Bayesian Optimization in Machine Learning

José Jiménez

Master's Thesis
Master's degree in Statistics and Operations Research

supervised by Josep Ginebra[1]
[1] Department of Statistics and Operations Research. ETSEIB.

UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

UNIVERSITAT DE
BARCELONA

Goals of this thesis

This Master's thesis has different and complimentary aims:

- Provide an introduction to both Gaussian Process regression and Bayesian optimization.
- Show that the Bayesian Optimization framework works in several real-world machine learning tasks.
- Write a complete software package (pyGPGO) for users to apply Bayesian Optimization in their research.

Organization of the work

Organized in 4 self-contained chapters.

- **Chapter 2** focuses on an introduction to regression problems using Gaussian Processes. These are surrogate models we will use for Bayesian Optimization.
- **Chapter 3** covers the main topic in this work, Bayesian Optimization.
- **Chapter 4** presents experiments using the Bayesian Optimization framework. Mostly mid-sized supervised-learning problems.
- **Chapter 5** provides technical explanations for pyGPGO, the software developed alongside this manual.

A brief introduction

Overall, Bayesian Optimization focuses on:

$$\max_{\boldsymbol{x} \in \mathcal{A}} f(x) \tag{1}$$

We make almost no assumptions about f:

- f may not have a closed-form expression.
- Evaluations of f may be noisy.
- No gradient information needed.

An example arises when optimizing the *loss* function of a machine-learning model, depending on its hyperparameters (e.g. log-loss):

$$\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\frac{1}{n} \sum_i \left(y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \right) \tag{2}$$

Gaussian Process Regression: basic definitions

Definition 1.

A Gaussian Process is a collection of random variables, any finite number of which have a joint Gaussian distribution. This process is defined by two functions. Its *mean function*:

$$m(\boldsymbol{x}) = \mathbb{E}\left[f(\boldsymbol{x})\right] \tag{3}$$

and its *covariance function*:

$$k(\boldsymbol{x}, \boldsymbol{x'}) = \mathbb{E}\left[\left(f(\boldsymbol{x}) - m(\boldsymbol{x})\right)\left(f(\boldsymbol{x'}) - m(\boldsymbol{x'})\right)\right] \tag{4}$$

We say that f is a Gaussian Process with mean $m(\boldsymbol{x})$ and covariance function $k(\boldsymbol{x}, \boldsymbol{x'})$ and write:

$$f(\boldsymbol{x}) \sim \mathcal{GP}\left(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x'})\right) \tag{5}$$

Gaussian Process Regression: basic definitions

Define then a covariance function, such as the *squared exponential* kernel:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{1}{2}|\boldsymbol{x} - \boldsymbol{x}'|^2\right) \tag{6}$$

where $|.|$ denotes the standard L_2 norm. Drawing samples from a Gaussian Process, assuming zero mean, for given finite inputs X_* simplifies to sampling from:

$$\boldsymbol{f}_* \sim \mathcal{N}(\mathbf{0}, K(X_*, X_*)) \tag{7}$$

Gaussian Process Regression: prediction

Assume training data $\mathcal{D} = \{(\boldsymbol{x_i}, y_i) \,|\, i = 1, \ldots, n\}$

Prediction using GP prior

Let $\boldsymbol{y}$ and $\boldsymbol{f_*}$ be jointly Gaussian:

$$\begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{f_*} \end{bmatrix} \sim \mathcal{N}\left(\boldsymbol{0}, \begin{bmatrix} K(X,X) + \sigma_n^2 I & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*) \end{bmatrix}\right) \tag{8}$$

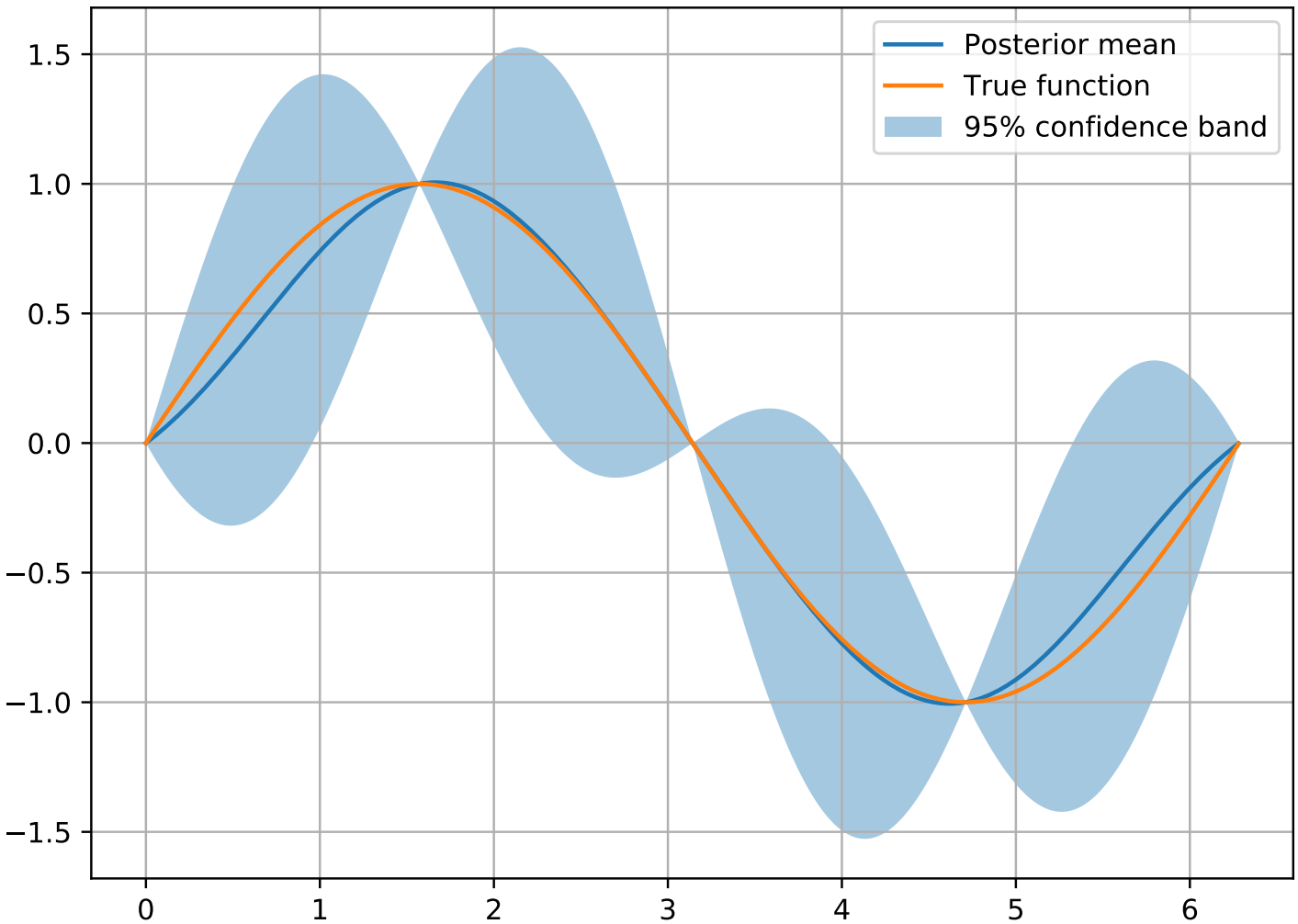
We want to condition $\boldsymbol{f_*}$ over $\boldsymbol{y}$.

$$\boldsymbol{f_*} | \boldsymbol{y} \sim \mathcal{N}(\overline{\boldsymbol{f_*}}, Cov(\boldsymbol{f_*})) \tag{9}$$

where:

$$\begin{aligned} \overline{\boldsymbol{f_*}} &= K(X_*, X)\left(K(X,X) + \sigma_n^2 I\right)^{-1} \boldsymbol{y} \\ Cov(\boldsymbol{f_*}) &= K(X_*, X_*) - K(X_*, X)\left(K(X,X) + \sigma_n^2 I\right)^{-1} K(X, X_*) \end{aligned} \tag{10}$$

Gaussian Process Regression: an example



Gaussian Process Regression: on covariance functions

Some common covariance function choices

$$k_{SE}(r) = \exp\left(-\frac{r^2}{2l^2}\right) \qquad k_{\text{Matèrn}}(r) = \frac{2^{1-\nu}}{\Gamma(\nu)}\left(\frac{\sqrt{2\nu}r}{l}\right)^{\nu} K_{\nu}\left(\frac{\sqrt{2\nu}r}{l}\right)$$

$$k_{\text{GE}}(r) = \exp\left(-\left(\frac{r}{l}\right)^{\gamma}\right) \qquad k_{RQ}(r) = \left(1 + \frac{r^2}{2\alpha l^2}\right)^{-\alpha} \tag{11}$$

where $r = |\boldsymbol{x} - \boldsymbol{x'}|$. Observations are noisy:

$$k^{y}(\boldsymbol{x}_p, \boldsymbol{x}_q) = \sigma_f^2 k(\boldsymbol{x}_p, \boldsymbol{x}_q) + \sigma_n^2 \delta_{pq} \tag{12}$$

