CHEN 5802: Guest lecture on Active Learning

Design of Materials on Computers

DMC LAB

Bartel Research Group

bartel.cems.umn.edu

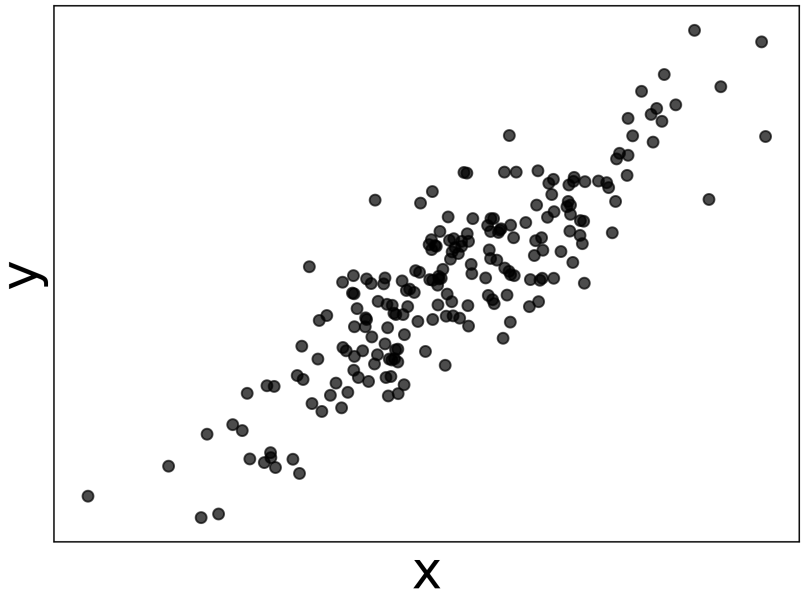
Chemical Engineering & Materials Science

University of Minnesota

Nathan J. Szymanski

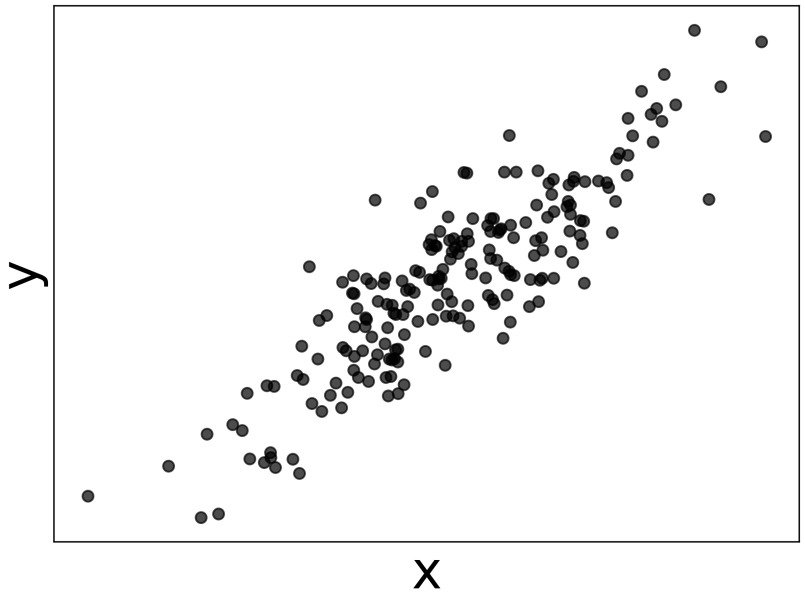
What you've learned so far in this course

Existing data (x_i, y_i)

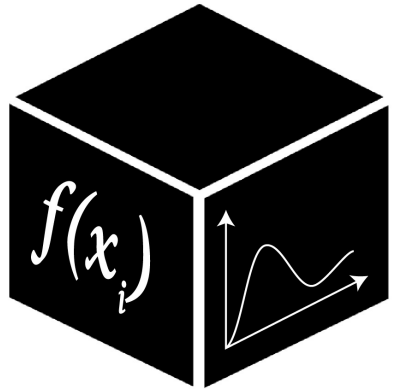


What you've learned so far in this course

Existing data (x_i, y_i)

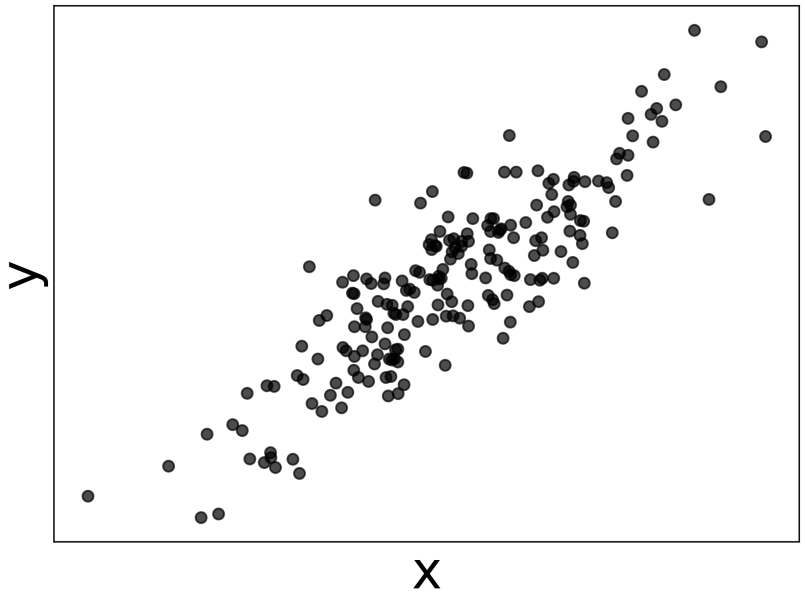


Used to train ML models

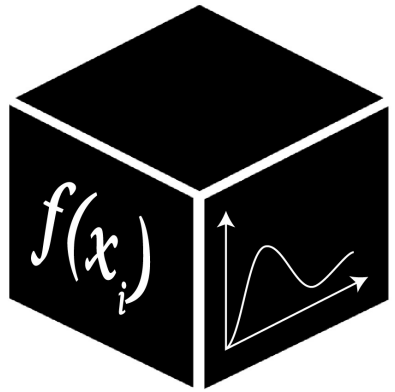


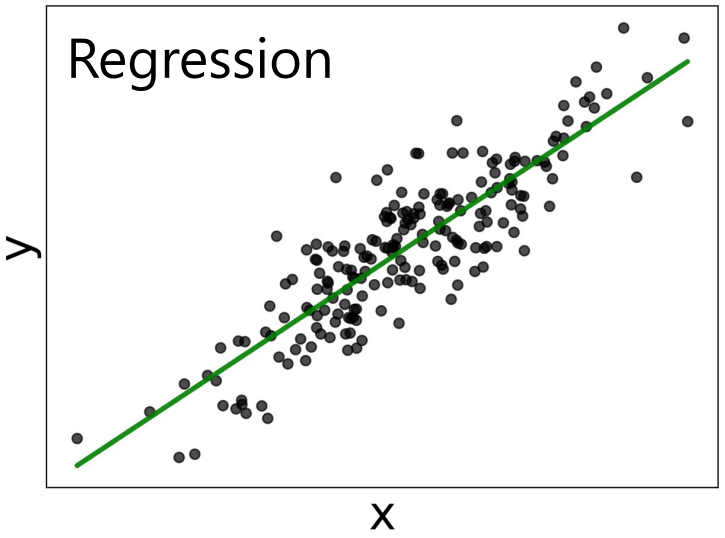
What you've learned so far in this course

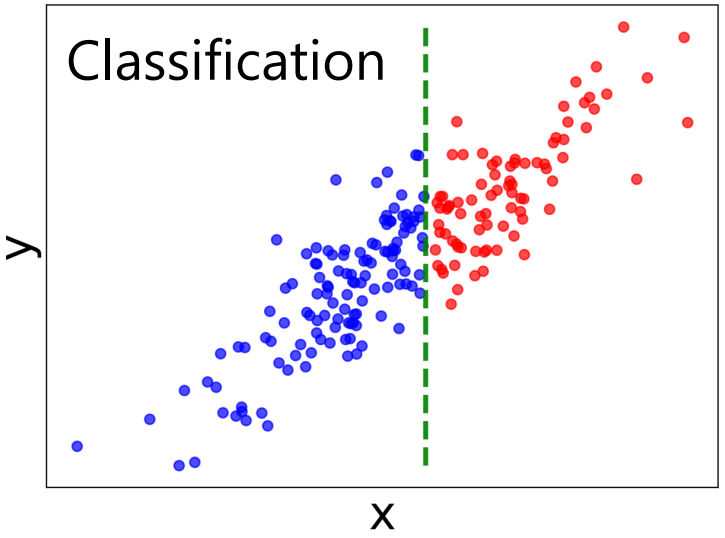
Existing data (x_i, y_i)



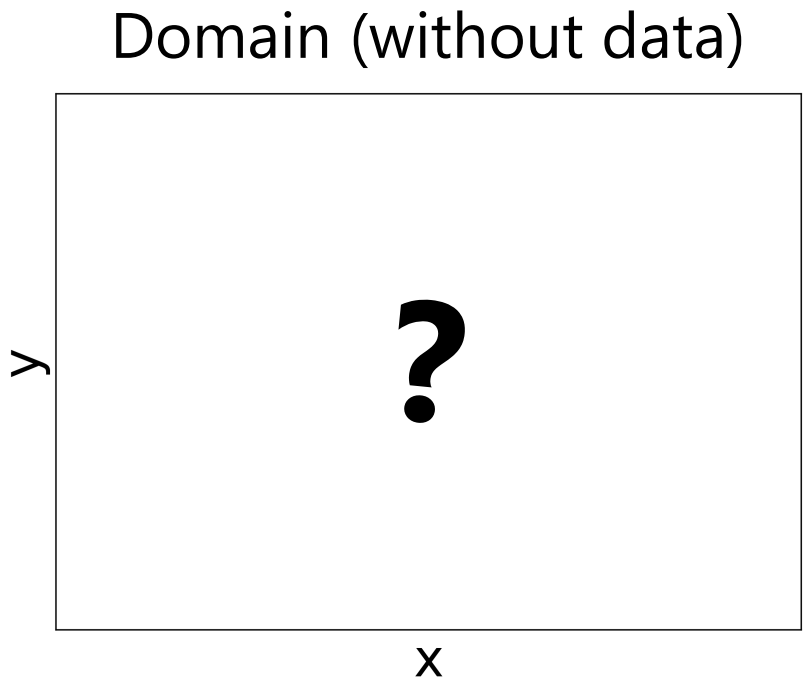
Used to train ML models







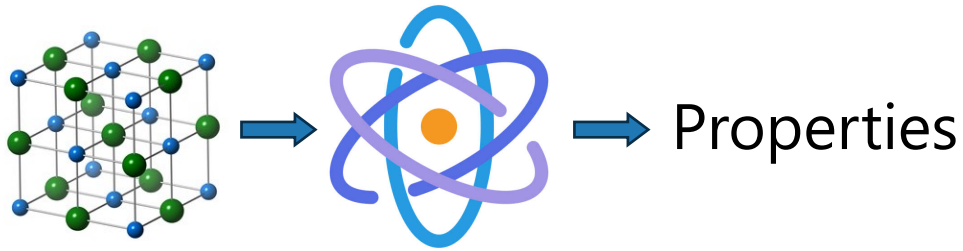
But often, training data is missing…we need to collect it!



- We have some domain over which new data (x_i, y_i) can be collected
- In practice, data collection tends to be **costly** and **time consuming**

