Bayesian optimization

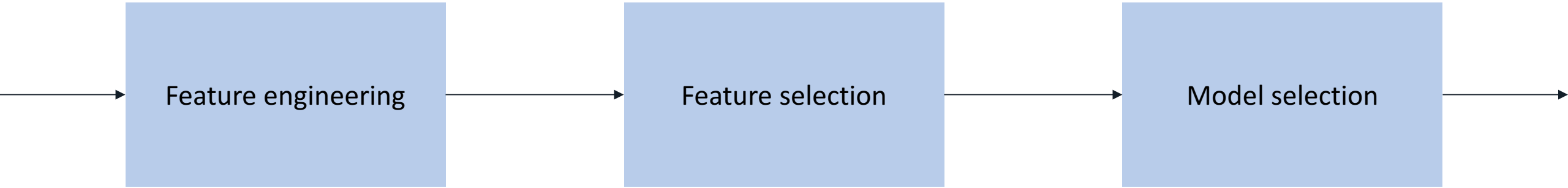
PyData London 2017

Thomas Huijskens

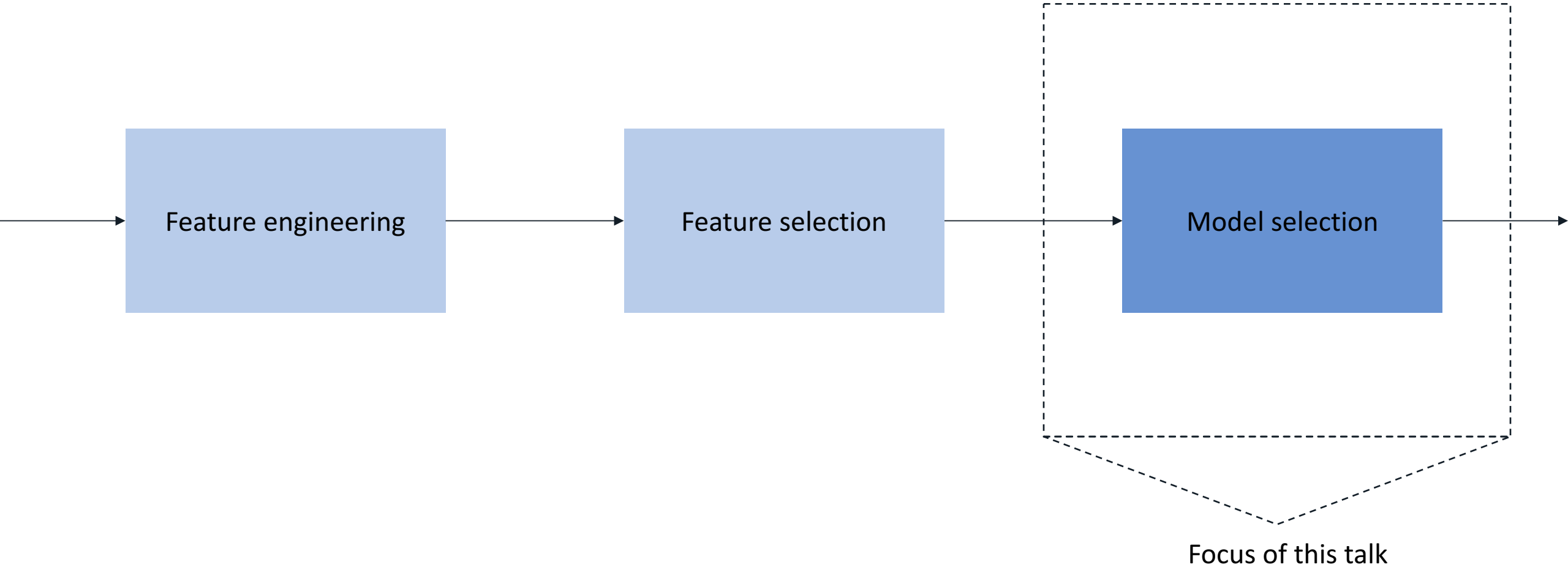
6th of May, 2017

QUANTUMBLACK

The *modelling* workflow of a data scientist roughly follows three steps



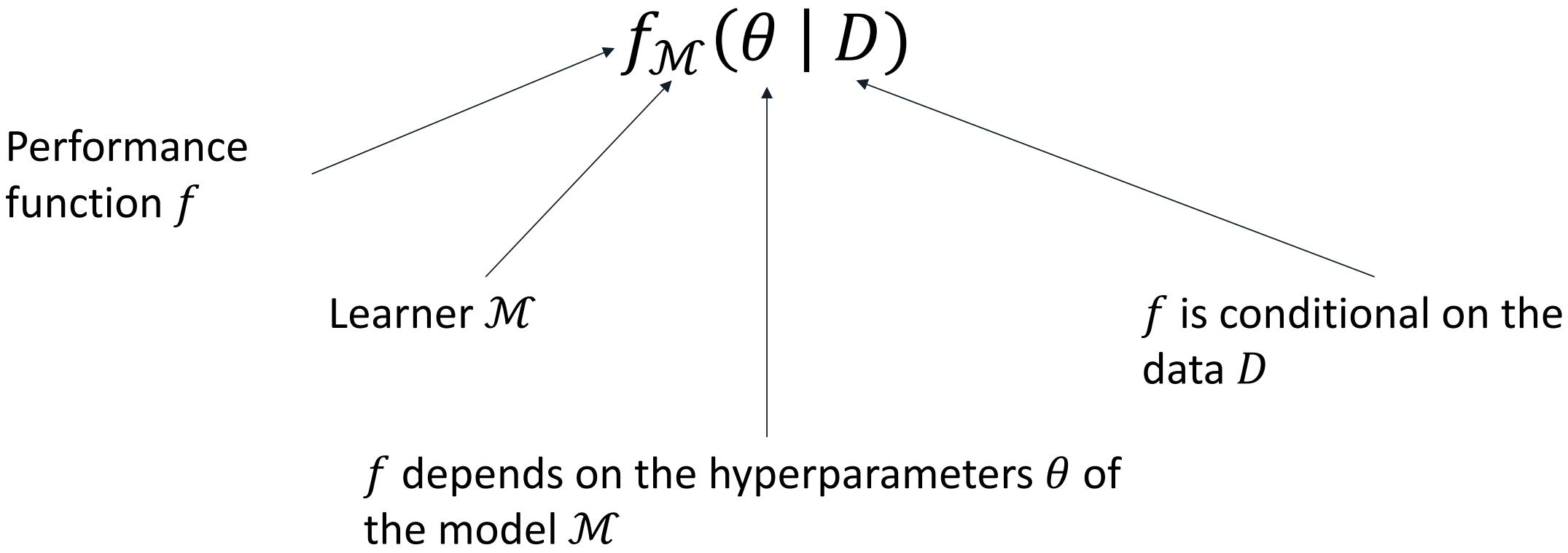
In this talk, we will focus on the last step of this workflow



Most models have a number of hyperparameters that need to be chosen a priori..

