A Survey of Model-based Methods for Global Optimization

Thomas Bartz-Beielstein

SPOTSeven Lab
www.spotseven.de

Technology
Arts Sciences
TH Köln

BIOMA
Keywords

- Abelson & Sussmann
- Wolpert
- Netflix
- Deep Learning
Agenda

- Part 1: Basics
- Part 2: Example
- Part 3: Considerations
- Part 4: Example
- Part 5: Conclusion
Overview

Introduction

Stochastic Search Algorithms

Quality Criteria: How to Select Surrogates

Examples

Ensembles: Considerations

SPO2 Part 2

Stacking: Considerations
Introduction

Model-based optimization (MBO)

- Prominent role in today's modeling, simulation, and optimization processes
- Most efficient technique for expensive and time-demanding real-world optimization problems
- Engineering domain, MBO is an important practice
- Recent advances in
  - computer science,
  - statistics, and
  - engineering
  - in combination with progress in high-performance computing
- Tools for handling problems, considered unsolvable only a few decades ago
Global optimization (GO)

- GO can be categorized based on different criteria.
- Properties of problems
  - continuous versus combinatorial
  - linear versus nonlinear
  - convex versus multimodal, etc.
- We present an algorithmic view, i.e., properties of algorithms
- The term GO will be used in this talk for algorithms that are trying to find and explore global optimal solutions with complex, multimodal objective functions [Preuss, 2015].
- GO problems are difficult: nearly no structural information (e.g., number of local extrema) available
- GO problems belong to the class of black-box functions, i.e., the analytic form is unknown
- Class of black-box function contains also functions that are easy to solve, e.g., convex functions
Introduction

Problem

- **Optimization problem** given by

\[
\text{Minimize: } f(\vec{x}) \quad \text{subject to } \vec{x}_l \leq \vec{x} \leq \vec{x}_u,
\]

where \( f : \mathbb{R}^n \to \mathbb{R} \) is referred to as the *objective function* and \( \vec{x}_l \) and \( \vec{x}_u \) denote the lower and upper bounds of the search space (region of interest), respectively.

- Setting arises in many real-world systems:
  - when the explicit form of the objective function \( f \) is not readily available,
  - e.g., user has no access to the source code of a simulator

- We cover *stochastic (random) search algorithms*, deterministic GO algorithms are not further discussed

- *Random* and *stochastic* used synonymously
Taxonomy of model-based approaches in GO

[1] Deterministic
[2] Random Search
  [2.1] Instance based
  [2.2] Model based optimization (MBO)
    [2.2.1] Distribution based
    [2.2.2] Surrogate Model Based Optimization (SBO)
      [2.2.2.1] Single surrogate based
      [2.2.2.2] Multi-fidelity based
      [2.2.2.3] Evolutionary surrogate based
      [2.2.2.4] Ensemble surrogate based
Overview

Introduction

Stochastic Search Algorithms

Quality Criteria: How to Select Surrogates

Examples

Ensembles: Considerations

SPO2 Part 2

Stacking: Considerations
Random Search

- Stochastic search algorithm: Iterative search algorithm that uses a stochastic procedure to generate the next iterate
- Next iterate can be
  - a candidate solution to the GO or
  - a probabilistic model, where solutions can be drawn from
- Do not depend on any structural information of the objective function such as gradient information or convexity ⇒ robust and easy to implement
- Stochastic search algorithms can further be categorized as
  - instance-based or
  - model-based algorithms [Zlochin et al., 2004]
[2.1] Instance-based Algorithms

- Instance-based algorithms: use a single solution, \( \vec{x} \), or population, \( P(t) \), of candidate solutions
- Construction of new candidates depends explicitly on previously generated solutions
- Examples: Simulated annealing, evolutionary algorithms

1: \( t = 0 \). InitPopulation(\( P \)).
2: Evaluate(\( P \)).
3: \textbf{while} not TerminationCriterion() \textbf{do}
4: \quad Generate new candidate solutions \( P'(t) \) according to a specified random mechanism.
5: \quad Update the current population \( P(t+1) \) based on population \( P(t) \) and candidate solutions in \( P'(t) \).
6: \quad Evaluate(\( P(t+1) \)).
7: \quad \( t = t + 1 \).
8: \textbf{end while}
[2.2] MBO: Model-based Algorithms

- MBO algorithms: generate a population of new candidate solutions $P'(t)$ by sampling from a model.
- In statistics: model $\equiv$ distribution.
- Model (distribution) reflects structural properties of the underlying true function, say $f$.
- Adapting the model (or the distribution), the search is directed into regions with improved solutions.
- One of the key ideas: replacement of expensive, high fidelity, fine grained function evaluations, $f(\vec{x})$, with evaluations, $\hat{f}(\vec{x})$, of an adequate cheap, low fidelity, coarse grained model, $M$. 
[2.2.1] Distribution-based Approaches