An example: ML trained on DFT calculations

Density Functional Theory



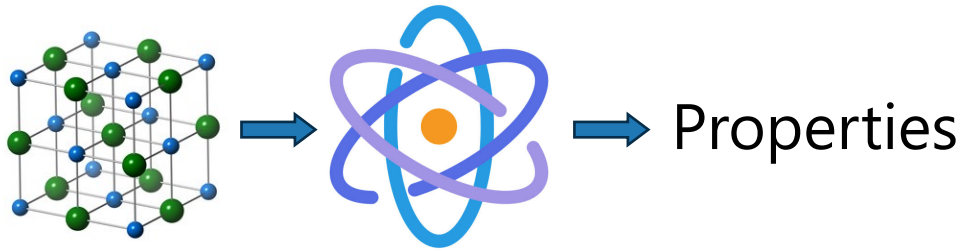
Properties

Required time: **min to hours**

An example: ML trained on DFT calculations

CEMS
Chemical Engineering
& Materials Science

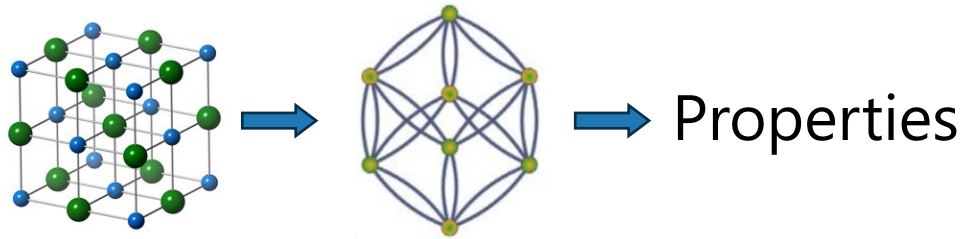
Density Functional Theory



Properties

Required time: **min to hours**

Machine learning potentials



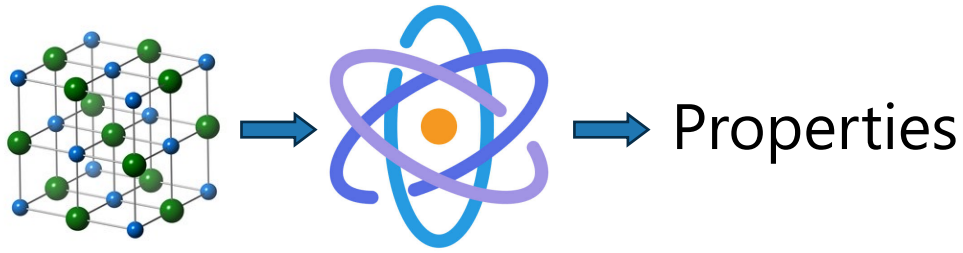
Properties

Required time: **seconds**

An example: ML trained on DFT calculations

CEMS
Chemical Engineering
& Materials Science

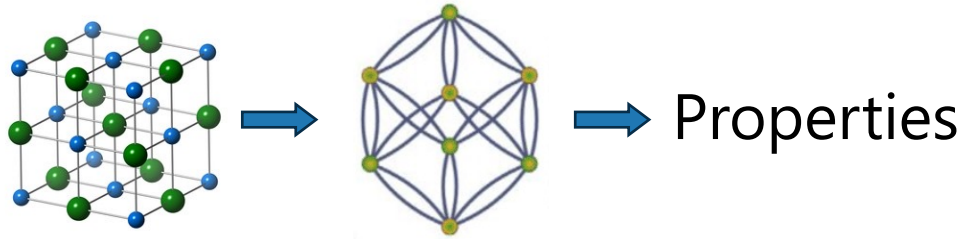
Density Functional Theory



Properties

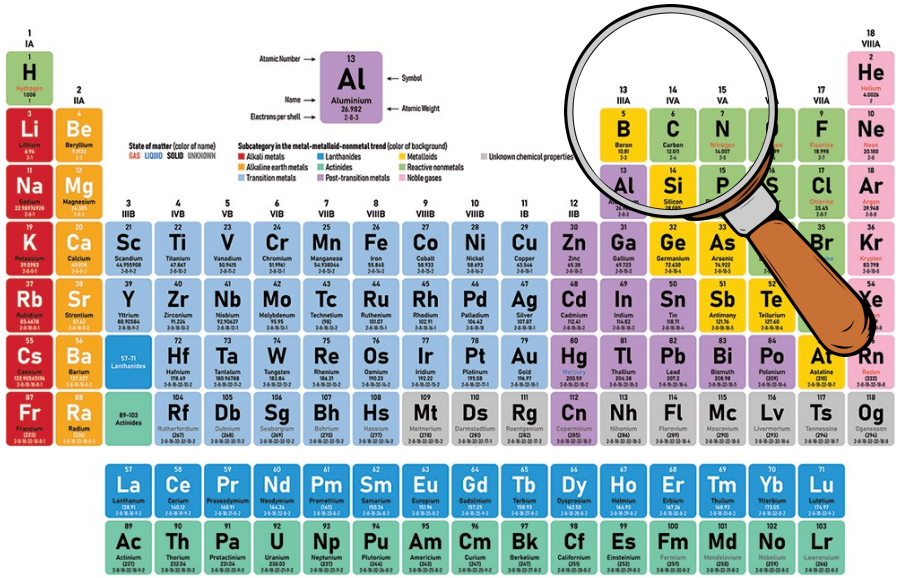
Required time: **min to hours**

Machine learning potentials



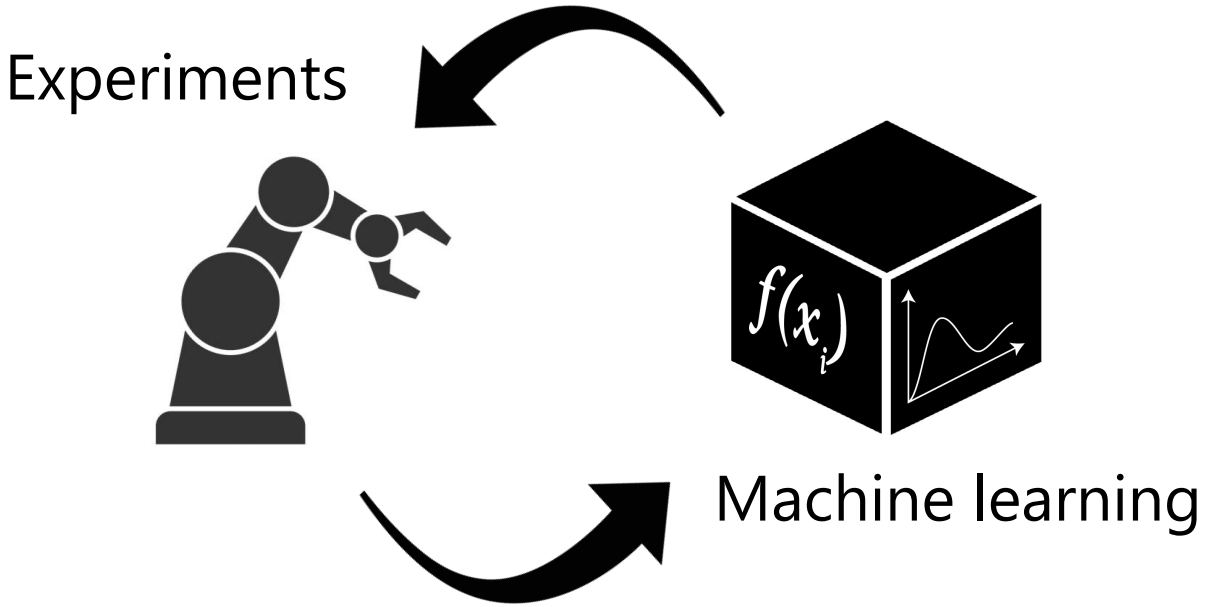
Properties

Required time: **seconds**



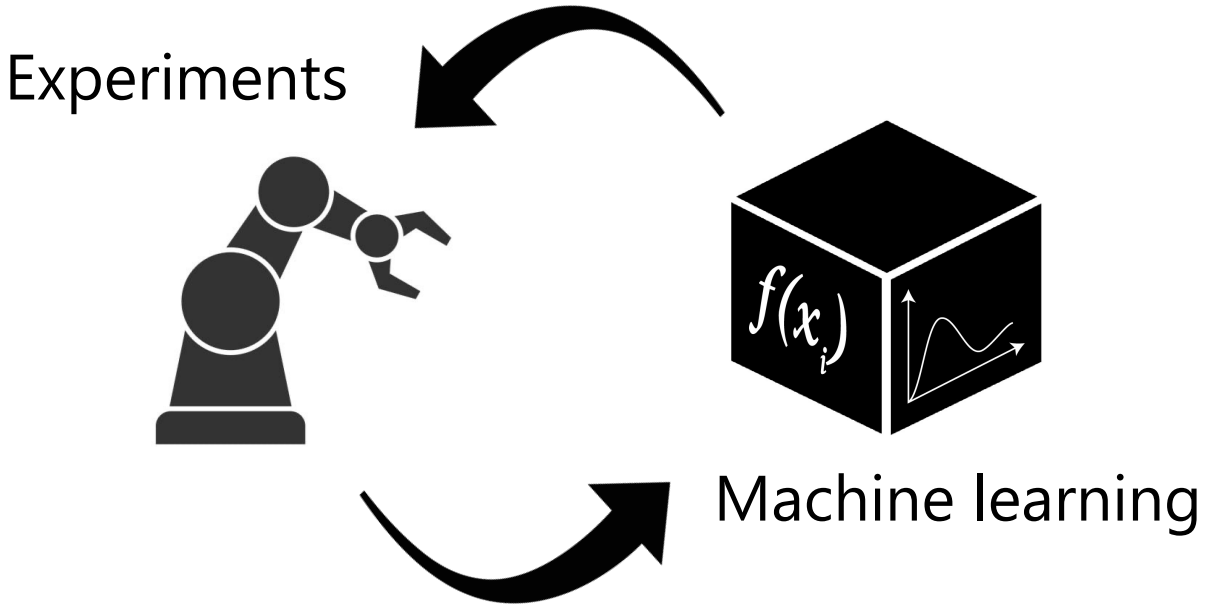
- We can train on existing DFT calculations, but these tend be biased in their sampling of certain chemistries…
- **New DFT calculations** can be run to supplement the training data

An example: autonomous (AI-driven) experiments



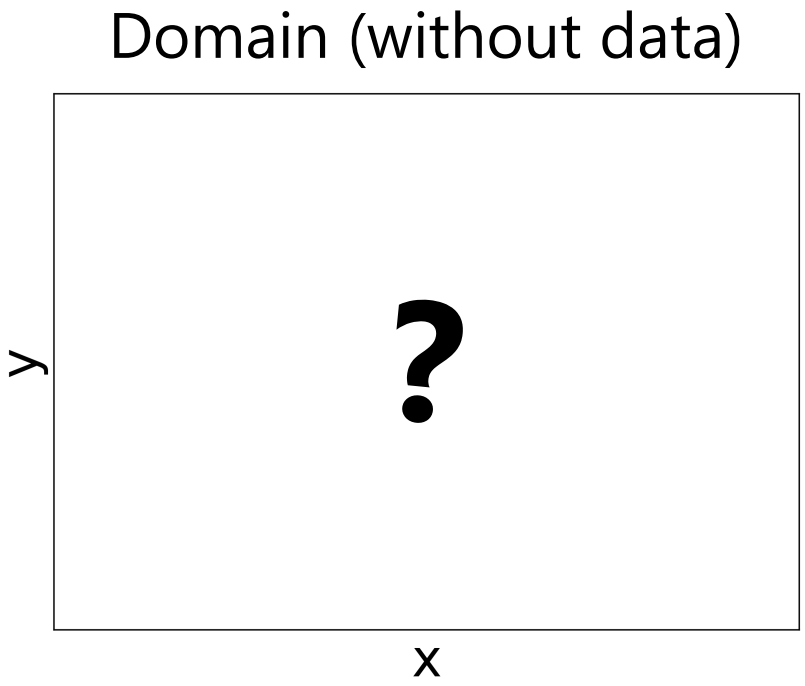
- The A-Lab tries to make new materials whose recipes are unknown

An example: autonomous (AI-driven) experiments



- The A-Lab tries to make new materials whose recipes are unknown
- But *thousands* of synthesis recipes are often possible, and each recipe requires **hours to days** of experiments → we cannot test them all!

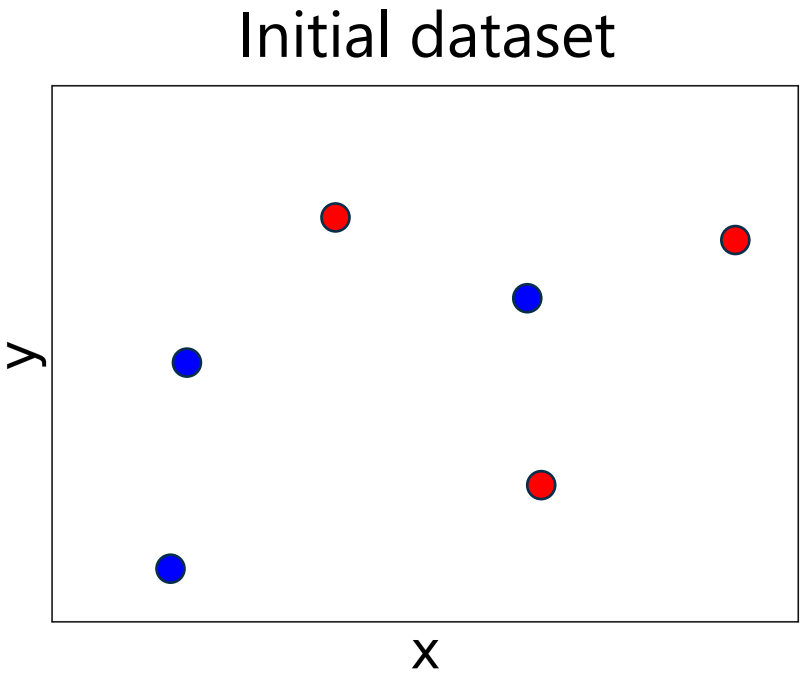
The key question: *which* data should we collect?



- We have some domain over which new data (x_i, y_i) can be collected
- In practice, data collection tends to be **costly** and **time consuming**
- In **which parts of the domain** should we collect training data? The goal is to achieve **good model performance** with **limited data**

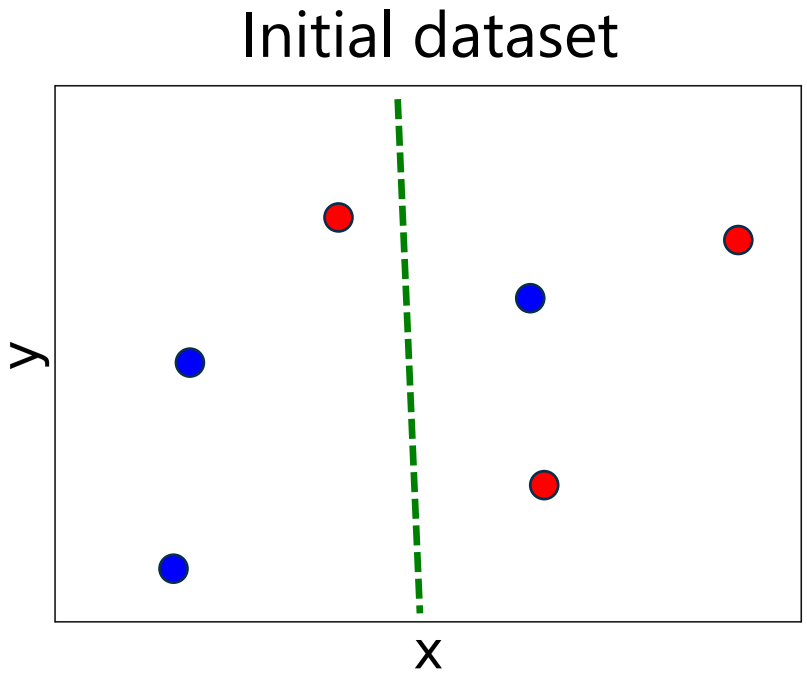
We can iterate between data collection with model training

Suppose we are collecting data to train a **classification** model in 2D (x-y) space



We can iterate between data collection with model training

Suppose we are collecting data to train a **classification** model in 2D (x-y) space

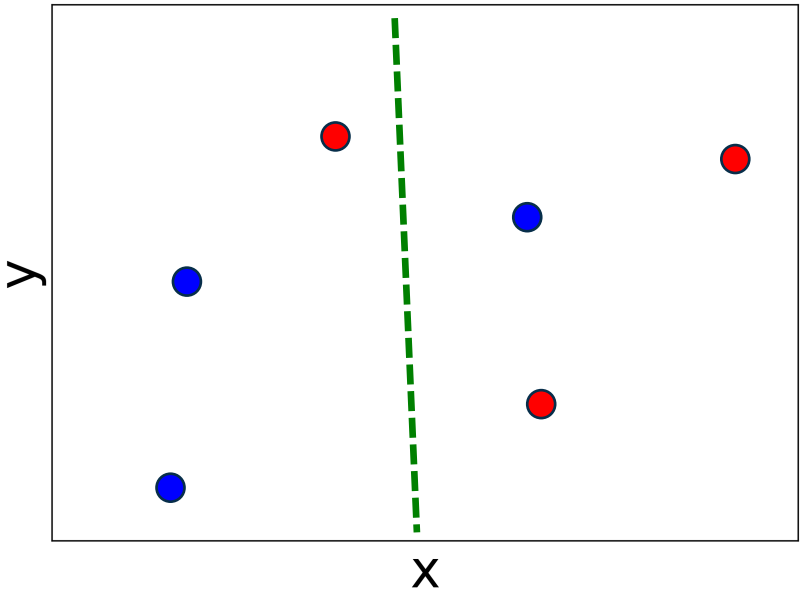


- Based on this initial training data, our model predicts a boundary to separate the two classes
- But there is likely a high amount of ***uncertainty*** in this prediction owing to limited data

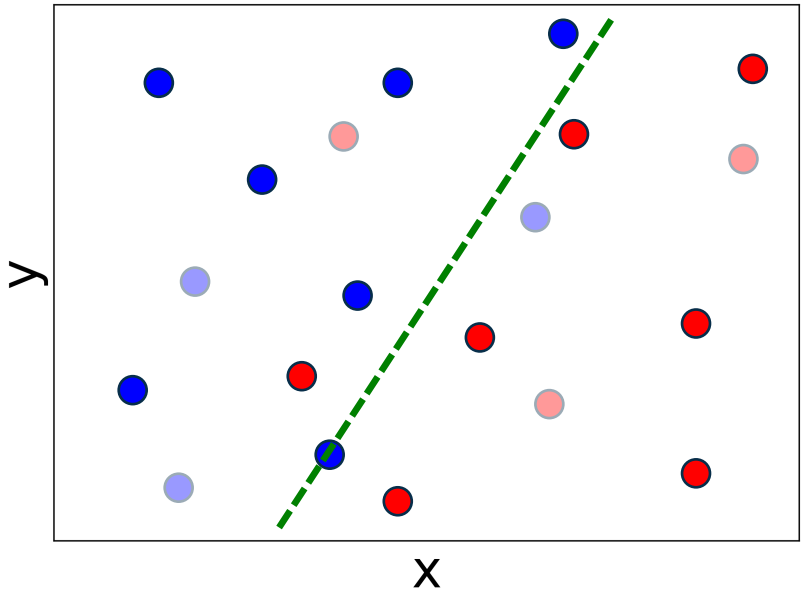
We can iterate between data collection with model training

Suppose we are collecting data to train a **classification** model in 2D (x-y) space

Initial dataset



2nd iteration

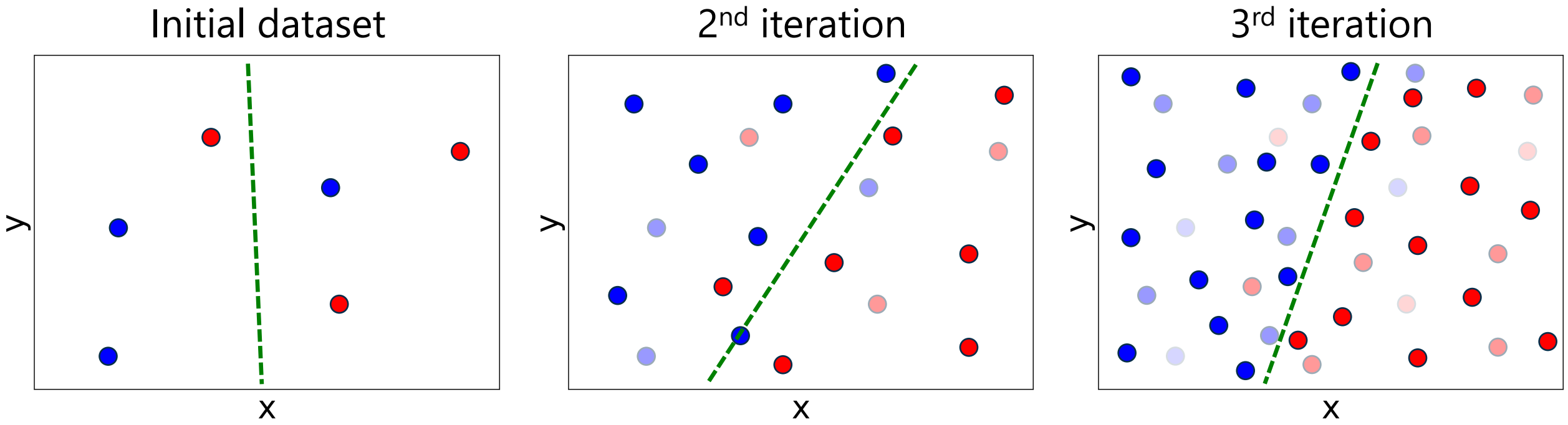


- With **more data**, the boundary changes
- This prediction should have **less uncertainty**

