Chapter 1
Introduction

1.1 Computational Intelligence

Computational intelligence is an increasingly important discipline with an impact on more and more domains. It mainly comprises the two large problem classes optimization and machine learning that are strongly connected to each other, but which also cover a broad field of individual problems and tasks. Examples for applications of optimization and learning methods range from robotics to civil engineering. Optimization is the problem of finding optimal parameters for all kinds of systems. Machine learning is an essential part of intelligence. It means that a system is able to recognize structure in data and to predict the future based on past observations from the past. It is therefore related to human learning and thinking. The benefit of learning is the ability to behave reasonable based on experience, in particular considering functional aspects that can be extracted from observations. Machine learning is basically an advancement of statistical methods, a reason why it also denoted as statistical learning.

Traditionally, computational intelligence consists of three main branches: (1) evolutionary algorithms, (2) neural networks, and (3) fuzzy logic. Evolutionary algorithms comprise methods that allow the optimization of parameters of a system oriented to the biological principle of natural evolution. Neural networks are also biologically inspired methods. They are abstract imitations of natural neural information processing. Fuzzy logic allows modeling of linguistic terms with linguistic inaccuracy. Meanwhile, numerous other methods have been developed and influence the three branches of computational intelligence.

1.2 Optimization

Optimization is an important problem class in computer science finding numerous applications in domains like electrical engineering, information management, and many more. Optimization variables may be numerical, discrete, or combinatorial.

O. Kramer, *Machine Learning for Evolution Strategies*,
Studies in Big Data 20, DOI 10.1007/978-3-319-33383-0_1

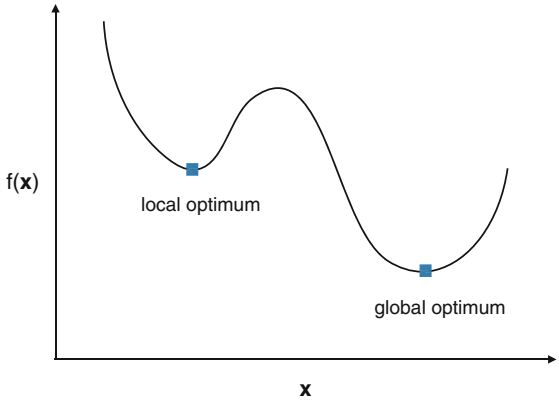

Fig. 1.1 Illustration of local and global optima of a minimization problem

Moreover, mixed-integer problems contain continuous and discrete variables. A famous combinatorial optimization problem is the traveling salesman problem (TSP), which is the task to find the shortest tour between a set of cities, which have to be visited. This problem can be arbitrarily difficult. For example, if we consider 100 cities, the number of possible tours exceeds 10^{150}, which is nearly impossible to compute. The search in this large solution space is a particularly difficult problem. But the world of combinatorial optimization profits from developments in computational complexity. This theory allows differentiating between easy and hard problems. For example, the TSP problem is NP-hard, which means that (probably—to the best of our current knowledge) no algorithm exists that computes a solution in polynomial time.

In this work, we focus on continuous optimization also referred to as numerical optimization. We define an optimization problem as the minimization of an objective function $f(\mathbf{x})$ w.r.t. a set of d parameters $\mathbf{x} = (x_1, \ldots, x_d)^T \in \mathbb{R}^d$. Every minimization problem can easily be treated as maximization problem by setting $f' = -f$. Figure 1.1 illustrates the definition of optimality. A local optimum employs the best fitness in its neighborhood, a global optimum employs the best overall fitness. A global optimum is also a (special) local optimum. Optimization methods must prevent getting stuck in local optima.

Some optimization problems are high-dimensional, i.e., a large number of parameters has to be optimized at the same time, while others only concern few variables. High-dimensional optimization problems often occur during computer experiments, when a complex simulation model replaces a lab experiment. The problem that comes up under such conditions is that due to the curse of dimensionality problem no reasonable search in the solution space is possible. In such cases, a stochastic trial-and-error method can yield good results. This also holds for problems that are difficult due to multiple local optima or non-smooth and noisy fitness landscapes.

Many conditions can make an optimization problem difficult to solve:

- Often, no analytic expression is available. In this case, it is not possible to apply methods like gradient descent, Newton methods, and other variants.
- Problems may suffer from noise, which means that the fitness function evolution of the same candidate solution may result in different fitness values.
- Solution spaces may be constrained, i.e., not the whole solution space is feasible. As optima often lie at the constraint boundary, where the success rate drops significantly, heuristics like evolutionary algorithms must be enhanced to cope with changing conditions.
- When more than two objectives have to be optimized at the same time, which may be conflictive, a Pareto set of solutions has to be generated. For this sake, evolutionary multi-objective optimization algorithms are a good choice as they naturally work with populations, which allow the evolution of a Pareto set.