Gaussian Process Regression: Type II Maximum-Likelihood

An empirical Bayes approach to choosing hyperparameters. Noticing that $\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{0}, K + \sigma_n^2 I)$

Marginal log-likelihood

$$\log p(\boldsymbol{y}|X) = -\frac{1}{2}\boldsymbol{y}^T(K + \sigma_n^2 I)^{-1}\boldsymbol{y} - \frac{1}{2}\log|K + \sigma_n^2 I| - \frac{n}{2}\log 2\pi \quad (13)$$

Derivative w.r.t hyperpameters

$$\frac{\partial}{\partial \theta_j}\log p(\boldsymbol{y}|X, \boldsymbol{\theta}) = \frac{1}{2}\boldsymbol{y}^T K^{-1}\frac{\partial K}{\partial \theta_j}K^{-1}\boldsymbol{y} - \frac{1}{2}\mathrm{tr}\left(K^{-1}\frac{\partial K}{\partial \theta_j}\right) \quad (14)$$

Gaussian Process Regression: incorporating diff. priors

We have assumed $m(\boldsymbol{x}) = 0$ for simplicity. If we dispose of prior knowledge, we can use it

Different prior mean

$$f(\boldsymbol{x}) \sim \mathcal{GP}\left(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}^*)\right) \tag{15}$$

Posterior mean becomes:

$$\boldsymbol{f}_* = \boldsymbol{m}(X_*) + k(X_*, X)K^{-1}\left(\boldsymbol{y} - \boldsymbol{m}(X)\right) \tag{16}$$

Posterior variance remains unchanged.

Gaussian Process Regression: marginalizing over hyperparameters

The full Bayesian approach does not optimize the marginal likelihood, but integrates the uncertainty of hyperparameters θ into the model, either using MCMC or Variational Inference techniques.

Full Bayesian approach

$$\theta \sim p_h(\theta) \tag{17}$$

$$\boldsymbol{f} \sim \mathcal{N}(0, \Sigma_\theta) \tag{18}$$

$$\mathcal{L}(\boldsymbol{f}) = p(\text{data}|\boldsymbol{f}) = p(\boldsymbol{y}|\boldsymbol{f}) \tag{19}$$

We wish to sample from the joint posterior under unknowns:

$$p(\boldsymbol{f}, \theta|\,\text{data}) \propto \mathcal{L}(\boldsymbol{f}) p(\boldsymbol{f}) p_h(\theta) \tag{20}$$

Several strategies for sampling from latent Gaussian models proposed in main text.

Bayesian Optimization: introduction

Again, assume:

$$\max_{\boldsymbol{x} \in \mathcal{A}} f(\boldsymbol{x}) \tag{21}$$

Assume that we have sampled our function f to optimize a small number of times n. Providing us with training data:

$$\mathcal{D}_n = \{\boldsymbol{x}_i, y_i, i = 1, \ldots, n\}. \tag{22}$$

Our steps here are:

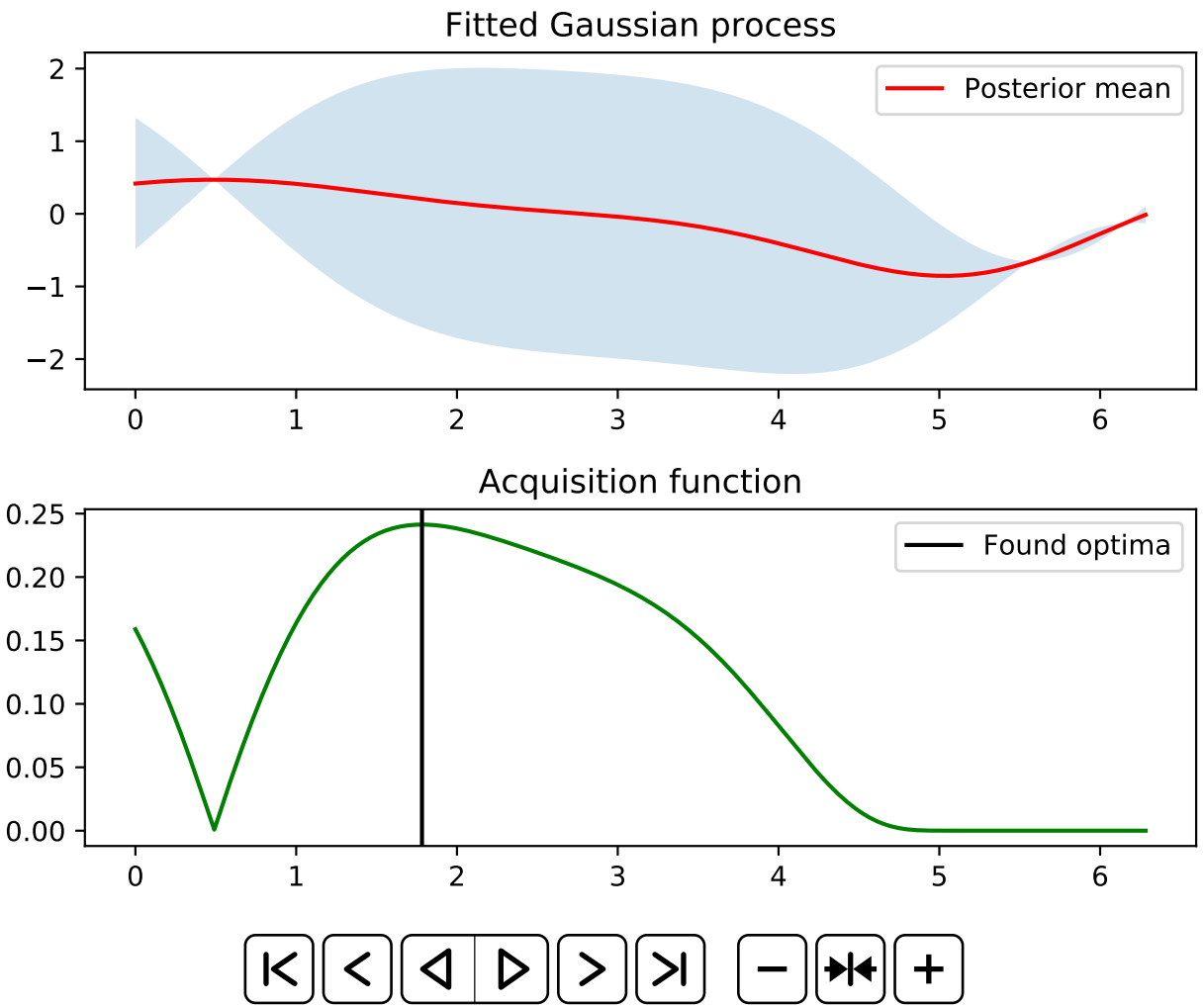
1. Fit a Gaussian Process regression model on $\mathcal{D}_n$.
2. Choose the next point to sample, according to an *acquisition function*, depending on GP.
3. Evaluate said point. Augment training data. Repeat.

Bayesian optimization: algorithm

Algorithm 1 Bayesian optimization framework

1: Sample a small number of points $\boldsymbol{x} \in \mathcal{A}$. Evaluate $f(\boldsymbol{x})$ to get $\mathcal{D}_n$
2: **for** $n = 1, 2, \ldots$ **do**
3: Fit a GP regression model on $\mathcal{D}_n$
4: $\boldsymbol{x}_{n+1} \leftarrow \arg\max_{\boldsymbol{x}} \alpha(\boldsymbol{x}, \mathcal{D}_n)$
5: Evaluate $f(\boldsymbol{x}_{n+1}) = y_{n+1}$
6: Augment data $\mathcal{D}_{n+1} = \{\mathcal{D}_n, (\boldsymbol{x}_{n+1}, y_{n+1})\}$
7: **end for**

Bayesian optimization: example



Bayesian optimization: on acquisition functions

Three popular categories: improvement-based, optimistic, and information-based policies.

