Automatic algorithm and hyperparameter selection based on data and users recommendations

Maxim Panov
Yermek Kapushev

Insitute for Information Transmission Problems
Datadvance
PreMoLab

12 February 2016

DATADVANCE

1. Problem statement, motivating examples
2. Hints based model choice
3. SBO for hyperparameter selection
4. pSeven Core framework
5. Experiments

Hyperparameters of algorithm

Hyperparameters are parameters defined outside training procedure.

Most ML algorithms have hyperparameters:

- Ridge regression and lasso — regularization term
- SVM — regularization and kernel parameters
- Neural network — number of layers and neurons
- XGBoost — number of trees, max depth, learning rate, ...
- ...

Example: hyperparameters of XGBoost

Hyperparameters of one of the most popular tool for gradient boosting **XGBoost**:

- Number of trees — integer
- Max depth — integer
- Objective function — categorical
- Learning rate — float
- Minimum loss reduction — float
- Minimum child weight — float
- Subsample ratio of instances — float
- Subsample ratio of columns — float

Motivation for hyperparameters tuning

Why automating hyperparameters tuning?

- Performance gain
- No universal set of hyperparameters, optimal hyperparameters depend on data set
- Optimization algorithm can do better than human
- Enable non-experts to use ML algorithms

Formal problem statement

Hyperparameter tuning problem involves:

- $\boldsymbol{\lambda}$ — vector of hyperparameters $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_n)$
- $A_{\boldsymbol{\lambda}}$ — learning algorithm with hyperparameters $\boldsymbol{\lambda}$
- $\mathcal{D}_{train}$ — training data set $\mathcal{D}_{train} = (X, Y) = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$
- $\mathcal{L}(A_{\boldsymbol{\lambda}}, \mathcal{D}_{train}, \mathcal{D}_{test})$ — loss function achieved by $A_{\boldsymbol{\lambda}}$ when trained on data set $\mathcal{D}_{train}$ and validated on $\mathcal{D}_{test}$

The task is to find

$$\boldsymbol{\lambda}^* = \arg\min_{\boldsymbol{\lambda}} \mathcal{L}\left(A_{\boldsymbol{\lambda}}, \mathcal{D}_{train}, \mathcal{D}_{test}\right)$$

Loss function

Example of loss function

$$\mathcal{L}(A_{\boldsymbol{\lambda}}, \mathcal{D}_{train}, \mathcal{D}_{test}) = \text{RMSE} = \sqrt{\frac{1}{N_{test}} \sum_{i=1}^{N_{test}} (y_i - \hat{y}_i)^2},$$

where $y_i \in \mathcal{D}_{test}$, N_{test} — size of $\mathcal{D}_{test}$,
$\hat{y}_i$ is a prediction of a model learned on $\mathcal{D}_{train}$ using algorithm $A_{\boldsymbol{\lambda}}$.

Cross-validation is used if $\mathcal{D}_{test}$ is not known.

Problem key properties

- Unknown, probably **non-convex** loss function $\mathcal{L}$
- **Derivatives are not available**
- Evaluation of loss function is **expensive**
- Evaluation time of $\mathcal{L}$ **depends** on $\boldsymbol{\lambda}$
- **Different types** of hyperparameters:
  - numerical (both integer and real-valued)
  - categorical
- **Tree-structured configuration space**

Example: artificial neural network

Hyperparameters:

- Number of layers
- Number of neurons in each layer
- Activation function for each layer
- Learning rate
- ...

$\rightarrow$ complex tree-structured configuration space

$\rightarrow$ mixed types of hyperparameters

Hints

Sometimes user **knows something about the data**

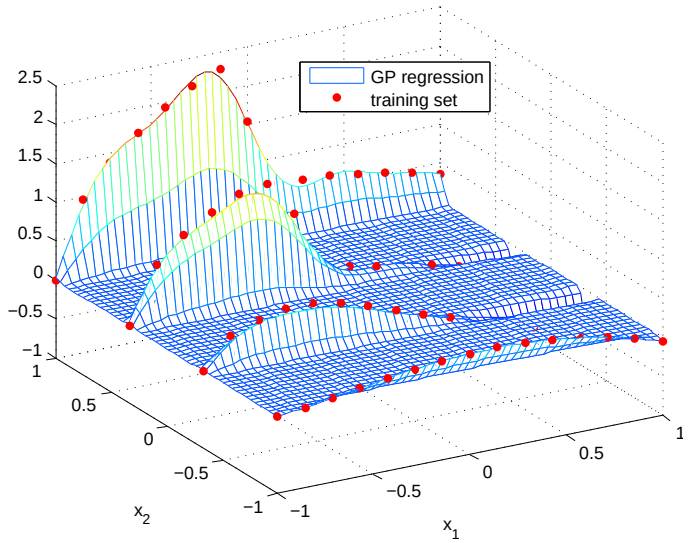
- Data is linear/quadratic
- Data is discontinuous
- Data has special structure (e.g. factorial design of experiments)

Sometimes user has **requirements for model**

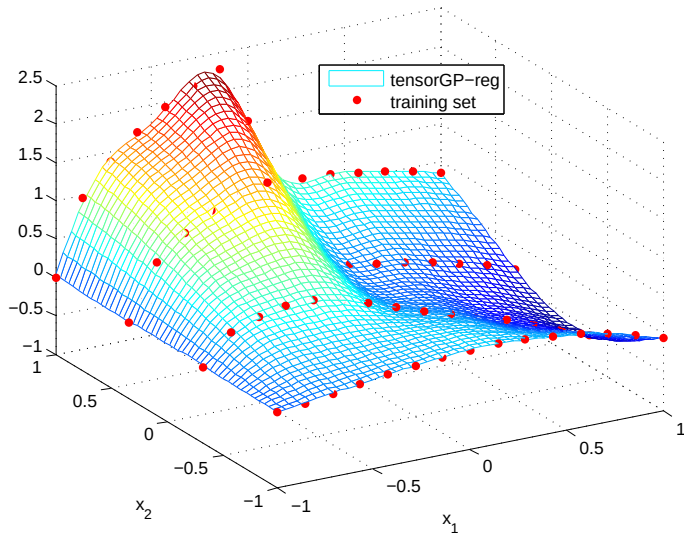
- to be smooth
- to be interpolating
- to allow to estimate uncertainty of prediction

$\rightarrow$ **Configuration space may be reduced**

Example



(a) Universal approximation technique



(b) Special technique for factorial samples

Universal techniques don't take into account sample structure

→ poor quality

→ high computational complexity

Existing approaches

- **Grid Search**
  - Simple to implement and parallelize
  - Requires large budget in high dimensional space
- **Random Search**
  - The maximum of 60 random observations lies within the top 5% of the true maximum, with 95% probability.

  $\rightarrow$ Random search finds better models than grid search if budget is small.
- **Bayesian optimization**
  - "Smart" selection of new candidate point
  - Based on modeling dependence of loss function $\mathcal{L}$ on $\boldsymbol{\lambda}$
  - Sequential, hard to parallelize

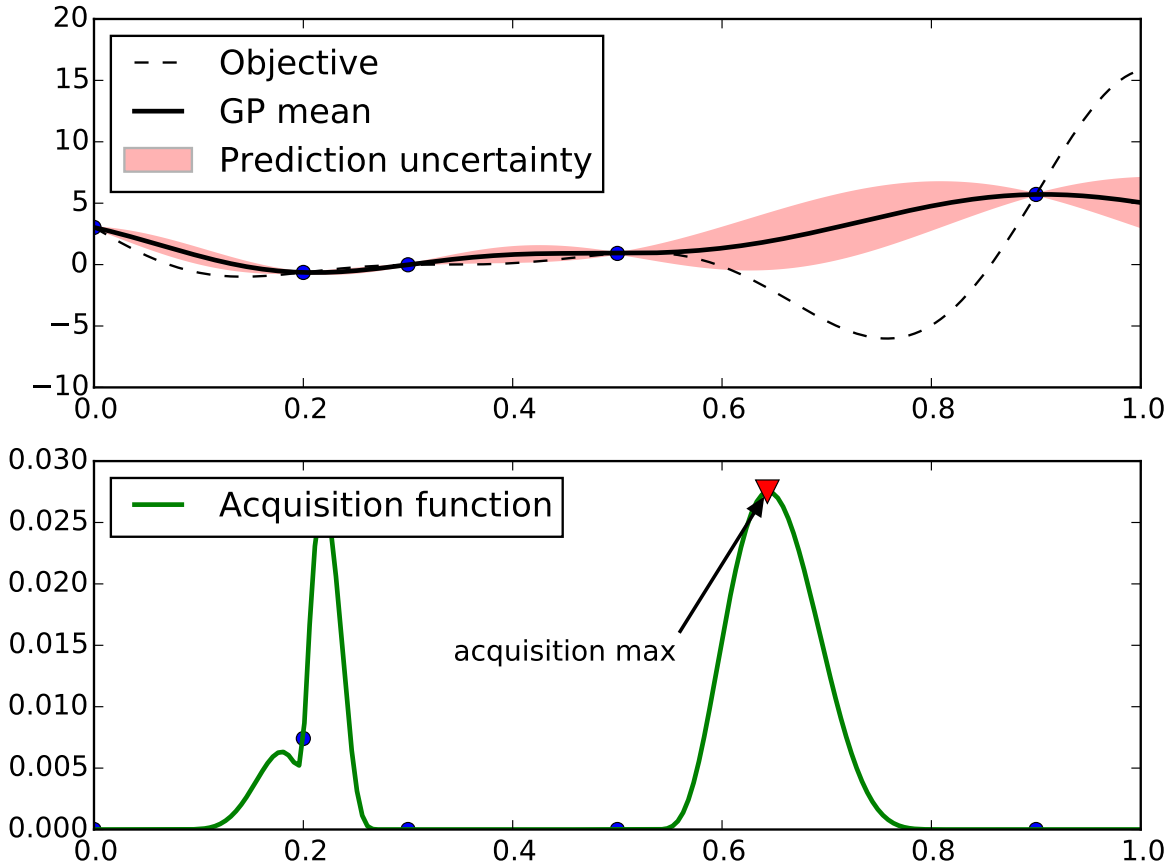
Surrogate model Based Optimization

SBO uses surrogate model $\mathcal{M}$ that captures dependency of loss function $\mathcal{L}$ on $\boldsymbol{\lambda}$.