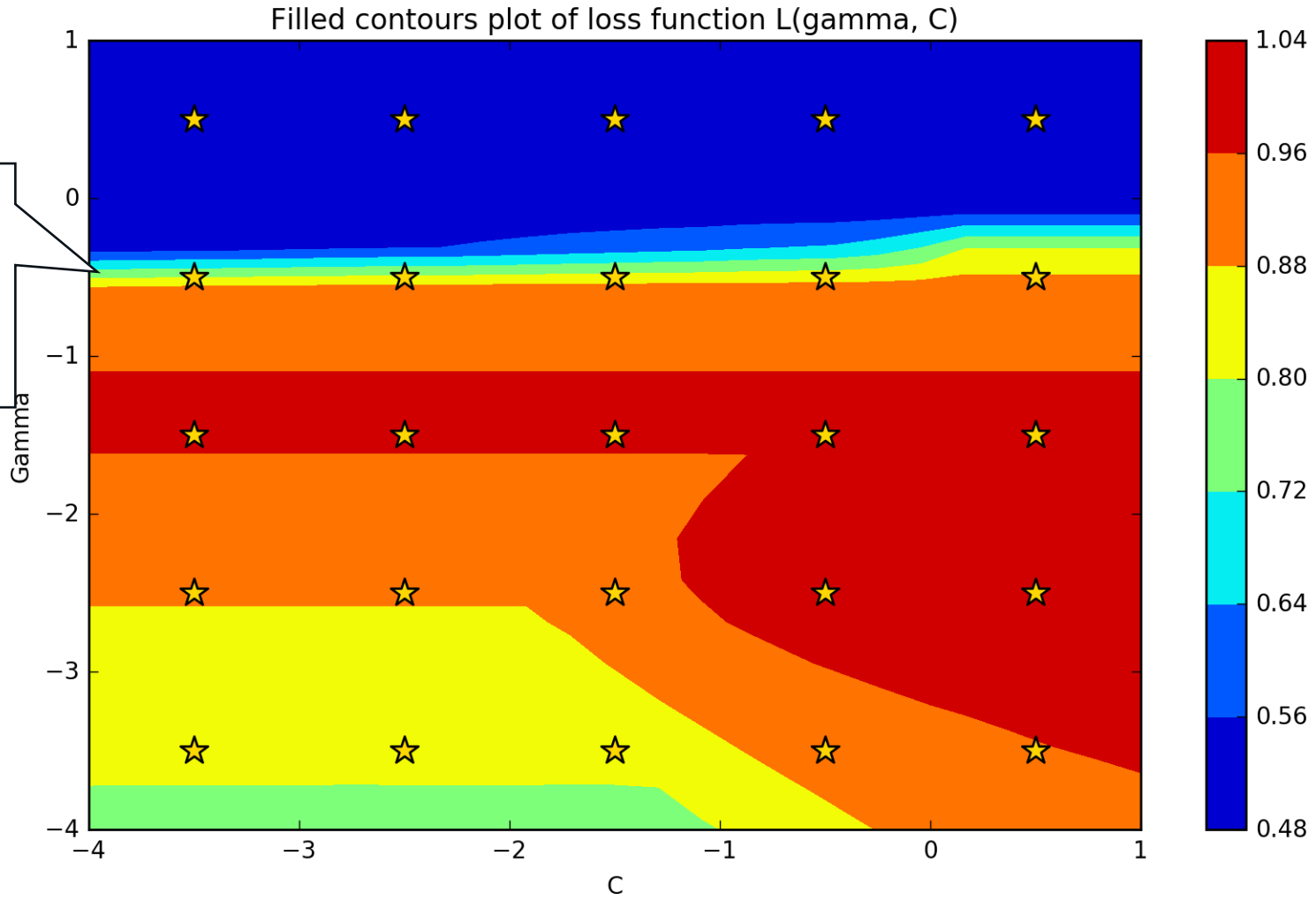
- Typically, we want to optimize some performance metric $f_{\mathcal{M}}$ for a model $\mathcal{M}$, on a hold-out set of data.
- The model $\mathcal{M}$ has some hyperparameters θ that we need to specify, and the performance of $\mathcal{M}$ is highly dependent on the settings of θ.
- Example: For a classification problem, $\mathcal{M}$ could be a support vector machine, and the performance metric $f_{\mathcal{M}}$ could be the out-of-sample AUC.
- Because the performance of the SVM depends on hyperparameters θ, the metric $f_{\mathcal{M}}$ depends on θ as well.

.. and our goal is to find the values of θ for which the value of the (out-of-sample) performance $f_{\mathcal{M}}$ is optimal