Class 1 Class 2

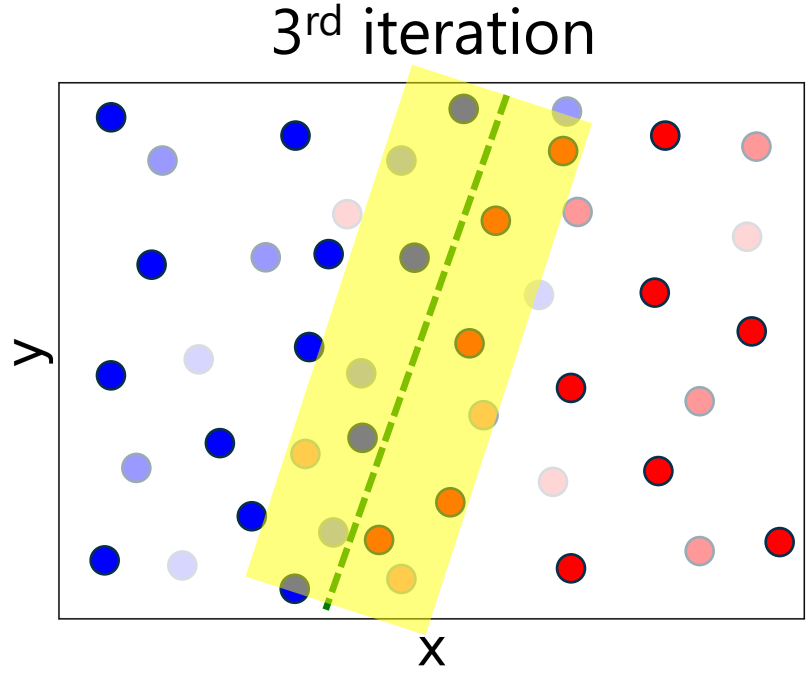
We can iterate between data collection with model training

Suppose we are collecting data to train a **classification** model in 2D (x-y) space



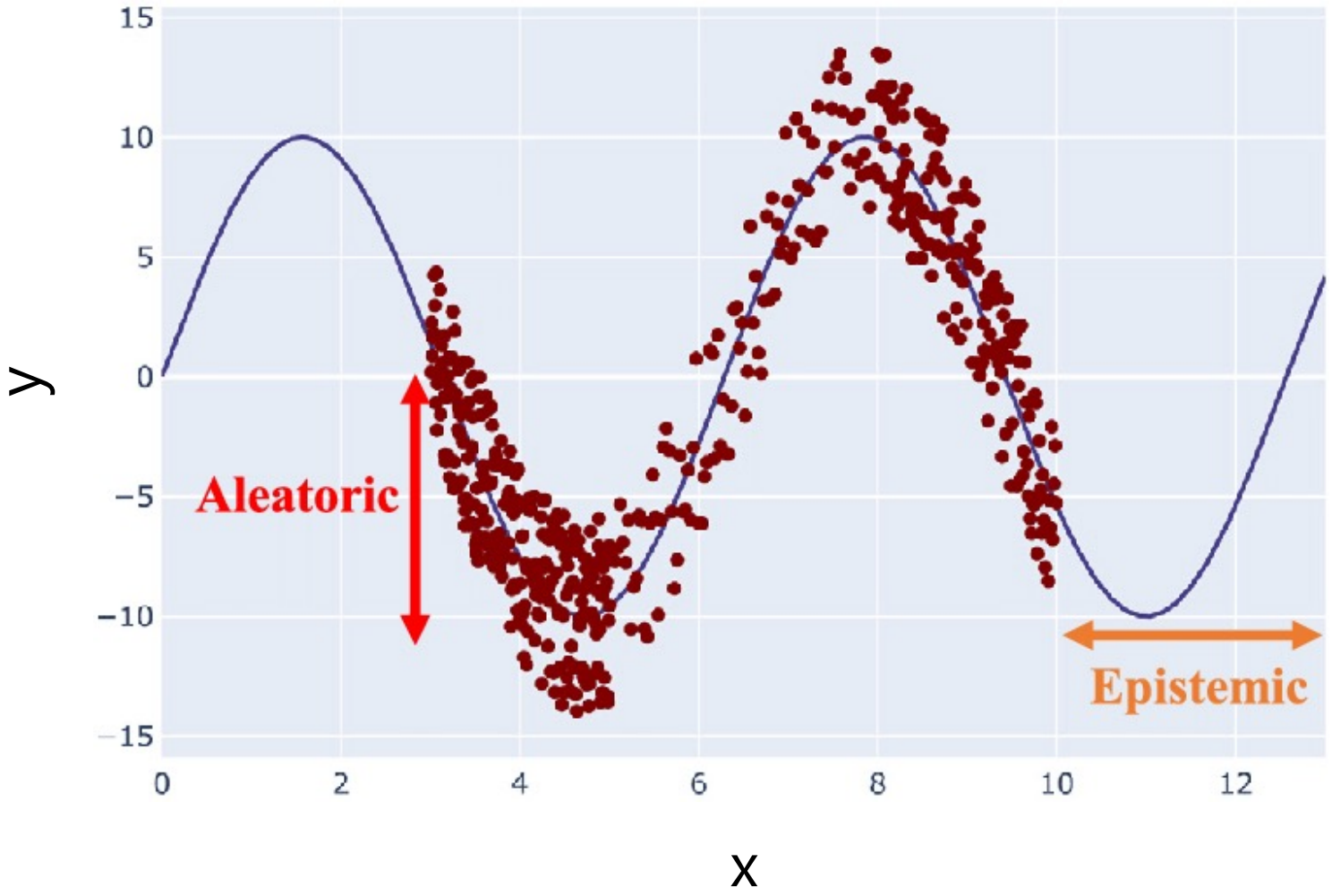
We can iterate between data collection with model training

- Ideally, we could focus our data collection on areas near the decision boundary
- For this, we need the ***model uncertainty*** as a **function of the domain space**



● Class 1 ● Class 2

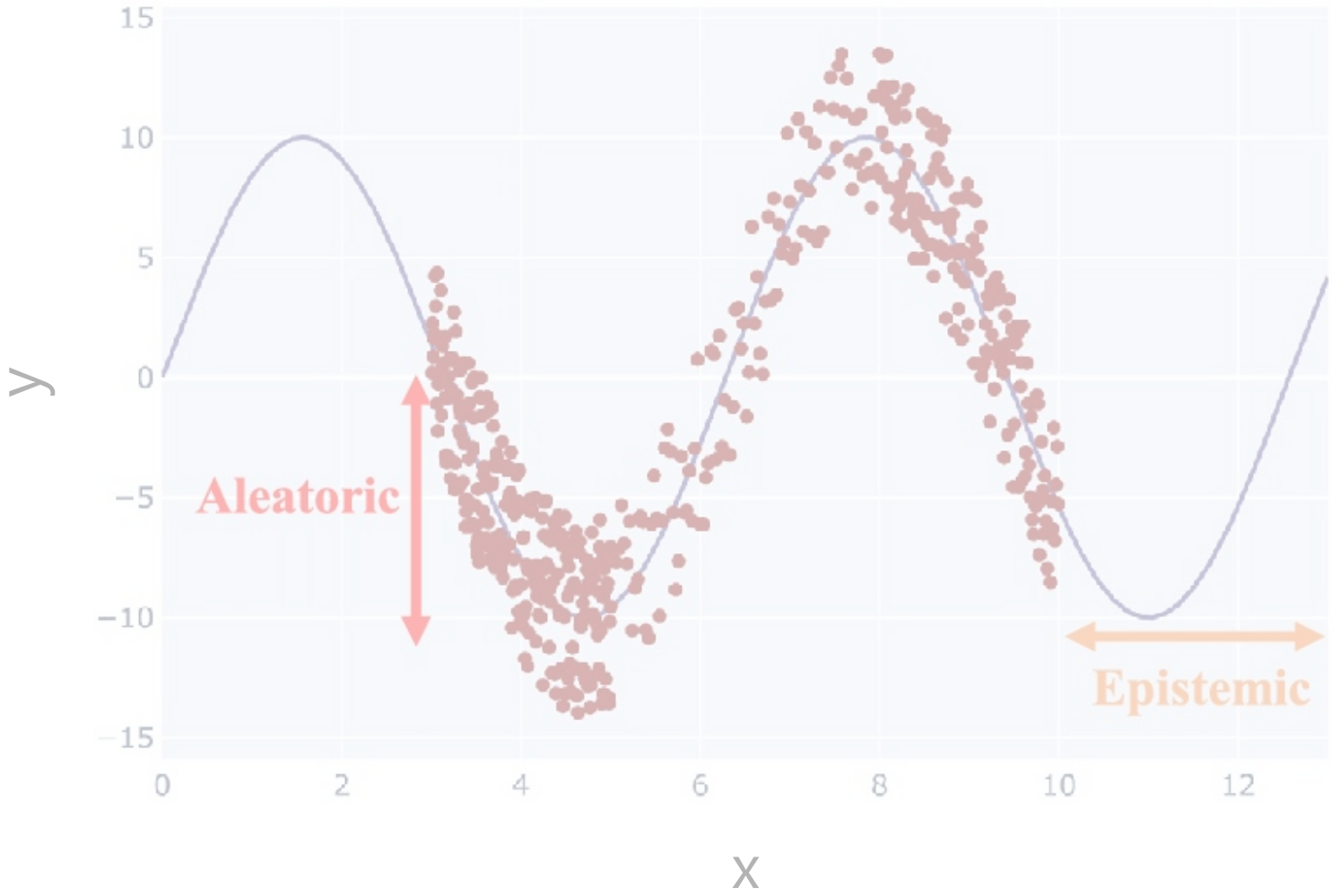
First, let's distinguish *epistemic* vs. *aleatoric* uncertainty



Aleatoric uncertainty:
Due to inherent randomness (noise) in the data

Epistemic uncertainty:
Due to a lack of knowledge, usually caused by the absence of training data

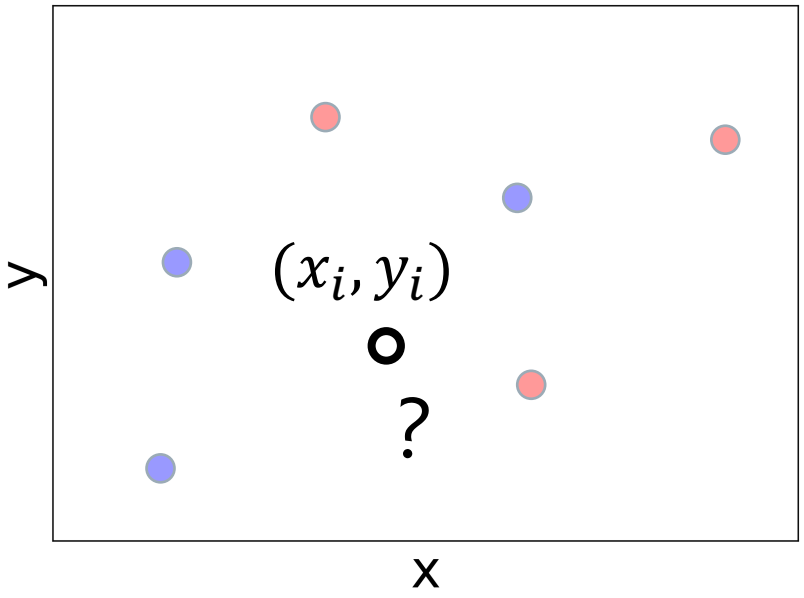
First, let's distinguish *epistemic* vs. *aleatoric* uncertainty

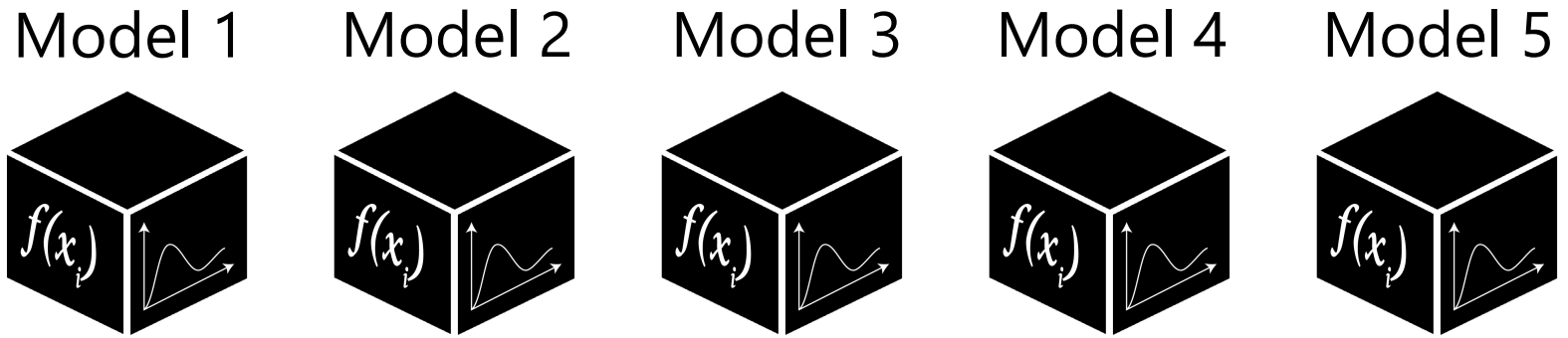


We'll focus on this type of uncertainty from here on out!

Epistemic uncertainty:
Due to a lack of knowledge, usually caused by the absence of training data

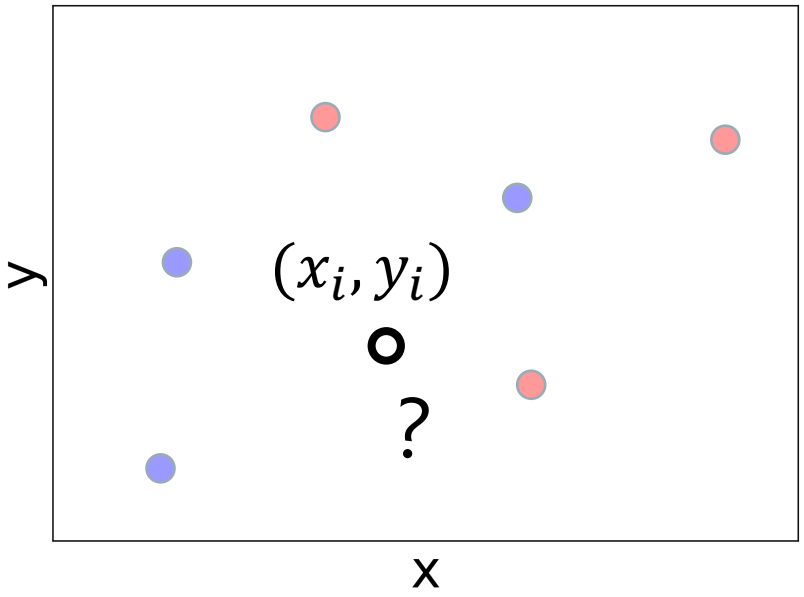
Method 1: ensemble methods for classification

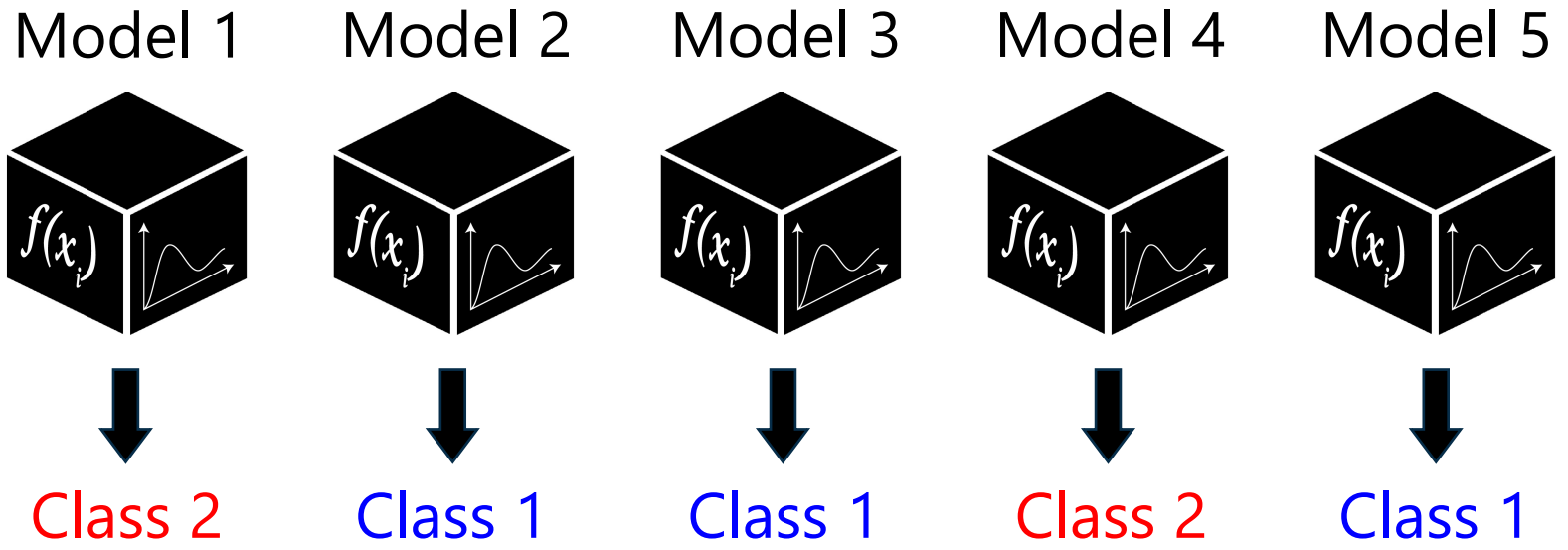




These models are each trained **independently**, either with **different initializations** or on **different parts of the training data**

