Don't model through the noise

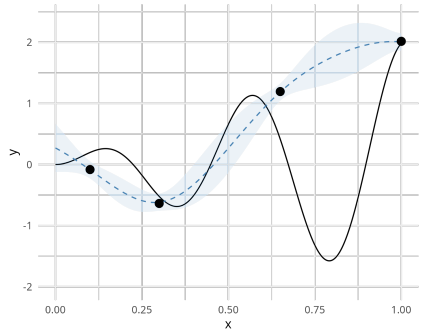
  $$s^2(\mathbf{x}^{[i]}) = 0$$

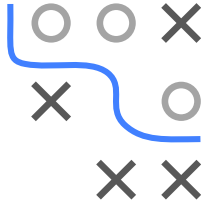
  for design points $(\mathbf{x}^{[i]}, y^{[i]}) \in \mathcal{D}^{[t]}$. This is problematic.
- **Acquisition functions:** Most acquisition functions are based on the best observed value $f_{\min}$ so far. If evaluations are noisy, we do not know this value (it is a random variable).
- **Final best point:** The design point evaluated best is not necessarily the true best point in design (overestimation).

SURROGATE MODEL

In case of noisy evaluations, a nugget-effect GP (GP regression) should be used instead of an interpolating GP.

The posterior predictive distribution for a new test point $\mathbf{x} \in \mathcal{S}$ under a GP assuming homoscedastic noise (σ^2_ϵ) is:





$$Y(\mathbf{x}) \mid \mathbf{x}, \mathcal{D}^{[t]} \sim \mathcal{N}\left(\hat{f}(\mathbf{x}), \hat{s}^2(\mathbf{x})\right)$$

with

"nugget effect" that's added

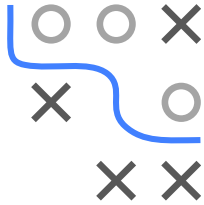
can be estimate by MLE

$$\begin{aligned} \hat{f}(\mathbf{x}) &= k(\mathbf{x})^\top (\boldsymbol{K} + \sigma^2_\epsilon \mathbf{I}_t)^{-1} \boldsymbol{y} \\ \hat{s}^2(\mathbf{x}) &= k(\mathbf{x}, \mathbf{x}) - k(\mathbf{x})^\top (\boldsymbol{K} + \sigma^2_\epsilon \mathbf{I}_t)^{-1} k(\mathbf{x}) \end{aligned}$$

NOISY ACQUISITION FUNCTIONS: AEI

Augmented Expected Improvement (Huang et al. 2006)

$$a_{\mathrm{AEI}}(\mathbf{x}) = a_{\mathrm{EI}_{f_{\min *}}}(\mathbf{x}) \left(1 - \frac{\sigma_\epsilon}{\sqrt{\hat{s}^2(\mathbf{x}) + \sigma_\epsilon^2}}\right).$$



Here, $a_{\mathrm{EI}_{f_{\min *}}}$ denotes the **Expected Improvement with Plugin**.
It uses the **effective best solution** as a plugin for the (unknown) best observed value $f_{\min}$

$$f_{\min *} = \min_{\mathbf{x} \in \{\mathbf{x}^{[1]}, \ldots, \mathbf{x}^{[t]}\}} \hat{f}(\mathbf{x}) + c\hat{s}(\mathbf{x}),$$

without this a point can have low value just by chance because of noise

upper bound of predicted value

conservative estimate

where $c > 0$ is a constant that controls the risk aversion.

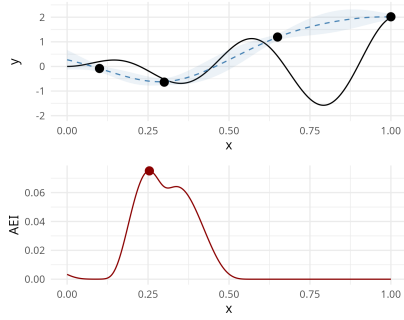
σ_ϵ^2 is the nugget-effect as estimated by the GP regression.