Improvement-based

τ denotes our best evaluation so far.
Probability of improvement:

$$\alpha_{\mathrm{PI}}(\boldsymbol{x}, \mathcal{D}_n) = P(\nu > \tau) = \Phi\left(\frac{\mu_n(\boldsymbol{x}) - \tau}{\sigma_n(\boldsymbol{x})}\right) \tag{23}$$

Expected improvement:

$$\alpha_{\mathrm{EI}}(\boldsymbol{x}, \mathcal{D}_n) = (\mu_n(\boldsymbol{x}) - \tau)\Phi\left(\frac{\mu_n(\boldsymbol{x}) - \tau}{\sigma_n(\boldsymbol{x})}\right) + \sigma_n(\boldsymbol{x})\phi\left(\frac{\mu_n(\boldsymbol{x}) - \tau}{\sigma_n(\boldsymbol{x})}\right) \tag{24}$$

Bayesian optimization: on acquisition functions

Optimistic acquisitions

Upper confidence bound:

$$\alpha_{\mathrm{UCB}}(\boldsymbol{x}, \mathcal{D}_n) = \mu_n(\boldsymbol{x}) + \beta_n \sigma_n(\boldsymbol{x}) \tag{25}$$

Information-based

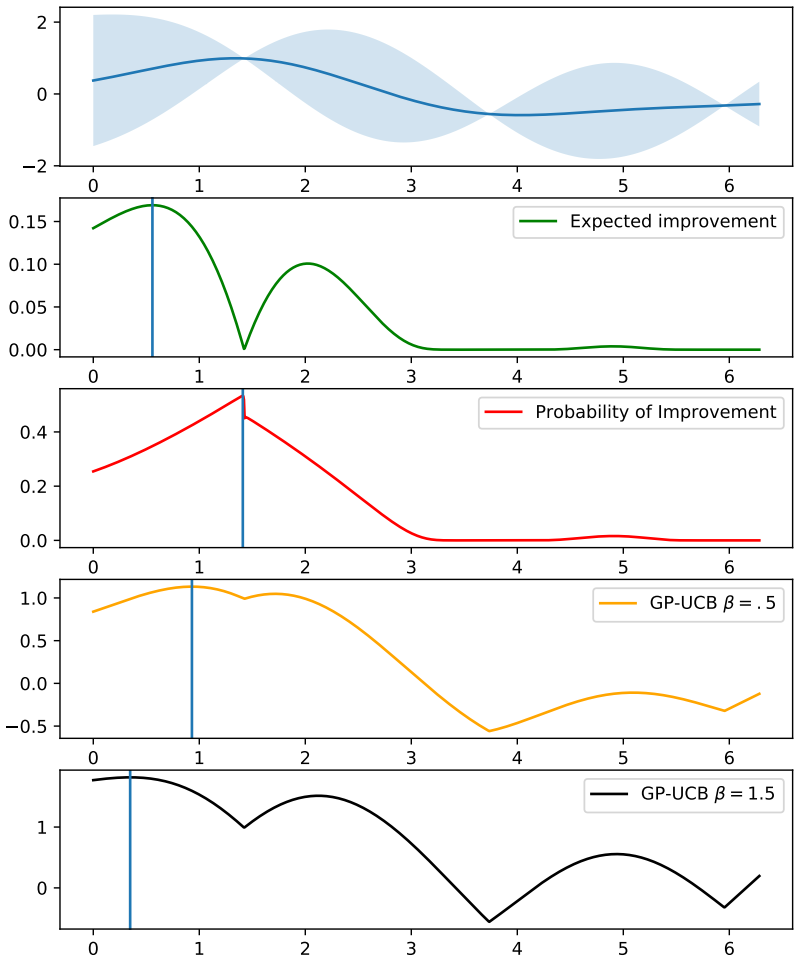
Entropy-based:

$$\alpha_{\mathrm{ES}}(\boldsymbol{x}|\mathcal{D}_n) = H(\boldsymbol{x}^*|\mathcal{D}_n) - \mathbb{E}_{y|\mathcal{D}_n,\boldsymbol{x}}\left[H(\boldsymbol{x}^*|\mathcal{D}_n \cup \{(\boldsymbol{x}, y)\})\right] \tag{26}$$

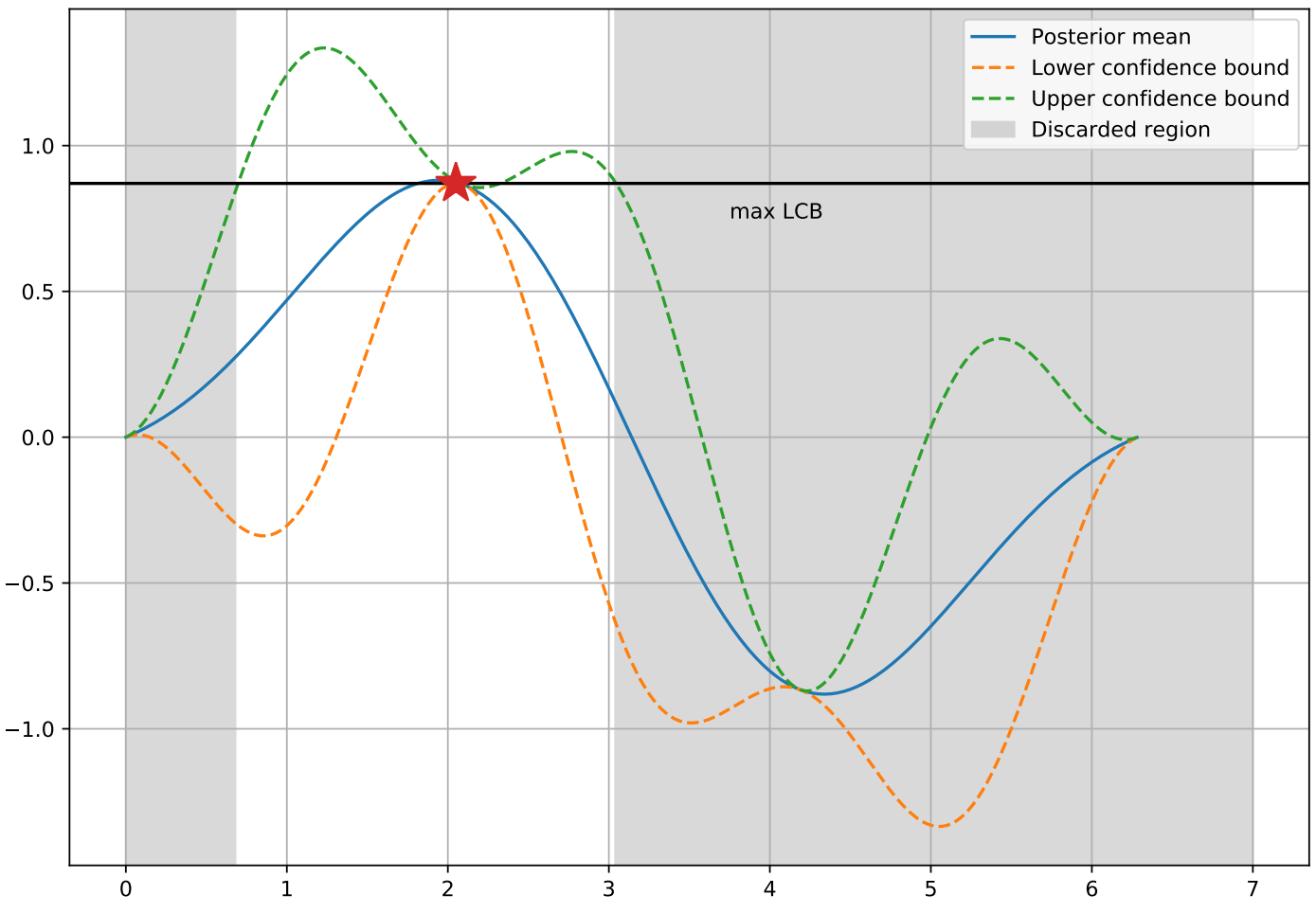
Predictive entropy search:

$$\alpha_{\mathrm{PES}}(\boldsymbol{x}, \mathcal{D}_n) = H(y|\mathcal{D}_n, \boldsymbol{x}) - \mathbb{E}_{\boldsymbol{x}^*|\mathcal{D}_n}\left[H(y|\mathcal{D}_n, \boldsymbol{x}, \boldsymbol{x}^*)\right] \tag{27}$$

Bayesian optimization: acquisition function behaviour



Bayesian optimization: why does it work?