Method 1: ensemble methods for classification

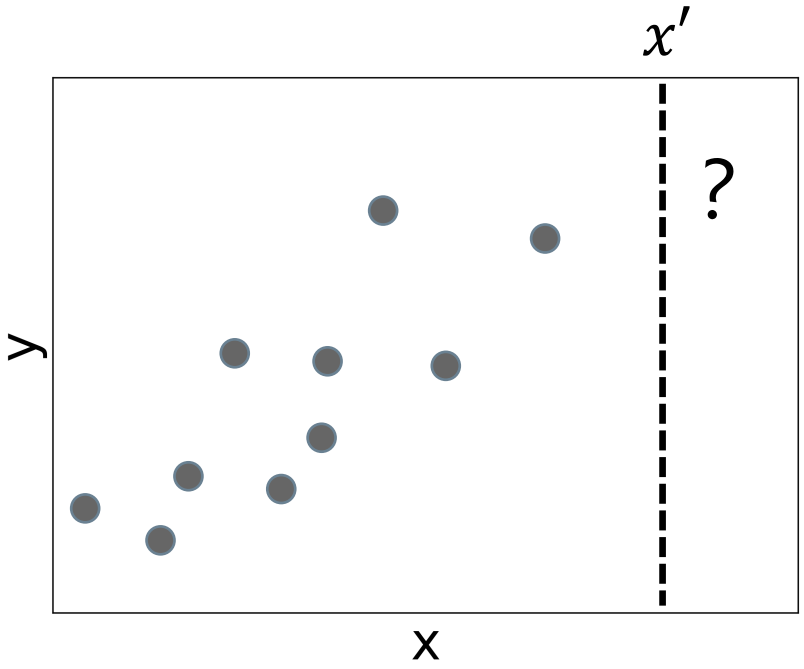


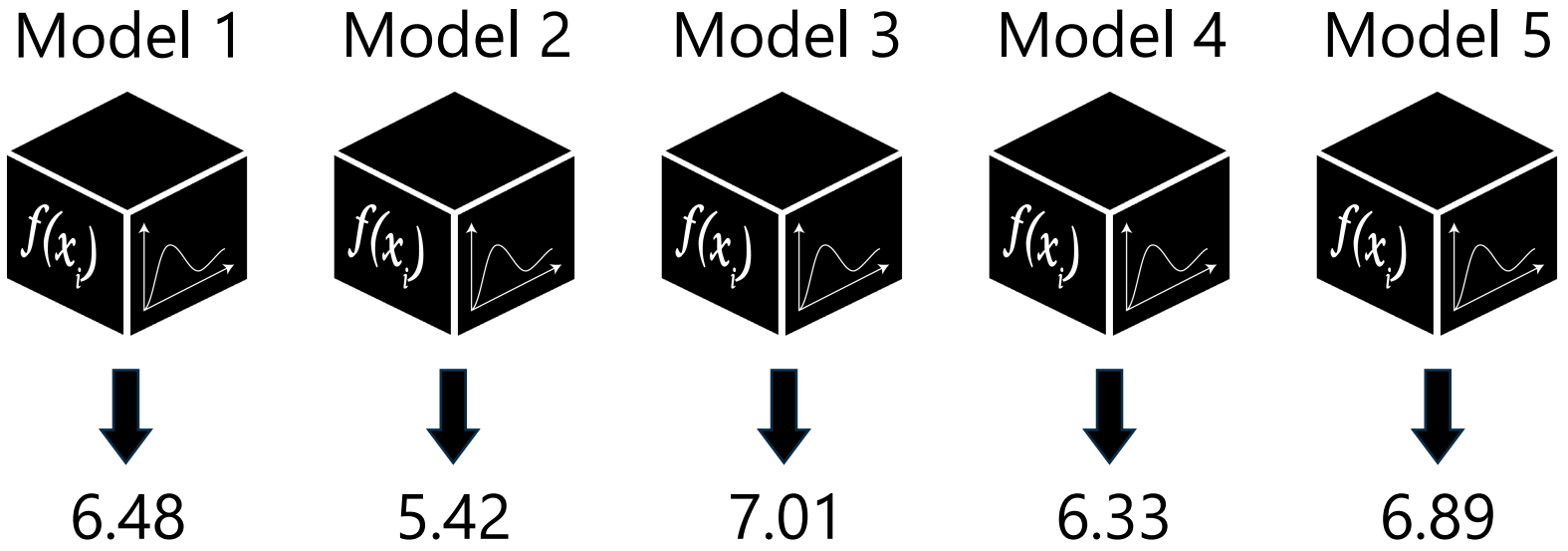


Class 1 is predicted most often (3/5 times) → **60% confidence**

Higher confidence ↔ Lower uncertainty

Method 1: ensemble methods for regression





We can use the **standard deviation** as a measure of uncertainty

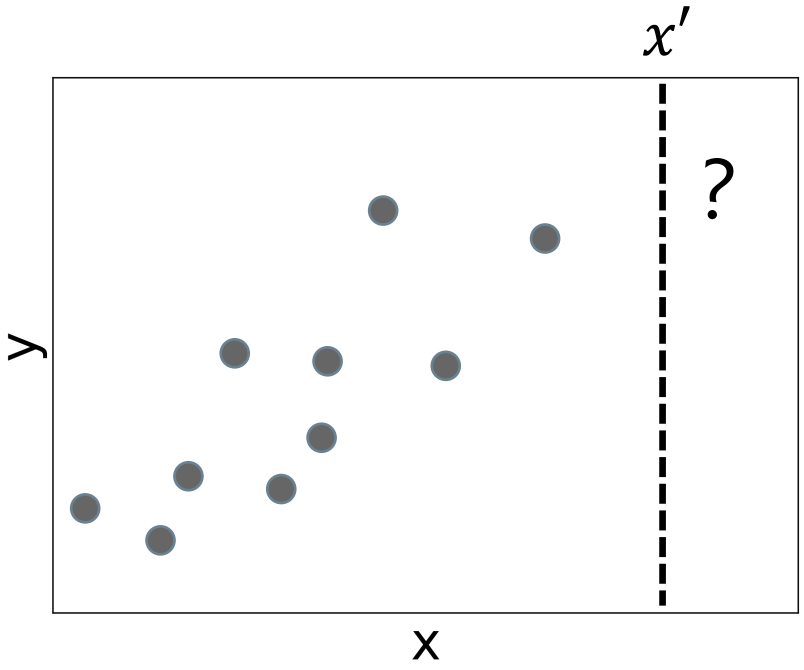
$$\sigma = \sqrt{\frac{\sum_i (y_i - \mu)^2}{N}}$$

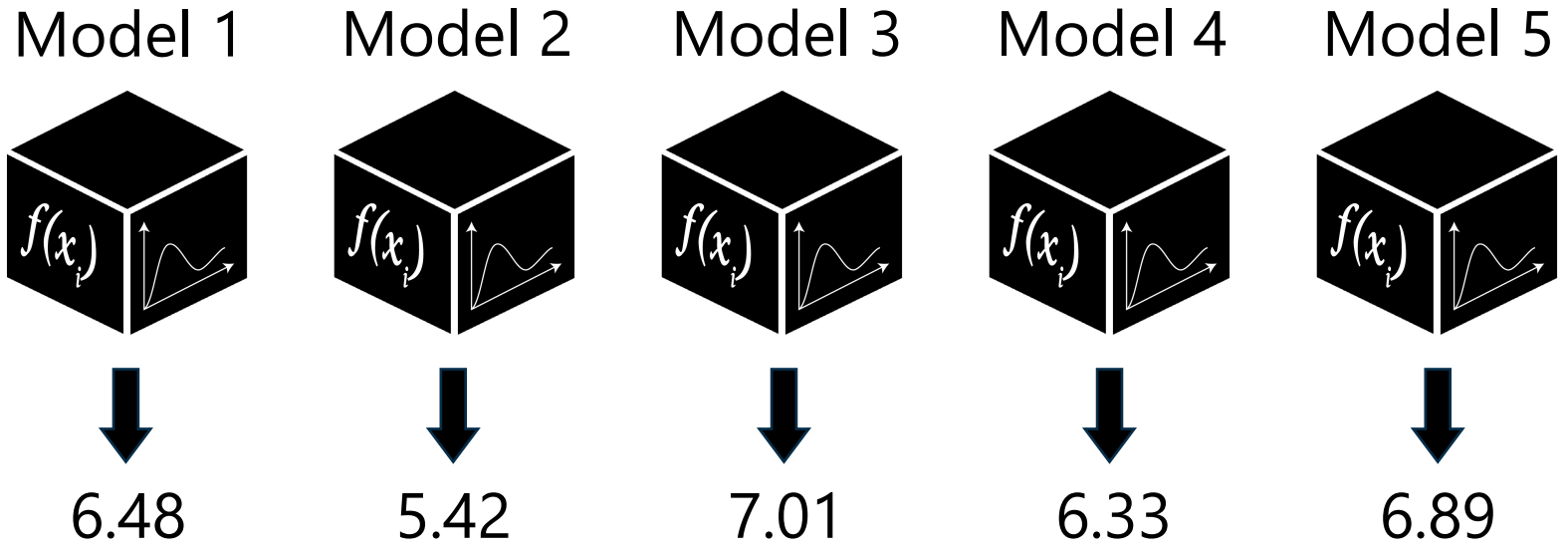
y_i = individual predictions

μ = mean of the predictions

N = number of predictions
(this should be as large as possible)

Method 1: ensemble methods for regression





We can use the **standard deviation** as a measure of uncertainty

$$\sigma = \sqrt{\frac{\sum_i (y_i - \mu)^2}{N}}$$

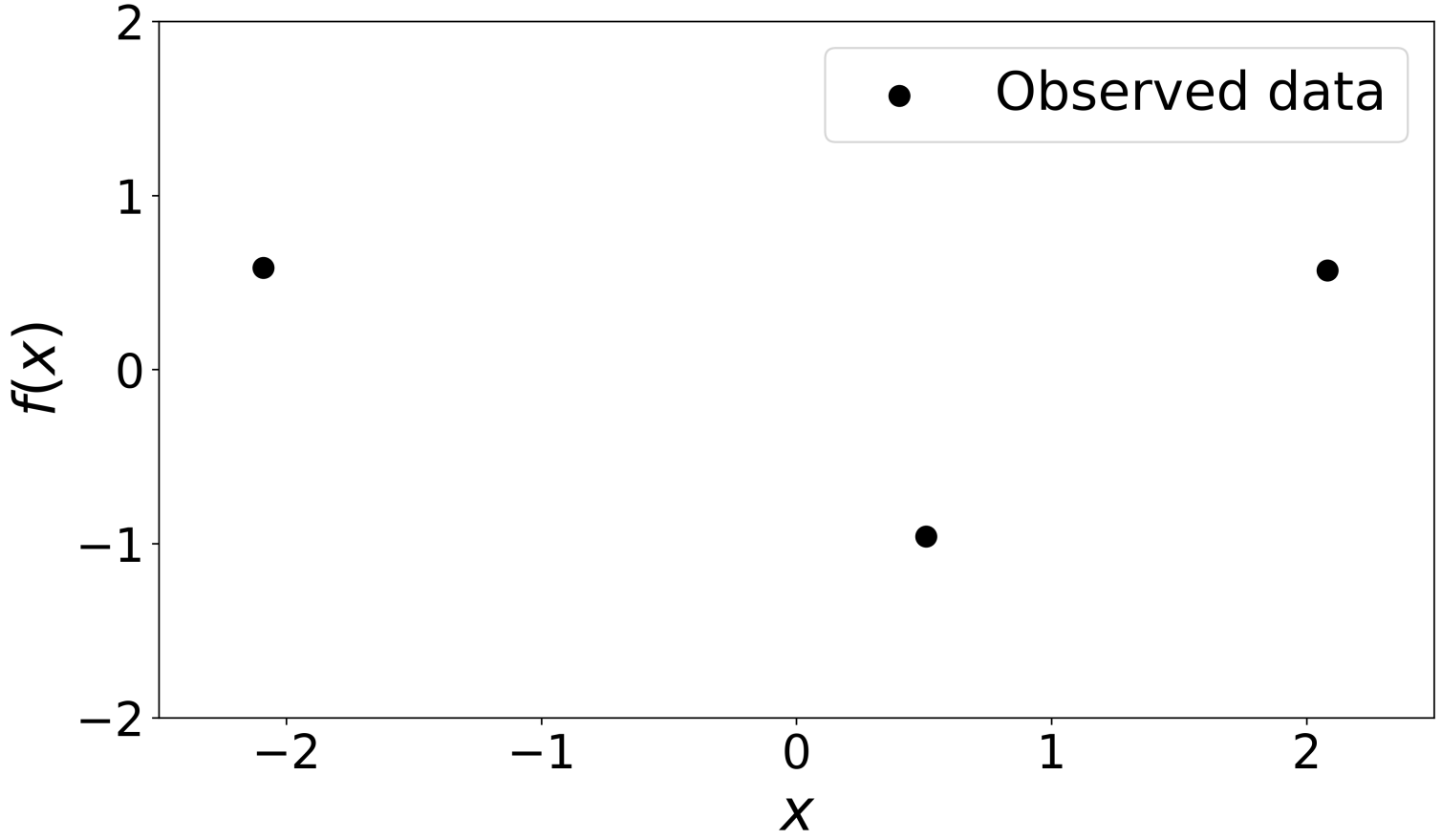
In this case:

Mean prediction = **6.43**

Std. deviation = **0.56**

Method 2: Gaussian processes have "built-in" uncertainty

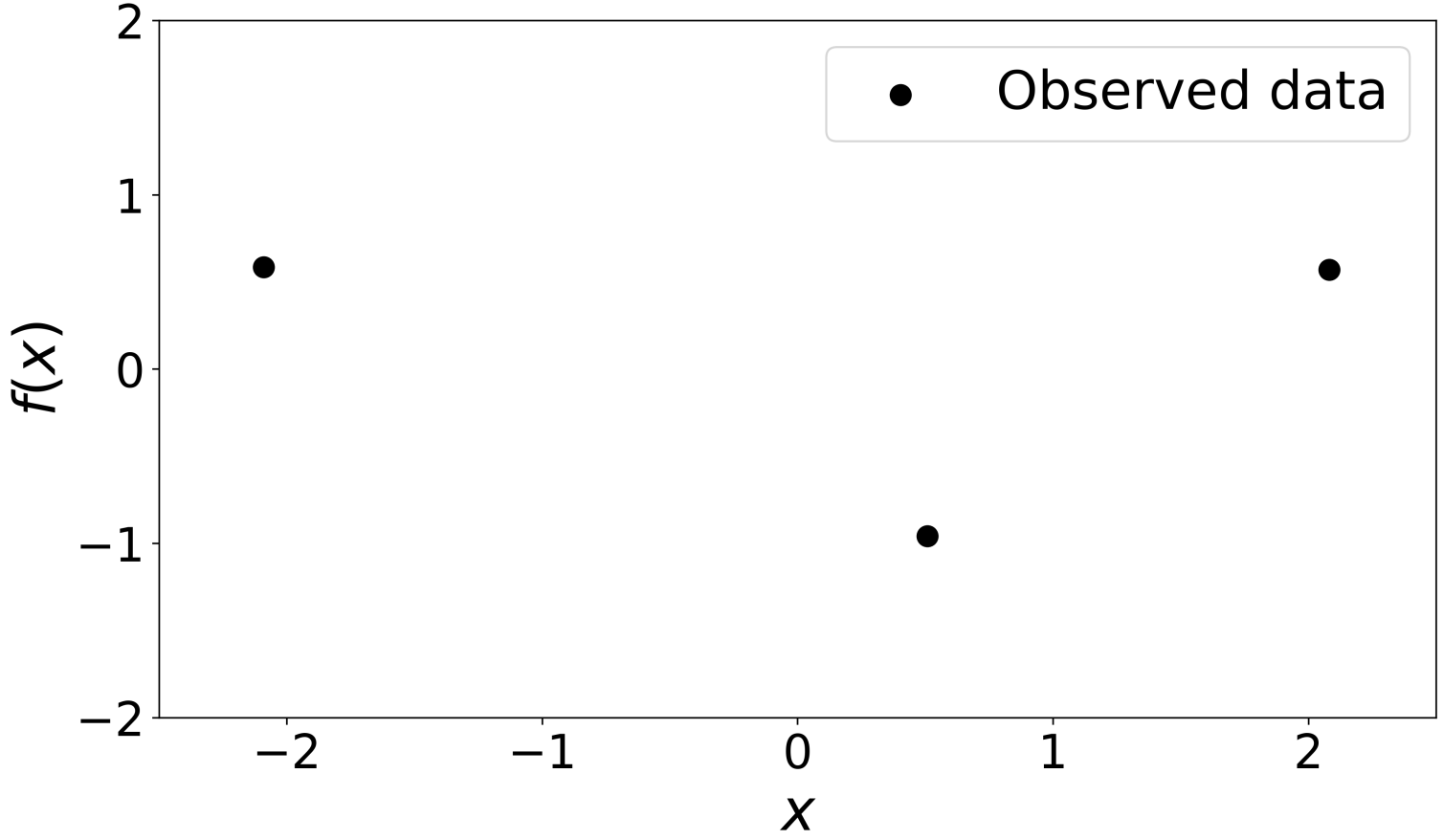
Consider a set of observed data $(\boldsymbol{x_i}, \boldsymbol{y_i})$ from which we aim to model $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$