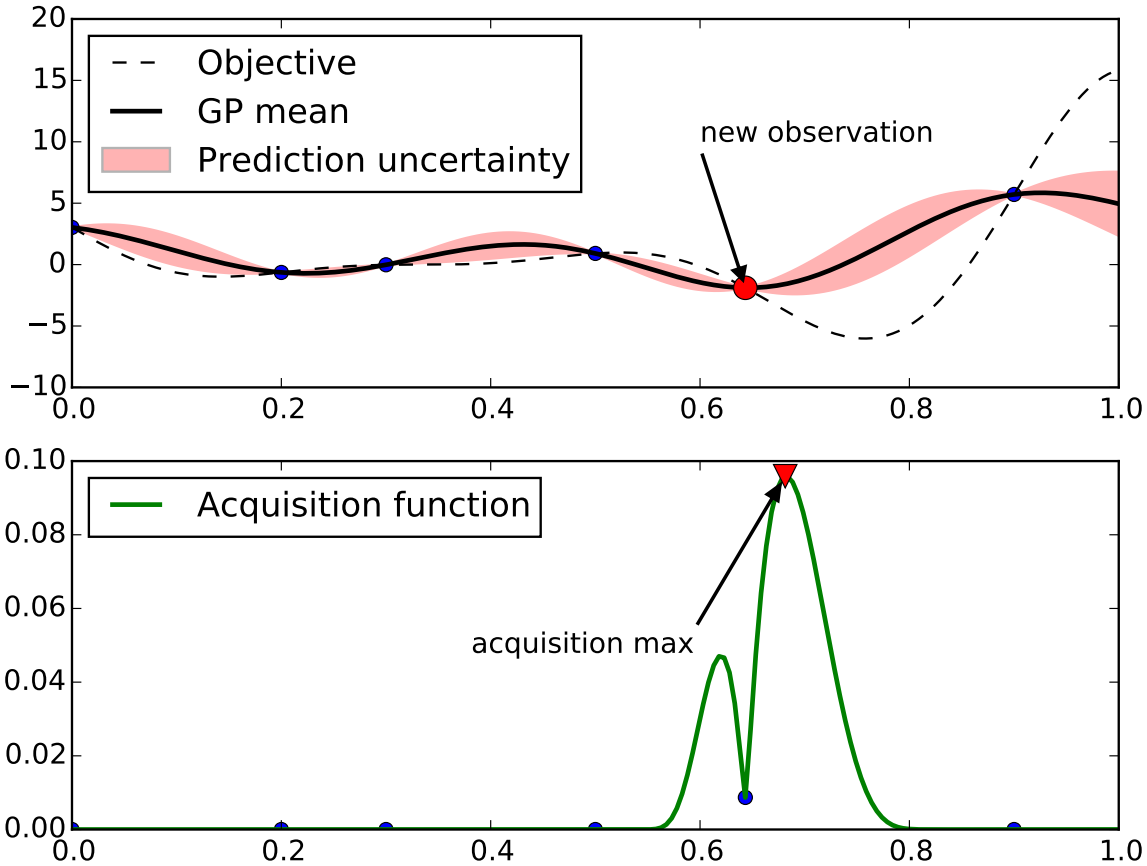
Generic SBO scheme

1. **Generate initial sample** of hyperparameters and evaluate objective function
2. **Build model**
3. **Select new candidates** using model-based criterion
4. **Evaluate objective** function
5. Augment current sample, **goto step 2**