- Metamodel is a distribution
- Generate a sequence of iterates (probability distributions) \{p(t)\} with the hope that

\[ p(t) \rightarrow p^* \text{ as } t \rightarrow \infty, \]

where \( p^* \): limiting distribution, assigns most of its probability mass to the set of optimal solutions

- Probability distribution is propagated from one iteration to the next
- Instance-based algorithms propagate candidate solutions

1: \( t = 0 \). Let \( p(t) \) be a probability distribution.
2: \textbf{while} not TerminationCriterion() \textbf{do}
3: Randomly generate a population of candidate solutions \( P(t) \) from \( p(t) \).
4: Evaluate\( (P(t)) \).
5: Update the distribution using population (samples) \( P(t) \) to generate a new distribution \( p(t + 1) \).
6: \( t = t + 1 \).
7: \textbf{end while}
[2.2.1] Estimation of distribution algorithms (EDA)

- EDA: very popular in the field of *evolutionary algorithms* (EA)
- Variation operators such as mutation and recombination replaced by a distribution based procedure:
  - Probability distribution estimated from promising candidate solutions from the current population ⇒ generate new population
- Larraaga and Lozano [2002] review different ways for using probabilistic models
- Hauschild and Pelikan [2011] discuss advantages and outline many of the different types of EDAs
- Hu et al. [2012] present recent approaches and a unified view
[2.2.2] Focus on Surrogates

- Although distribution-based approaches play an important role in GO, they will not be discussed further in this talk.
- We will concentrate on surrogate model-based approaches.
- Origin in statistical design and analysis of experiments, especially in response surface methodology [G E P Box, 1951, Montgomery, 2001].
2.2.2] Surrogate Model-based Approaches

- In general: Surrogates used, when outcome of a process cannot be directly measured.
- Imitate the behavior of the real model as closely as possible while being computationally cheaper to evaluate.
- Surrogate models also known as:
  - the cheap model, or
  - a response surface,
  - meta model,
  - approximation,
  - coarse grained model.
- Simple surrogate models constructed using a data-driven approach.
- Refined by integrating additional points or domain knowledge, e.g., constraints.
[2.2.2] Surrogate Model-based Approaches

- Validation step (e.g., via CV) is optional
- Samples generated iteratively to improve the surrogate model accuracy
2.2.2 Surrogate Model Based Optimization (SBO) Algorithm

1. \( t = 0 \). \text{InitPopulation}(P(t))
2. \text{Evaluate}(P(t))
3. \textbf{while} not \text{TerminationCriterion()} \textbf{do}
4. \quad \text{Use } P(t) \text{ to build a cheap model } M(t)
5. \quad \text{Evaluate}(P'(t+1))
6. \quad \text{GlobalSearch}(M(t))
7. \quad P(t+1) \subseteq P(t) + P'(t+1)
8. \quad t = t + 1
9. \textbf{end while}
[2.2.2] Surrogates

- Wide range of surrogates developed in the last decades ⇒ complex design decisions [Wang and Shan, 2007]:
  - (a) Metamodels
  - (b) Designs
  - (c) Model fit

- (a) Metamodels:
  - Classical regression models such as polynomial regression or response surface methodology [G E P Box, 1951, Montgomery, 2001]
  - support vector machines (SVM) [Vapnik, 1998],
  - neural networks [Zurada, 1992],
  - radial basis functions [Powell, 1987], or
  - Gaussian process (GP) models, design and analysis of computer experiments, Kriging [Schonlau, 1997], [ Büche et al., 2005], [Antognini and Zagoraiou, 2010], [Kleijnen, 2009], [Santner et al., 2003]

- Comprehensive introduction to SBO in [Forrester et al., 2008]
[2.2.2] Surrogates: Popular metamodeling techniques

- (b) Designs [Wang and Shan, 2007]:
  - Classical
    - Fractional factorial
    - Central composite
    - Box-Behnken
    - A-, D-optimal (alphabetically)
    - Plackett-Burmann
  - Space filling
    - Simple grids
  - Latin hypercube
  - Orthogonal
  - Uniform
  - Minimax and Maximin
  - Hybrid methods
  - Random or human selection
  - Sequential methods
[2.2.2] Surrogates: Popular metamodeling techniques

- (c) Model fitting [Wang and Shan, 2007]:
  - Weighted least squares regression
  - Best linear unbiased predictor (BLUP)
  - Likelihood
  - Multipoint approximation
  - Sequential metamodeling
  - Neural networks: backpropagation
  - Decision trees: entropy
[2.2.2] Applications of SBO

- One of the most popular application areas for SBO:
  - Simulation-based design of complex engineering problems
    - *computational fluid dynamics* (CFD)
    - *finite element modeling* (FEM) methods
  - Exact solutions → solvers require a large number of expensive computer simulations
- Two variants of SBO
  - (i) metamodel [2.2.2.1]: uses one or **several** different metamodels
  - (ii) multi-fidelity approximation [2.2.2.2]: uses several instances with different parameterizations of the **same** metamodel
[2.2.2.1] Applications of Metamodels and [2.2.2.2] Multi-fidelity Approximation