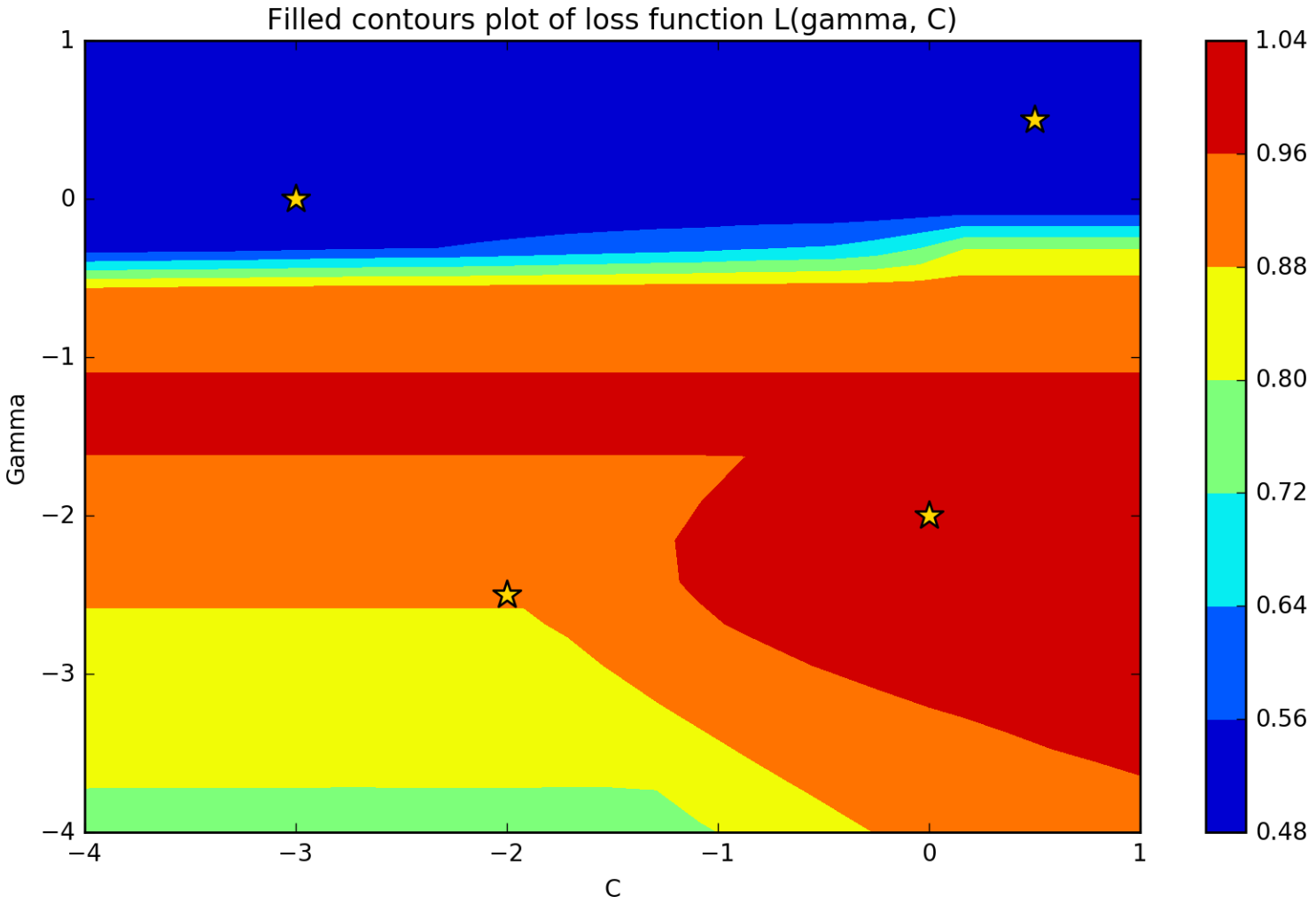


Manual grid search works for low-dimensional problems, but does not scale well to higher dimensions

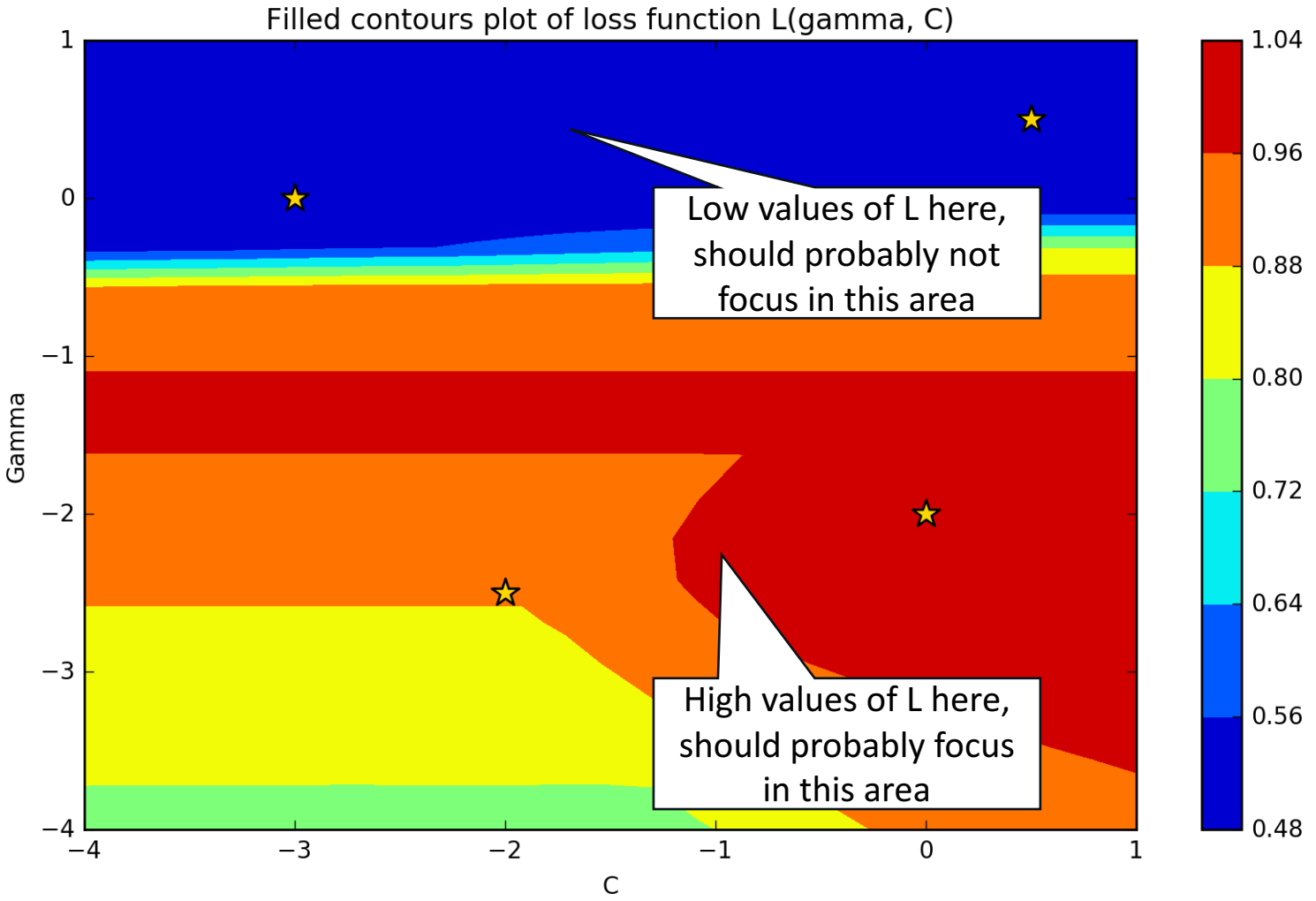
Curse of dimensionality makes this inefficient in higher dimensional problems



Random grid search works better in higher dimensions, but..



.. shouldn't previously evaluated hyperparameter values guide us in the search for the true optimal value?