For a nice explanation, see: **Gaussian Processes for Dummies**

Method 2: Gaussian processes have "built-in" uncertainty

Consider a set of observed data $(\boldsymbol{x_i}, \boldsymbol{y_i})$ from which we aim to model $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$

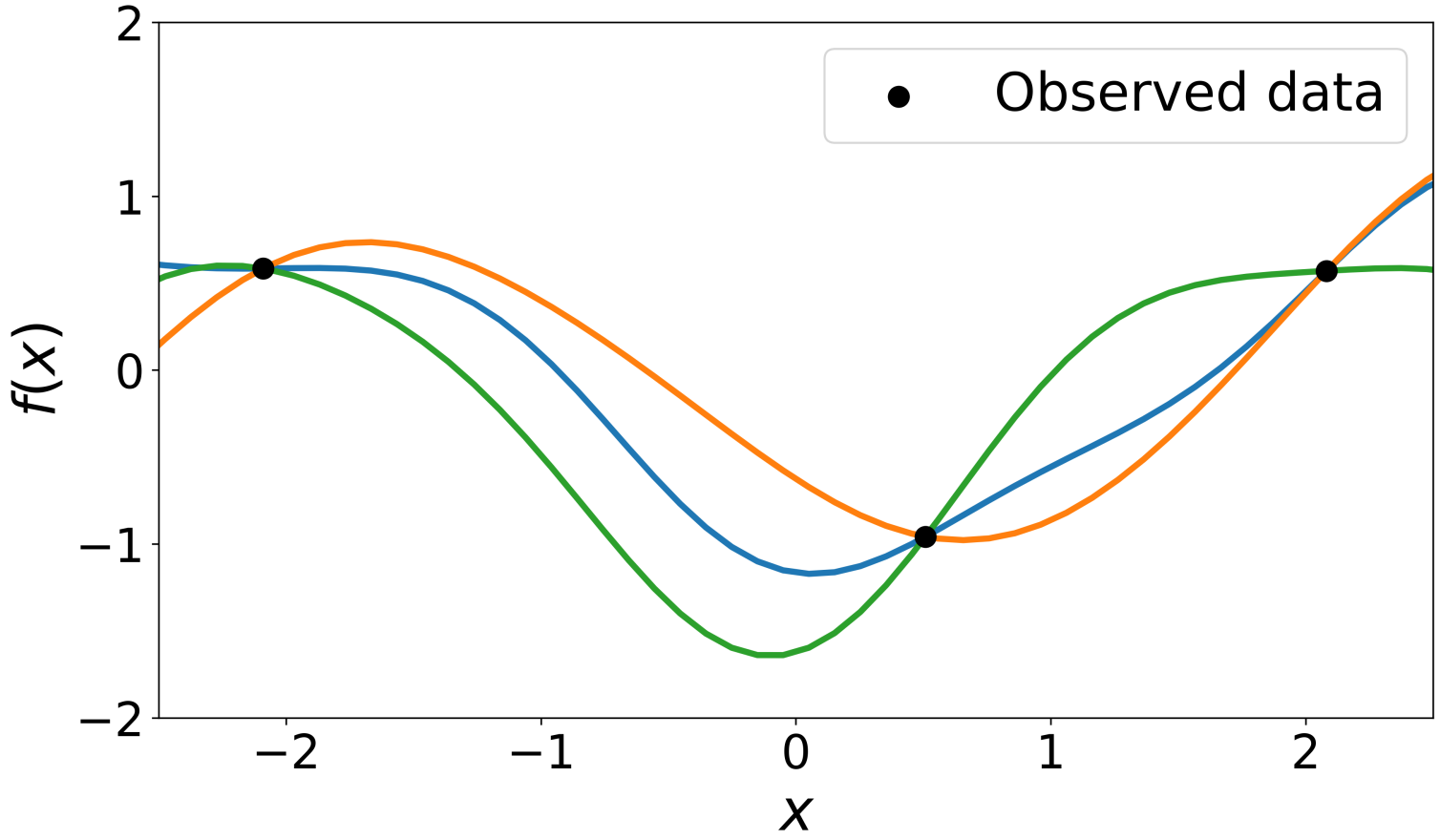


A **Gaussian process** (GP) describes a **distribution** over the **possible functions** $\boldsymbol{f}(\boldsymbol{x})$ that are consistent with the observed datapoints (x_i, y_i)

For a nice explanation, see: **Gaussian Processes for Dummies**

Method 2: Gaussian processes have "built-in" uncertainty

Consider a set of observed data $(\boldsymbol{x_i}, \boldsymbol{y_i})$ from which we aim to model $\boldsymbol{y = f(x)}$

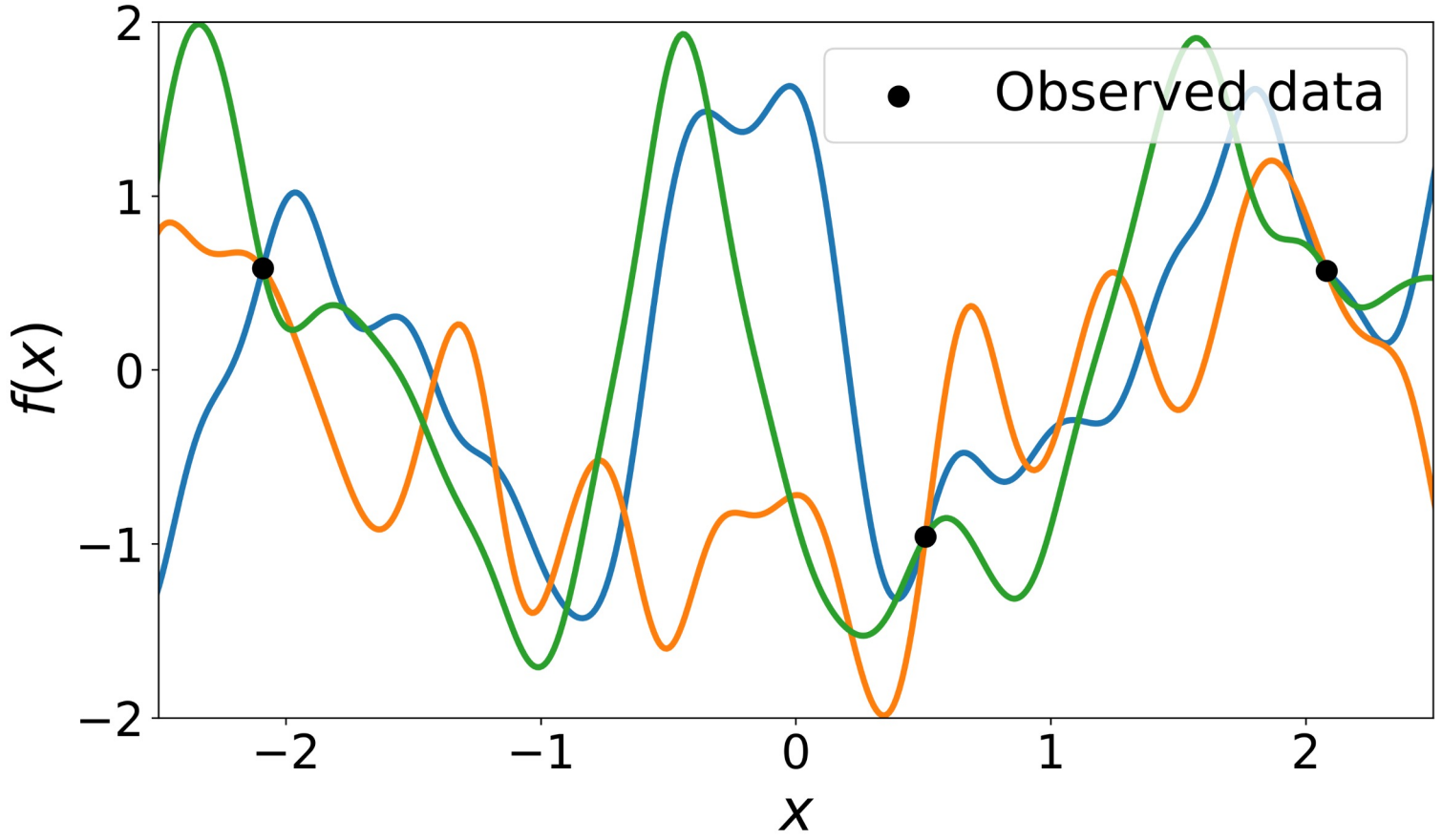


A **Gaussian process** (GP) describes a **distribution** over the **possible functions** $\boldsymbol{f(x)}$ that are consistent with the observed datapoints (x_i, y_i)

For a nice explanation, see: **Gaussian Processes for Dummies**

Method 2: Gaussian processes have "built-in" uncertainty

Consider a set of observed data (x_i, y_i) from which we aim to model $y = f(x)$



But in principle, these functions could be *anything*…

How do we reasonably **constrain** them?

For a nice explanation, see: **Gaussian Processes for Dummies**

Building a Gaussian process requires a *prior*

A ***prior*** describes an initial set of assumptions or beliefs we have about our system

To build a Gaussian process, we'll need **two things** to define our prior:

Building a Gaussian process requires a *prior*

A ***prior*** describes an initial set of assumptions or beliefs we have about our system

To build a Gaussian process, we'll need **two things** to define our prior:

The **mean** function: $\mu(x)$

This defines the average value of all functions at each point in the domain space

Usually, we start with $\mu(x) = 0$ and then modify it as more data is collected

Building a Gaussian process requires a *prior*

A ***prior*** describes an initial set of assumptions or beliefs we have about our system

To build a Gaussian process, we'll need **two things** to define our prior:

The **mean** function: $\mu(x)$

This defines the average value of all functions at each point in the domain space

Usually, we start with $\mu(x) = 0$ and then modify it as more data is collected

The **covariance** function: $K(x, x')$

This defines how function values at different points in the domain are correlated with one another

In simpler terms, it defines how "smooth" our functions are

Commonly used for covariance: the radial basis function

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

- When $\boldsymbol{x}$ **and** $\boldsymbol{x'}$ **are close to one another**, the RBF kernel $\boldsymbol{K(x, x') \to 1}$, which means $\boldsymbol{f(x)}$ **and** $\boldsymbol{f(x')}$ are highly correlated and **should exhibit similar values**

Commonly used for covariance: the radial basis function

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

- When $\boldsymbol{x}$ **and** $\boldsymbol{x'}$ **are close to one another**, the RBF kernel $\boldsymbol{K(x, x') \to 1}$, which means $\boldsymbol{f(x)}$ **and** $\boldsymbol{f(x')}$ are highly correlated and **should exhibit similar values**

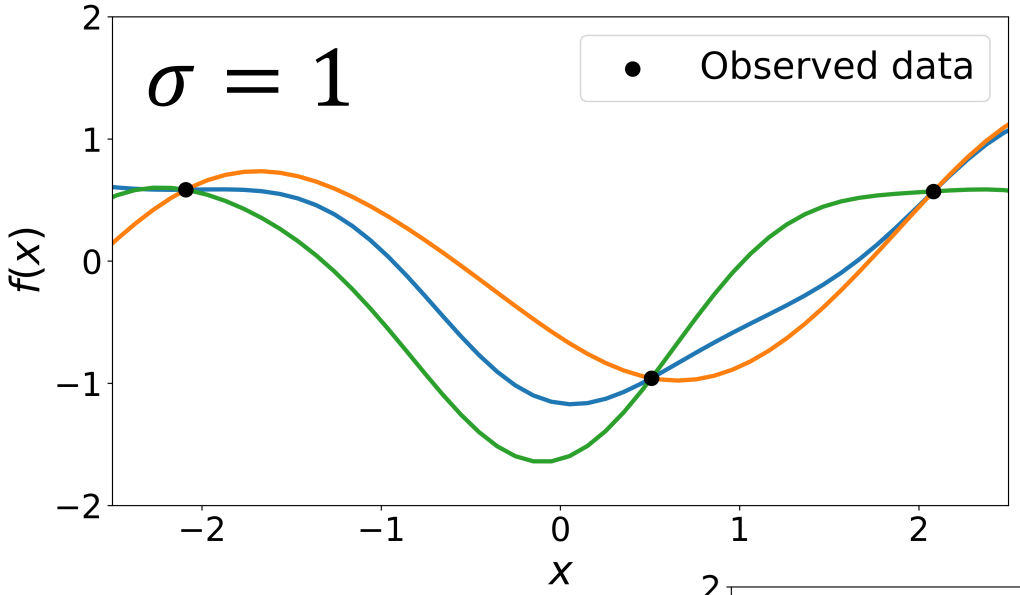
- When $\boldsymbol{x}$ **and** $\boldsymbol{x'}$ **are far from one another**, the RBF kernel $\boldsymbol{K(x, x') \to 0}$, which means $\boldsymbol{f(x)}$ **and** $\boldsymbol{f(x')}$ are ***not*** **correlated**

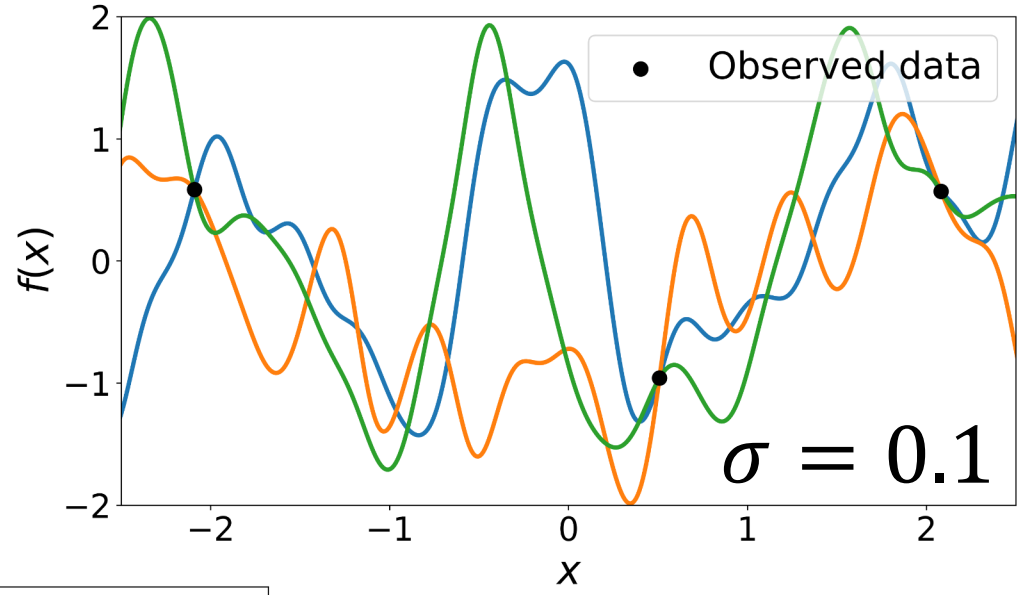
Commonly used for covariance: the radial basis function