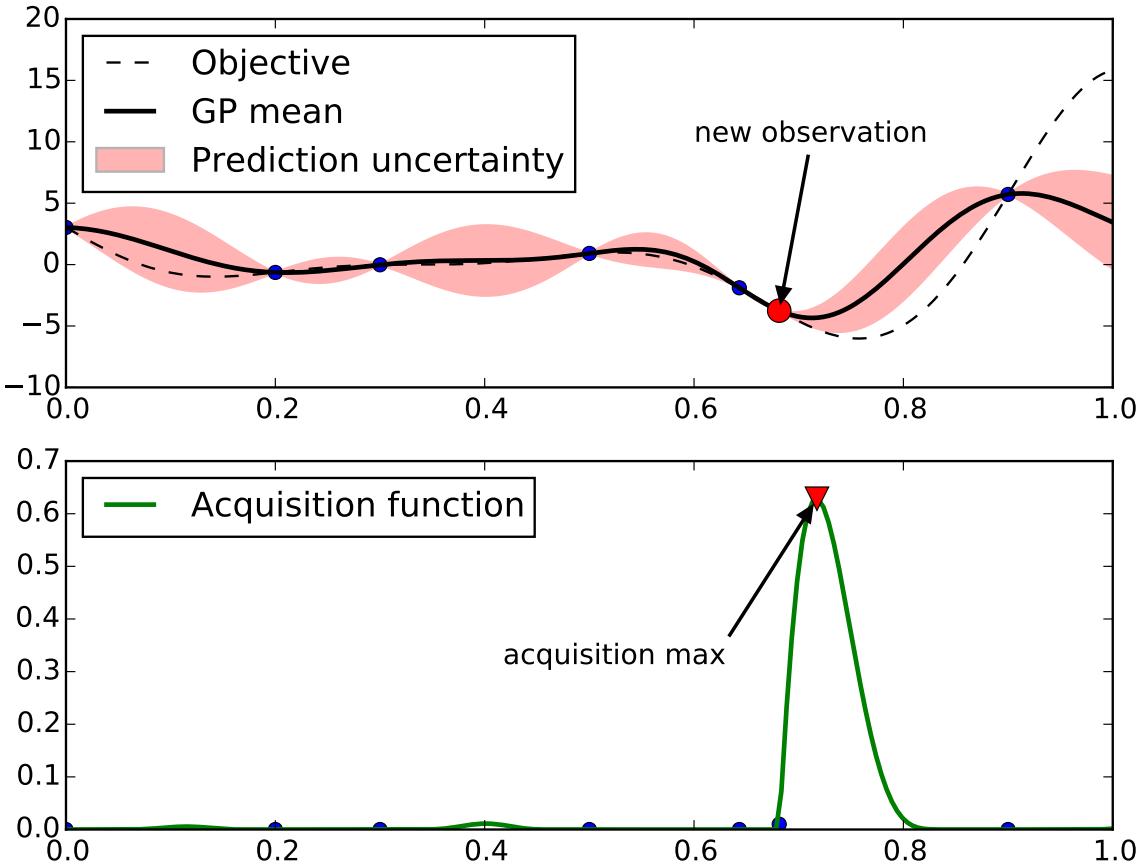
SBO illustration



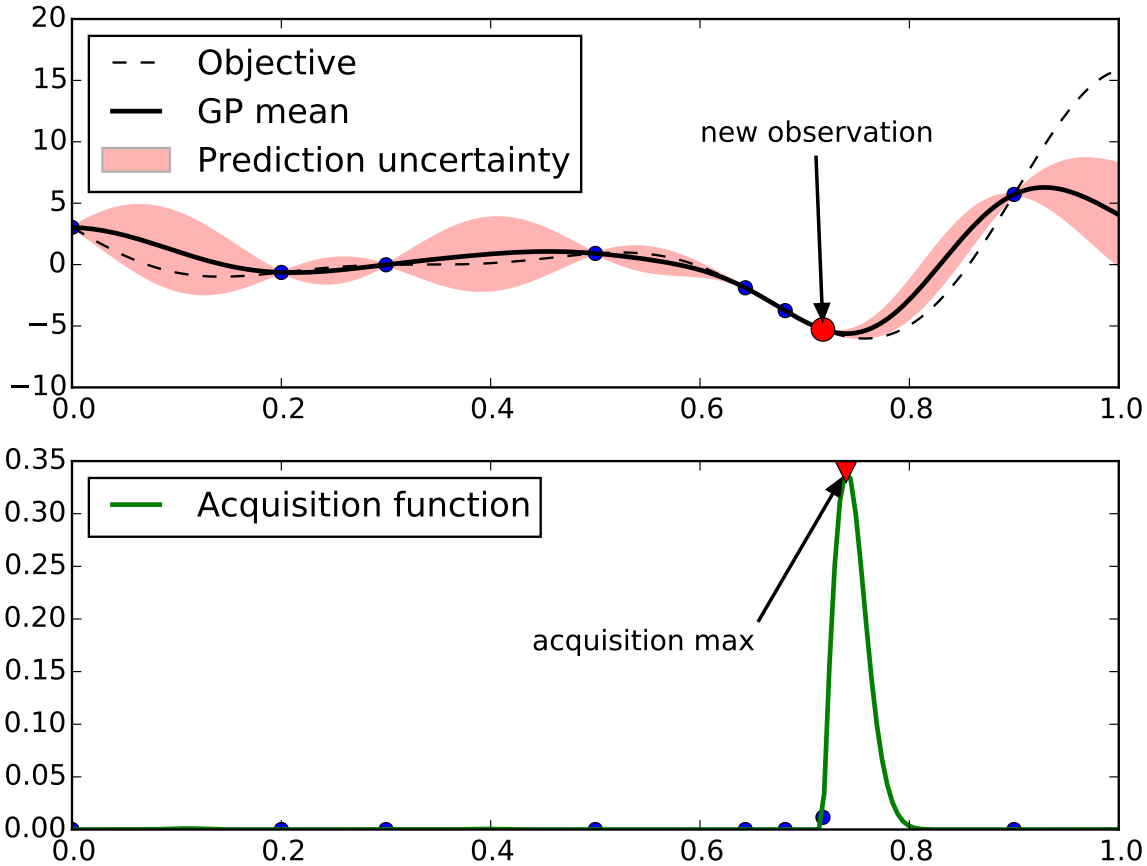
SBO illustration



SBO illustration



SBO illustration



Surrogate model Based Optimization

So, we need

1. Surrogate modeling function
2. Procedure to choose candidate points

Surrogate models

Requirements for surrogate models:

uncertainty estimate $\sigma(\boldsymbol{\lambda})$ at any point $\boldsymbol{\lambda}$.

Popular choices:

1. Gaussian Processes
2. Random Forest

Gaussian Process Regression

- $g(x) = f(x) + \varepsilon(x)$,
  where $f(x)$ — Gaussian process (GP), ε — Gaussian white noise.
- GP is fully defined by its mean and covariance function.
- The covariance function of $g(x)$:

$$K_g(x, x') = K(x, x') + \sigma^2_{noise}\delta(x, x'),$$

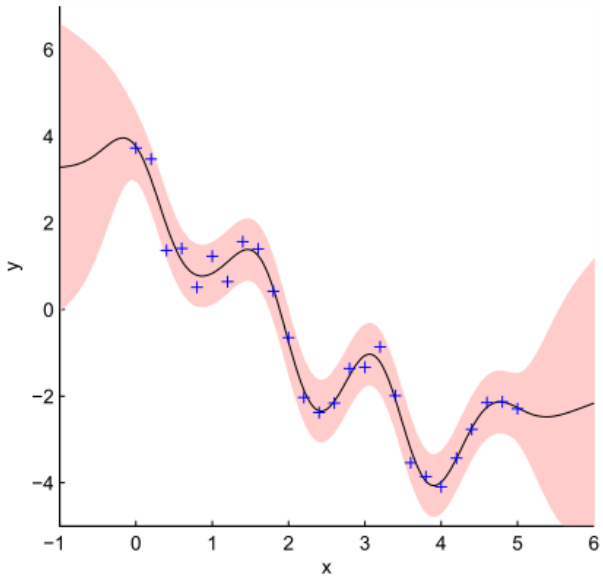
$K(x, x')$ — covariance function of $f(x)$,
$\delta(x, x')$ — Kronecker delta.

Gaussian Process Regression

- Prediction of $g(x)$ at point x

$$\hat{f}(x) = \mathbf{k}^T \mathbf{K}_g^{-1} \mathbf{y},$$

where $\mathbf{k} = (K_f(x_1, x), \ldots, K_f(x_n, x))$,
$\mathbf{K}_g = ||K_g(x_i, x_j)||_{i,j}^N$,
$\mathbf{y} = (g(x_1), \ldots, g(x_N)$, $\{(x_i, g(x_i))\}_{i=1}^N$ is a given data set.



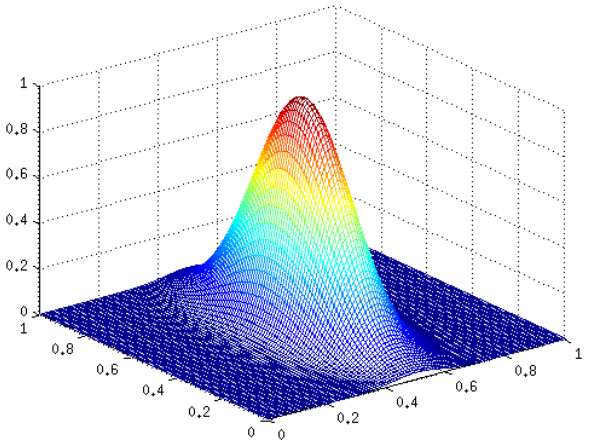
- Posterior variance

$$\sigma^2(x) = K_f(x, x) - \mathbf{k}^T \mathbf{K}_g^{-1} \mathbf{k}.$$

Covariance Function

- Squared exponential function

$$K_{se}(x, x') = \sigma^2 \exp\left(-\sum_{i=1}^{d} \theta_i (x_k - x'_k)^2\right)$$



Some observations of SBO:

1. Evaluated candidates **cluster** in promising regions
2. **Hierarchy of cluster sizes** could be observed

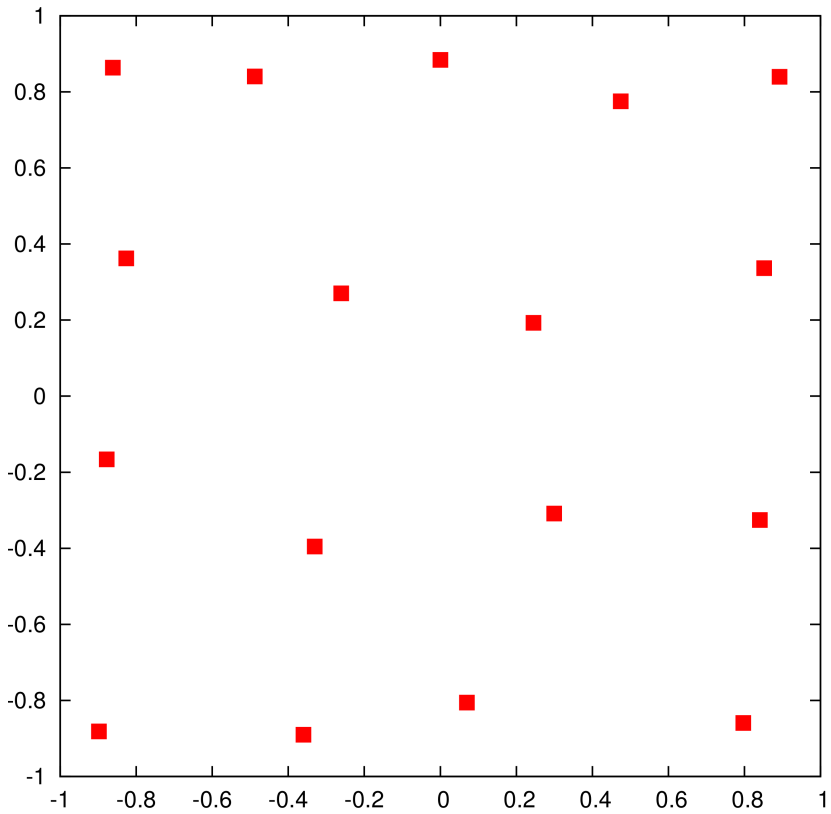


Figure : Initial sample

Some observations of SBO:

1. Evaluated candidates **cluster** in promising regions
2. **Hierarchy of cluster sizes** could be observed

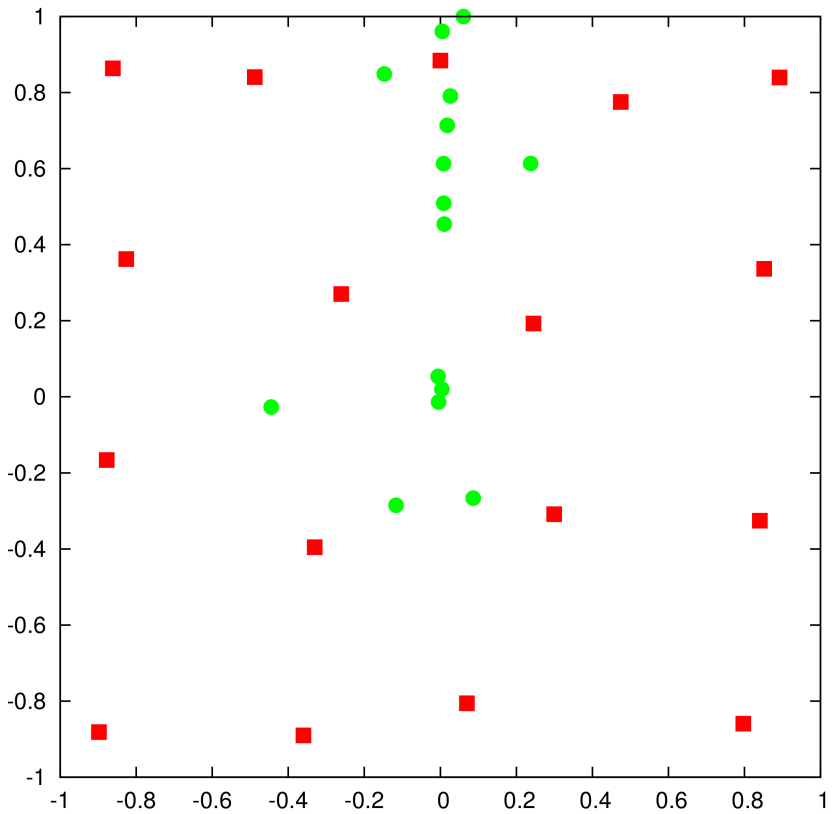


Figure : After few iterations

Some observations of SBO:

1. Evaluated candidates **cluster** in promising regions
2. **Hierarchy of cluster sizes** could be observed

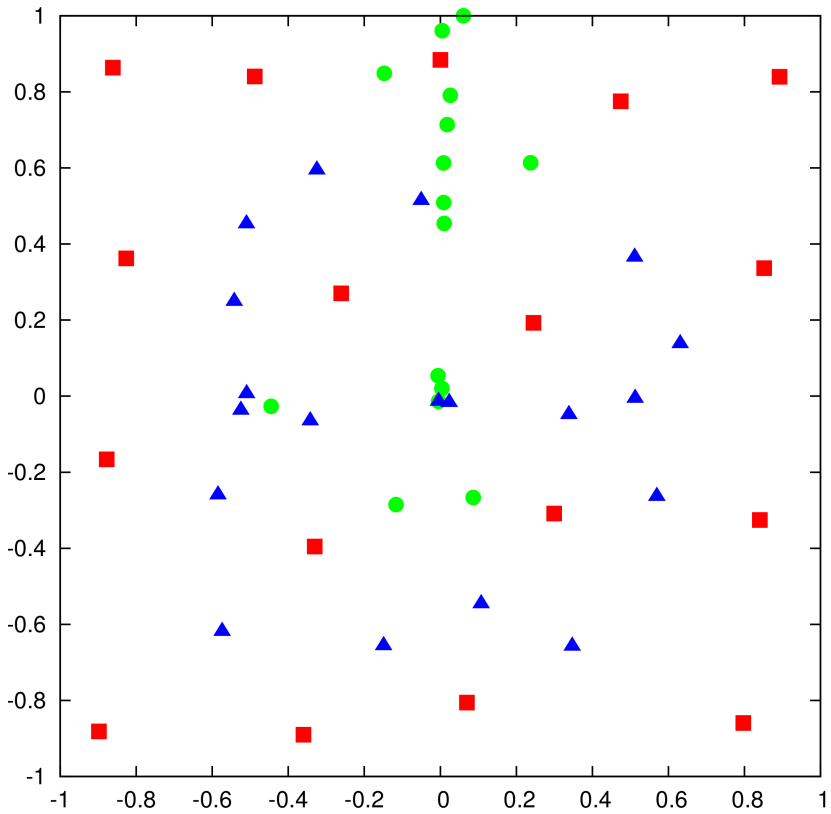
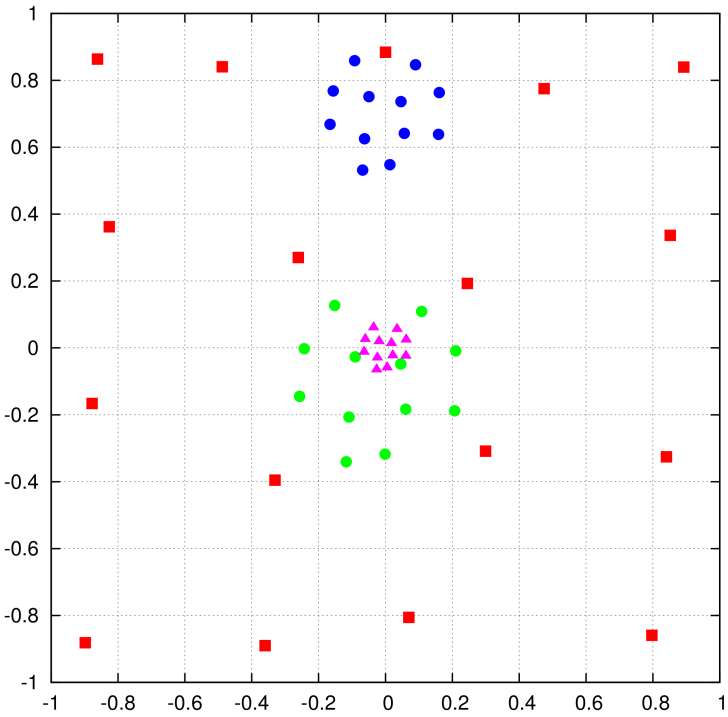


Figure : Converged solution

Some observations of SBO:

1. Evaluated candidates **cluster** in promising regions
2. **Hierarchy of cluster sizes** could be observed



- Instead of single evaluation of candidate we perform sampling in candidate vicinity.
- Vicinity of μ-th candidate we will denote by Ω_μ.

Covariance Function

- **Multi-resolution** covariance function

$$K(x,y) = K_0(x,y) + \sum_{\mu} \alpha_\mu \sum_{i,j} K^{(\mu)}(x, x_\mu^{(i)})[K^{(\mu)}]^{-1} K^{(\mu)}(x_\mu^j, y),$$

where $K^{(\mu)}$ is an Ω_μ-specific covariance vector/matrix, Ω_μ is a vicinity of candidate point.

Advantages of such covariance function:

- **Non-stationary**
- **Robust w.r.t. sample augmentation**

Illustration of multi-resolution GP

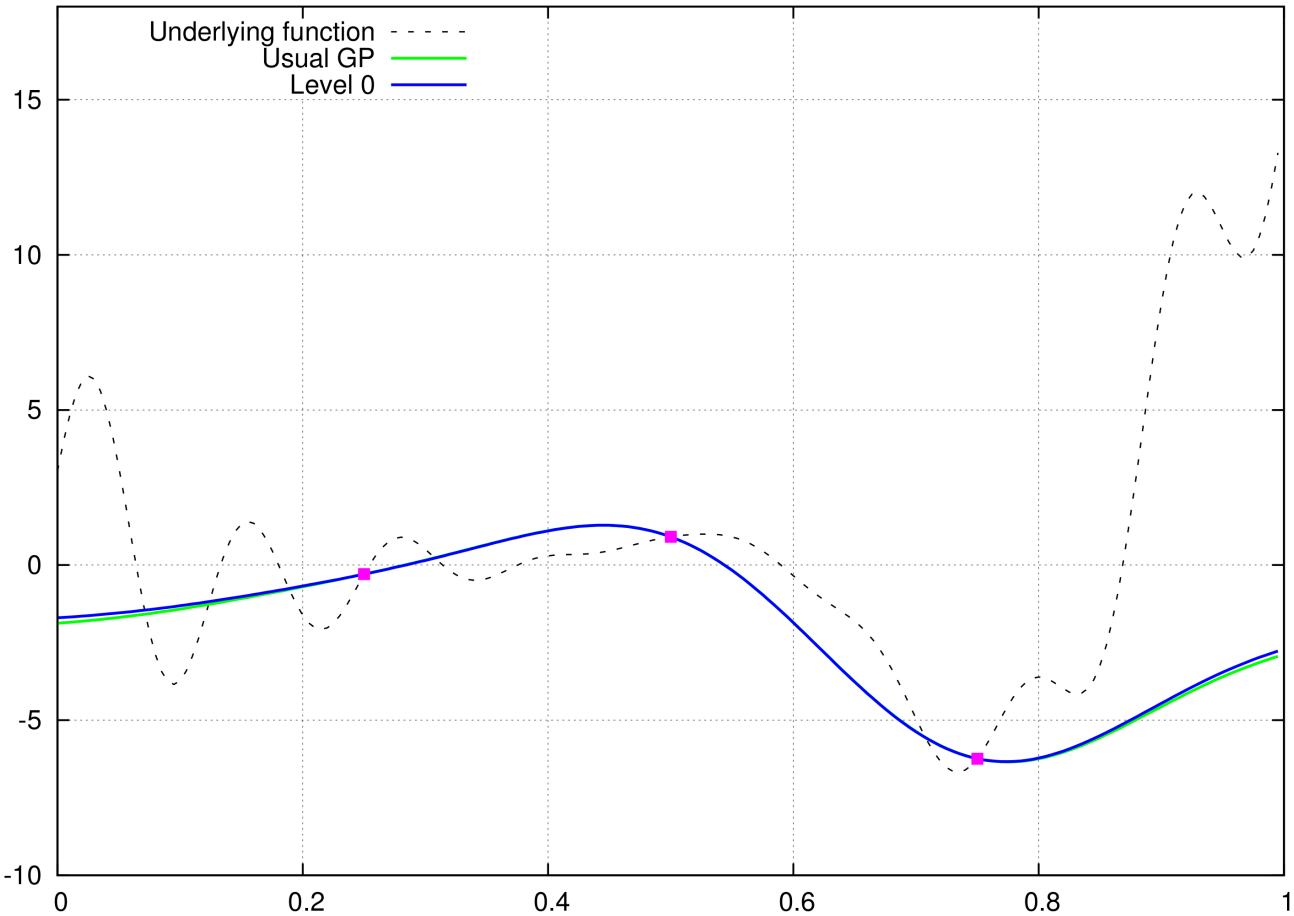


Illustration of multi-resolution GP

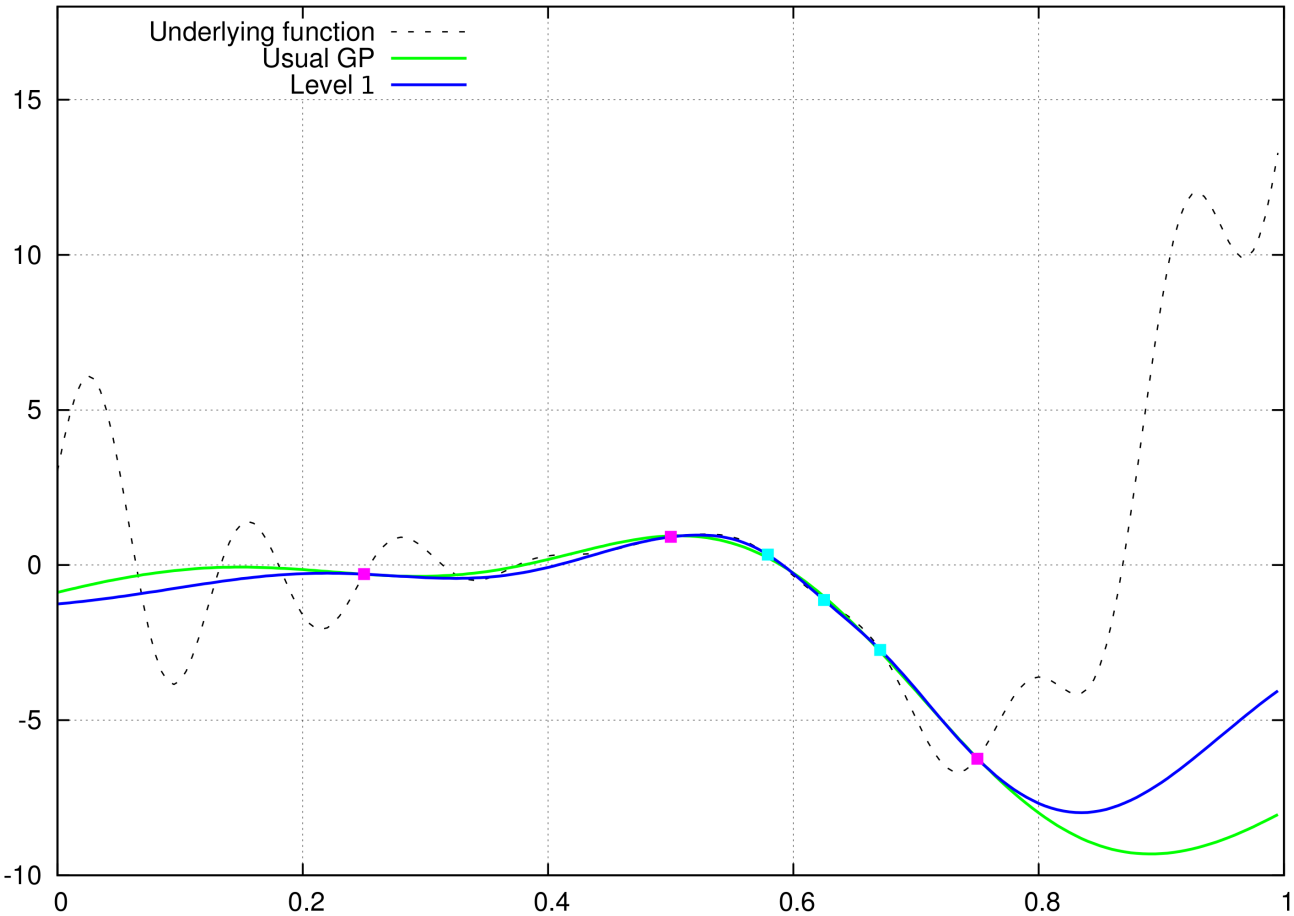


Illustration of multi-resolution GP

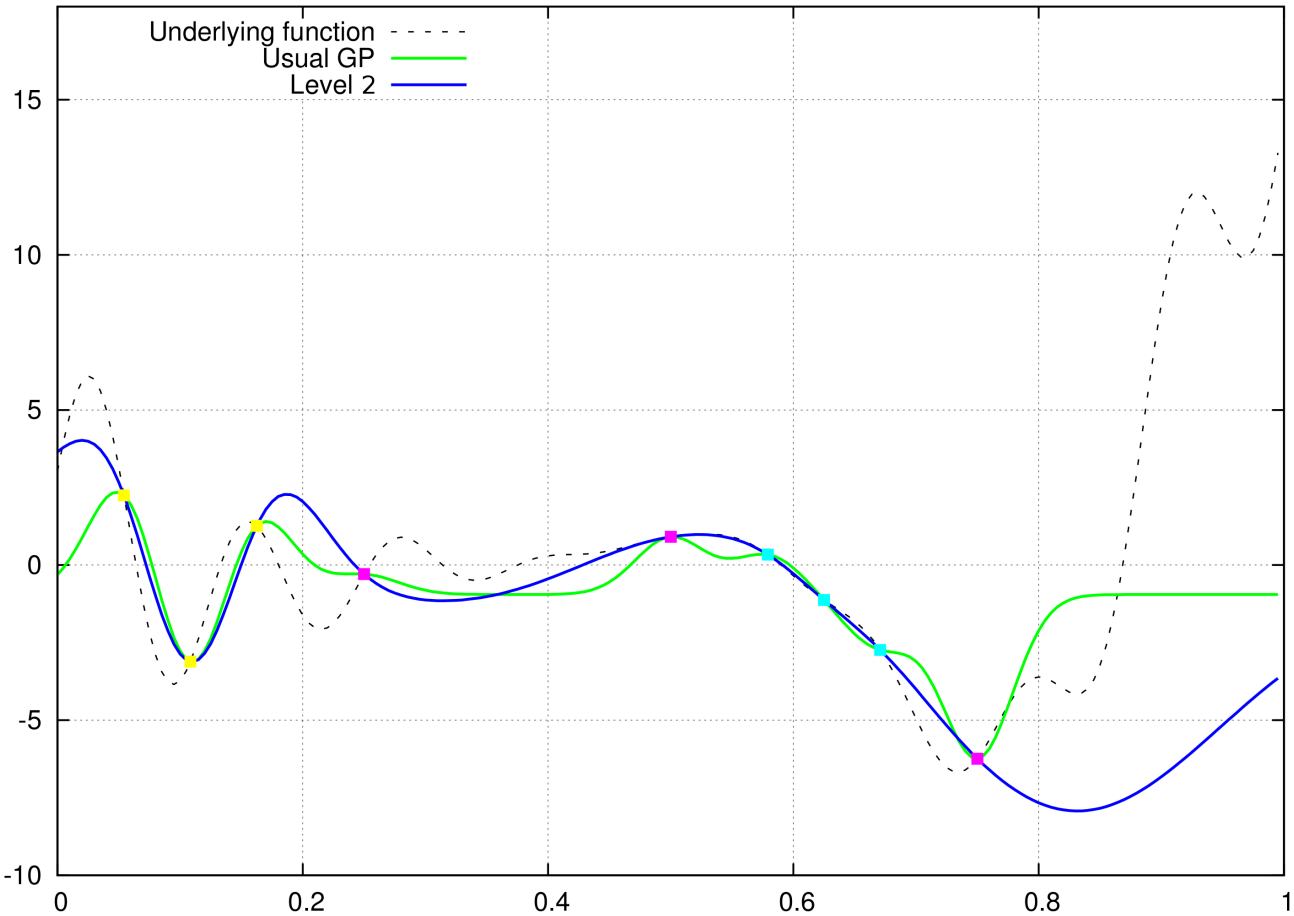


Illustration of multi-resolution GP

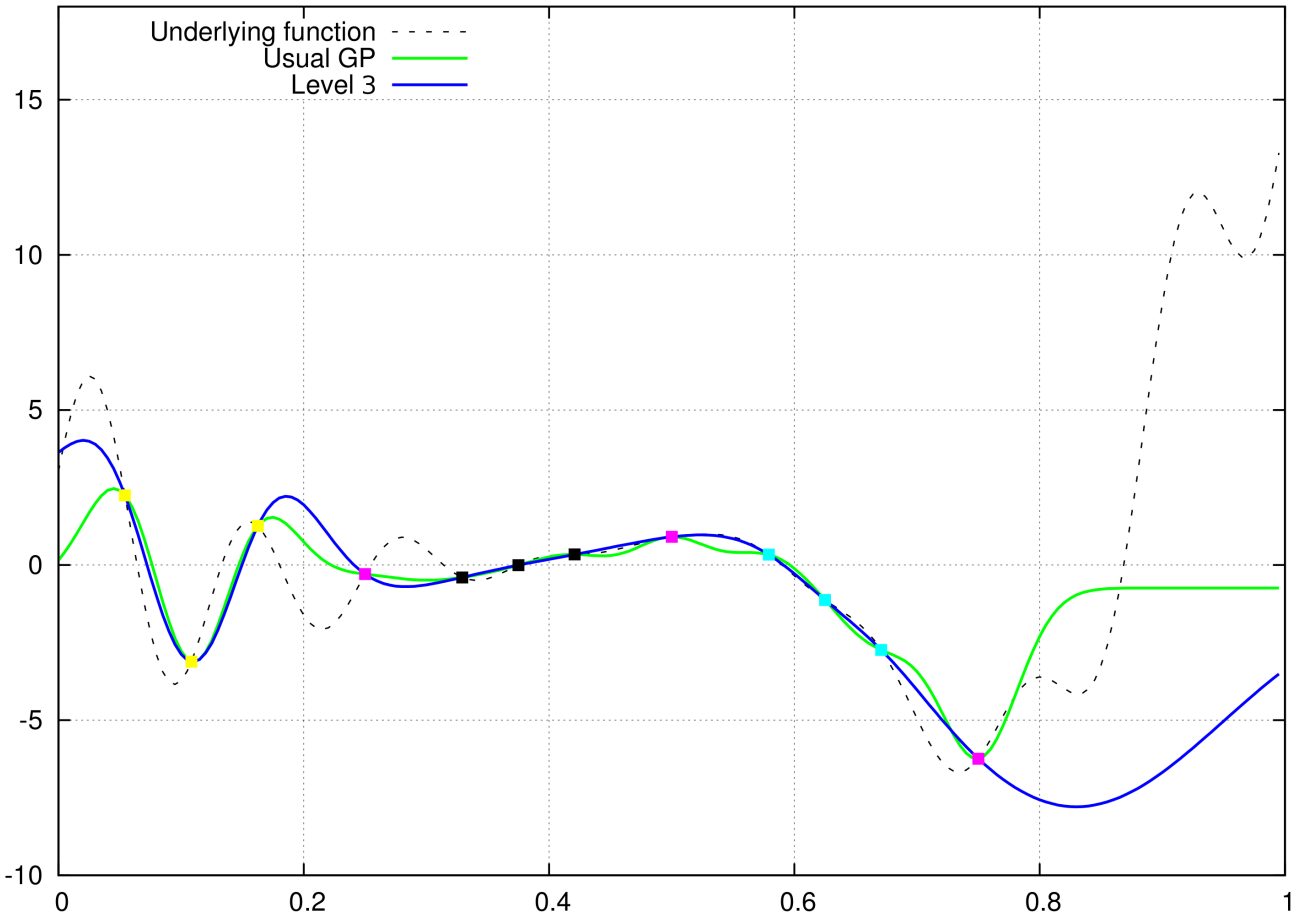
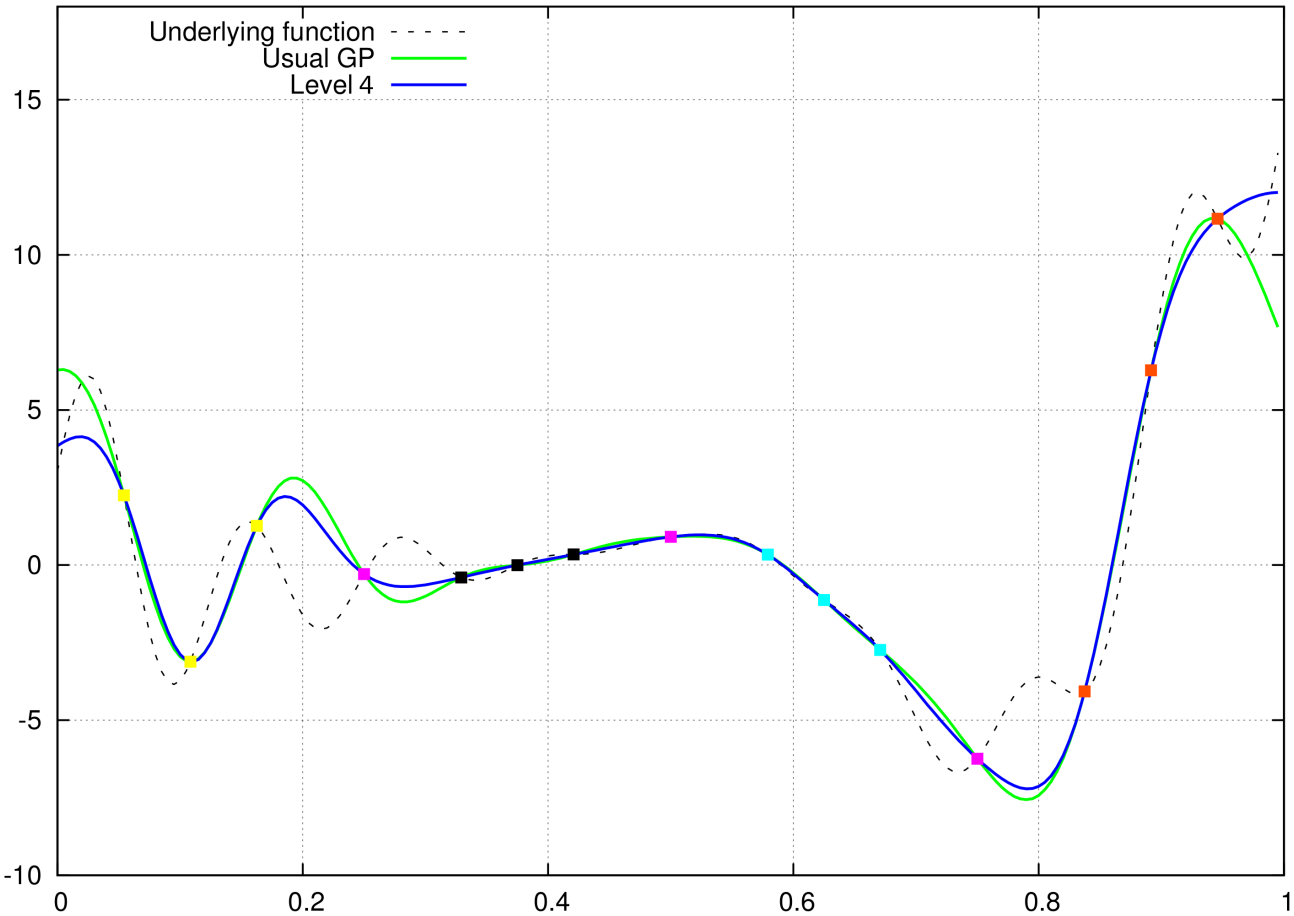


Illustration of multi-resolution GP



Choosing candidate point

Acquisition function $a(\boldsymbol{\lambda}, R)$ is used to choose new candidate point

$$\boldsymbol{\lambda}_{k+1} = \arg\max_{\boldsymbol{\lambda}} a(\boldsymbol{\lambda}, R_k),$$

where $R_k = \{(\boldsymbol{\lambda}_i, c_i), c_i = \mathcal{L}(A_{\boldsymbol{\lambda}_i}, \mathcal{D}_{train}, \mathcal{D}_{test})\}_{i=1}^{k}$.

High values of $a(\boldsymbol{\lambda}, R)$ corresponds to *potentially* low values of loss

- because of low prediction
- because of great uncertainty in prediction
- both

Acquisition function

Let $c' = \min c$ — minimal value of objective found so far,
$\mu(\boldsymbol{\lambda}|R)$ — model prediction,
$\sigma(\boldsymbol{\lambda}|R)$ — uncertainty of prediction.

1. **Probability of Improvement**

$$a_{PI}(\boldsymbol{\lambda}, R) = \mathbb{P}(c < c') = \boldsymbol{\Phi}(\gamma(\boldsymbol{\lambda})),$$

$$\gamma(\boldsymbol{\lambda}) = \frac{c' - \mu(\boldsymbol{\lambda}|R)}{\sigma(\boldsymbol{\lambda}|R)}.$$

Pros: takes into account both prediction and uncertainty.

Cons: doesn't take into account value of improvement.

Acquisition function

Let $c' = \min c$ — minimal value of objective found so far,