- Meta-modeling approaches
  - 31 variable helicopter rotor design [Booker et al., 1998]
  - Aerodynamic shape design problem [Giannakoglou, 2002]
  - Multi-objective optimal design of a liquid rocket injector [Queipo et al., 2005]
  - Airfoil shape optimization with CFD [Zhou et al., 2007]
  - Aerospace design [Forrester and Keane, 2009]

- Multi-fidelity Approximation
  - Several simulation models with different grid sizes in FEM [Huang et al., 2015]
  - Sheet metal forming process [Sun et al., 2011]

- “How far have we really come?” [Simpson et al., 2012]
[2.2.2.3] Surrogate-assisted Evolutionary Algorithms

- **Surrogate-assisted EA**: EA that decouple the evolutionary search and the direct evaluation of the objective function.
- Cheap surrogate model replaces evaluations of expensive objective function.
[2.2.2.3] Surrogate-assisted Evolutionary Algorithms

- Combination of a genetic algorithm and neural networks for aerodynamic design optimization [Hajela and Lee, 1997]
- Approximate model of the fitness landscape using Kriging interpolation to accelerate the convergence of EAs [Ratle, 1998]
- Evolution strategy (ES) with neural network based fitness evaluations [Jin et al., 2000]
- Surrogate-assisted EA framework with online learning [Zhou et al., 2007]
- Not evaluate every candidate solution (individual), but to just estimate the objective function value of some of the neighboring individuals [Branke and Schmidt, 2005]
- Survey of surrogate-assisted EA approaches [Jin, 2003]
- SBO approaches for evolution strategies [Emmerich et al., 2002]
Instead of using one surrogate model only, several models $M_i$, $i = 1, 2, \ldots, p$, generated and evaluated in parallel.

Each model $M_i : X \rightarrow y$ uses:
- same candidate solutions, $X$, from the population $P$ and
- same results, $y$, from expensive function evaluations.

Multiple models can also be used to partition the search space:
- The tree-based Gaussian process (TGP): regression trees to partition the search space, fit local GP surrogates in each region [Gramacy, 2007].
- Tree-based partitioning of an aerodynamic design space, independent Kriging surfaces in each partition [Nelson et al., 2007].

Combination of an evolutionary model selection (EMS) algorithm with expected improvement (EI) criterion: select best performing surrogate model type at each iteration of the EI algorithm [Couckuyt et al., 2011].
Ensembles of surrogate models gained popularity:
- Adaptive weighted average model of the individual surrogates [Zerpa et al., 2005]
- Use the best surrogate model or a weighted average surrogate model instead [Goel et al., 2006]
- Weighted-sum approach for the selection of model ensembles [Sanchez et al., 2006]
  - Models for the ensemble chosen based on their performance
  - Weights are adaptive and inversely proportional to the local modeling errors
Overview

Introduction

Stochastic Search Algorithms

Quality Criteria: How to Select Surrogates

Examples

Ensembles: Considerations

SPO2 Part 2

Stacking: Considerations
Model Refinement: Selection Criteria for Sample Points

- An initial model refined during the optimization ⇒ Adaptive sampling
- Identify new points, so-called infill points
- Balance between
  - exploration, i.e., improving the model quality (related to the model, global), and
  - exploitation, i.e., improving the optimization and determining the optimum (related to the objective function, local)

- Expected improvement (EI): popular adaptive sampling method [Mockus et al., 1978], [Jones et al., 1998]
Model Selection Criteria

- EI approach handles the initialization and refinement of a surrogate model
- But not the selection of the model itself
- Popular *efficient global optimization* (EGO) algorithm uses a Kriging model
  - Because Kriging inherently determines the prediction variance (necessary for the EI criterion)
- But there is no proof that Kriging is the best choice
- Alternative surrogate models, e.g., neural networks, regression trees, support vector machines, or lasso and ridge regression may be better suited
- An *a priori* selection of the best suited surrogate model is conceptually impossible in the framework treated in this talk, because of the black-box setting
Single or Ensemble

- Regarding the model choice, the user can decide whether to use
  - one single model, i.e., one unique global model or
  - multiple models, i.e., an ensemble of different, possibly local, models

The static SBO uses a single, global surrogate model, usually refined by *adaptive sampling*, but did not change ⇒ category [2.2.2.1]
Criteria for Selecting a Surrogate

- Here, we do **not consider** the selection of a new sample point (as done in EI)
- Instead: Criteria for the selection of one (or several) surrogate models
- Usually, surrogate models chosen according to their estimated true error [Jin et al., 2001], [Shi and Rasheed, 2010]
- Commonly used performance metrics:
  - *mean absolute error* (MAE)
  - *root mean square error* (RMSE)
- Generally, attaining a surrogate model that has **minimal error** is the desired feature
- Methods from statistics, statistical learning [Hastie, 2009], and machine learning [Murphy, 2012]:
  - Simple holdout
  - Cross-validation
  - Bootstrap
Overview