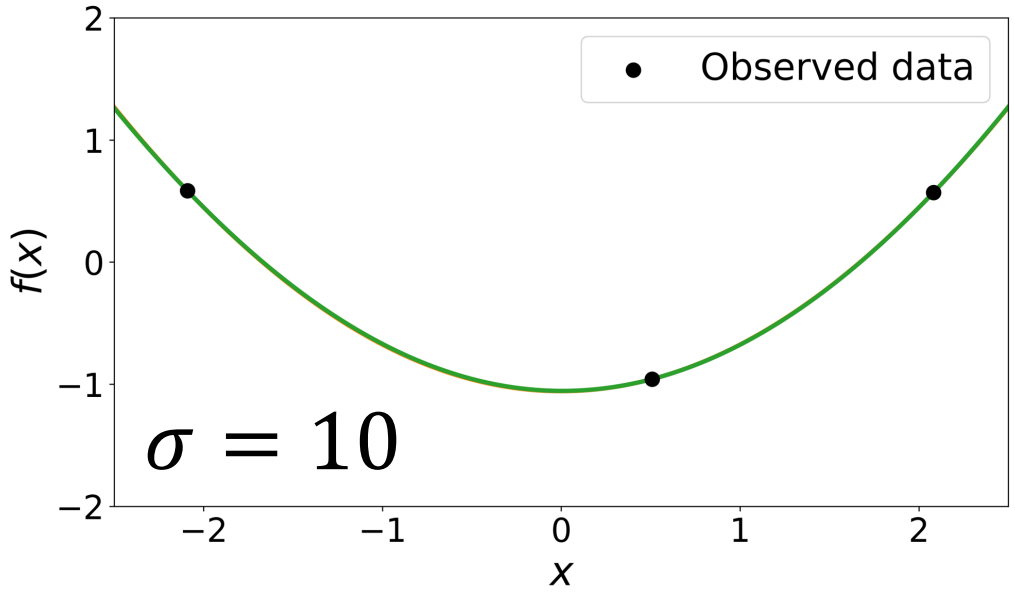
$$K(x, x') = \exp\left(-\frac{|x - x'|}{2\boldsymbol{\sigma}^2}\right)$$

- When x **and** x' **are close to one another**, the RBF kernel $K(x, x') \to 1$, which means $f(x)$ **and** $f(x')$ are highly correlated and **should exhibit similar values**
- When x **and** x' **are far from one another**, the RBF kernel $K(x, x') \to 0$, which means $f(x)$ **and** $f(x')$ are ***not*** **correlated**
- σ **is a hyperparameter** that measures the **correlation length**. Essentially, this controls **how smooth** we want our functions to be.

σ is a user-chosen parameter that controls "smoothness"

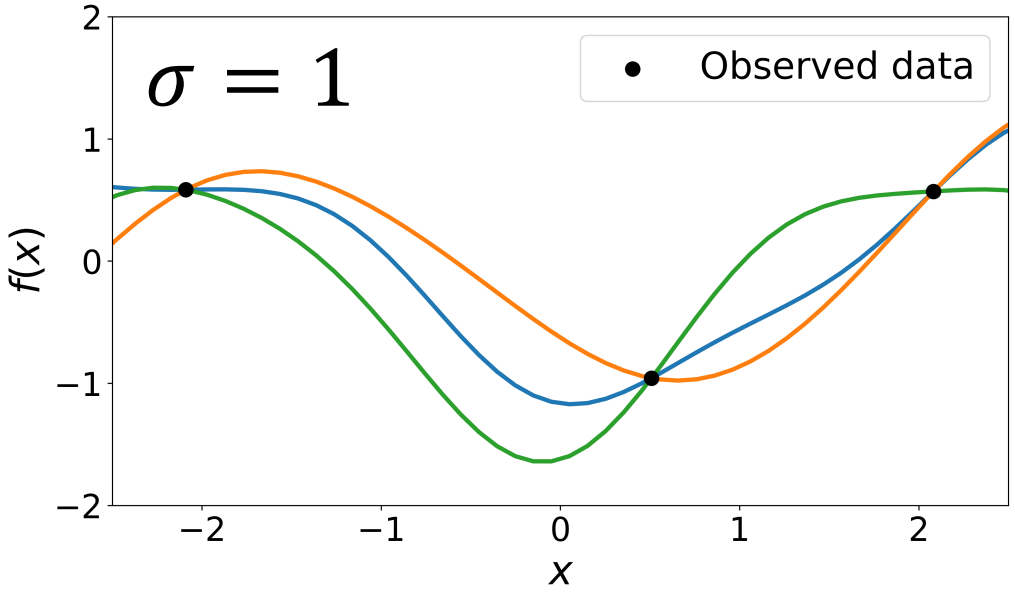






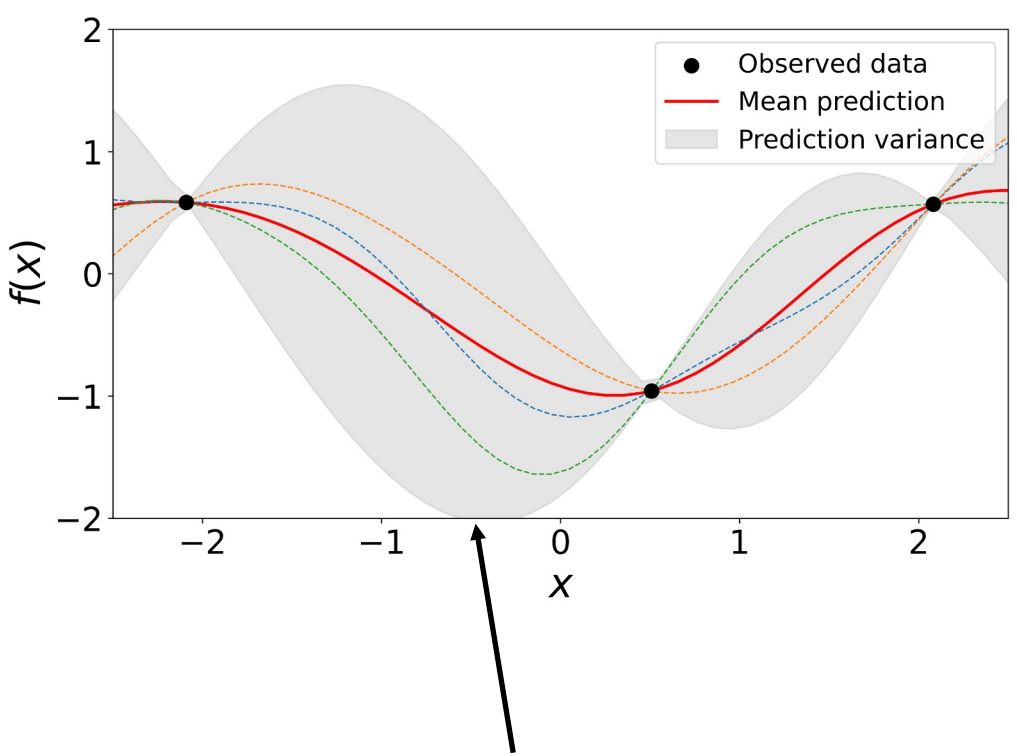
Looks reasonable, but notice there is no spread…These simple models are likely to be overconfident

We can get uncertainty from the prediction variance



The variance is the square of the standard deviation: $\mathbf{Var}(\boldsymbol{x}) = \boldsymbol{\sigma}^{\mathbf{2}}(\boldsymbol{x})$

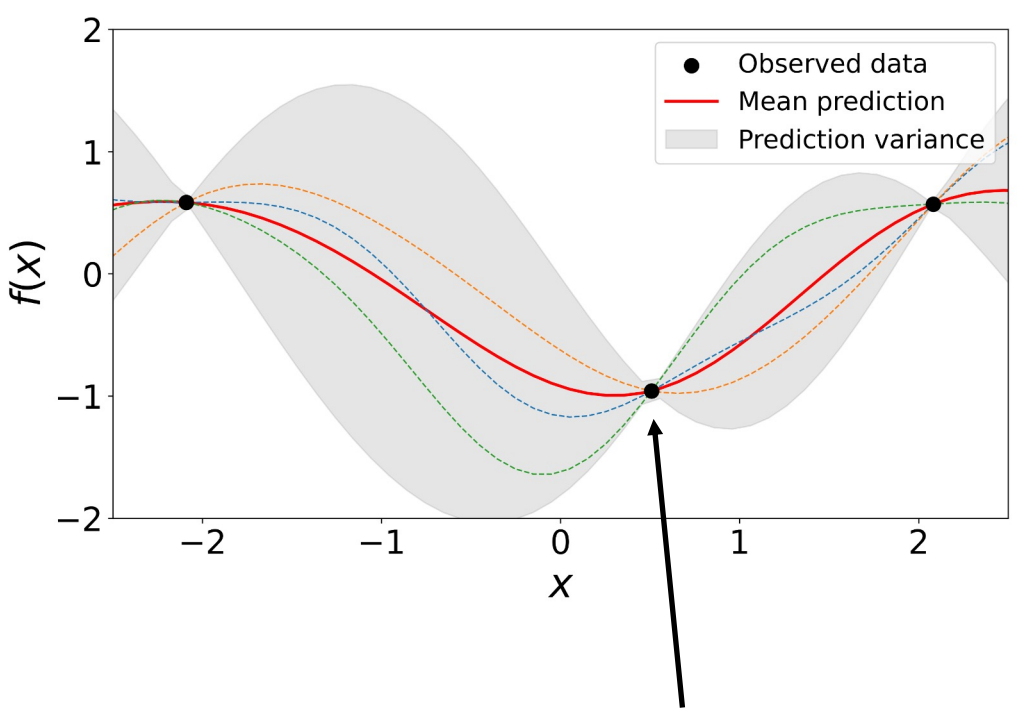
We can get uncertainty from the prediction variance



The variance is the square of the standard deviation: $\mathbf{Var}(\boldsymbol{x}) = \boldsymbol{\sigma}^{\mathbf{2}}(\boldsymbol{x})$

Large variance exist in regions without much data → **high uncertainty!**

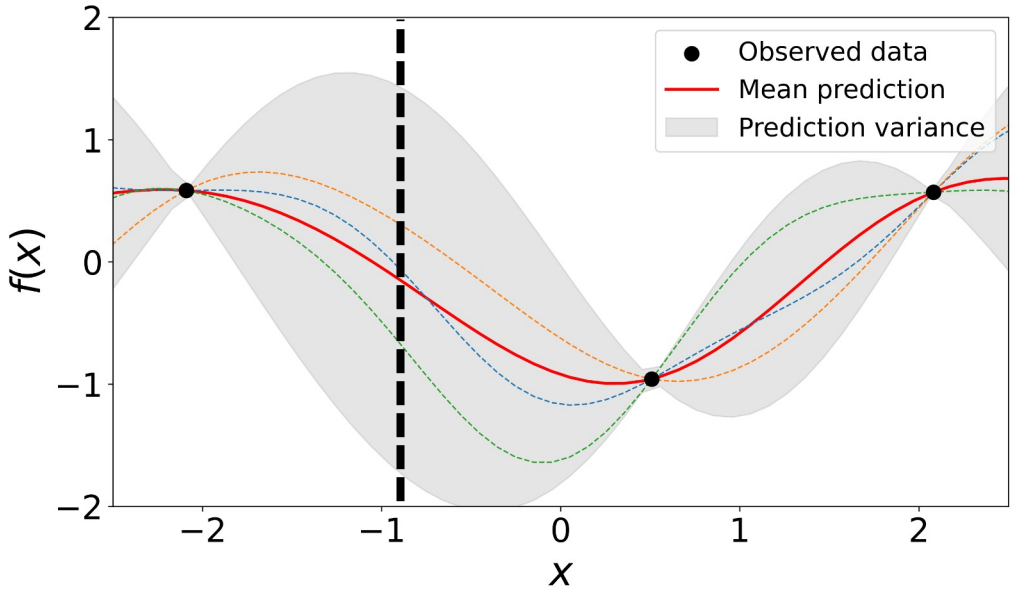
We can get uncertainty from the prediction variance



The variance is the square of the standard deviation: $\mathbf{Var}(\boldsymbol{x}) = \boldsymbol{\sigma}^{\mathbf{2}}(\boldsymbol{x})$

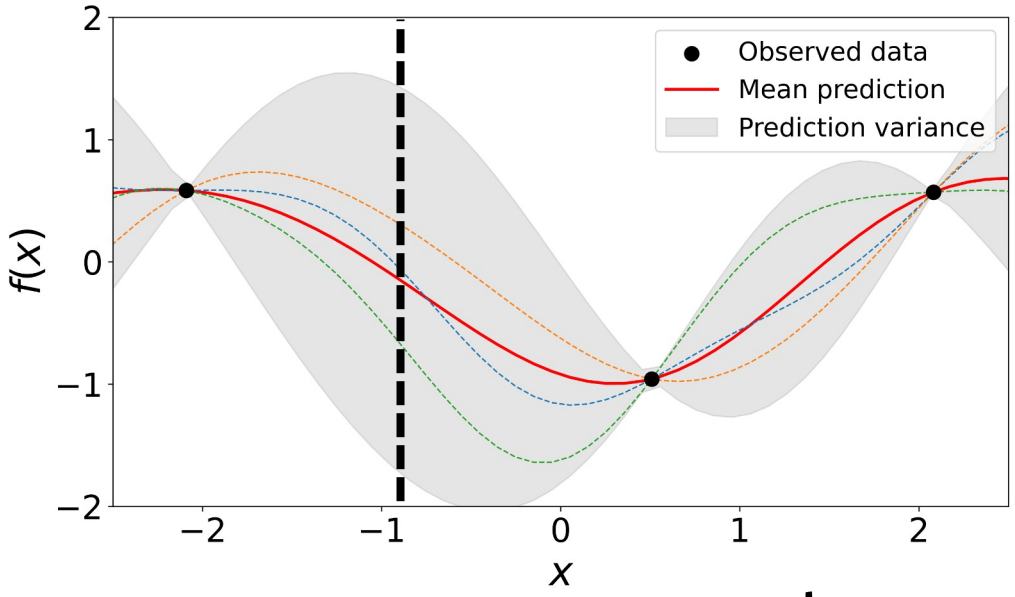
Little to no variance around the known data → **low uncertainty!**

To improve model accuracy: sample high-uncertainty areas



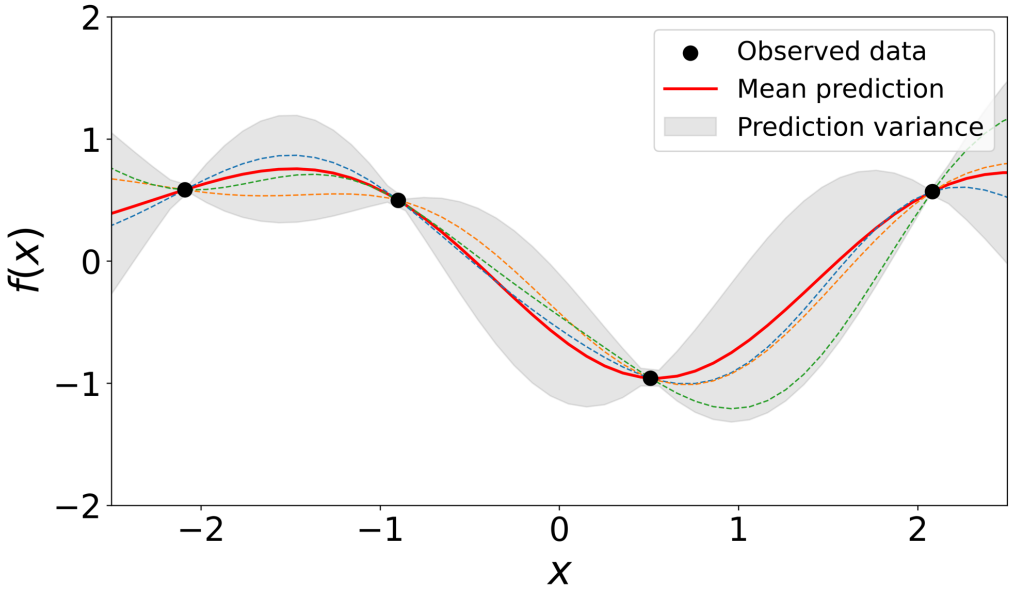
If all we care about is model accuracy, then we should sample the part of the domain with the **highest uncertainty**