Bayesian optimization

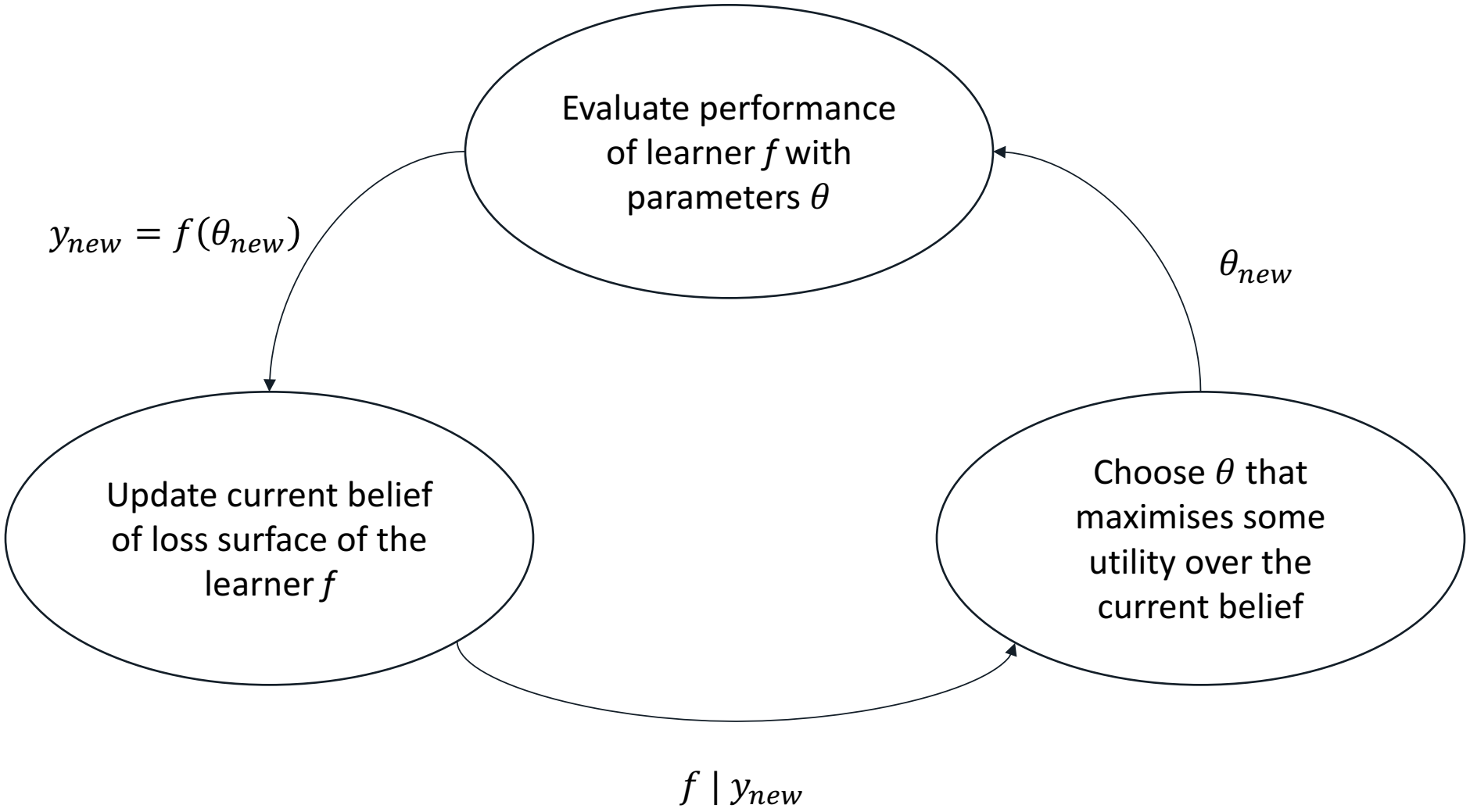
Bayesian optimization is only beneficial in certain situations

In Bayesian optimization, we are building a probabilistic model for the performance metric $f_{\mathcal{M}}(\theta)$. This introduces computational overhead.

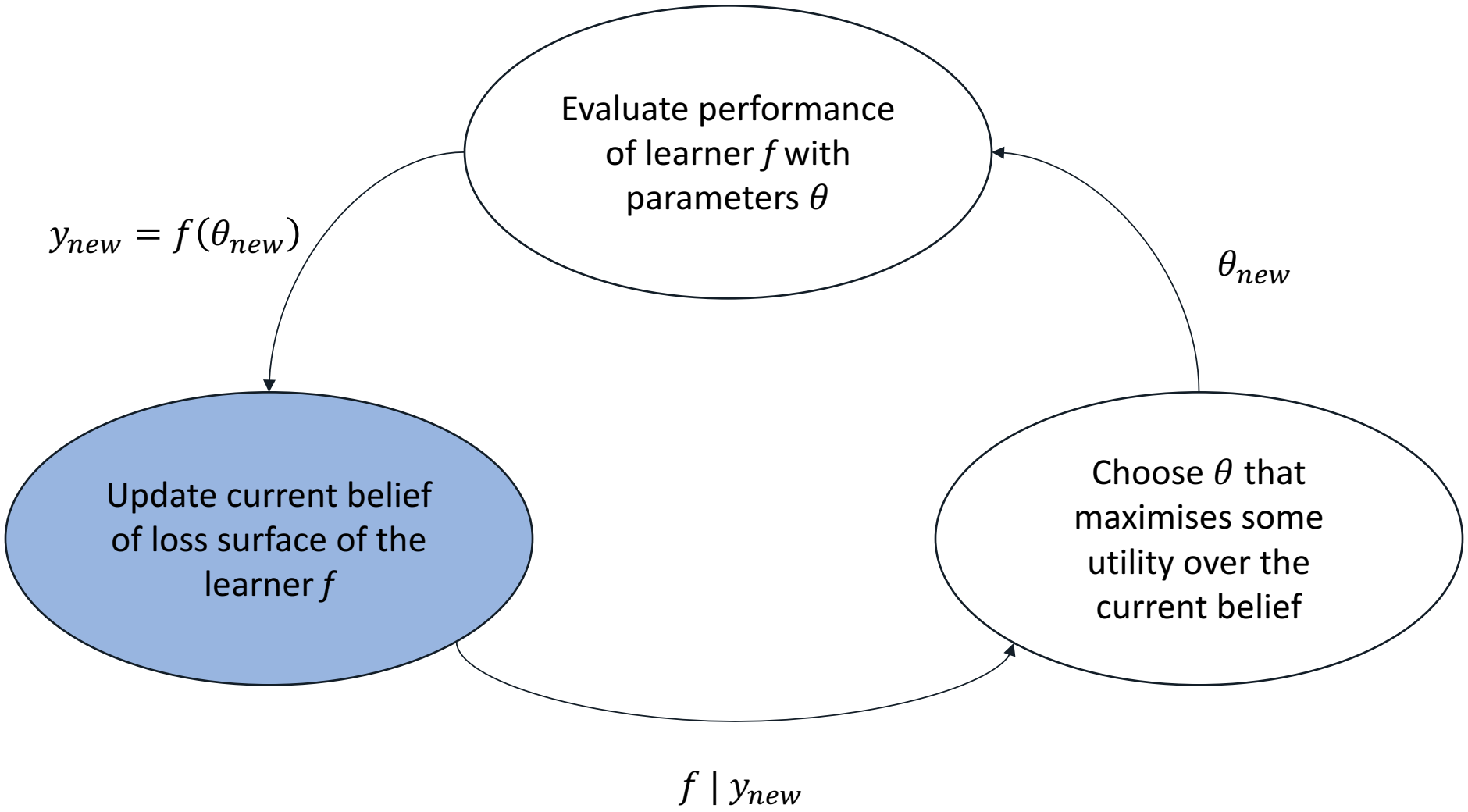
As a result, applying Bayesian optimization only really makes sense if

- The number of hyperparameters is very high (θ is of high dimension); or
- It is computationally very expensive to evaluate $f_{\mathcal{M}}(\theta)$ for a single point θ.