$\mu(\boldsymbol{\lambda}|R)$ — model prediction,
$\sigma(\boldsymbol{\lambda}|R)$ — uncertainty of prediction.

2. **GP Upper Confidence Bound**

$$a_{UCB}(\boldsymbol{\lambda}, R) = \mu(\boldsymbol{\lambda}|R) - \beta\sigma(\boldsymbol{\lambda}|R),$$

where β — exploitation-exploration ratio.

Pros: takes into account both prediction and uncertainty.

Cons: has hyperparameter itself.

Acquisition function

Let $c' = \min c$ — minimal value of objective found so far,
$\mu(\boldsymbol{\lambda}|R)$ — model prediction,
$\sigma(\boldsymbol{\lambda}|R)$ — uncertainty of prediction.

3. **Expected Improvement**

$$a_{EI}(\boldsymbol{\lambda}, R) = \mathbb{E}((c' - c)_+) = \\ = \sigma(\boldsymbol{\lambda}|R)\left[\gamma(\boldsymbol{\lambda})\boldsymbol{\Phi}(\gamma(\boldsymbol{\lambda})) + \mathcal{N}(\gamma(\boldsymbol{\lambda}); 0, 1)\right]$$

Pros: takes into account prediction, its uncertainty and value of improvement.

Standard approach for SBO:

- Use Gaussian Processes regression to build model for $c(\boldsymbol{\lambda})$
- Use Expected Improvement

Gaussian Processes model $p(c|\boldsymbol{\lambda})$ which allows to compute Expected Improvement.

There is an alternative approach!

Tree-structured Parzen estimator (TPE)

Previous approaches model $p(c|\boldsymbol{\lambda})$ explicitly. TPE separately estimates $p(c)$ and $p(\boldsymbol{\lambda}|c)$

$$p(\boldsymbol{\lambda}|c) = \begin{cases} l(\boldsymbol{\lambda}), \text{ if } c < c^* \\ g(\boldsymbol{\lambda}), \text{ if } c \geq c^* \end{cases},$$

where c^* is a γ-quantile of the losses obtained so far ($\gamma = 0.15$).