To improve model accuracy: sample high-uncertainty areas

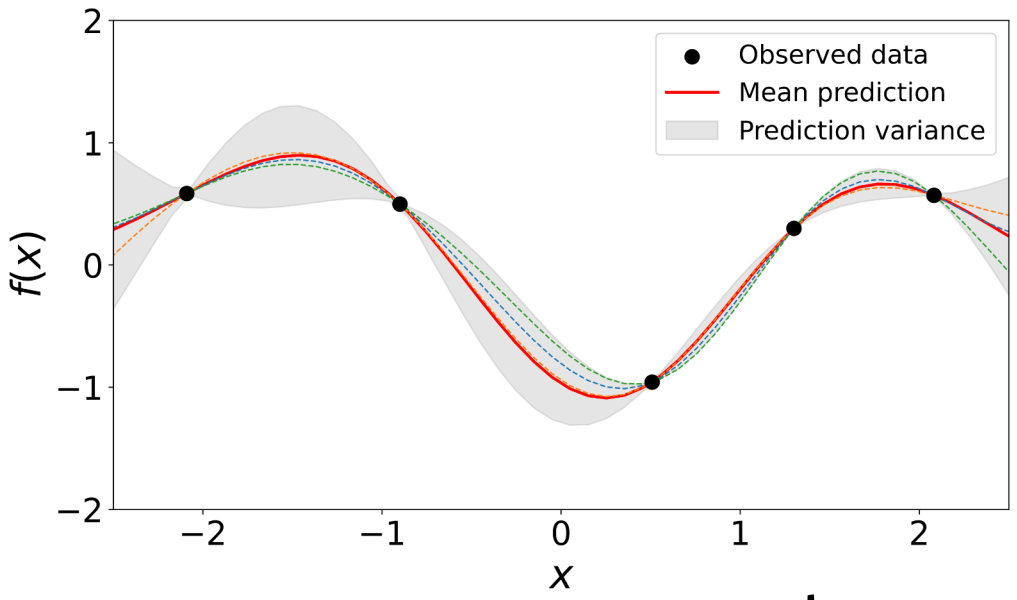


If all we care about is model accuracy, then we should sample the part of the domain with the **highest uncertainty**

Re-train the GP using the updated dataset

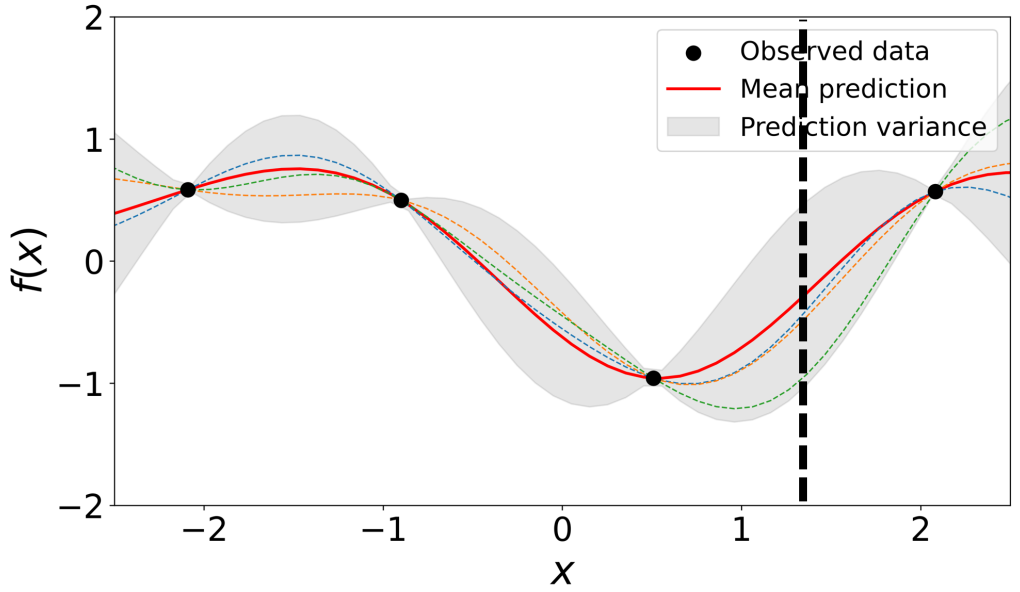


To improve model accuracy: sample high-uncertainty areas

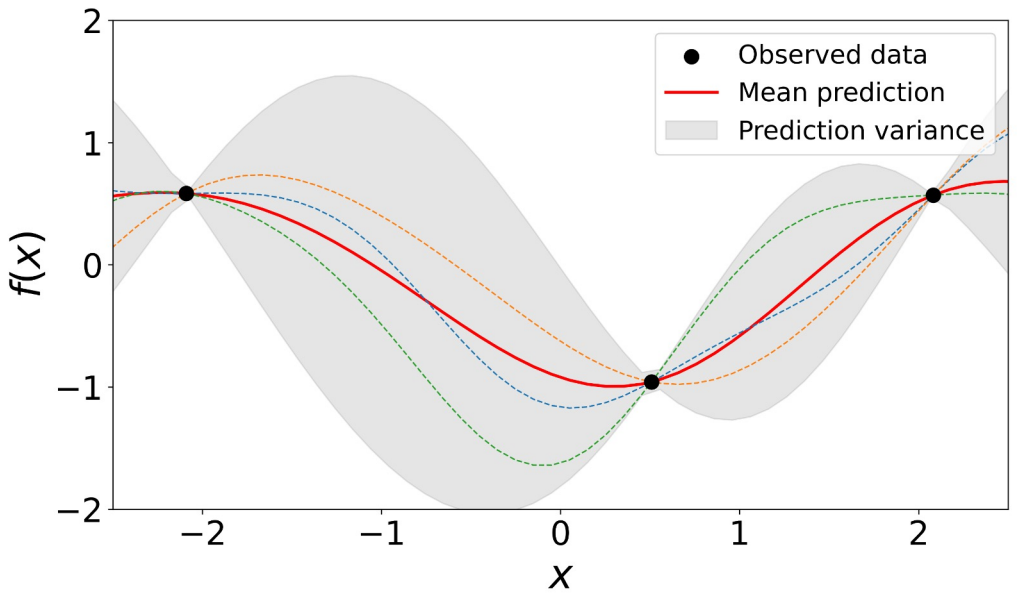


Repeat this process until you are happy with your model's uncertainty over the entire domain...

Re-train the GP using the updated dataset



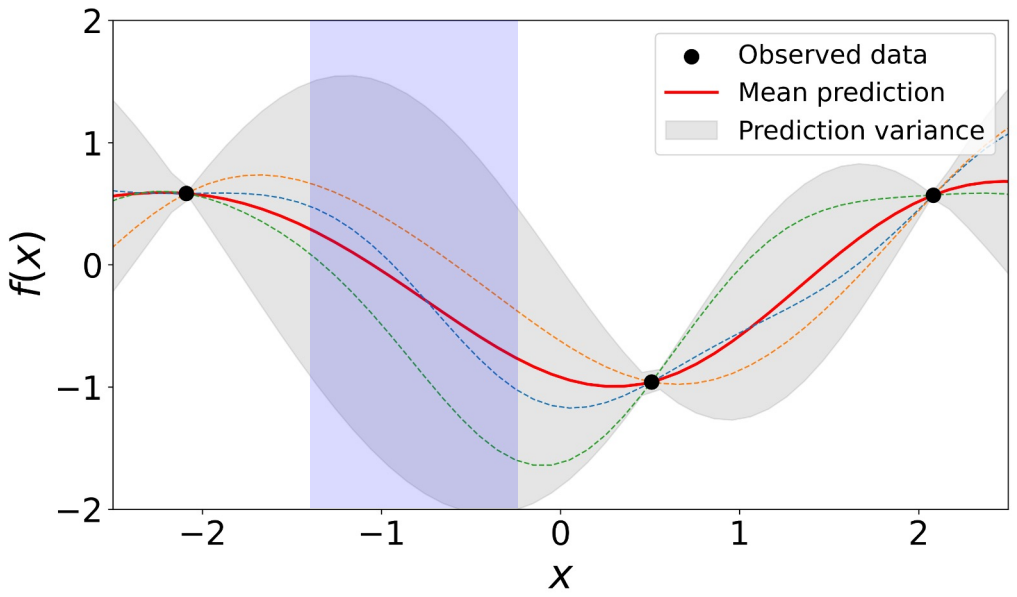
Now, what if we want to use our model for *optimization?*



Objective:
Find the value of x where $f(x)$ is maximal

To accomplish this, we need to balance **exploration** with **exploitation**

Now, what if we want to use our model for *optimization?*



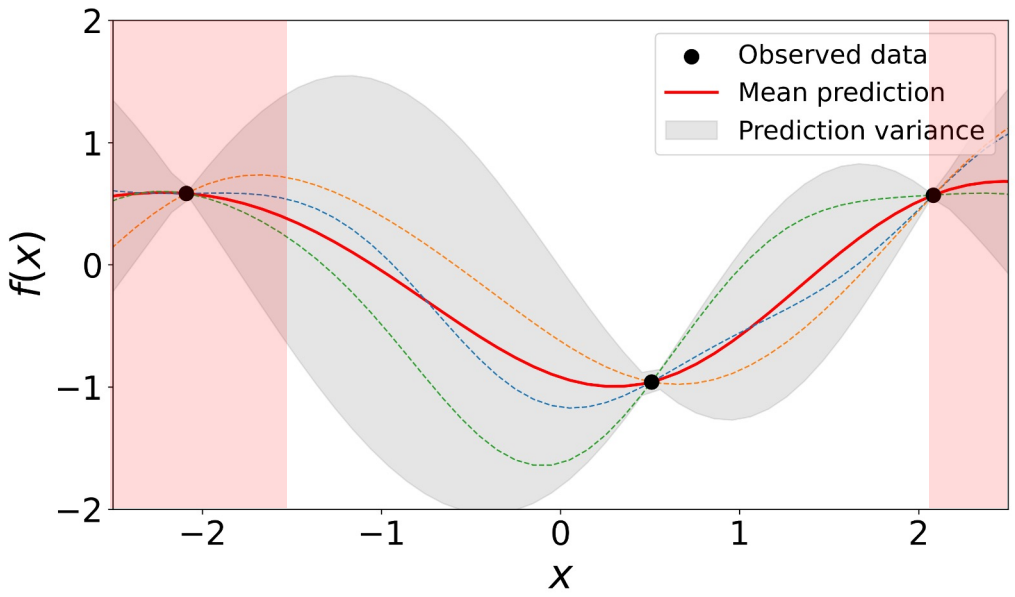
Objective:
Find the value of x where $f(x)$ is maximal

To accomplish this, we need to balance **exploration** with **exploitation**

Exploration:

Collecting new training data in parts of the domain that are under-sampled and have high model uncertainty → This is basically what we've been prioritizing so far

Now, what if we want to use our model for *optimization?*



Objective:
Find the value of x where $f(x)$ is maximal

To accomplish this, we need to balance ***exploration*** with ***exploitation***

Exploration:

Collecting new training data in parts of the domain that are under-sampled and have high model uncertainty

Exploitation:

Collecting new training data in parts of the domain where $f(x)$ is expected to be optimal

Acquisition functions balance exploration/exploitation

Acquisition functions quantify the **anticipated benefit** of sampling a point (x)

Acquisition functions balance exploration/exploitation

Acquisition functions quantify the **anticipated benefit** of sampling a point (x)

Upper Confidence Bound (UCB):

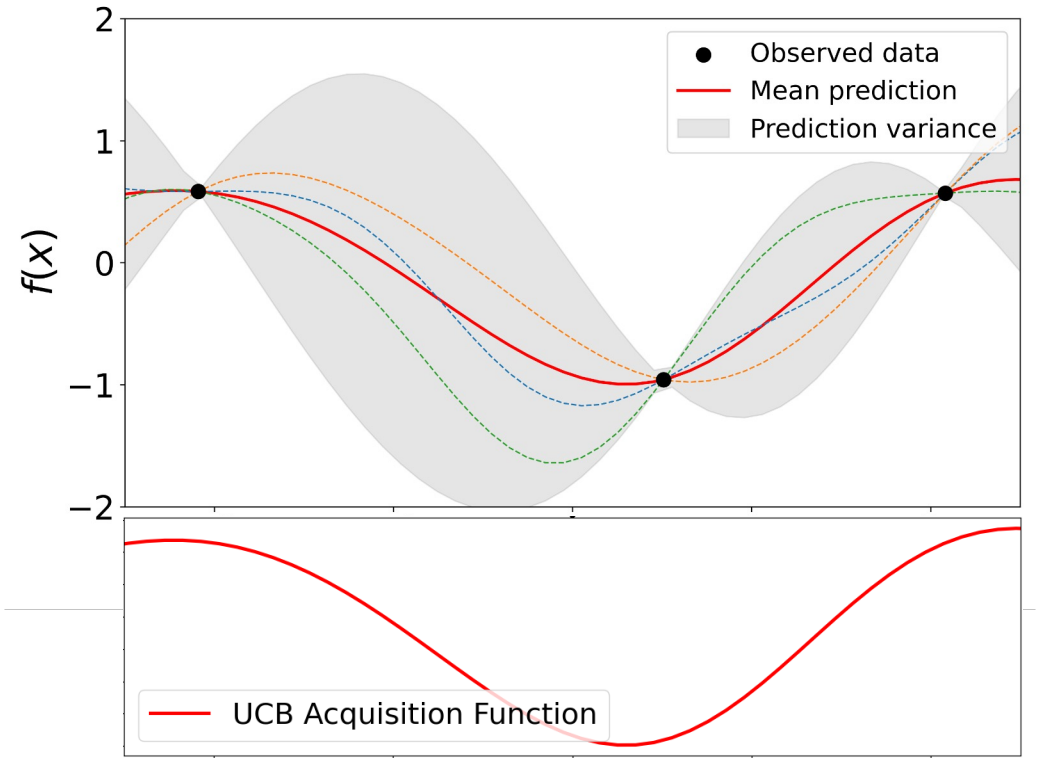
$$a(x, \beta) = \mu(x) + \beta\sigma(x)$$

Mean of the predicted function

Uncertainty of the predicted function

β **is a hyperparameter** which we can use to control the **exploration-exploitation tradeoff**

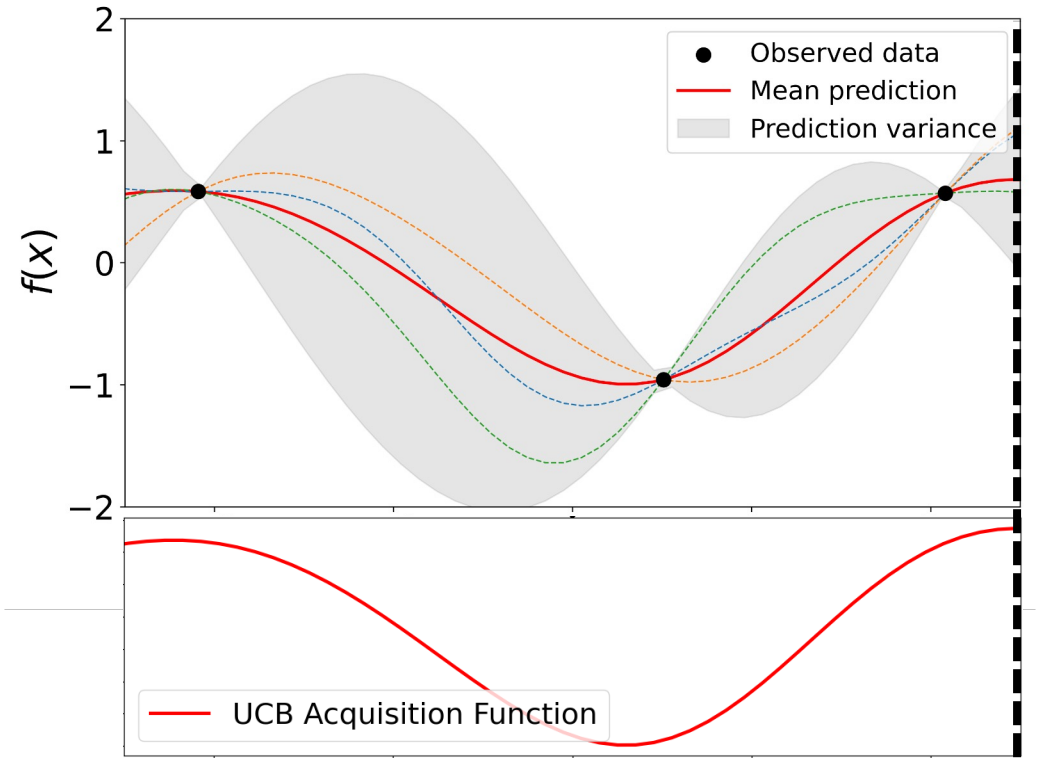
Controlling the exploration-exploitation tradeoff with β