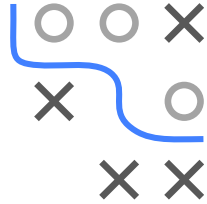
NOISY ACQUISITION FUNCTIONS: AEI

In addition, it takes into account the nugget-effect σ_ϵ^2 by a penalty term:

$$\left(1 - \frac{\sigma_\epsilon}{\sqrt{\hat{s}^2(\mathbf{x}) + \sigma_\epsilon^2}}\right)$$

The penalty is justified to "account for the diminishing return of additional replicates as the predictions become more accurate" (Huang et al. 2006)



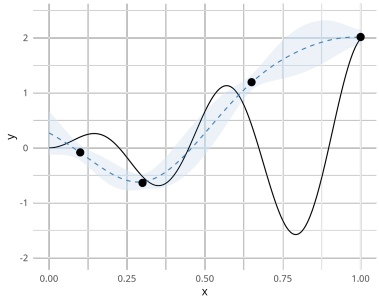


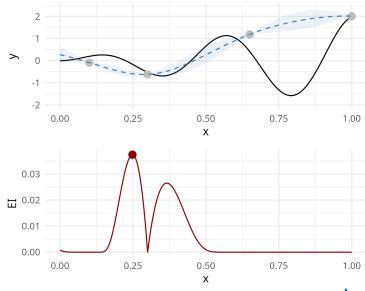
- Designs with small predictive variance $\hat{s}^2(\mathbf{x})$ are penalized in favor of more exploration.
- If $\sigma_\epsilon^2 = 0$ (noise-free), the AEI corresponds to the EI with plugin.

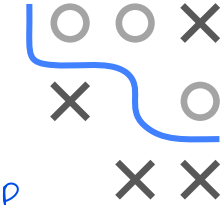
REINTERPOLATION

Clean noise from the model and then apply a general acquisition function (EI, PI, LCB, ...)

The RP suggests to build **two models**: a nugget-effect GP (regression model; left) and then, on the predictions from the first model (grey), an interpolating GP (right)



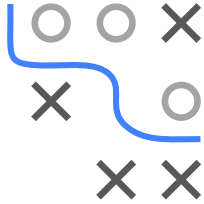




fit reg GP, predict, and use that (smoothed data) kind of like smoothing the noise and doing regular BO

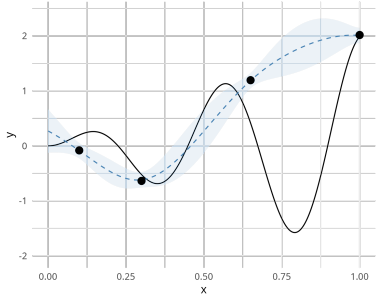
Can be very time consuming to speed up one may copy kernel params from 1st GP to 2nd

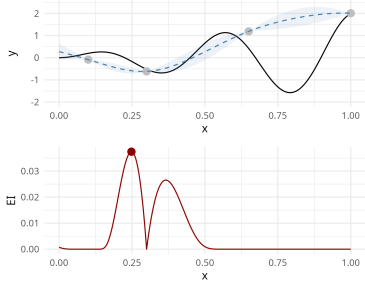
REINTERPOLATION



Algorithm Reinterpolation Procedure

1: Build a nugget-effect GP model based on noisy evaluations
2: Compute predictions for all points in the design $\hat{f}(\mathbf{x}^{[1]}), \ldots, \hat{f}(\mathbf{x}^{[t]})$
3: Train an interpolating GP on $\left\{ \left(\mathbf{x}^{[1]}, \hat{f}(\mathbf{x}^{[1]}) \right), \ldots, \left(\mathbf{x}^{[t]}, \hat{f}(\mathbf{x}^{[t]}) \right) \right\}$
4: Based on the interpolating model, obtain a new candidate using a noise-free acquisition function

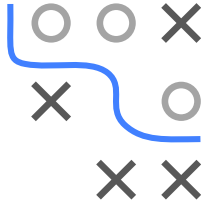


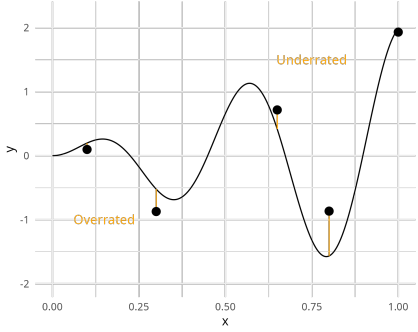


IDENTIFICATION OF FINAL BEST POINT

Another problem is the identification of a final best point:

- Assume that all evaluations are noisy
- The probability is high that **by chance**
  - bad points get overrated
  - good points get overlooked

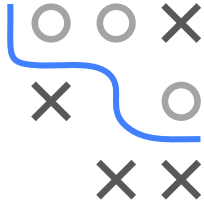




IDENTIFICATION OF FINAL BEST POINT

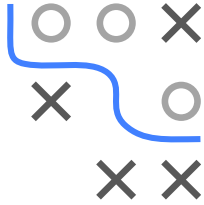
Possibilities to reduce the risk of falsely returning a bad point:

- Return the best predicted point: $\arg\min_{\mathbf{x} \in \{\mathbf{x}^{[1]}, \ldots, \mathbf{x}^{[t]}\}} \hat{f}(\mathbf{x})$
- Repeated evaluations of the final point: infer guarantees about final point (however if final point is “bad” unclear how to find a better one)
- Repeated evaluations of all design points: reduce noise during optimization and risk of falsely returning a bad point
- More advanced replication strategies, e.g. incumbent strategies: also re-evaluate the “incumbent” in each iteration



Optimization in Machine Learning

Bayesian Optimization
Practical Aspects of BO

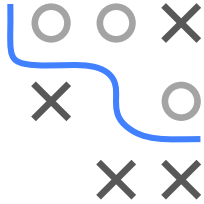


Learning goals

- Size of the initial design
- Optimizing the acquisition function
- When to terminate
- BO Components and robustness

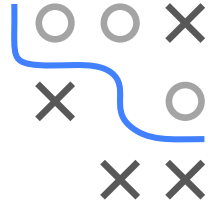
SIZE OF THE INITIAL DESIGN

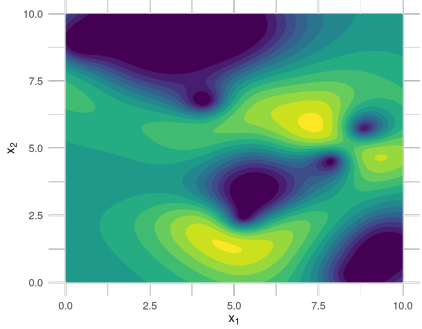
- Should not be too small → bad SM
- Should not be too large
- Scale with the dimensionality of the search space
- Certain SM may impose restrictions on the lower bound of the size of the initial design
- Rule of thumb $4d$ to $10d$ (d may be too many points to place randomly)



OPTIMIZING THE ACQUISITION FUNCTION

- Optimizing the acquisition function to find the next candidate point is comparably cheap
- Still can be a hard optimization problem: non-linear, multimodal
- Properly optimizing the acquisition function can be crucial for performance





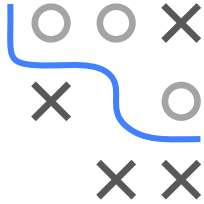
Example EI landscape for a 2D problem.

more iterations = more modes

we may have Gradient for some acquisition fun and kernels

Optim can be done by Cool idea
evolutionary + some gradient based
1) start with X points
2) do some iter of GD with shg
3) select best points
4) you never pop.

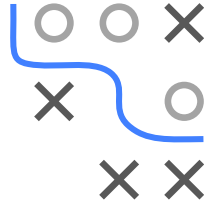
OPTIMIZING THE ACQUISITION FUNCTION



- Optimizing the acquisition function to find the next candidate point is comparably cheap
- Still can be a hard optimization problem: non-linear, multimodal
- Properly optimizing the acquisition function can be crucial for performance
- Choice of optimizer depends on the search space
  - Numeric: L-BFGS-B with restarts, DIRECT, ...
  - Mixed: EAs, local search with restarts, ...
  - A random search can always be used but may not find a good solution (even if the budget is large)
- Sometimes, gradient-based optimizers can be used (e.g. when using a GP as SM)

WHEN TO TERMINATE

- After a certain number of evaluations
  - Potentially scaling the budget with the dimensionality of the search space
- After a certain runtime
- Specify a target threshold (of the objective or acquisition function)
- Based on stagnation (of the objective or acquisition function)



how?

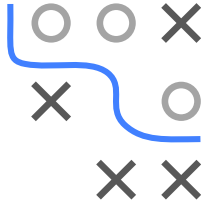
can be bad because we usually have

no improv, no, no, jump, no, n, .. jump and not

We can also check EI over whole region, but this means we're trusting GP, which may have not got seen many points.

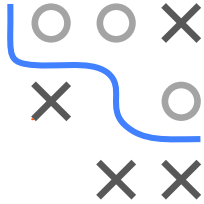
ROBUSTNESS

A good BO implementation should be **robust**:



- Handle crashing SM
  - Especially a GP can result in errors during training (Kernel matrix not invertible, training points too close to each other)
  - Use a fallback SM (e.g., Random Forest) or catch errors and propose a new candidate uniformly at random
- Automatically detect input types (numerical, categorical; hierarchical) and choose an appropriate SM
- Have sensible defaults that work well for most scenarios

CHOOSING THE RIGHT COMPONENTS



- BO is modular: SM, acquisition function, acquisition function optimizer
- Different choices induce different overhead, e.g., GP vs. RF, entropy based acquisition functions vs. cheap to compute ones like EI, thorough acquisition function optimization with a large budget can be expensive
- Choosing the right components usually depends on the concrete application at hand and how expensive the evaluation of the black box itself is