Introduction

Stochastic Search Algorithms

Quality Criteria: How to Select Surrogates

Examples

Ensembles: Considerations

SPO2 Part 2

Stacking: Considerations
Examples

Criteria for Selecting a Surrogate: Evolvability

- Model error is **not the only criterion** for selecting surrogate models
- *Evolvability learning of surrogates* approach (EvoLS) [Le et al., 2013]:
  - Use fitness improvement for determining the quality of surrogate models
- EvoLS belongs to the category of *surrogate-assisted evolutionary algorithms* ([2.2.2.3])
- Distributed, local information
Evolvability Learning of Surrogates

- **EvoLS**: select a surrogate models that enhance search improvement in the context of optimization

- **Process information about the**
  - (i) different fitness landscapes,
  - (ii) state of the search, and
  - (iii) characteristics of the search algorithm to statistically determine the so-called *evolvability* of each surrogate model

- **Evolvability of a surrogate model estimates the expected improvement of the objective function value that the new candidate solution has gained after a local search has been performed on the related surrogate model** [Le et al., 2013]
Evolvability

- Local search: After recombination and mutation, a local search is performed
- It uses an individual local meta-model, $M$, for each offspring
- The local optimizer, $\varphi_M$, uses an offspring $\vec{y}$ as an input and returns $\vec{y}^*$ as the refined offspring
- Evolvability measure can be estimated as follows [Le et al., 2013]:

\[ Ev_M(\vec{x}) = f(\vec{x}) - \sum_{i=1}^{K} f(\vec{y}_i^*) \times w_i(\vec{x}) \]

with weights (selection probabilities of the offsprings):

\[ w_i(\vec{x}) = \frac{P(\vec{y}_i | P(t), \vec{x})}{\sum_{j=1}^{K} P(\vec{y}_j | P(t), \vec{x})} \]
SPO

- EvoLS: distributed, local information. Now: more centralized, global information ⇒ *sequential parameter optimization* (SPO)
- Goal: Analysis and *understanding* of algorithms
- Early versions of the SPO [Bartz-Beielstein, 2003, Bartz-Beielstein et al., 2005] combined methods from
  - *design of experiments* (DOE) [Pukelsheim, 1993]
  - *response surface methodology* (RSM) [Box and Draper, 1987, Montgomery, 2001]
  - *design and analysis of computer experiments* (DACE) [Lophaven et al., 2002, Santner et al., 2003]
  - regression trees [Breiman et al., 1984]
- Also: SPO as an optimizer
SPO

- SPO: sequential, model based approach to optimization
- Nowadays: established parameter tuner and an optimization algorithm
- Extended in several ways:
  - For example, Hutter et al. [2013] benchmark an SPO derivative, the so-called *sequential model-based algorithm configuration* (SMAC) procedure, on the BBOB set of blackbox functions.
  - Small budget of $10 \times d$ evaluations of $d$-dimensional functions, SMAC in most cases outperforms the state-of-the-art blackbox optimizer CMA-ES
The most recent version, SPO2, is currently under development

Integration of state-of-the-art ensemble learners

SPO2 ensemble engine:

- Portfolio of surrogate models
- regression trees and random forest, least angle regression (lars), and Kriging
- Uses cross validation to select an improved model from the portfolio of candidate models
- Creates a weighted combination of several surrogate models to build the improved model
- Use stacked generalization to combine several level-0 models of different types with one level-1 model into an ensemble [Wolpert, 1992]
- Level-1 training algorithm: simple linear model
Promising preliminary results

SPO2 ensemble engine can lead to significant performance improvements

Rebolledo Coy et al. [2016] present a comparison of different data driven modeling methods
  - Bayesian model
  - Several linear regression models
  - Kriging model
  - Genetic programming

Models build on industrial data for the development of a robust gas sensor

Limited amount of samples and a high variance
Two sensors are compared

1st sensor (MSE)
- Linear model (0.76), OLS (0.79), Lasso (0.56), Kriging (0.57), Bayes (0.79), and genetic programming (0.58)
- SPO2 0.38

2nd sensor (MSE)
- Linear model (0.67), OLS (0.80), Lasso (0.49), Kriging (0.49), Bayes (0.79), and genetic programming (0.27)
- SPO2 0.29
The following part of this talk is based on an interactive jupyter notebook [Pérez and Granger, 2007]

The next slides (43 - 56) summarize the output from the jupyter notebook
Examples

Preparation [jupyter]

- Basically
  - 1. import libraries and
  - 2. set the SPO2 parameters, i.e., the number of folds
- Code implements ideas from Wolpert [1992], based on Olivetti [2012]
- Libraries shown explicitly, because we will comment on this topic later!

```python
import matplotlib.pyplot as plt
from IPython.display import set_matplotlib_formats
from sklearn.linear_model import LassoCV, LassoLarsCV, LassoLarsIC...
import statsmodels.formula.api as sm
```
The Complete Data Set [jupyter]

