# Other SMBO Algorithms

1. Bayesian optimization

* Maximize or minimize an unknown (black-box) objective function f(x)
* Where x are the hyperparams
* F(x) is costly to compute
* Approximate f(x) with a surrogate
* Gaussian processes (section 6)
* **Random Forests (SMAC) and GBMs**
* **TPE, non-standard Bayesian optimization**

1. SMBO

* Sample several hyperparam combinations -> determine f(x) for those combinations -> decide where to sample next (with acquisition function) -> determine f(x) for those combinations ->…

# SMAC (Sequential Model-based Algorithm Configuration)

1. Bayesian Hyperparam optimization

* See figure in slides
* Red – f(x) – the hyperparam response surface
* Green – our estimation of f(x): In Bayes terms P(y | x) with y is model score f(x) and x are hyperparams
* Gaussian Process
* Random Forests
* Gradient boosted trees

1. SMAC

* Approximating f(x) is a regression problem
* Hyperparams are the input ‘predictive’ variables
* f(x) is response variable or target
* Predict f(x) for unseen values of the hyperparams
* Decide where to sample next -> Acquisition functions

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Hyperparam 1 | Hyperparam 2 | Hyperparam 3 | Hyperparam 4 | F(x) – model performance |
| 0.1 | 1 | 512 | 32 | 0.65 |
| 0.01 | 2 | 48 | 16 | 0.78 |
| 0.005 | 3 | 12 | 8 | 0.89 |
| 0.03 | 2 | 250 | 1 | 0.66 |

1. Decision tree / Random Forests – mean and error

* With a trained decision tree, every new observation will be allocated to an end node
* In the trained tree, each end node contains a bunch of training observations
* It is possible to derive a mean and std for f(x)
* Similar procedure for Random Forests

1. Acquisition functions – scikit-optimize

* Probability of Improvement (PI)
* Expected Improvement (EI)
* Upper (Lower) confidence bound (UCB or LCB)

# Tree-structured Parzen Estimators (TPE)

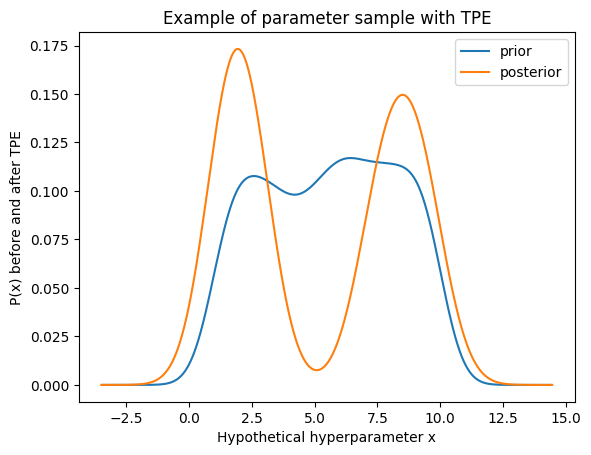
1. Bayes’ rule

* Y is the score -> f(x)
* X is the hyperparameters
* **Approximate P(y|x)** with surrogate function – Gaussian process
* Approximate f(x), the probability of the score given by the hyperparams with a surrogate model (Gaussian process)

1. TPE

* **Approximate P(x|y)** with surrogate function
* Approximate the probability of the hyperparameters using 2 different equations (functions)
* Guides where to sample next for hyperparameters

1. Intuition

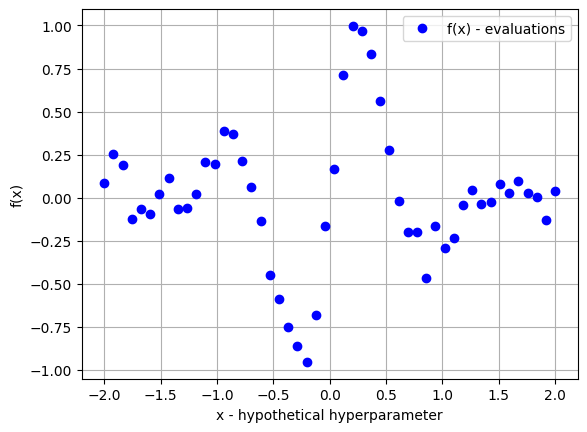


* With TPE, we’re trying to find the spaces within the original distribution that we give as a prior, that are actually more promising to improve the performance of the model

