BAYESIAN OPTIMIZATION IN JULIA

Google Summer of Code 2023

Name of the project: Implementing Scalable Bayesian Optimization in Julia
Organization: The Julia Language
Supervisor: Johanni Brea

AUTHOR

- Samuel Belko
- enrolled in a master's program in mathematics at Technical University of Munich

CONTACT

- github: samuelbelko
- https://belko.xyz
- samuelbelko@protonmail.com

PROJECT TASKS

Implement Trust Region Bayesian Optimization (TuRBO) algorithm *and* integrate it into an existing package BayesianOptimization.jl, currently implementing traditional Bayesian optimization algorithms in Julia.

RESULTS

BayesianOptimization.jl was refactored into multiple packages such that they

- serve as a framework for implementing BO-like algorithms
  - making it easier to research, test and develop novel BO-like algorithms in Julia
- provide implementations of some sucessful BO-like algorithms
  - traditional methods that work well in low dimensions
  - recent developments aiming to work well in higher dimensions (TuRBO)

The project is currently still in an experimental, partly documented stage but it is already working on examples. All related packages to this project are available in *JuliaBayesianOptimization* organization on github.

Moving the Gaussian process surrogate where it belongs

- TuRBO as well as traditional BO methods depend crucially on Gaussian proceses (GPs) with hyperparameter optimization
- there is a package AbstractGPs.jl that provides tools for working with GPs but there are some utilities that need to be added
  - idea: isolate them into a package and integrate the package into an existing system of surrogates
    - refactor existing SurrogatesAbstractGPs module in Surrogates.jl package (SciML org) into own package and extend it
- Surrogates.jl formalizes what a deterministic surrogate is via the AbstractSurrogate interface
  - idea: extend the interface to include stochastic surrogates, in particular SurrogatesAbstractGPs.jl should use the same, unified surrogate API
    - move and rework AbstractSurrogate interface into SurrogatesBase.jl, where only surrogate-related generic function definitions are provided
- this is currently *work-in-progress*

Bayesian optimization (BO) algorithms

- a family of methods for *sample efficient* search for global optimizers
- useful for optimizing objective functions that exhibit one or more of the following properties:
  - are expensive to evaluate
  - are restricted by a limited evaluation budget
  - evaluations can be corrupted by noise
  - are non-convex
  - are black box functions, i.e., lack a closed-form expression
  - there is no efficient mechanism for estimating the gradient
- traditional methods work well for continuous domains of less than 20 dimensions, as mentioned in [1]
- “killer app”: hyperparameter tuning
- commonly used with a Gaussian process surrogate, for details of the algorithms, please see below

Trust Region Bayesian Optimization (TuRBO) algorithm

- belongs to recent advances in Bayesian optimization that aim to work well in higher dimensions
- was first introduced in [2]
- maintains local Gaussian process surrogates in parallel that are accurate inside respective trust regions that adapt in size and postion based on performance
  - improved approximating capability of heterogeneous objective functions (different hyperparameters for each trust region)
  - effectively deal with large posterior uncertainty in higher dimensional problems and related over exploration behavior occuring in traditional methods
  - enjoy local Bayesian optimization properties:
    - robust to noice, sample efficient
- for details of the algorithm, please see below

`AbstractBayesianOptimization.jl`

- implements a prototypical BO-like algorithm using abstract data types
- is motivated by a mathematical framework introduced in chapter 1 of the book [3]
- implements an OptimizationHelper that provides basic utilities for defining unconstrained problems and logging progress
- currently supports gradient free optimization only
- up to some datails, it implements the following while loop inside exported function optimize!(dsm, policy, oh; verbose=true)

```
while run_optimization(dsm, oh)
    xs = next_batch!(policy, dsm, oh)
    ys = evaluate_objective!(oh, xs)
    update!(dsm, oh, xs, ys)
end
```

`dsm isa AbstractDecisionSupportModel`

- aggregates evaluations into some kind of model

`policy isa AbstractPolicy`

- uses a decision support model to decide where to evaluate the objective function next

`oh isa OptimizationHelper`

- logs best observed optimizer, history, overall time
- normalizes input, i.e., transforms the problem into a maximization problem on a unit cube
- unified interface for getting results, history etc..

Notable functions from the interface:

- initialize!(dsm, oh)
- next_batch!(policy, dsm, oh)
- update!(dsm, oh, xs, ys)

`BayesianOptimization.jl`

- refactors the original BayesianOptimization.jl to fit the abstract framework above
- implements initialize! to perform initial sampling
- implements BasicGP <: AbstractDecisionSupportModel
  - maintains a global Gaussian process surrogate
- implements various policies that are of the following form:
  - evaluate the objective at a maximizer of an acquistion function, that is defined on the domain of the objective function using aggregated information in BasicGP
  - acquisition functions quantify how useful an evaluation at a point in the domain could be
    - search for *the optimal* compromise between exploration and exploitation, i.e., sampling where we are uncertain about the objective and sampling where we expect high objective values, respectively
  - currently implemented acquisition functions:
    - expected improvement
    - max mean
    - mutual information
    - probability of improvement
    - Thompson sampling
    - upper confidence bound

`TrustRegionBayesianOptimization.jl`

- evaluate objective in batches
- implements a method initialize! for initializing local models, i.e, performing initial sampling
- implements Turbo <: AbstractDecisionSupportModel
  - maintains local Gausian process surrogates and corresponding trust regions
  - strategy for adjusting trust regions:
    - if we could improve current optimizer via the current batch, enlarge trust region, else shrink trust region
    - if a trust region has converged, initalize a new one
    - set center to current optimizer / max. posterior mean in case of noice
    - run hyperparameter optimization, use lengthscales parameter for side ratio of the box trust region
- implements TurboPolicy <: AbstractPolicy
  - strategy for advancing local models in parallel:
    - use Thomson sampling across all trust regions simultaneously
      - automatically evaluate the objective function more frequently in trust regions that perform better

LITERATURE

[1] Peter I. Frazier. A Tutorial on Bayesian Optimization. 2018
[2] David Eriksson, Michael Pearce, Jacob Gardner, Ryan D. Turner, Matthias Poloczek. Scalable Global Optimization via Local Bayesian Optimization. 2019
[3] Roman Garnett, Bayesian Optimization. 2023. Cambridge University Press