The classic Bayesian optimization algorithm consists of three steps



The classic Bayesian optimization algorithm consists of three steps



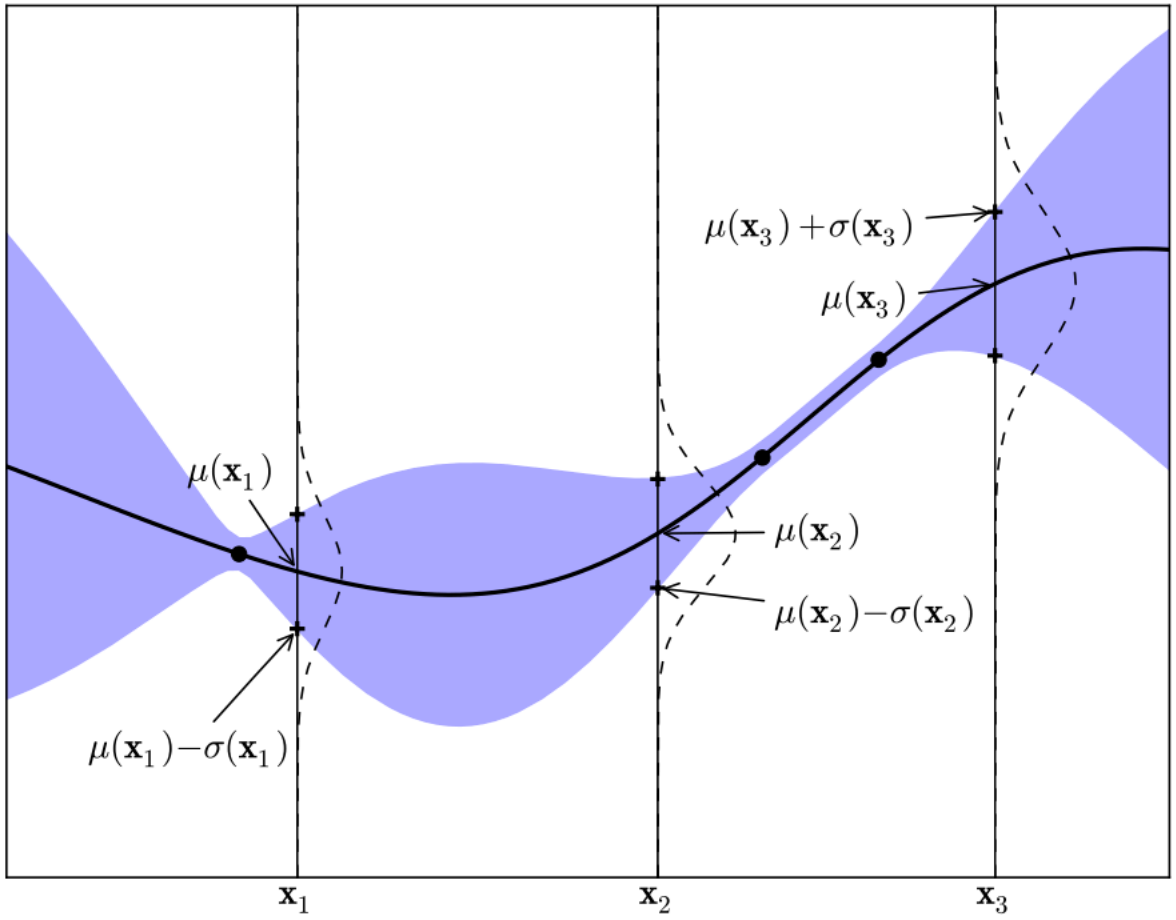
The performance function $f_{\mathcal{M}}$ is modelled using Gaussian Processes (GPs)

- GPs are the generalization of a Gaussian distribution to a distribution over *functions*, instead of random variables

- Just as a Gaussian distribution is completely specified by its mean and variance, a GP is completely specified by its **mean function** and **covariance function**.

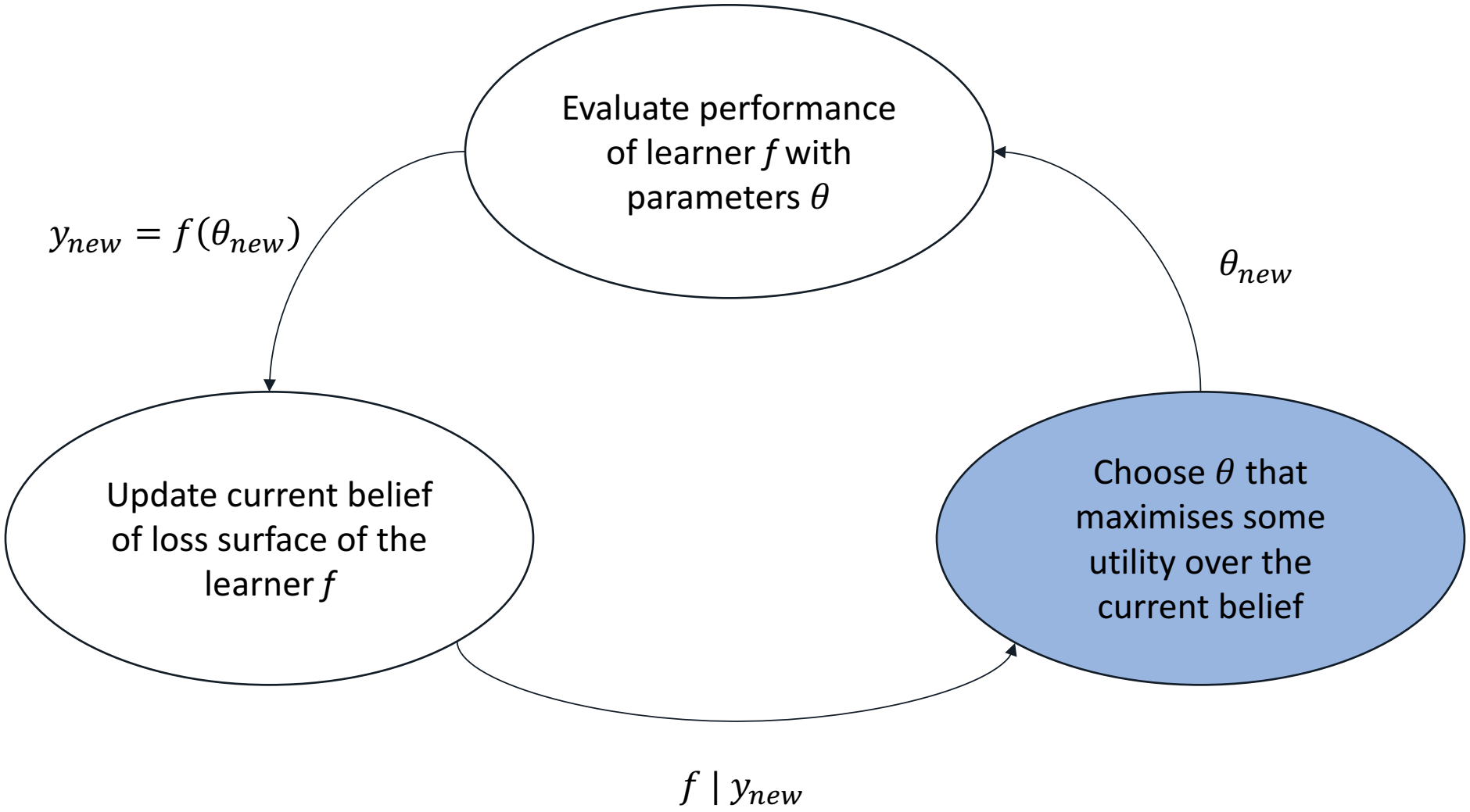
- We can think of a GP as a function that, instead of returning a scalar $f(\boldsymbol{x})$, returns the mean and variance of a normal distribution over the possible values of f at $\boldsymbol{x}$.