Tree-structured Parzen estimator (TPE)

Actually, in such setting we don't need $p(c)$, because

$$a_{EI}(\boldsymbol{\lambda}, R) = \mathbb{E}((c' - c)_+) \propto \left(\gamma + \frac{g(\boldsymbol{\lambda})}{l(\boldsymbol{\lambda})}(1-\gamma)\right)^{-1}$$

$\rightarrow$ TPE minimizes $\dfrac{g(\boldsymbol{\lambda})}{l(\boldsymbol{\lambda})}$.

Modeling $l(\boldsymbol{\lambda})$ and $g(\boldsymbol{\lambda})$

For each hyperparameter $\lambda_i \in \boldsymbol{\lambda}$ a 1-D Parzen estimator is constructed.

- **Continuous** hyperparameters: Gaussian densities are placed at each hyperparameter value λ_i.
  Standard deviation — the largest distance to neighbors of λ_i.
- **Discrete** hyperparameters: probability are proportional to number of occurrences.

pSeven Core framework

pSeven Core is a Python library developed by DATADVANCE.

It contains generic tools for

- **Approximation**
- Design of Experiment
- Dimension Reduction
- Sensitivity Analysis
- **Optimization**

DATADVANCE

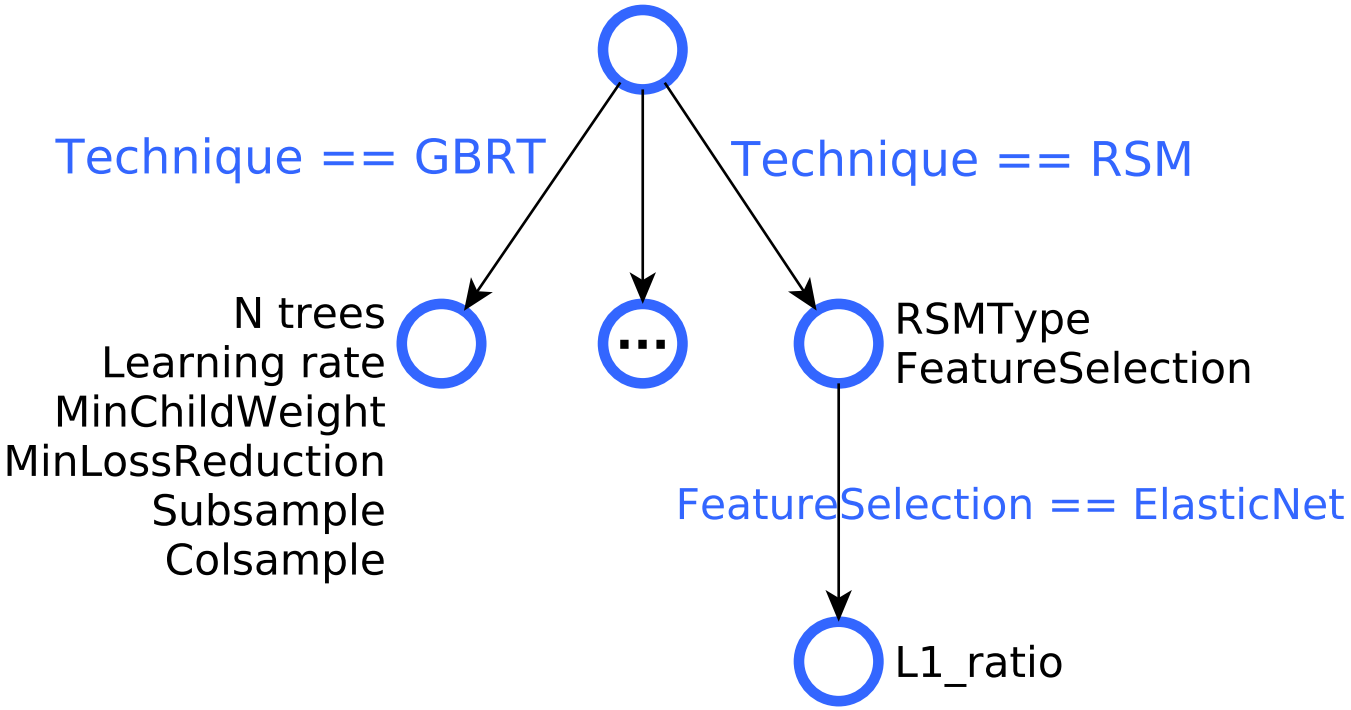
Requirements to hyperparameter tuning algorithm

Current state: some default values of hyperparameters, don't give desired performance for some problems.

Requirements according to problems arising in DATADVANCE practice

- Automatic approach to tune hyperparemters
- Support human-friendly hints
- Control over training time

Options structure



- Most of the branches have depth $= 1$
- Maximum depth $= 2$

Hyperparameters tuning in pSeven Core

1. **Reduce configuration space** according to hints, sample size and input dimension

Examples:

- Hints: DataFeatures=Quadratic
  $\rightarrow$ **RSM** with **RSMType** $\in$ [Quadratic, PureQuadratic, Interactions]

Hyperparameters tuning in pSeven Core

1. **Reduce configuration space** according to hints, sample size and input dimension

Examples:

- Sample size = 10 000
  $\rightarrow$ don't use **GP**

Hyperparameters tuning in pSeven Core

1. **Reduce configuration space** according to hints, sample size and input dimension

Examples:

- Input dimension $>$ sample size
  $\rightarrow$ use **RSM** with **ElasticNet** or **GBRT**.

Hyperparameters tuning in pSeven Core

2. For each technique find optimal hyperparameters via
   - **SBO for numeric** hyperparmeters
   - **brute force search for categorical** hyperparameters

Techniques are either

- Cheap (e.g. RSM, GBRT)
- Medium expensive with few categorical levels (e.g. **GP** has only 6 combinations of categorical parameters)
- Expensive and have no hyperparameters to tune
- Expensive with tunable hyperparameters, but there is only 1 such technique

3. Choose the best performing technique and its hyperparameters.

Experimental Setup for Regression

- A set of toy functions (about 30) is used
- Sample sizes: 80, 320
- Comparing algorithms:
  - **Hyperopt** (uses **TPE**)
  - **SMAC** (uses **Random Forest**)
  - **SmartSelection** (uses **GP**)
- Quality criterion:

$$\mathrm{RRMS} = \frac{\sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}}{\sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \bar{y})^2}},$$

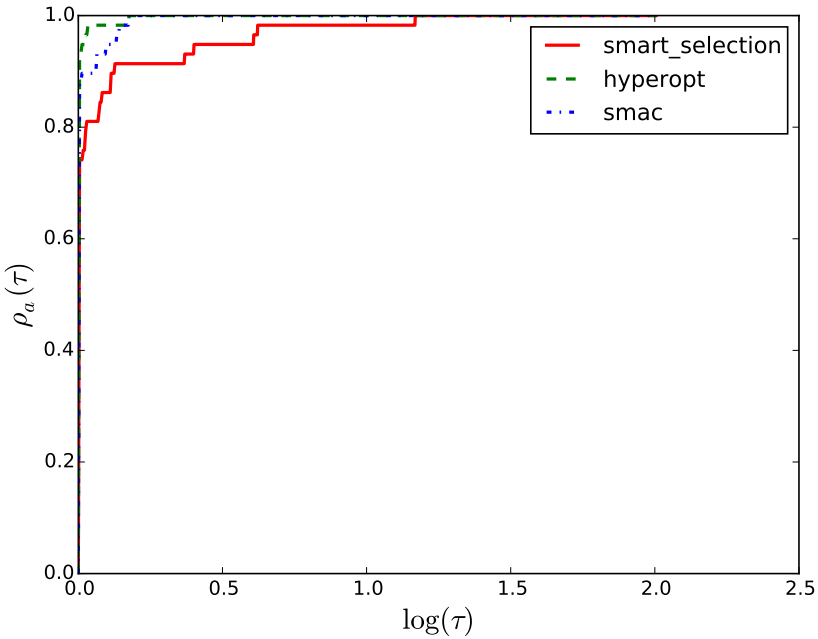
where $\bar{y}$ is mean over training set