- Problem description in Rebolledo Coy et al. [2016]
- One training data set and one test data set

```python
In [2]: dfTrain = read_csv('/Users/bartz/workspace/svnbib/Python.d/projects/pyspot2/data/training.csv')
    dfTest = read_csv('/Users/bartz/workspace/svnbib/Python.d/projects/pyspot2/data/validation.csv')
```

RangeIndex: 80 entries, 0 to 79
Data columns (total 9 columns):
X1 80 non-null float64
X2 80 non-null float64
..  
X7 80 non-null float64
Y1 80 non-null float64
Y2 80 non-null float64
dtypes: float64(9)
memory usage: 5.7 KB

```
0 -1.132054 -1.206144 ...
0.132054 -1.206144 ...
```
Data Used in this Study [Jupyter]

- Here, we consider data from the seconds sensor.
- There are seven input values and one output value ($y$).
- The goal of this study: predict the outcome $y$ using the seven input measurements ($X_1, \ldots, X_7$).
- Output ($y$) plotted against input ($X_1, \ldots, X_7$).
2. Cross Validation (CV Splits) [jupyter]

- The training data are split into folds
- KFold divides all the samples in $k = n_{folds}$ folds, i.e., groups of samples of equal sizes (if possible)
- If $k = n$, this is equivalent to the Leave One Out strategy
- Prediction function is learned using $(k - 1)$ folds, and the fold left out is used for test
3. Models in the Ensemble [jupyter]

- **Linear Regression**
  - 1. Normalized predictors
    - LinearRegression(normalize=False)
    - LinearRegression(normalize=True)
  - 2. Intercept
    - LinearRegression(normalize=False, fit_intercept = False)
    - LinearRegression(normalize=True, fit_intercept = False)

- **Random Forest**
  - RandomForestRegressor(n_estimators = 10, random_state=0),
  - RandomForestRegressor(n_estimators = 100, oob_score = True, random_state=2),

- **Lasso**
  - Lasso(alpha=0.1, fit_intercept = False)
  - LassoCV(positive = True)
3. Models in the Ensemble [jupyter]

- Gaussian Processes
  - The kernel specifying the covariance function of the GP
  - Parameterized with different kernels:
  - For example RBF, Matern, RationalQuadratic, ExpSineSquared, DotProduct, ConstantKernel
  - If none is passed, the kernel RBF() is used as default
  - Kernel’s hyperparameters are optimized during fitting
  - Kernel combinations are allowed.
  - Example:
    ```python
    kern = 1.0 * RBF(length_scale=100.0
                     , length_scale_bounds = (1e-2, 1e3))
    + WhiteKernel(noise_level=1
                  , noise_level_bounds=(1e-10, 1e+1))
    ⇒ GaussianProcessRegressor(kern)
    ```
3. Models in the Ensemble [jupyter]

```python
In [5]: clfs = [ LinearRegression(), RandomForestRegressor()
           #, GaussianProcessRegressor() ]
```
**Matrix Preparation, Dimensions [jupyter]**

- $n$: size of the training set (samples): `X.shape[0]`
- $k$: number of folds for CV: `n_folds`
- $p$: number of models: `len(clfs)`
- $m$: size of the test data (samples): `XTest.shape[0]`

We will use two matrices:

1. $Y_{CV}$ is a $(n \times p)$-matrix. It stores the results from the cross validation for each model. The training set is partitioned into $k$ folds ($n\_folds=k$).
2. $Y_{BT}$ is a $(m \times p)$-matrix. It stores the aggregated results from the cross validation models on the test data. For each fold, $p$ separate models are build, which are used for prediction on the test data. The predicted values from the $k$ folds are averaged for each model, which results in $(m \times p)$ different values.

```python
In [6]: YCV = np.zeros((X.shape[0], len(clfs)))
YBT = np.zeros((XTest.shape[0], len(clfs)))
```
Cross-Validation [jupyter]

- Each of the \( p \) algorithms is run separately on the \( k \) folds
- The training data set is split into folds
- Each fold contains training data (train) and validation data (val)
- For each fold, a model is build using the train data
- The model is used to make predictions on
  - 1. the validation data from the \( k \)-th fold These values are stored in the matrix \( Y_{CV} \)
  - 2. the test data \( \Rightarrow Y_{BT}[i] \)
- The average of these predictions are stored in the matrix \( Y_{BT} \)
The $Y_{CV}$ Matrix [jupyter]

- The $Y_{CV}$ matrix has $p$ columns and $n$ rows
- Each row contains the prediction for the $n$-th sample from the training data set

$$
(80, 2)
[[ 0.98940238 1.72943761] \\
[ 0.14445345 0.17159034] \\
[ -0.0969411 0.86240816] \\
[ 1.24820656 0.27214466] \\
[ 0.0897393 0.1542136 ] \\
... \\
[ -0.56801858 -0.50319222]]
$$
The $Y_{BT}$ Matrix [jupyter]