Bayesian optimization: role of hyperparameters

GP hyperparameters θ cause uncertainty in the optimization procedure:

$$\alpha(\boldsymbol{x}) = \mathbb{E}_{\theta|\mathcal{D}_n}\left[\alpha(\boldsymbol{x}, \theta)\right] = \int_{\Theta} \alpha(\boldsymbol{x}|\theta) p(\theta|\mathcal{D}_n) d\theta. \tag{28}$$

Approach 1. MAP estimate $\hat{\theta}_{\mathrm{MAP}}$

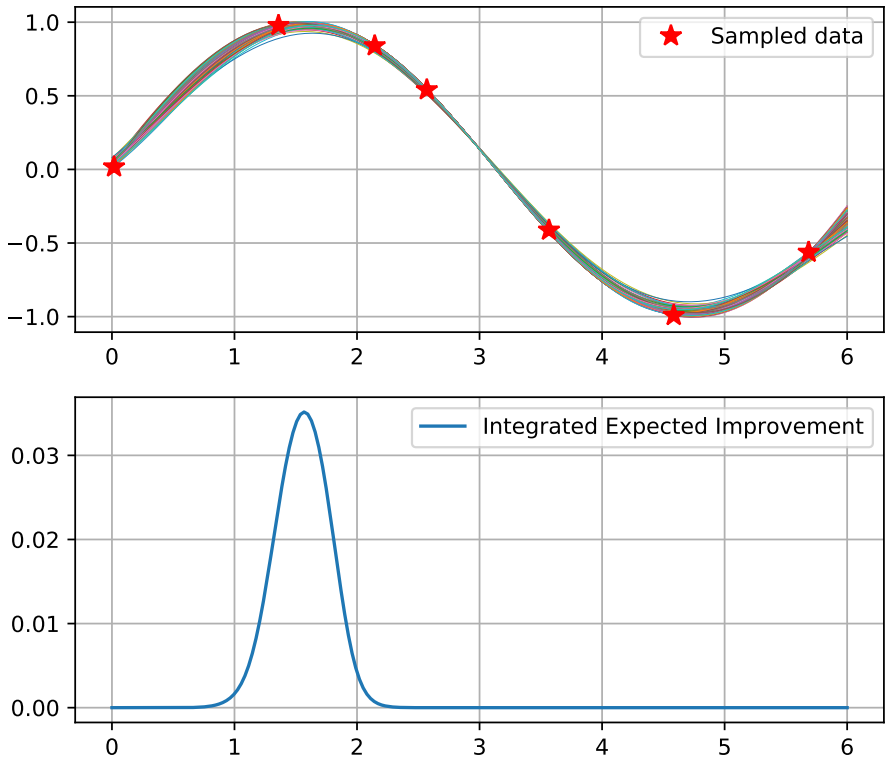
$$\hat{\alpha}(\boldsymbol{x}) = \alpha(\boldsymbol{x}, \hat{\theta}) \tag{29}$$

Approach 2. Sample posterior $p(\theta|\mathcal{D}_n)$. Integrated acquisition functions

$$\mathbb{E}_{\theta|\mathcal{D}_n}\left[\alpha(\boldsymbol{x}, \theta)\right] \approx \frac{1}{M} \sum_{i=1}^{M} \alpha(\boldsymbol{x}, \theta_n^{(i)}) \tag{30}$$

Bayesian optimization: Integrated acquisition example

Figure: Means of 200 posterior predictive distributions, taken from sampled posterior $p(\theta|\mathcal{D}_n)$



Bayesian optimization: optimizing acquisitions

We have assumed that the acquisition function is easily optimized. This is not necessarily the case:

- It is often multimodal.
- Or non-convex.
- Furthermore theoretical convergence is guaranteed when finding true maximum $\boldsymbol{x}^*$.

One may be thinking that we have switched optimization problems! (f by α). However, α is very cheap to compute.

- In practice α is optimized using multi-start quasi-Newton methods (e.g. L-BFGS-B) or evolutionary approaches (CMA-ES).

Bayesian optimization: Other surrogates

- Random Forests: empirical estimate of variance provided by predictions of trees.
- Gradient Boosting Machines: using quantile regression for variance estimate.
- Sparse pseudo-input GPs
- Student-t Processes
- ...

Student-t Process example

$$\boldsymbol{f}^* | \boldsymbol{f} \sim \mathcal{T}\left(\nu + n, \phi_2, \frac{\nu + \beta_1 - 2}{\nu + n - 2}\tilde{K}_{**}\right) \tag{31}$$

Experiments

The aim is to demonstrate that Bayesian Optimization works in real hyperparameter optimization problems.

- Mid-sized supervised learning problems. Both regression and classification considered.
- Two examples explained in detail: datasets retrieved from computational chemistry problems.
- Other examples drawn from other areas, such as medicine. Available from other sources (UCI ML repository).

Experiments: benchmarking rules

Loss functions used:

Binary classification

Logarithmic loss:

$$\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\frac{1}{n} \sum_i \left(y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \right) \tag{32}$$

Regression

Mean Squared Error (MSE)

$$\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{n} \sum_i (y_i - \hat{y}_i)^2 \tag{33}$$

Experiments: benchmarking rules

- Shuffled $k = 5$ fold cross-validation scheme for loss evaluation.
- Features scaled: zero mean, unit variance.
- $n = 50$ ($+3$ to account GP fitting) total function evaluations.
- Gaussian Process surrogate, with squared exponential covariance function.
- Type II ML estimation of all hyperparameters at each stage (l, σ_n^2, σ_f^2).
- Expected Improvement and GP-UCB ($\beta = 0.5$, $\beta = 1.5$) acquisitions.
- Machine-learning models used: Support Vector Machines, K-Nearest-Neighbours, Gradient Boosting Machines and One-Hidden Layer Neural Networks.

Experiments: benchmarking rules

Table: SVM parameters.

Parameter	**Type**	**Bounds**
C	$\mathbb{R}^+$	$\left[10^{-5}, 10^5\right]$ (log-scaled)
γ	$\mathbb{R}^+$	$\left[10^{-5}, 10^5\right]$ (log-scaled)

Table: KNN parameters.

Parameter	**Type**	**Bounds**
k	Integer	$\{10, \ldots, 50\}$

Experiments: benchmarking rules

Table: GBM parameters.

Parameter	**Type**	**Bounds**
`learning_rate`	$\mathbb{R}^+$	$[10^{-5}, 10^{-2}]$
`n_estimators`	Integer	$\{10, \ldots, 100\}$
`max_depth`	Integer	$\{2, \ldots, 100\}$
`min_samples_split`	Integer	$\{2, \ldots, 100\}$

Table: MLP parameters.

Parameter	**Type**	**Bounds**
`hidden_layer_size`	Integer	$[5, 50]$
`alpha`	$\mathbb{R}^+$	$[0, 0.9]$

Experiments: Binding affinity prediction dataset

- Problem is to predict some binding/inhibition constant between a protein and a ligand (K_d, K_i). Target variable is defined as $y = -\log_{10} K$

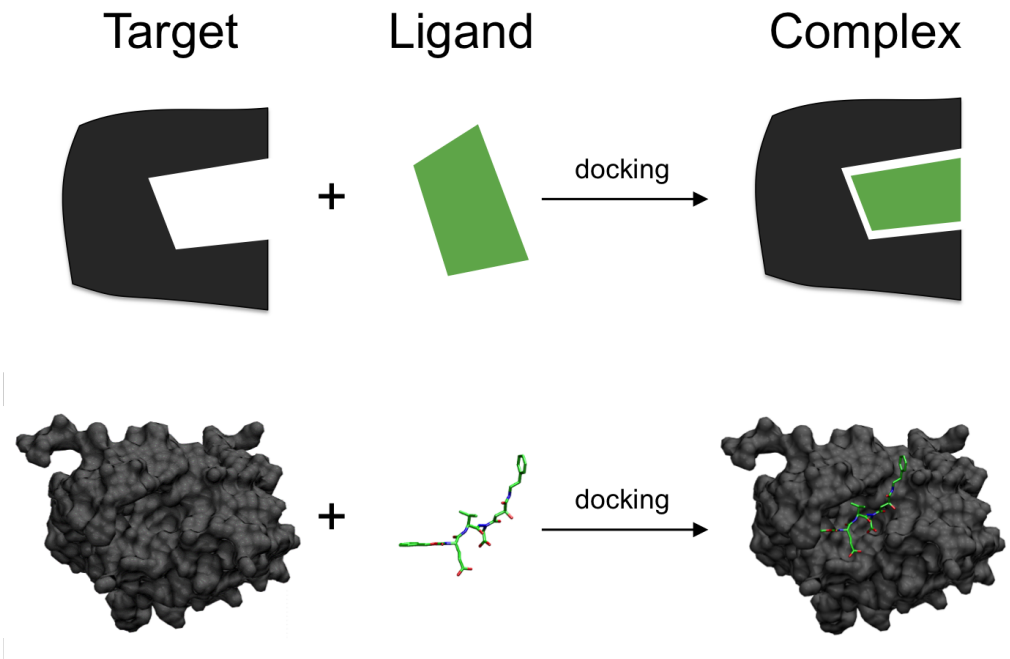


Figure: Illustration of the protein-ligand binding scenario.