$$\beta = 0$$

Pure exploitation

Controlling the exploration-exploitation tradeoff with β

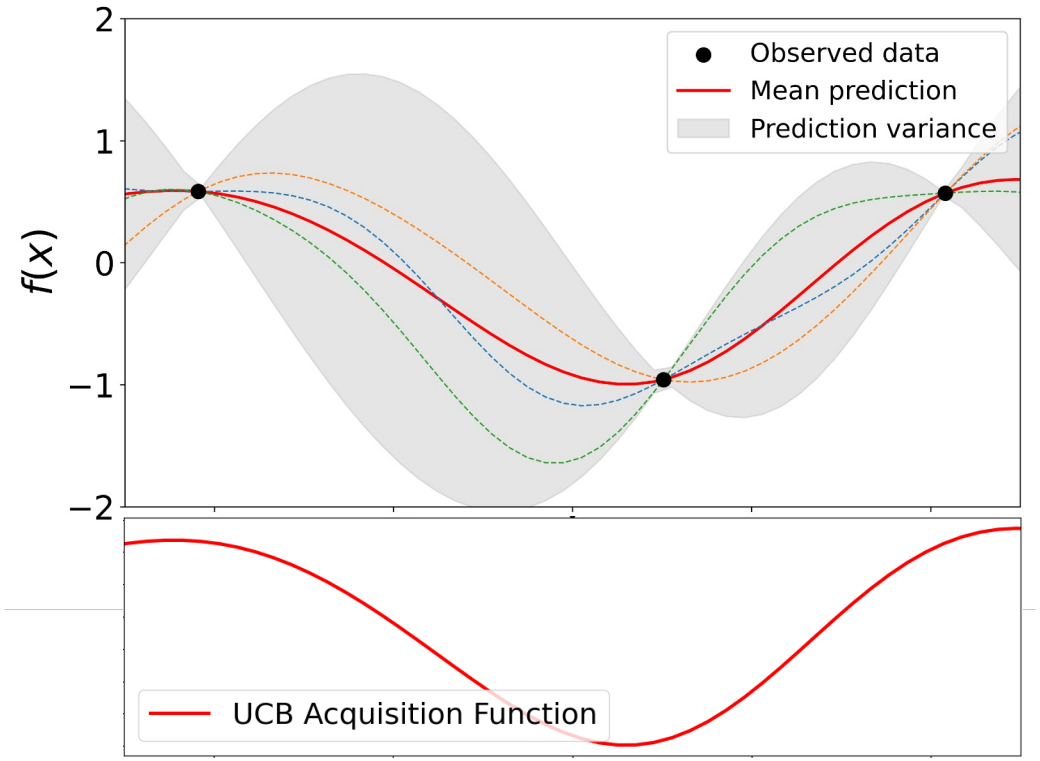


$\beta = 0$

Pure exploitation

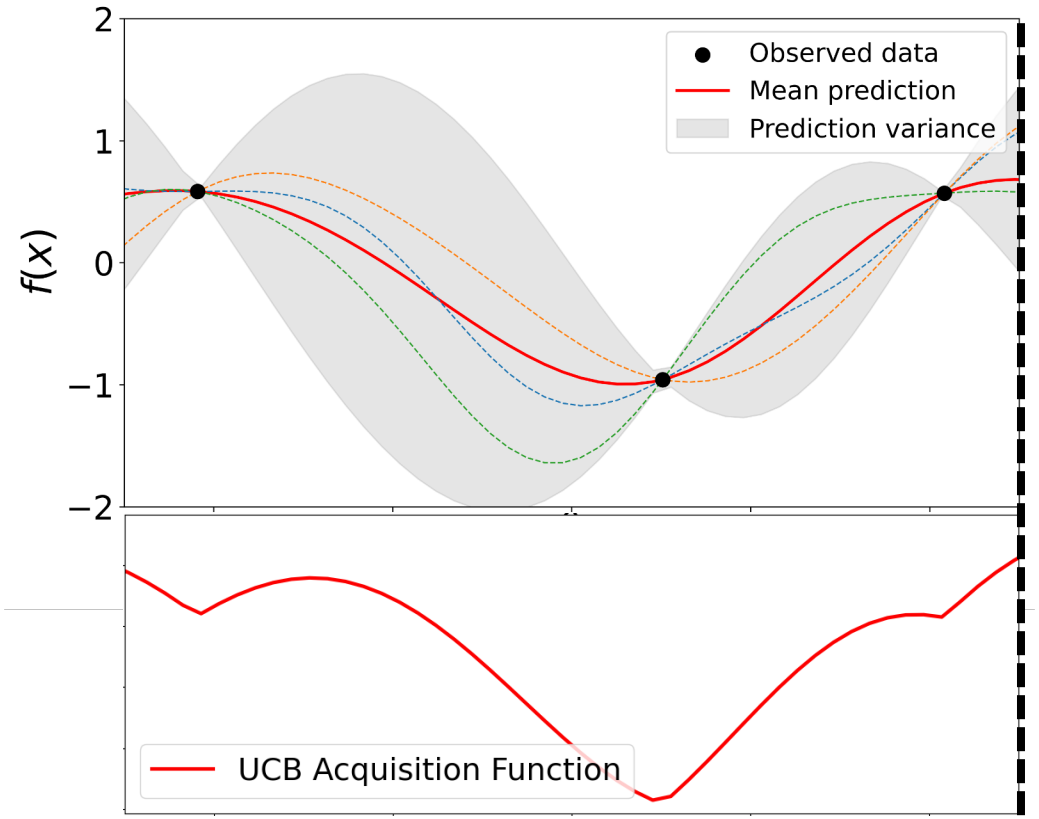
Sample the point corresponding to the maximum in the acquisition function

Controlling the exploration-exploitation tradeoff with β



$\beta = 0$

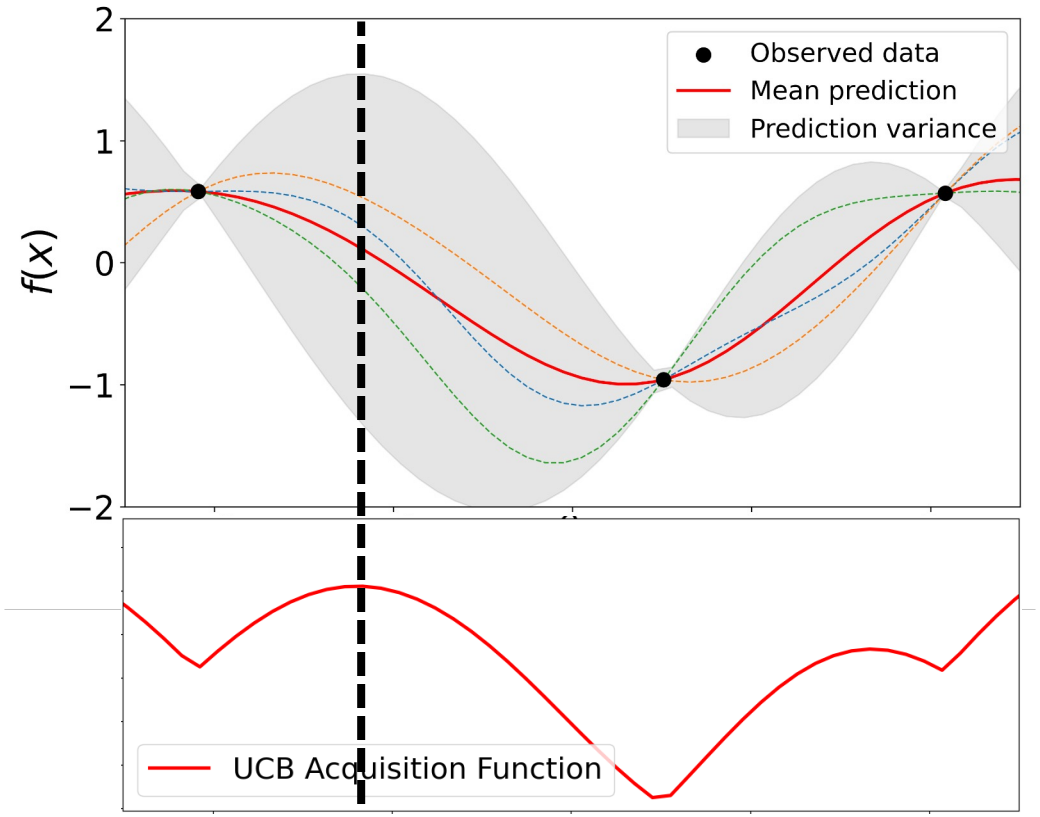
Pure exploitation



$\beta = 1$

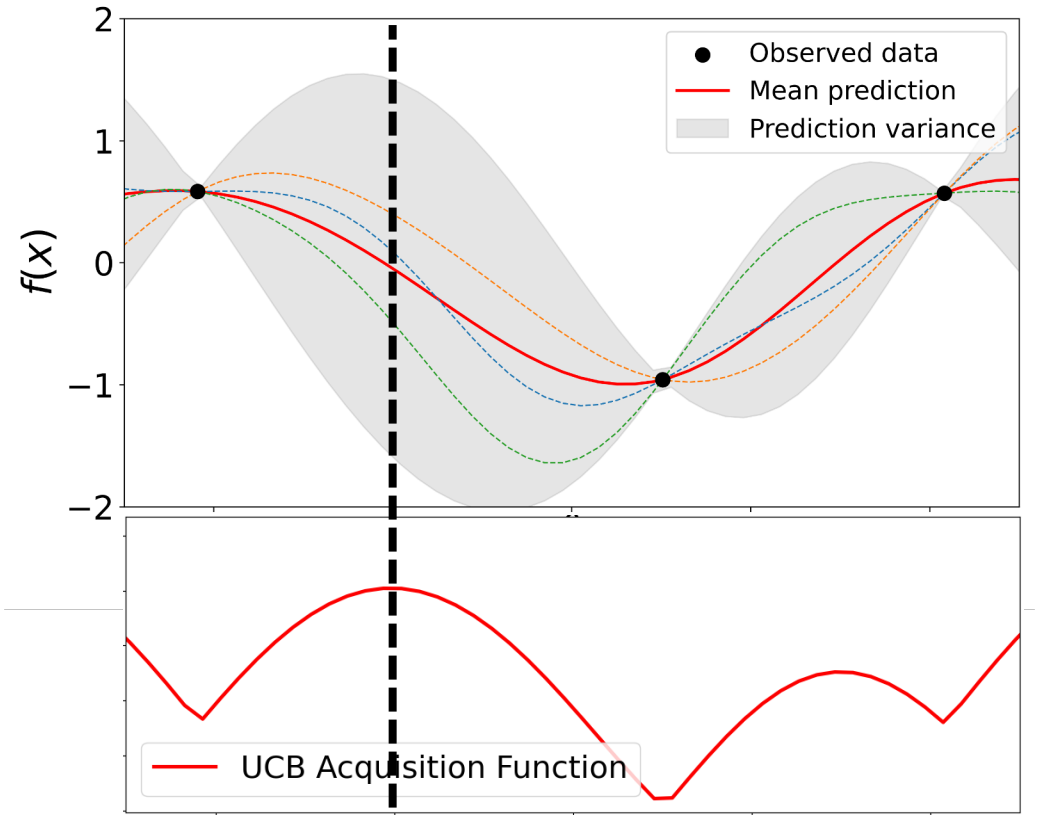
Some exploration

Controlling the exploration-exploitation tradeoff with β



$\beta = 2$

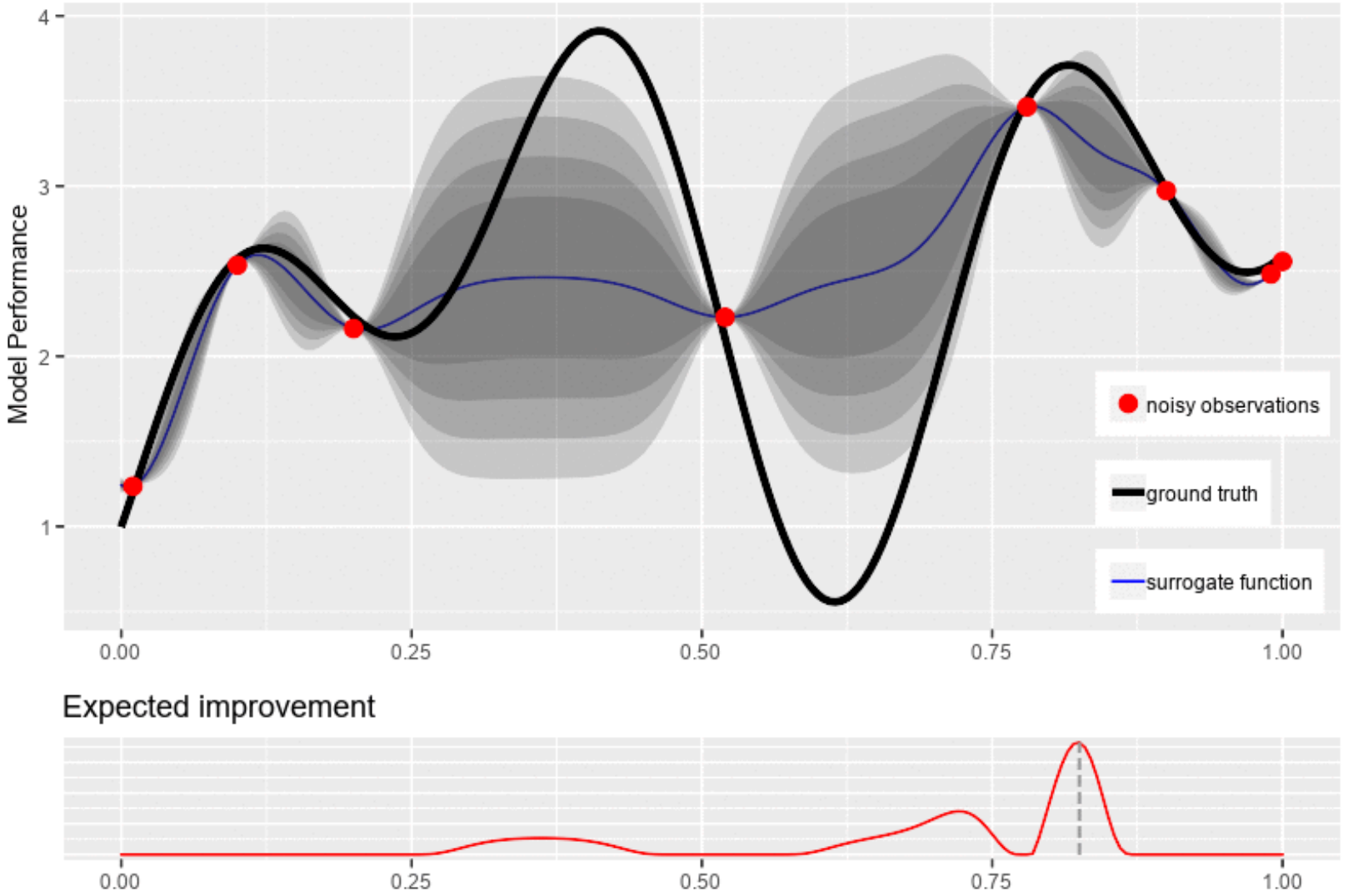
More exploration



$\beta = 4$

And even more exploration

Putting it all together: Bayesian optimization!



Widely used for optimization in the **physical sciences**, where experiments require a lot of time, money, and effort

From "Bayesian Hyperparameter Optimization" by Matti Karppanen

There is *a lot* that can be tuned in Bayesian optimization!

CEMS
Chemical Engineering
& Materials Science

Surrogate models:

- Gaussian processes are most popular since they provide uncertainty
- But any ML model can be used – neural networks are increasingly common

Acquisition functions:

- Many different ones exist: upper confidence bound, expected improvement, entropy search, and so on…All balance exploration/exploitation differently

Prior:

- The mean and covariance should be set to reflect your system
- For example, a periodic system should use periodic covariance (trig functions)

There is *a lot* that can be tuned in Bayesian optimization!

CEMS
Chemical Engineering
& Materials Science

Surrogate models:

- Gaussian processes are most popular since they provide uncertainty
- But any ML model can be used – neural networks are increasingly common

Acquisition functions:

- Many different ones exist: upper confidence bound, expected improvement, entropy search, and so on…All balance exploration/exploitation differently

Prior:

- The mean and covariance should be set to reflect your system
- For example, a periodic system should use periodic covariance (trig functions)

There is *a lot* that can be tuned in Bayesian optimization!

CEMS
Chemical Engineering
& Materials Science

Surrogate models:

- Gaussian processes are most popular since they provide uncertainty
- But any ML model can be used – neural networks are increasingly common

Acquisition functions:

- Many different ones exist: upper confidence bound, expected improvement, entropy search, and so on…All balance exploration/exploitation differently

Prior:

- The mean and covariance should be set to reflect your system
- For example, a periodic system should use periodic covariance (trig functions)