GPs are the generalization of a Gaussian distribution to a distribution over *functions*, instead of random variables[1]

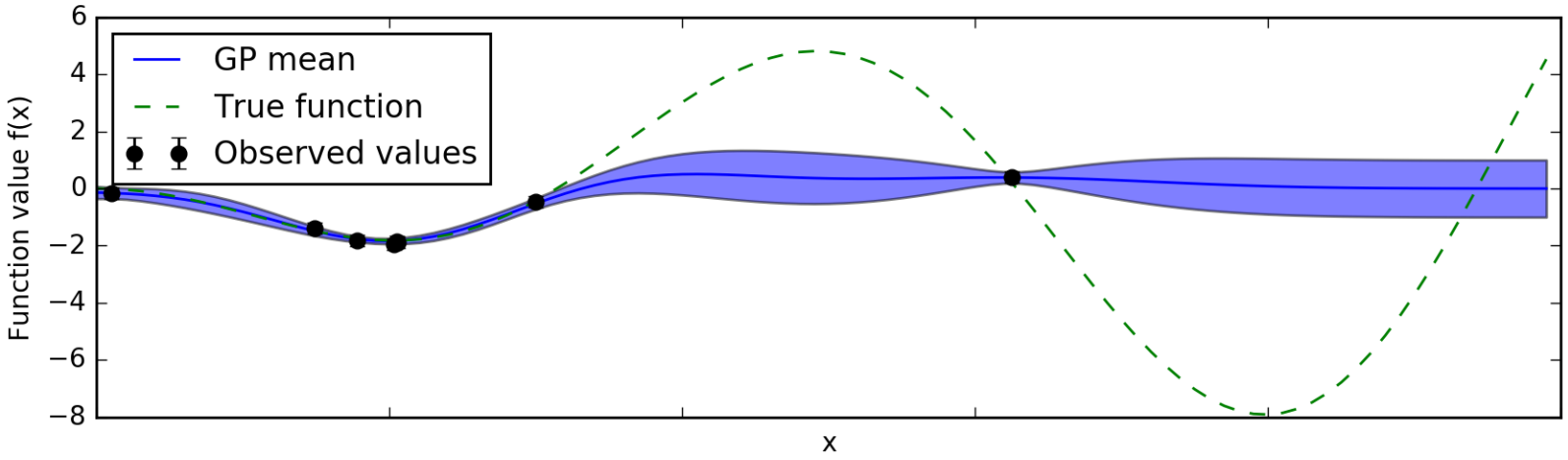


[1]A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning, *Eric Brochu, Vlad M. Cora, Nando de Freitas*, 2010. https://arxiv.org/abs/1012.2599

The classic Bayesian optimization algorithm consists of three steps



How do we use our current belief to make a good guess about what point to evaluate next?



Acquisition functions are used to formalize what constitutes a ”best guess”

$$EI(\theta) = \mathbb{E}[\max_{\theta}\{0, f_{\mathcal{M}}(\theta) - f_{\mathcal{M}}(\hat{\theta})\}],$$

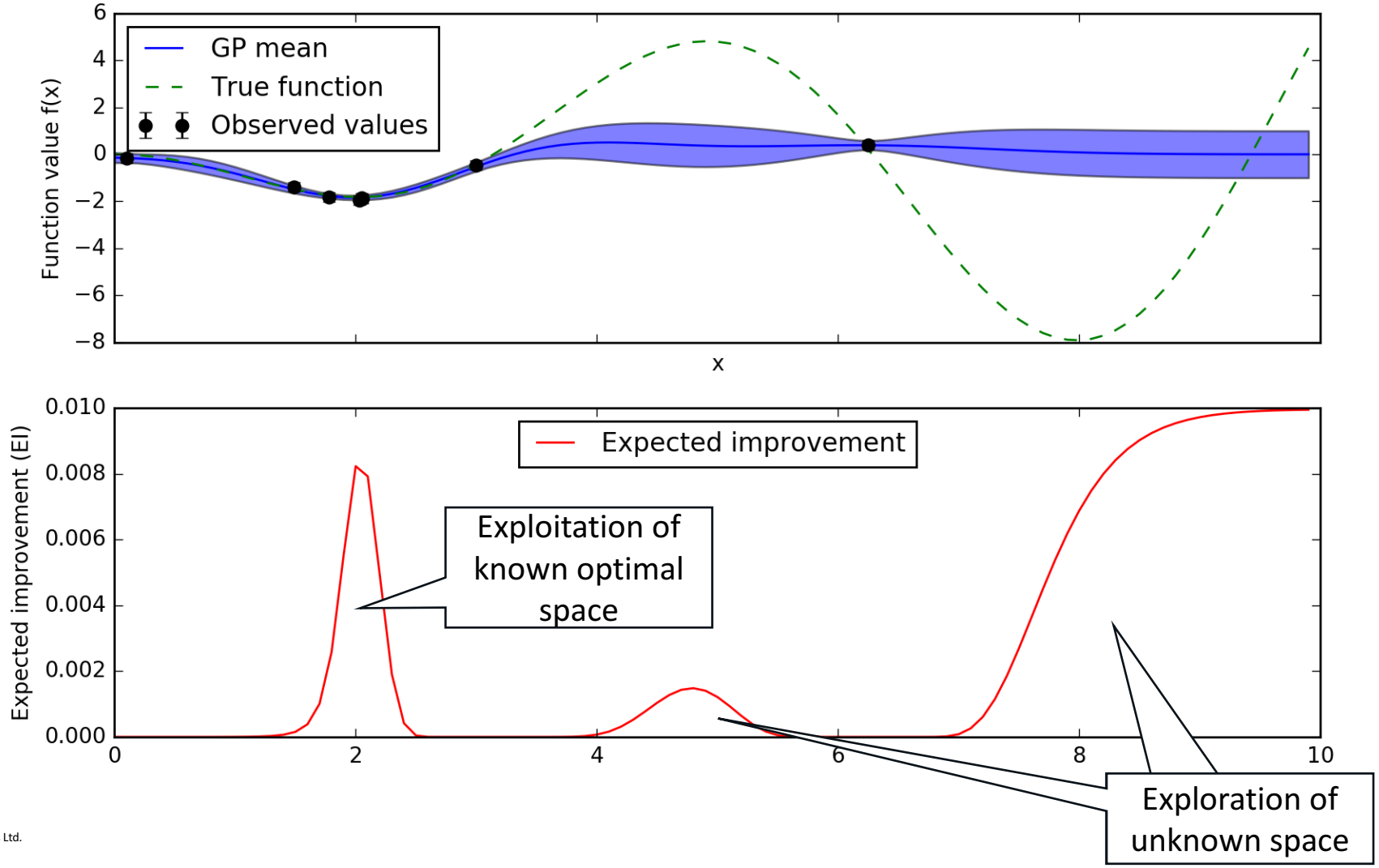
$$\theta_{new} = \operatorname{argmax}_{\theta} EI(\theta)$$