Experiments: Binding affinity prediction dataset

Set of features used

Pair of atom-type counts from protein and ligand in the binding site.

$$\{P_j\}_{j=1}^{9} = \{C, N, O, F, P, S, Cl, Br, I\}$$
$$\{L_i\}_{i=1}^{9} = \{C, N, O, F, P, S, Cl, Br, I\}$$

$$\boldsymbol{x}\left(Z(P_j), Z(L_i)\right) = \sum_{k=1}^{P_j} \sum_{l=1}^{L_i} \Theta(d_{\text{cutoff}} - d_{kl}) \tag{34}$$

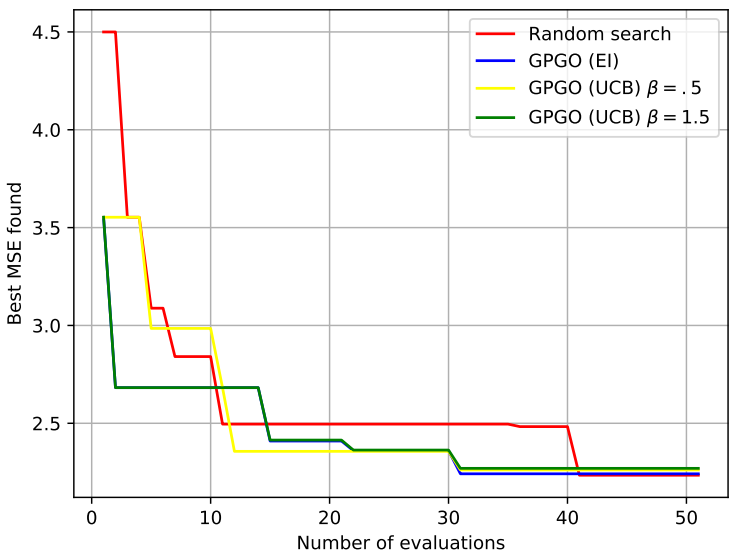
Data

- PDBbind v.2015 database of affinity and kinetic biological data.
- A refined set of $n = 3623$ of protein-ligand pairs for training, and another $m = 195$ structurally diverse for testing.

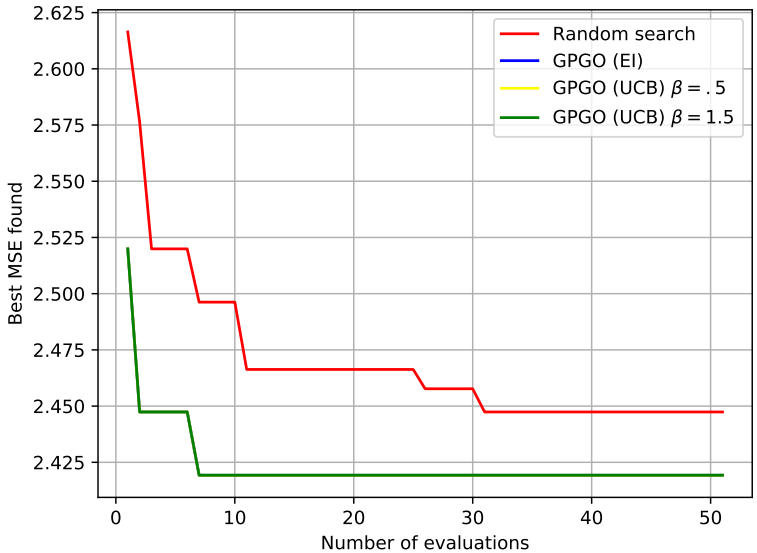
Experiments: Binding affinity prediction dataset

Figure: Benchmarking results for the binding affinity dataset (1/2).

(a) SVM



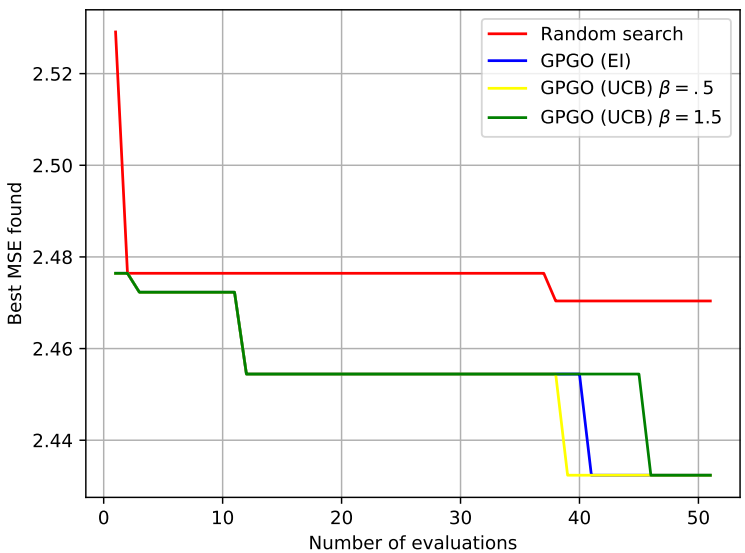
(b) KNN



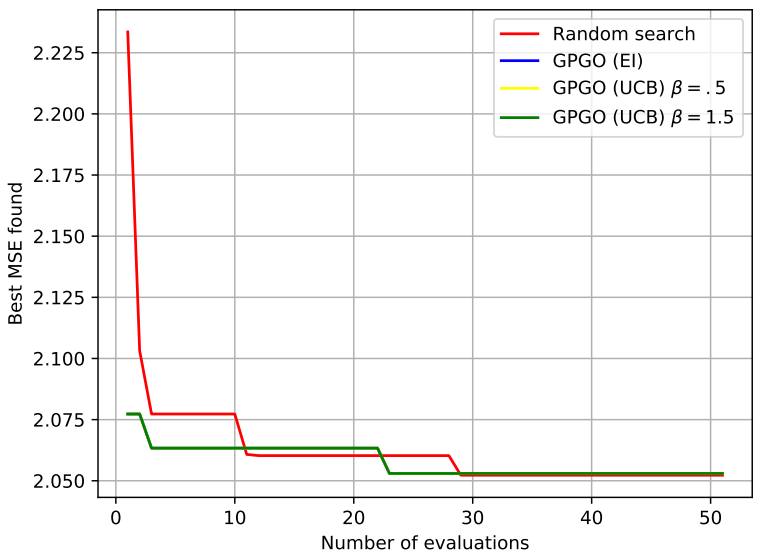
Experiments: Binding affinity prediction dataset

Figure: Benchmarking results for the binding affinity dataset (2/2).

(a) MLP



(b) GBM



Experiments: Protein-protein interface prediction

- Predict whether a particular residue is in contact with a chain of another protein.
- Interfaces are **not** considered target-specific.

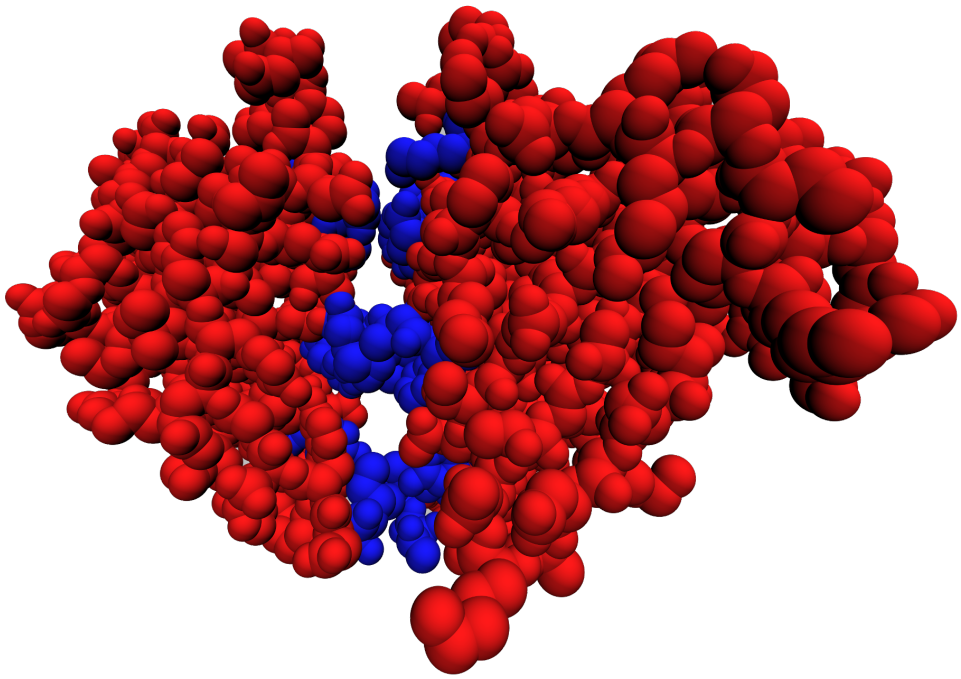


Figure: An interface between two chains of a dimer. Residues considered interficial are marked in blue.