- The $Y_{BT}$ matrix has $p$ columns and $m$ rows
- Each row contains the prediction for the $m$-th sample from the test data set
- These values are the mean values of the $k$ fold predictions calculated with the training data
- Each fold generates one model, which is used for prediction on the test data
- The mean value from these $k$ predictions are stored in the matrix $Y_{BT}$

$$(60, 2)$$

$$
\begin{bmatrix}
6.85725928e-01 & 1.18237211e+00 \\
1.19810293e+00 & -2.93516856e-01 \\
-8.16703631e-01 & -1.87229011e+00 \\
2.06323037e+00 & 7.25281970e-01 \\
1.37816630e+00 & -1.35272556e-01 \\
\vdots \\
3.22714124e+00 & 1.36502285e+00
\end{bmatrix}
$$
Model Building [jupyter]

- The level-1 model is a function of the CV-values of each model to the known, training $y$-values
- It provides an estimate of the influence of the single models
- For example, if a linear level-1 model is used, the coefficient $\beta_i$ represents the effect of the $i$-th model
Model Prediction [jupyter]

- The level-1 model is used for predictions on the $Y_{BT}$ data, i.e., on the averaged predictions of the CV-models.
- Constructed using the effects of the predicted values of the single models (determined by linear regression) on the true values of the training data.
- If a model predicts a similar value as the true value during the CV, then it has a strong effect.
- The final predictions are made using the coefficients (weights) of the single models on the $Y_{BT}$ data.
- $Y_{BT}$ data: predicted values from the corresponding models on the final test data.

\[
\begin{bmatrix}
0.98940238 & 1.72943761 & 0.55617508 \\
0.14445345 & 0.17159034 & 0.28026176 \\
\vdots \\
-0.56801858 & -0.50319222 & -0.28009123 \\
\end{bmatrix}
\]

Intercept: $-0.0199327428866$  
Coefficients: $[0.34705597, 0.68483785]$
Comparing MSE [jupyter]

- Comparison of the mean squared error from the SPO2 ensemble and the single models:

  SPO2 (MSE): 0.284948273406
  L (MSE): 0.673695001324
  R (MSE): 0.367652881967
End: Jupyter Interactive Document

- End of the interactive jupyter notebook [Pérez and Granger, 2007]
- The next slides (59 - 64) are based on statics slides
Overview

Introduction

Stochastic Search Algorithms

Quality Criteria: How to Select Surrogates

Examples

Ensembles: Considerations

SPO2 Part 2

Stacking: Considerations
Why are Ensembles Better?

- The following considerations are based on van Veen [2015]

**Example 1 (Error Correcting Codes)**

- Signal in the form of a binary string like: 01110101010000111001010101 gets corrupted with just one bit flipped as in the following: 011010101010000111001010101
- Repetition code: Simplest solution
- Repeat signal multiple times in equally sized chunks and have a majority vote
- 1. Original signal: 001010101 2. Encoded: (Relay multiple times) 000000111000111000111000111 3. Decoding: Take three bit and calculate majority
A Simple Machine Learning Example

- Test set of ten samples
- The ground truth is all positive: 1111111111
- Three binary classifiers (A, B, C) with a 70% accuracy
- View classifiers as pseudo-random number generators
  - output a 1 70% of the time and
  - a 0 30% of the time
- Pseudo-classifiers are able to obtain 78% accuracy through a voting ensemble [van Veen, 2015]
A Simple Machine Learning Example

- All three are correct: \( (0.7 \times 0.7 \times 0.7) = 0.3429 \)
- Two are correct: \( (0.7 \times 0.7 \times 0.3 + 0.7 \times 0.3 \times 0.7 + 0.3 \times 0.7 \times 0.7) = 0.4409 \)
- Two are wrong: \( (0.3 \times 0.3 \times 0.7 + 0.3 \times 0.7 \times 0.3 + 0.7 \times 0.3 \times 0.3) = 0.189 \)
- All three are wrong \( (0.3 \times 0.3 \times 0.3) = 0.027 \)
- This majority vote ensemble will be correct an average of \( \approx 78\%: (0.3429 + 0.4409 = 0.7838) \)
Correlation

- Uncorrelated models clearly do better when ensembled than correlated
  - 1111111100 = 80% accuracy
  - 1111111100 = 80% accuracy
  - 1011111100 = 70% accuracy

- Models highly correlated in their predictions. Majority vote results in no improvement:
  - 1111111100 = 80% accuracy

- Now 3 less-performing, but highly uncorrelated models:
  - 1111111100 = 80% accuracy
  - 0111011101 = 70% accuracy
  - 1000101111 = 60% accuracy

- Ensembling with a majority vote results in:
  - 1111111101 = 90% accuracy

- Lower correlation between ensemble model members seems to result in an increase in the error-correcting capability [van Veen, 2015]
Ensembles: Considerations

Beyond Averaging: Stacking

- Averaging, i.e., taking the mean of individual model predictions, works well for a wide range of problems (classification and regression) and metrics
- Often referred to as *bagging*
- Averaging predictions often reduces overfit [van Veen, 2015]
- Wolpert [1992] introduced *stacked generalization* before *bagging* was proposed by Breiman [1996]
- Wolpert is famous for *There is no free lunch in search and optimization* theorem
- Basic idea behind stacking: use a pool of base classifiers, then using another classifier to combine their predictions
2-fold Stacking

- Stacker model gets information on the problem space by using the first-stage predictions as features [van Veen, 2015]:
  - 1. Split the train set in 2 parts: train_a and train_b
  - 2. Fit a first-stage model on train_a and create predictions for train_b
  - 3. Fit the same model on train_b and create predictions for train_a
  - 4. Finally fit the model on the entire train set and create predictions for the test set.
  - 5. Now train a second-stage stacker model on the predictions from the first-stage model(s).