The expected improvement acquisition function can be evaluated analytically

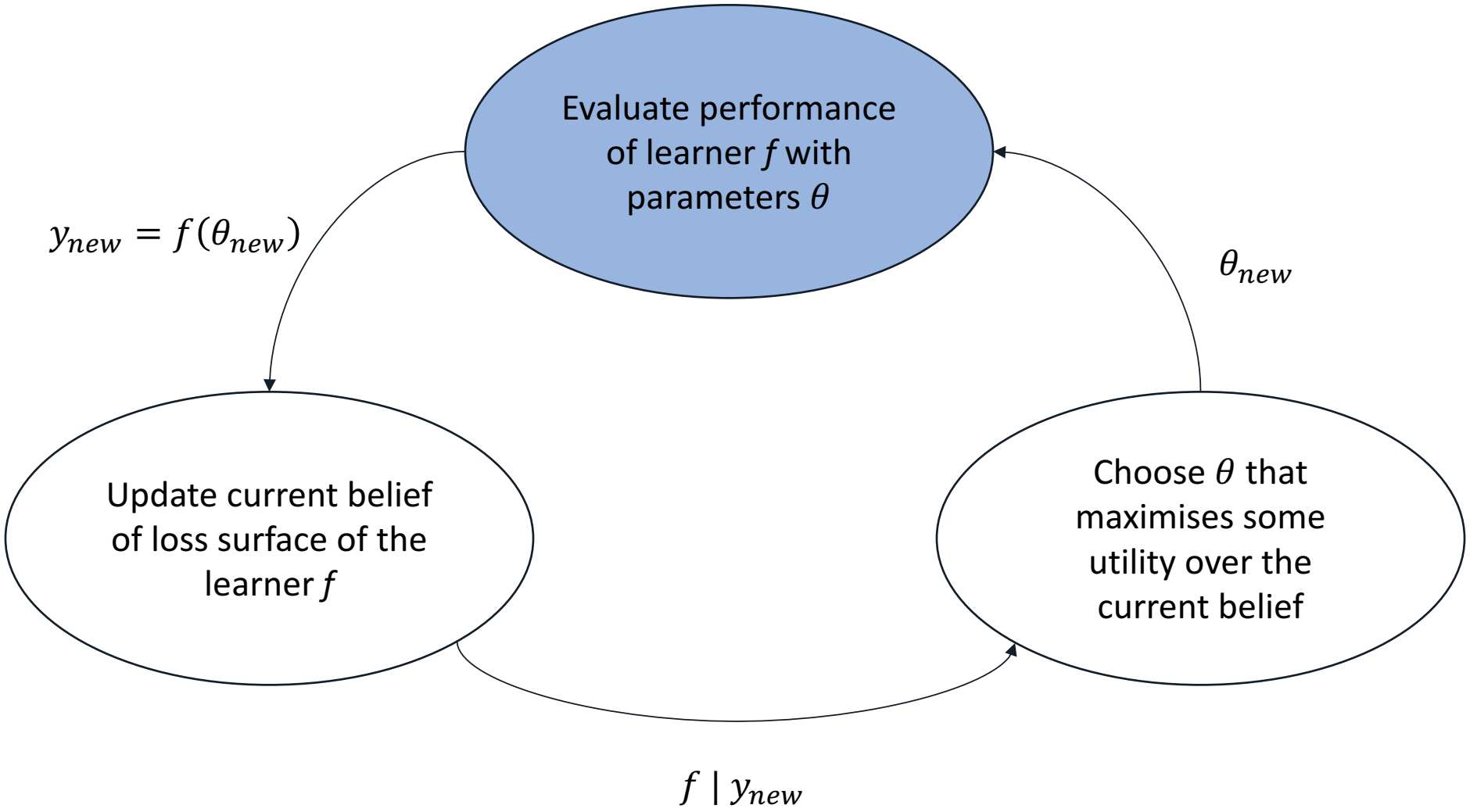
$$EI(\theta) = \begin{cases} \left(\mu(\theta) - f\left(\hat{\theta}\right)\right)\Phi(Z) + \sigma(\theta)\phi(Z), & \sigma(\theta) > 0 \\ 0, & \sigma(\theta) = 0 \end{cases},$$

$$Z = \frac{\mu(\theta) - f(\hat{\theta})}{\sigma(\theta)}$$

The expected improvement acquisition trades off *exploitation* of known optimal areas, versus *exploration* of unexplored areas of the loss surface



The classic Bayesian optimization algorithm consists of three steps



How do we use this in real-life?

```
import sklearn.gaussian_process as gp
```

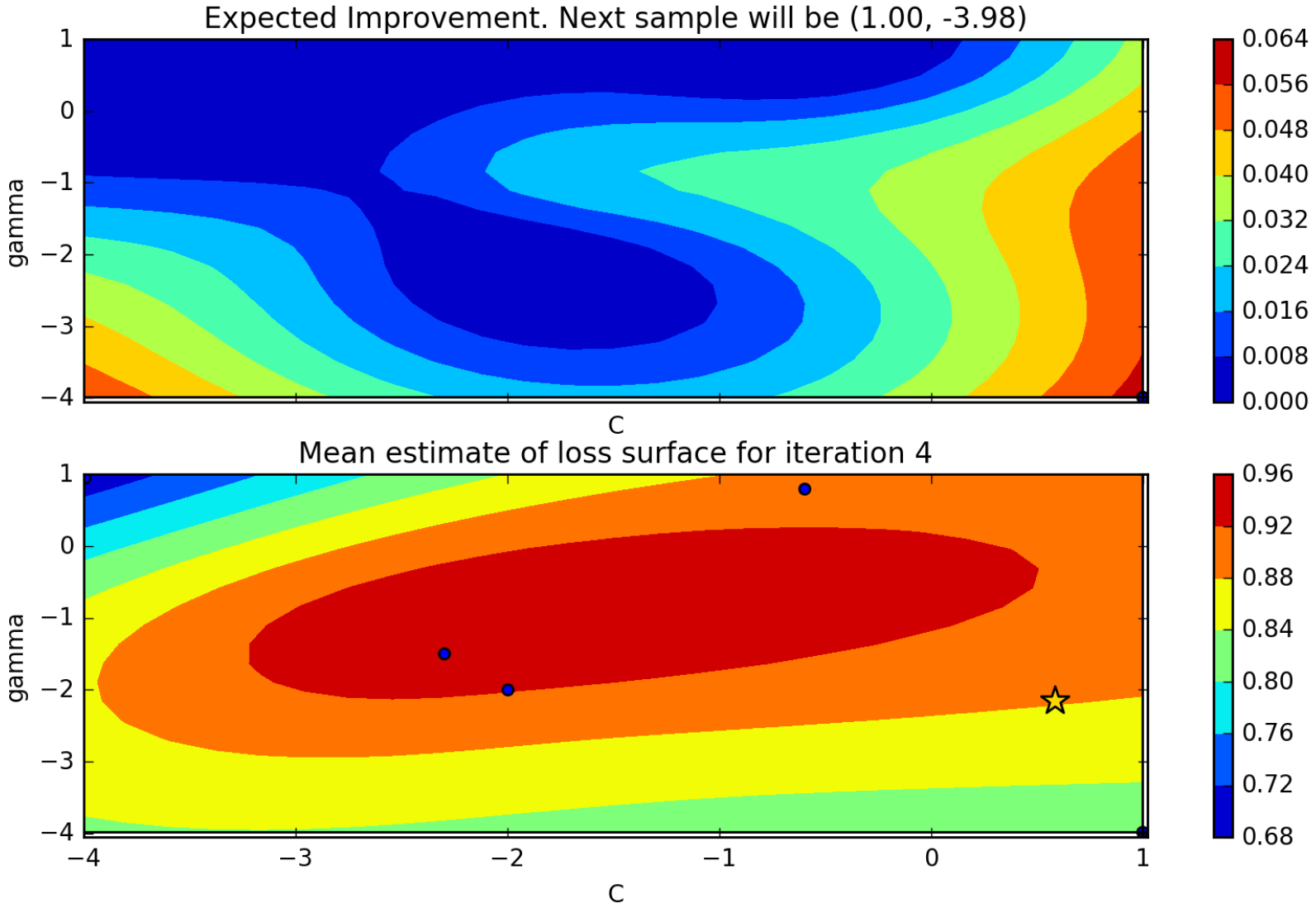
Example – Bayesian optimization can be used to tune the hyperparameters of an SVM model

```
def sample_loss(X, y, params, random_state):
    """ Sample loss of an SVM model """

    # C and gamma are on the log scale
    return cross_val_score(SVC(C=10 ** params[0],
                               gamma=10 ** params[1],
                               random_state=random_state),
                           X=X,
                           y=y,
                           scoring='roc_auc',
                           cv=3).mean()
```