Experiments: Protein-protein interface prediction

Features used

Descriptors resemble those detailed in (Jiménez *et al*. 2017). Average pharmacophoric-like features computed in a particular neighbourhood (20Å^3) of a residue.

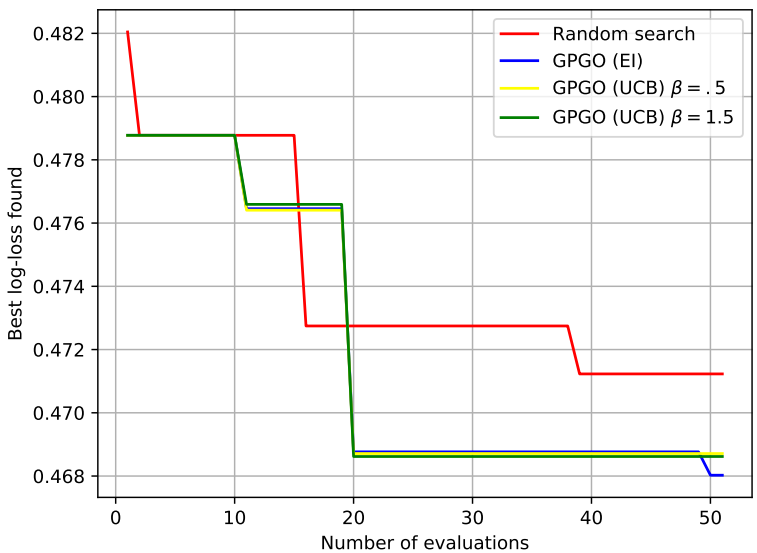
Data

PIFACE database of clustered protein interface. Over 22k structurally unique interfaces extracted from the Protein Data Bank.

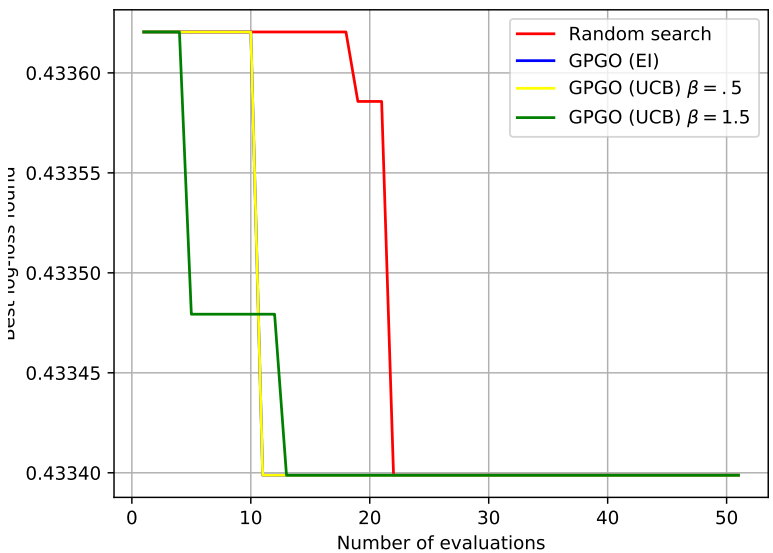
Experiments: Protein-protein interface prediction

Figure: Benchmarking results for the protein-protein interface dataset (1/2).

(a) SVM



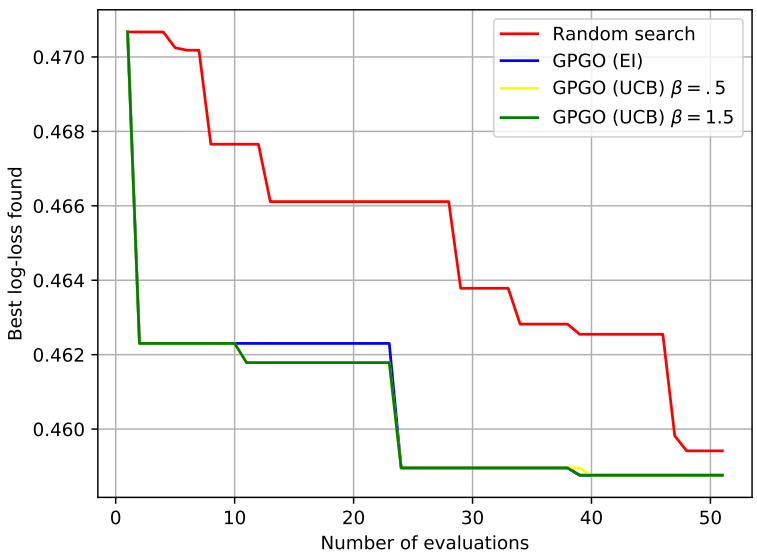
(b) KNN



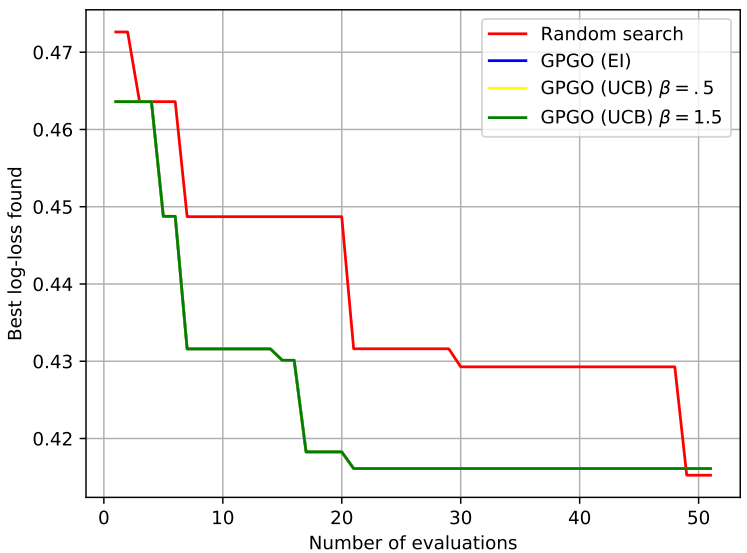
Experiments: Protein-protein interface prediction

Figure: Benchmarking results for the protein-protein interface dataset (2/2).

(a) MLP



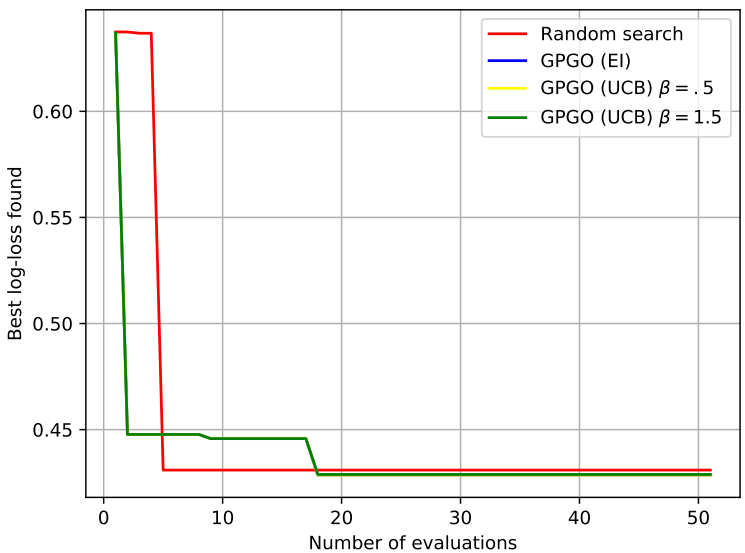
(b) GBM



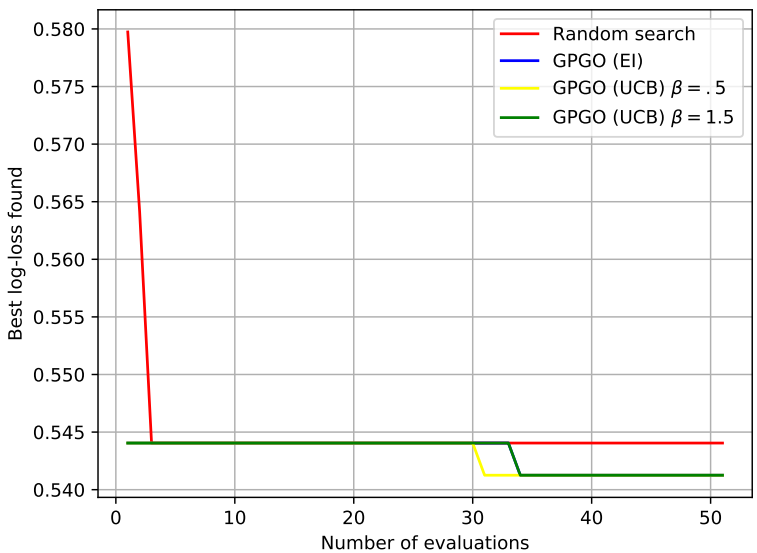
Experiments: other datasets

Figure: Benchmarking results for the LSVT Voice Rehabilitation dataset.