⇒ Python Example
The following part of this talk is based on an interactive jupyter notebook [Pérez and Granger, 2007]

The next slides (67 - 85) summarize the output from the jupyter notebook
Overview

Introduction

Stochastic Search Algorithms

Quality Criteria: How to Select Surrogates

Examples

Ensembles: Considerations

SPO2 Part 2

Stacking: Considerations
Motivated by van der Laan and Polley [2010], we consider six test functions.

All simulations involve a univariate $X$ drawn from a uniform distribution in $[-4, +4]$

Test functions:

- $f_1(x)$: \[ \text{return } -2 \cdot \mathbb{I}(x < -3) + 2.55 \cdot \mathbb{I}(x > -2) - 2 \cdot \mathbb{I}(x > 0) + 4 \cdot \mathbb{I}(x > 2) - 1 \cdot \mathbb{I}(x > 3) + \epsilon \]
- $f_2(x)$: \[ \text{return } 6 + 0.4 \cdot x - 0.36x^2 + 0.005x^3 + \epsilon \]
- $f_3(x)$: \[ \text{return } 2.83 \cdot \sin(\pi/2 \cdot x) + \epsilon \]
- $f_4(x)$: \[ \text{return } 4.0 \cdot \sin(3 \cdot \pi \cdot x) \cdot \mathbb{I}(x \geq 0) + \epsilon \]
- $f_5(x)$: \[ \text{return } x + \epsilon \]
- $f_6(x)$: \[ \text{return } \text{np.random.normal(0,1,len(x))} + \epsilon \]

$I(\cdot)$ indicator function, $\epsilon$ drawn from an independent standard normal distribution, sample size $r = 100$ (repeats)
Function Definitions [jupyter]

- **f1**: Step function

![Graph of f1: Step function](image)
Function Definitions [jupyter]

- **f2**: Polynomial function
Function Definitions [jupyter]

- **f3: Sine function**

![Graph of the sine function with data points and curve.]
Function Definitions [jupyter]

- **f4**: Composite function
Function Definitions [jupyter]

- **f5: Linear function**

![Graph showing a linear function with data points scattered around a straight line.](image-url)
**f6: Noise function**

![Graph of f6: Noise function](image)
f1: Coefficients of the Level-1 Model [jupyter]

- The coefficients can be interpreted as weights in the linear combination of the models. $0 = \text{intercept}; 1, 2, \text{ and } 3 \text{ denote the } \beta_1, \beta_2, \text{ and } \beta_3 \text{ values, respectively}$
**f1: $R^2$ Values [jupyter]**

- $R^2$ (larger values are better) and standard deviation.
  - SPO: 0.78211976, 0.03308847
  - L: 0.4024831, 0.07134356
  - R: 0.78556947, 0.03187105
  - G: 0.76547433, 0.03564519
f2: Coefficients of the Level-1 Model [jupyter]

- The coefficients can be interpreted as weights in the linear combination of the models. 0 = intercept; 1, 2, and 3 denote the $\beta_1$, $\beta_2$, and $\beta_3$ values, respectively.


**f2: $R^2$ Values [jupyter]**

- $R^2$ (larger values are better) and standard deviation.
  - SPO: 0.79514735 0.03602018
  - L: 0.21445917 0.07656562
  - R: 0.79488344 0.03604606
  - G: 0.79514727 0.03602018
**f3: Coefficients of the Level-1 Model [jupyter]**

- The coefficients can be interpreted as weights in the linear combination of the models. 0 = intercept; 1, 2, and 3 denote the $\beta_1$, $\beta_2$, and $\beta_3$ values, respectively.
f3: $R^2$ Values [jupyter]

- $R^2$ (larger values are better) and standard deviation.
  - SPO: 0.7939634 0.02777211
  - L: 0.11677184 0.05688847
  - R: 0.79244941 0.02743085
  - G: 0.79396338 0.02777211
The coefficients can be interpreted as weights in the linear combination of the models. 0 = intercept; 1, 2, and 3 denote the $\beta_1$, $\beta_2$, and $\beta_3$ values, respectively.
**f4: $R^2$ Values [jupyter]**

- $R^2$ (larger values are better) and standard deviation.
  - SPO: 0.74144195 0.05779718
  - L: 0.00651219 0.01489886
  - R: 0.75301025 0.05133169
  - G: 0.31721598 0.07939812
f5: Coefficients of the Level-1 Model [jupyter]