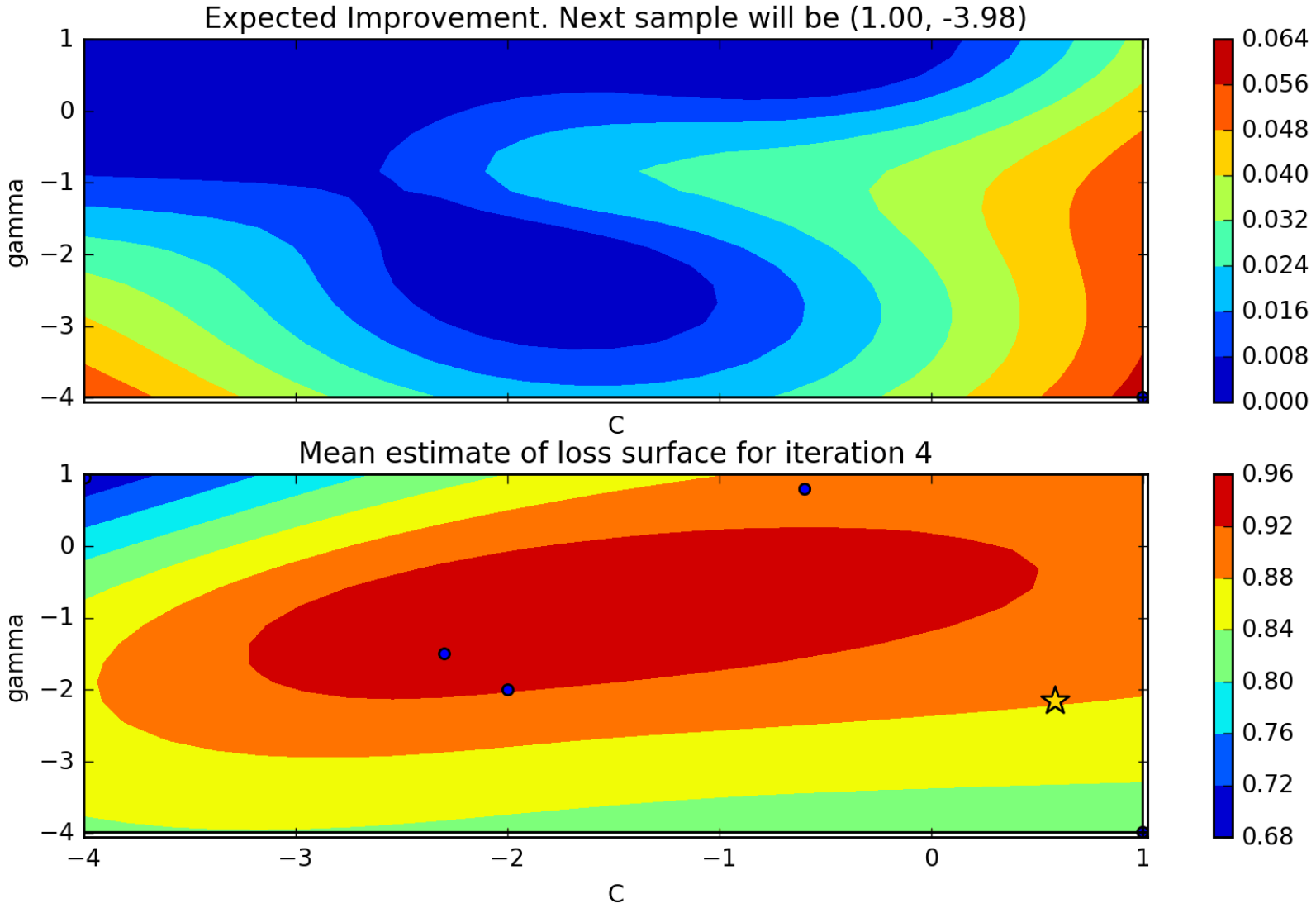
Example – A pseudocode implementation of Bayesian optimization shows its simple API

```
def bayesian_optimisation(n_iters, sample_loss, xp, yp):
    kernel = gp.kernels.Matern()
    model = gp.GaussianProcessRegressor(kernel=kernel,
                                        alpha=1e-4,
                                        n_restarts_optimizer=10,
                                        normalize_y=True)
    for i in range(n_iters):
        # Update our belief of the loss function
        model.fit(xp, yp)

        # sample_next_hyperparameter is a method that computes the argmax
        # of the acquisition function
        next_theta = sample_next_hyperparameter(model, yp)

        # Evaluate the loss for the new hyperparameters
        next_loss = sample_loss(next_sample)

        # Update xp and yp
```

Example – Bayesian optimization can be used to tune the hyperparameters of an SVM model



Example – Bayesian optimization can be used to tune the hyperparameters of an SVM model



GPs can not be used as a simple black-box, and some care must be taken

1. **Choose an appropriate scale for your hyperparameters:** For parameters like a learning rate, or regularization term, it makes more sense to sample on the log-uniform domain, instead of the uniform domain.

2. **Choose the kernel of the GP carefully:** Each kernel implicitly assumes different properties on the loss function, in terms of differentiability and periodicity.

Multiple production-ready, and open-source, modules for Bayesian optimization are available online

1. Spearmint
2. Hyperopt
3. MOE
4. Hyperband (model-free)
5. SMAC (model-free)