(a) SVM



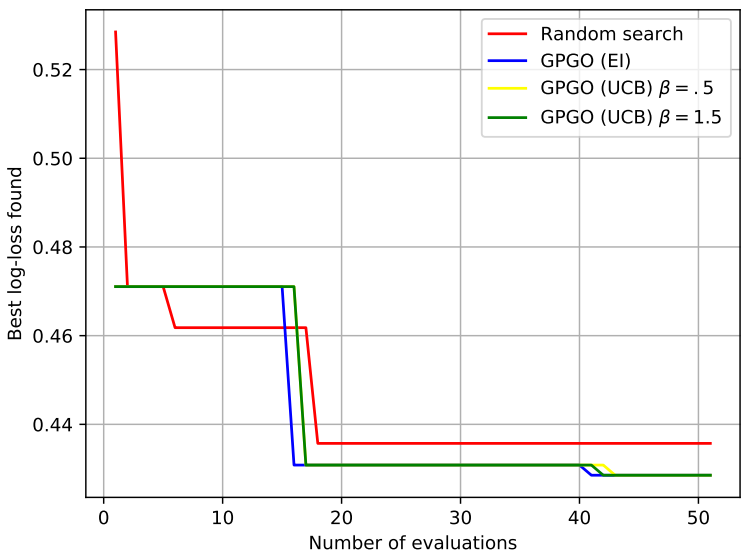
(b) KNN



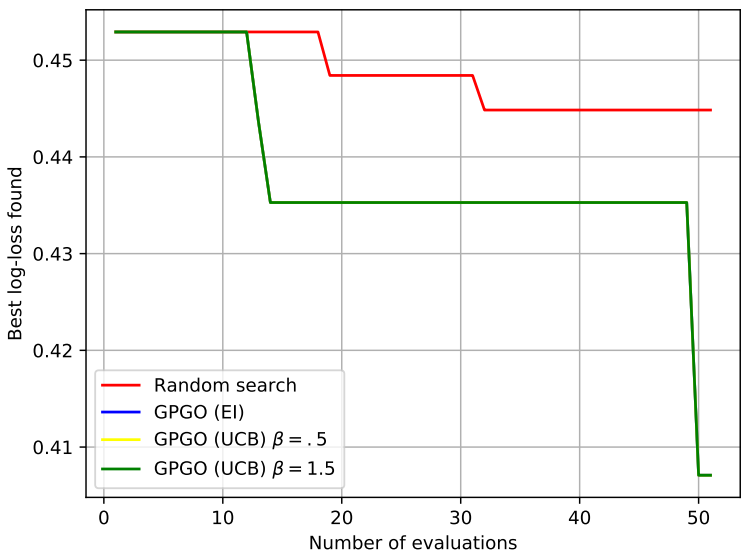
Experiments: other datasets

Figure: Benchmarking results for the LSVT Voice Rehabilitation dataset.

(a) MLP



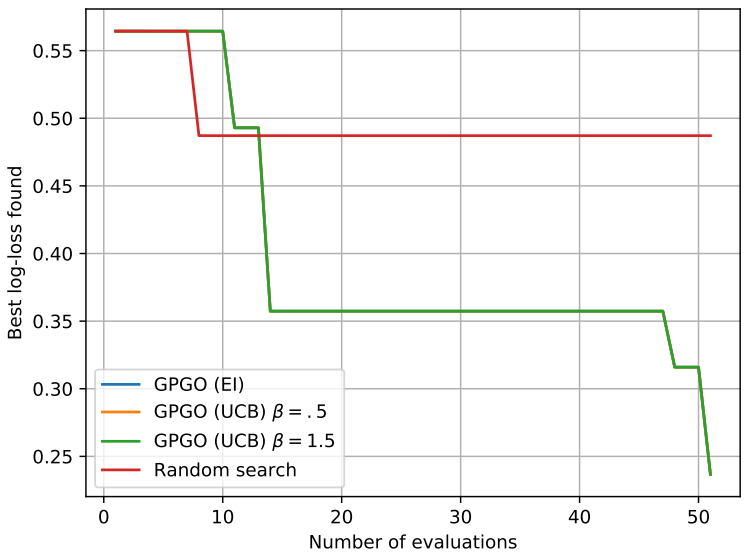
(b) GBM



Experiments: other datasets

Figure: Benchmarking results for the Parkinson's dataset.

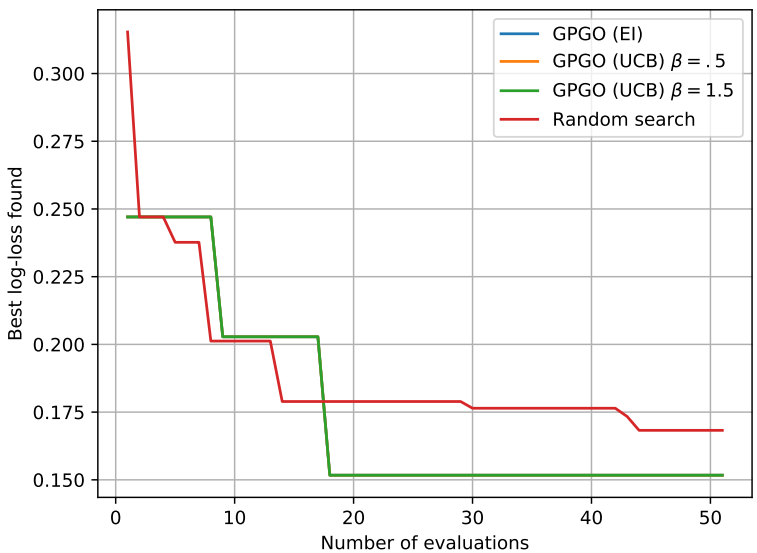
(a) SVM



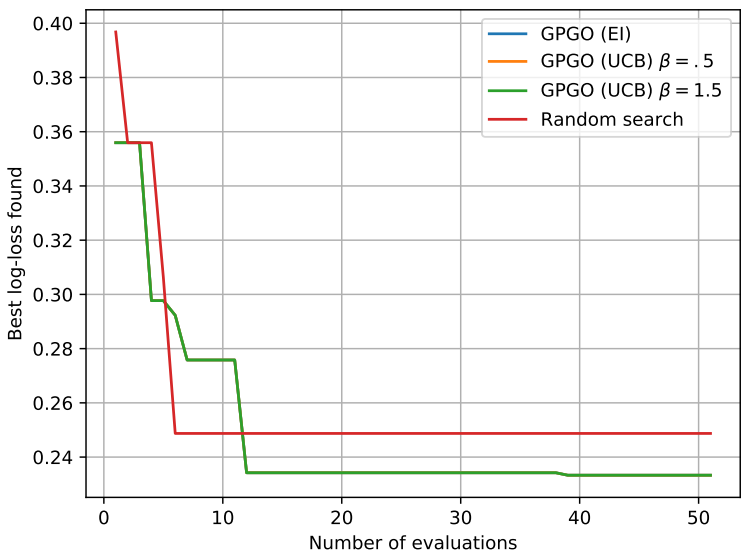
Experiments: other datasets

Figure: Benchmarking results for the Parkinson's dataset.