▶ The coefficients can be interpreted as weights in the linear combination of the models. 0 = intercept; 1, 2, and 3 denote the $\beta_1$, $\beta_2$, and $\beta_3$ values, respectively.
f5: $R^2$ Values [jupyter]

- $R^2$ (larger values are better) and standard deviation.
  - SPO: 0.8362937 0.02381472
  - L: 0.8362937 0.02381472
  - R: 0.83628043 0.02374492
  - G: 0.8362937 0.02381472
The coefficients can be interpreted as weights in the linear combination of the models. $0 = \text{intercept}; 1, 2, \text{and } 3 \text{ denote the } \beta_1, \beta_2, \text{and } \beta_3 \text{ values, respectively}$. 

![Graph showing coefficients of the Level-1 Model]
f5: \( R^2 \) Values [jupyter]

- \( R^2 \) (larger values are better) and standard deviation.
  - SPO: -0.02025601 0.10308039
  - L: -0.00035958 0.01505964
  - R: 0.3586063 0.06232495
  - G: 0.10037904 0.05356867
End: Jupyter Interactive Document

- End of the interactive jupyter notebook [Pérez and Granger, 2007]
- The next slides (88 - 93) are based on statics slides
Stacking: Considerations

Data

Training: \( \{(X_i, y_i)\}_{i=1}^{n} \)

\( \hat{y}_{A_1} \)
\( \hat{y}_{A_2} \)

Val_Training: \( \{(X_i, y_i)\}_{i=1}^{q} \)

Val_Test: \( \{(X_i, y_i)\}_{i=q+1}^{n} \)

A_1
A_2

Val_Test: \( \{(y_i)\}_{i=q+1}^{n} \)

\( \hat{y}_{A_1} \)
\( \hat{y}_{A_2} \)

Test: \( \{\{X_t, y_t\}\}_{i=1,\ldots,m} \)

\( \hat{y}_{A_3} \)

A_3

\( \hat{y} \)

TH Köln

Bartz-Beielstein

MBO
Blending and Meta-Meta Models

- Continue with the summary of van Veen [2015]:
  - Blending is a word introduced by the Netflix winners
  - Instead of creating out-of-fold predictions for the train set: use a small holdout set of say 10% of the train set
    - Benefit: Simpler than stacking
    - Con: The final model may overfit to the holdout set
  - Further improvements: Combine multiple ensembled models
  - Use averaging or voting on manually-selected well-performing ensembles:
    - Start with a base ensemble
    - Add a model when it increases the train set score the most
    - By allowing put-back of models, a single model may be picked multiple times (weighting)
    - Use of genetic algorithms and CV-scores as the fitness function
  - van Veen [2015] proposes a fully random method:
    - Create a 100 or so ensembles from randomly selected ensembles (without placeback)
    - Then pick the highest scoring model
“Frankenstein Ensembles"

Why stacking and combining 1000s of models and computational hours?

van Veen [2015] lists the following pros for these “monster models”:

- Win competitions (Netflix, Kaggle, ...)
- Beat most state-of-the-art academic benchmarks with a single approach
- Transfer knowledge from the ensemble back to a simpler shallow model
- Loss of one model is not fatal for creating good predictions
- Automated large ensembles don’t require much tuning or selection
- 1% increase in accuracy may push an investment fund from making a loss, into making a little less loss. More seriously: Improving healthcare screening methods helps save lives
Video Sequence

- The following part of this talk refers to a sequence from the video “Gerald Jay Sussman on Flexible Systems, The Power of Generic Operations”, which is available on https://vimeo.com/151465912
- 1:01:42 - 1:04:25 (to be deleted from the videolectures.net version)
Summary: Structure and Interpretation of Computer Programs (SICP)

- PROGRAMMING BY POKING: WHY MIT STOPPED TEACHING SICP
- 1. Hal Abelson and Gerald Jay Sussman and got tired of teaching it
- 2. SICP curriculum no longer prepared engineers for what engineering is like today
- In the 80s and 90s, engineers built complex systems by combining simple and well-understood parts
- Programming today is
  
  "[m]ore like science. You grab this piece of library and you poke at it. You write programs that poke it and see what it does. And you say, ‘Can I tweak it to do the thing I want?’"

- MIT chose Python as an alternative for SCIP
Summary

▶ Keywords:
  ▶ Abelson & Sussmann ✓
  ▶ Wolpert ✓
  ▶ Netflix ✓
  ▶ Deep learning ✓

▶ Related report “Stacked Generalization of Surrogate Models—A Practical Approach” [Bartz-Beielstein, 2016]

▶ More: http://www.spotseven.de
This work has been supported by the Bundesministeriums für Wirtschaft und Energie under the grants KF3145101WM3 und KF3145103WM4.

This work is part of a project that has received funding from the European Union’s Horizon 2020 research and innovation program under grant agreement No 692286.


References


References

Evolutionary Computation, pages 1209–1216, New York, NY, USA, 2013. ACM.