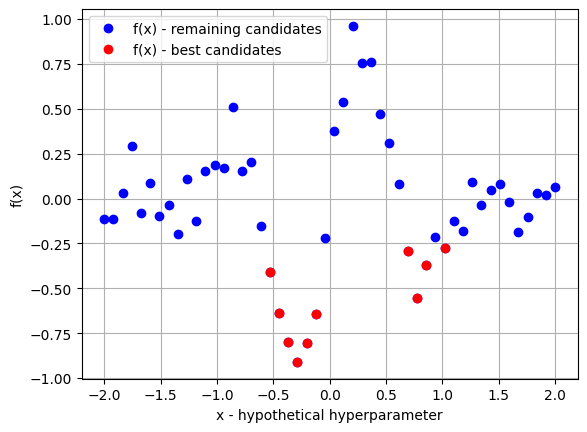
# TPE procedure

1. TPE – procedure
2. TPE – Steps (see figures in slides)

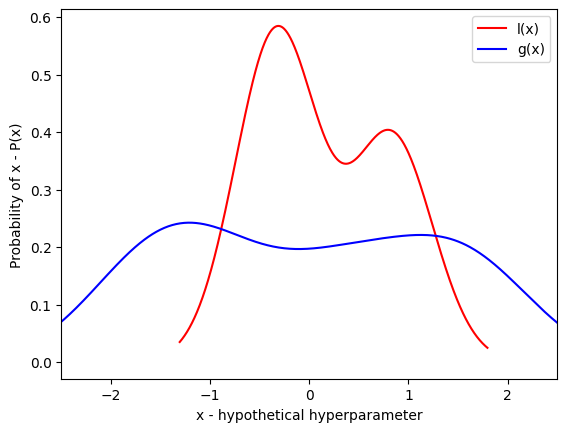
* Step 1 – Sample f(x): Evaluate f(x) at different values of the hyperparams



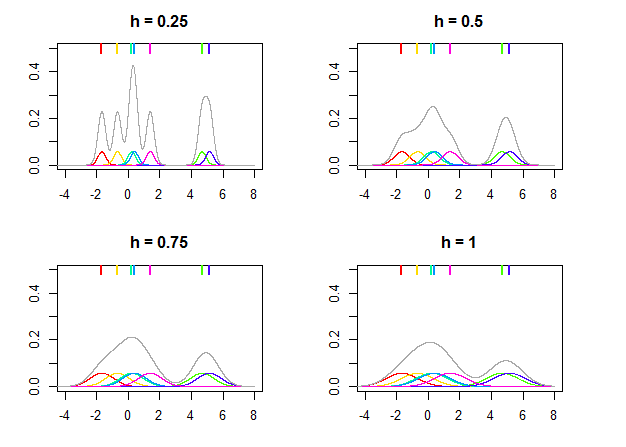
* Step 2 – Divide observations: Separate observations in 2 groups:



* First group: observations with best scores, smallest values of f(x)
* Second group: the rest of the observations
* Step 3 – Estimate distribution of x



* Estimate the distributions of each hyperparam and each group
* Distributions are estimated by Parzen windows, which is another name for kernel density estimation
* The name Parzen estimators in TPE
* Parzen Windows:



* Step 4 – Estimate EI
* Draw samples from L(x)
* Determine EI
* Find max EI
* To maximize EI we would like points x (hyperparams) with high prob under l(x) and low prob under g(x)
* Instead of optimizing 1 function for all the hyperparam space together like Gaussian processes or RF, here we **estimate distributions for each one of the hyperparams individually** (e.g., 20 hyperparams – repeat this process 20 times)

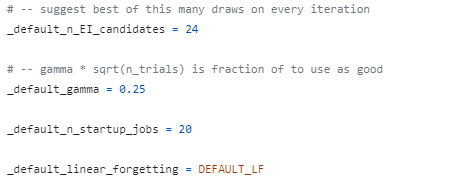
# TPE Hyperparameters

1. TPE procedure (see previous section)

* Sample f(x): How many hyperparam vales should we examine at random?
* Sort observations: How many observations should we send to group 1 (quantiles usually used, so which quantile)?
* Estimate distribution: Parzen window size?
* Draw samples: How many?

1. TPE.suggest (see slides) & change parameters

* Default:



* Change parameters:

from functools import partial

from hyperopt import tpe, fmin

# ...

# this line effectively creates a new function that is the same as tpe.suggest but with some parameter values already set

algo = partial(tpe.suggest, n\_startup\_jobs=125, n\_EI\_candidates=50)

# pass in the new `algo` function that you created instead

best = fmin(fn=ae, space=parameterspace, algo=algo, trials=bayes\_trials, max\_evals=100)

# TPE - Why Tree-structured?

1. GP vs TPE

|  |  |
| --- | --- |
| **GP** | **TPE** |
| Surrogate model for f(x) – P(y|x)  F(x) takes a value given all hyperparams | Surrogate model for P(x|y)  1 probability per hyperparameter  Hyperparam spaces can be nested |
| GP gives us the whole view of the hyperparam space for one particular point of f(x) | TPE guides the search hyperparam per hyperparam |

1. Why tree-structured

* Because TPE estimates a distribution for each hyperparam, it will only estimate distributions for parameters when they ‘exist’
* Example: see slide
* Parameters only make sense inside each node / sub-node

# Discussion: Bayesian optimization and Basic search

1. Search strategies

* Basic
* Manual search
* Grid search
* Random search
* SMBO – Bayesian
* GP
* SMAC
* TPE

1. Searches in a nutshell

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Grid Search** | **Random Search** | **Bayesian** |
| **Parallelize** | Yes | Yes | ish |
| **Effective in high dimension** | No | Yes | Yes |
| **Effective in (very) low dimension** | Yes | - | - |
| **Suited for continuous hyperparameters** | No | Yes | Yes |
| **Complex ML models** | No | ish | Yes |
| **Resource intensive** | - | Yes | No |
| **Wall-clock time intensive** | - | No | Yes |
| **Hyperparameter values** | Manually defined | Drawn from a distribution | Drawn from a distribution |

1. Bayesian searches in a nutshell

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **GP** | **SMAC** | **TPE** | **Random** |
| **Approximate** | F(x) | F(x) | Hyperparam distribution | - |
| **Surrogate** | Gaussian processes | Random Forests | Density kernels (tree parzen estimators) | - |
| **Hyperparam interaction** | Yes | Yes | No | Yes |
| **Speed** | Less fast | Fast | Fast | - |
| **Implementation** | Scikit-optimize | Scikit-optimize | Hyperopt, Optuna | Sklearn, **Hyperopt, Optuna**, others |
| **Nested spaces** | No | No | Yes | Yes |
| **Performance** | Second best | Best | Third best | Best (if allowed sufficient iterations |

1. **Which method should I use?**

* Training sklearn model
* Random Search (sklearn)
* Bayesian Search (scikit-optimize)
* Training other models (xgb, lightGBM)
* Random Search (scikit-optimize, optuna)
* SMAC (scikit-optimize)
* Nested spaces
* Random Search (optuna, hyperopt)
* TPE (optuna, hyperopt)
* Neural nets
* TPE or other methods (optuna, Keras tuner, ray tune)