(a) MLP



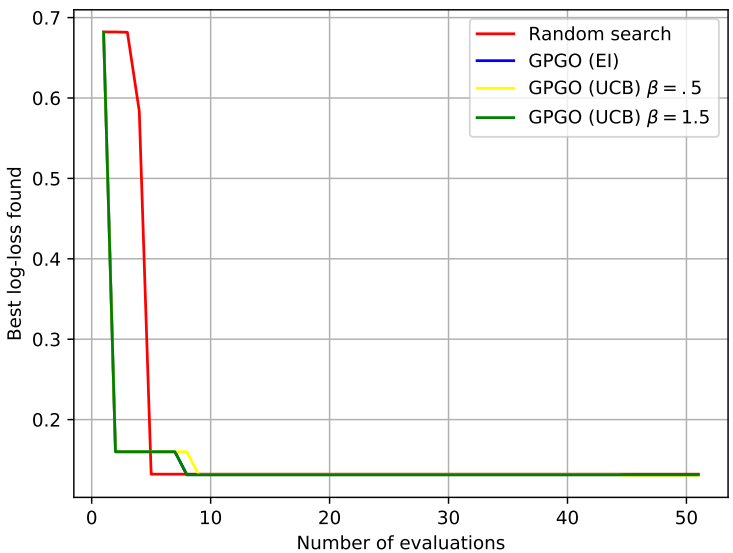
(b) GBM



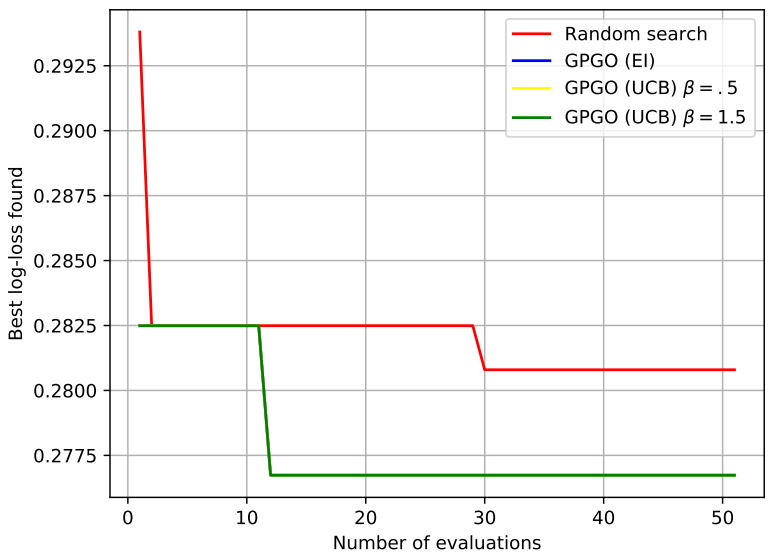
Experiments: other datasets

Figure: Benchmarking results for the Breast cancer dataset.

(a) SVM



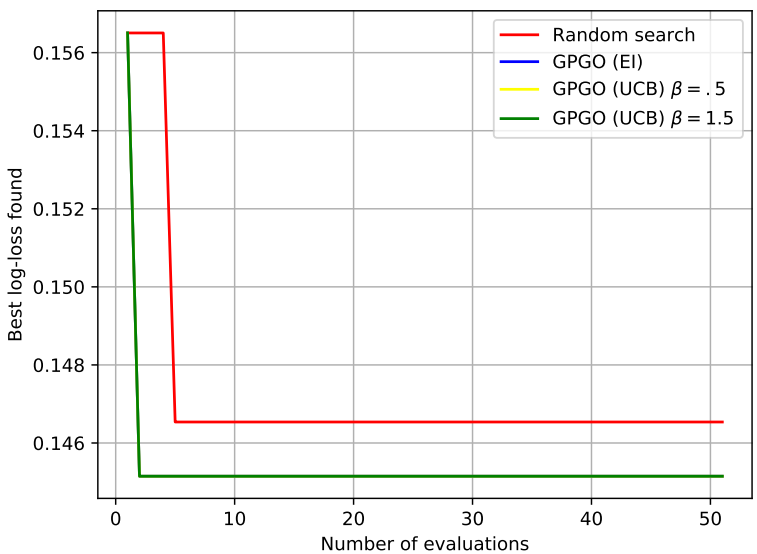
(b) KNN



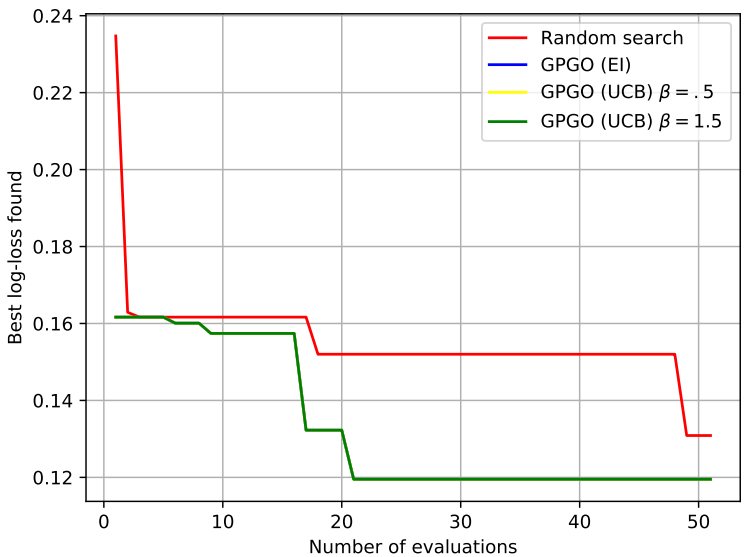
Experiments: other datasets

Figure: Benchmarking results for the Breast cancer dataset.

(a) MLP



(b) GBM



pyGPGO: Bayesian Optimization for Python

There are many possible choices in the Bayesian Optimization framework:

- Choice of surrogate model.
- Covariance function to use
- Acquisition behaviour
- Hyperparameter treatment

They naturally motivate a modular design. Most of the available software focuses on a particular implementation of the Bayesian optimization algorithm.

pyGPGO is a package I wrote to cover as many of these scenarios as possible.

pyGPGO: features

pyGPGO features:

- A completely modular and customizable design. Easy to setup and minimal dependencies.
- A wide range of surrogate models: Gaussian Processes, Student-t Processes, Random Forests, Extra Random Forests and Gradient Boosting Machines.
- Most of the usual covariance functions, as well as its derivatives: squared exponential, Matérn, γ-exponential, rational quadratic, exponential sine, and dot product.
- Many acquisition function behaviours: probability of improvement, expected improvement, upper confidence bound and entropy-based, as well as their integrated versions.
- Type II Maximum-Likelihood of covariance function hyperparameters.
- MCMC sampling for full-Bayesian treatment of hyperparameters (via `pyMC3`).

pyGPGO: a simple example

```
1  import numpy as np
2  from pyGPGO.covfunc import matern32
3  from pyGPGO.acquisition import Acquisition
4  from pyGPGO.surrogates.GaussianProcess import GaussianProcess
5  from pyGPGO.GPGO import GPGO
6
7
8  def f(x, y):
9      # Franke's function (https://www.mathworks.com/help/curvefit/franke.html)
10     one = 0.75 * np.exp(-(9 * x - 2) ** 2 / 4 - (9 * y - 2) ** 2 / 4)
11     two = 0.75 * np.exp(-(9 * x + 1) ** 2/ 49 - (9 * y + 1) / 10)
12     three = 0.5 * np.exp(-(9 * x - 7) ** 2 / 4 - (9 * y -3) ** 2 / 4)
13     four = 0.25 * np.exp(-(9 * x - 4) ** 2 - (9 * y - 7) ** 2)
14     return one + two + three - four
```

pyGPGO: a simple example

```
cov = matern32()
gp = GaussianProcess(cov)
acq = Acquisition(mode='ExpectedImprovement')
param = {'x': ('cont', [0, 1]),
         'y': ('cont', [0, 1])}

np.random.seed(1337)
gpgo = GPGO(gp, acq, f, param)
gpgo.run(max_iter=10)
gpgo.getResult()
```

pyGPGO: optimizing SVM hyperparameters

```
1  import numpy as np
2  from sklearn.datasets import make_moons
3  from sklearn.svm import SVC
4  from sklearn.model_selection import cross_val_score
5
6
7  from pyGPGO.GPGO import GPGO
8  from pyGPGO.surrogates.GaussianProcess import GaussianProcess
9  from pyGPGO.acquisition import Acquisition
10 from pyGPGO.covfunc import squaredExponential
11
12
13 def evaluateModel(C, gamma):
14     clf = SVC(C=10**C, gamma=10**gamma)
15     return np.average(cross_val_score(clf, X, y))
```

pyGPGO: optimizing SVM hyperparameters

```
X, y = make_moons(n_samples = 200, noise = 0.3)

sexp = squaredExponential()
gp = GaussianProcess(sexp, optimize = True, usegrads = True)
acq = Acquisition(mode = 'UCB', beta = 1.5)

params = {'C':      ('cont', (-4, 5)),
          'gamma':  ('cont', (-4, 5))}

gpgo = GPGO(gp, acq, evaluateModel, params)
gpgo.run(max_iter = 50)
gpgo.getResult()
```

pyGPGO: Bayesian Optimization for Python

- The software is MIT licensed and can be downloaded from its GitHub repository @ https://github.com/hawk31/pyGPGO
- Extensive documentation can be found in ReadTheDocs. pygpgo.readthedocs.io/en/latest
- Anyone can contribute to the project! (Implement new features, report bugs, improve docs...)
- All benchmarking, examples and thesis code is also available.

Thank you!