Dolan-Moré curves

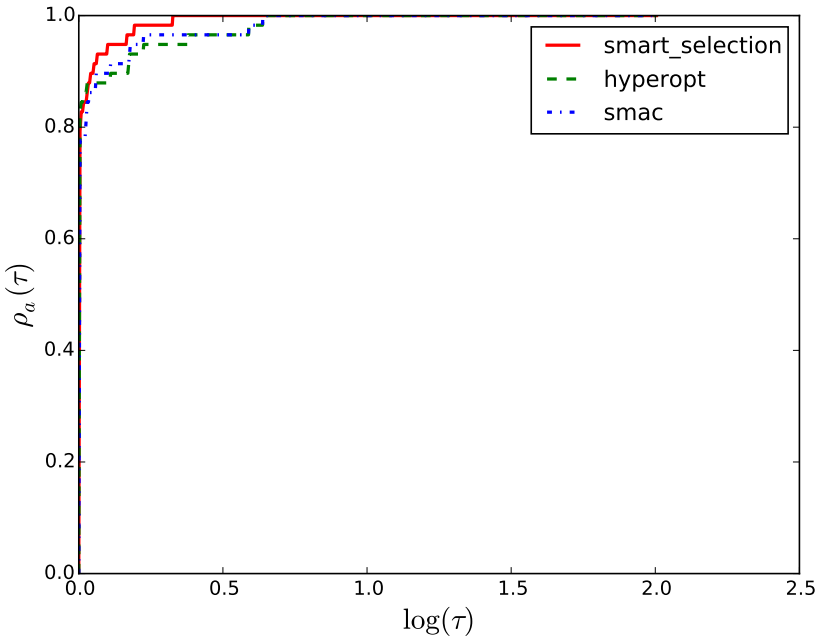
- T problems, A algorithms
- e_{ta} — approximation error (or training time) of a-th algorithm on t-th problem
- $\tilde{e}_t = \min_a e_{ta}$

$$\rho_a(\tau) = \frac{\#\{t : e_{ta} < \tau \tilde{e}_t\}}{T}$$

- The higher the curve is the better works corresponding algorithm
- $\rho_a(1)$ — fraction of problems for which a-th algorithm worked the best



(a) Cross-validation errors



(b) Test errors

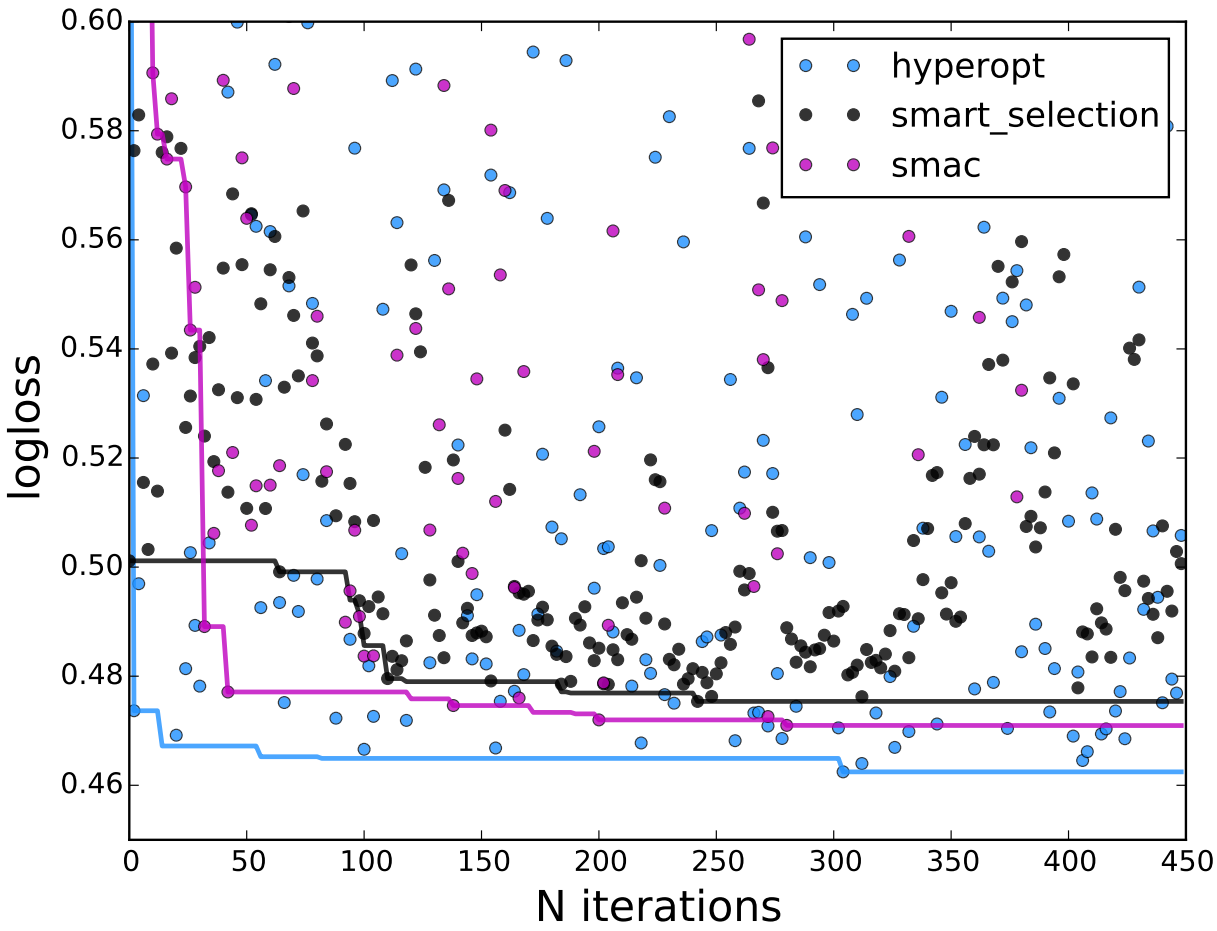
Experimental Setup for Classification

- Otto Group Product Classification Challenge from kaggle
- XGBoost is used for classification
- Comparing algorithms:
  - **Hyperopt** (uses **TPE**)
  - **SMAC** (uses **Random Forest**)
  - **SmartSelection** (uses **GP**)
- Quality criterion:

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log(p_{ij}),$$

where N is the number of products in the test set, M is the number of class labels, y_{ij} is 1 if observation i is in class j and 0 otherwise, and p_{ij} is the predicted probability that observation i belongs to class j.

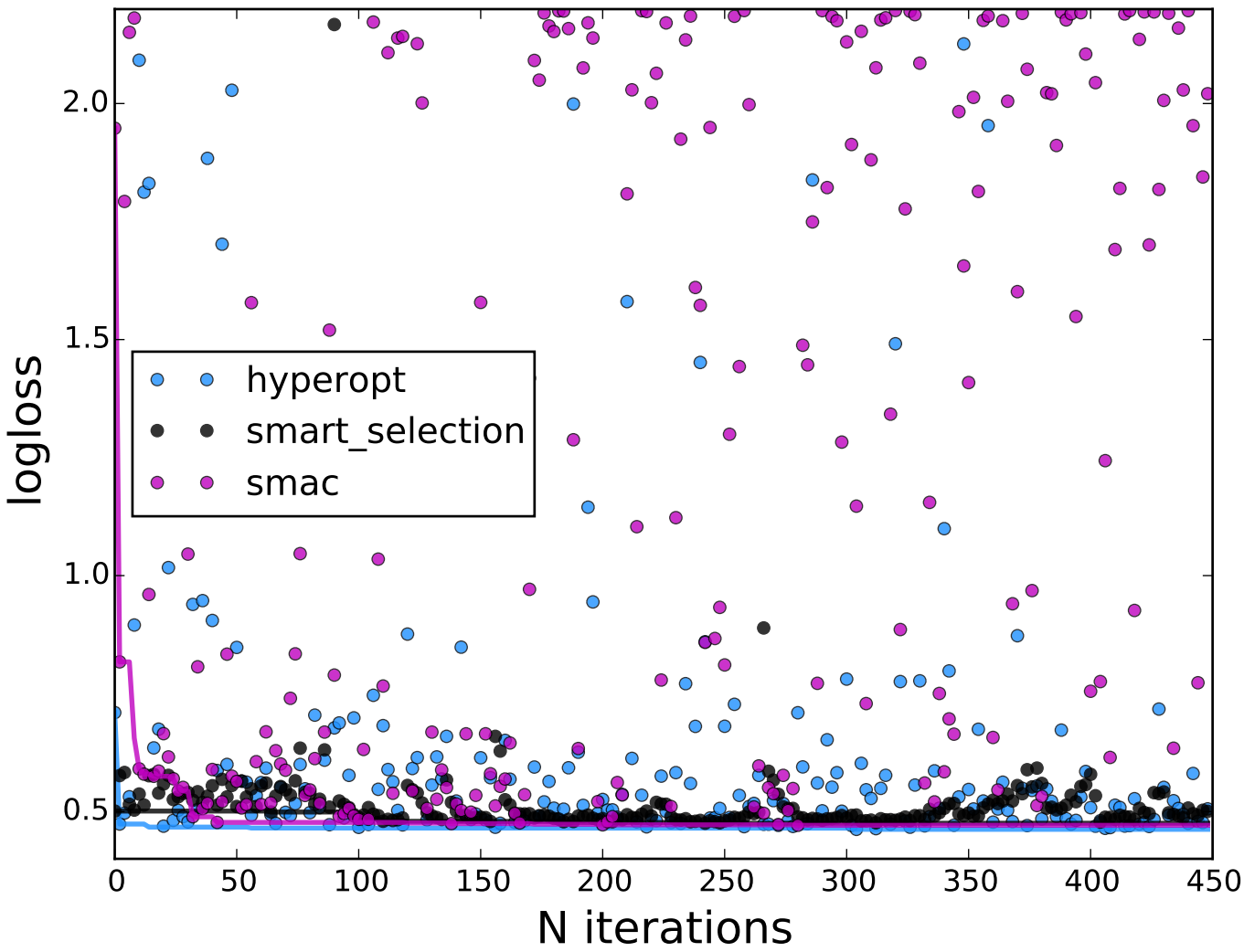
History of evaluations



Smart Selection	0.4757
SMAC	0.4710
Hyperopt	0.4624

Table : Best errors

History of evaluations



Thank you for your attention!