Under such conditions classic optimization strategies often fail and heuristics perform good results. Evolutionary algorithms (EAs) are excellent methods to find optima for blackbox optimization problems. EAs are inspired by the natural process of evolution. Different branches began to develop in the mid of the last century. In Germany, Rechenberg and Schwefel [1–3] developed evolution strategies (ES) in the sixties in Berlin. A similar development took place in the United States initiated by Holland [4] at the same time. Methods developed in this line of research were called genetic algorithms. All EA research branches have grown together in the recent decade, as similar heuristic extensions can be used and exchanged within different EA types.

1.3 Machine Learning and Big Data

Machine learning allows learning from data. Information is the basis of learning. Environmental sensors, text and image data, time series data in economy, there are numerous examples of information that can be used for learning. There are two different types of learning: supervised and unsupervised. Based on observations, the focus of supervised learning is the recognition of functional relationships between patterns. Labels are used to train a model. Figure 1.2 illustrates a supervised learning scenario. In classification, a new pattern $\mathbf{x}'$ without label information is assigned to a label based on the model that is learned from a training data set. Many effective and efficient machine learning methods have been proposed in the past. Strong machine learning libraries allow their fast application to practical learning problems. Famous methods are nearest neighbor methods, decision trees, random forests, and support vector machines. Deep learning is a class of methods that recently attracts a lot of attention, in particular in image and speech recognition.

In unsupervised learning scenarios, the idea is to learn from the data distribution without further label information. Clustering, dimensionality reduction, and outlier detection are the most important unsupervised learning problems. Clustering is the task to learn groups (clusters) of patterns, mostly based on the data distributions

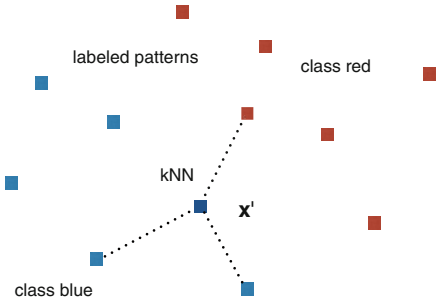

Fig. 1.2 Illustration of supervised learning with the nearest neighbor classifier, see Chap. 6. We seek for the label of a novel pattern $\mathbf{x}'$

and on data densities. Dimensionality reduction is a further typically unsupervised[1] problem class that maps patterns from high-dimensional spaces to spaces with lower dimensionality. Applications are visualization of high-dimensional data spaces and pre-processing of supervised learning methods. Famous examples for unsupervised learning methods are k-means, Spatial-Density Clustering of Applications with Noise (DBSCAN), isometric mapping, and self-organizing maps.

Besides the development and analysis of new methods, their theoretical investigation is an important aspect. Theory helps to understand aspects like runtime behavior and convergence properties. It often requires the reduction of problems and methods to simplified variants that are not applied in practice. However, theoretical investigations often reveal the general aspects of problem hardness and method complexity that are required to understand.

The application aspect plays a very important role in machine learning. To improve the applicability of machine learning concepts strong machine learning libraries have been developed. An example is the SCIKIT-LEARN library that will exhaustively be used in this book. A deeper knowledge about the problem domain usually helps to adapt and improve the machine learning process chain. The optimization of this process chain comprises the selection of appropriate methods for aggregating data sets, pre-processing, learning, post-processing, and visualization of results. Deeper knowledge about various technologies is required in practice like heterogeneous data bases, low-level programming languages to access embedded sensors, network technologies for the collection of data and distribution of computations, and computing architectures like multi-processor systems and distributed computing settings.

With Big Data, we usually associate the situation of very large data sets, from which we want to learn. This scenario often imposes difficult challenges on various aspects of the machine learning process chain. An optimization of the tool chain is often required. If billions of patterns have to be processed in a short time this affords the scaling of the computing architecture or the distribution of processes and tasks on many machines. For the machine learning expert, the proper development and choice of efficient machine learning methods with low runtime complexity but large effectivity is the most important task.

[1] Also supervised dimensionality reduction methods exist.

1.4 Motivation

The objective of this book is to show that the intersection of both worlds, evolutionary computation and machine learning, lead to efficient and powerful hybridizations. Besides the line of research of evolutionary computation for machine learning, the other way around, i.e., machine learning for evolutionary computation, is a fruitful research direction. The idea is to support the evolutionary search with machine learning models, while concentrating on the specific case of continuous solution spaces. Various kinds of hybridizations are possible reaching from covariance matrix estimation to meta-modeling of fitness and constraint functions. Numerous challenges arise when both worlds are combined with each other.

This book is an introductory depiction building a bridge between both worlds from an algorithmic and experimental perspective. Experiments on small sets of benchmark problems illustrate algorithmic concepts and give first impressions of their behaviors. There is much space for further investigations, e.g., for theoretical bridges that help to understand the interplay between methods. For the future, we expect a lot of further kinds of ways to improve evolutionary search with machine learning techniques.

The book only requires basic knowledge in linear algebra and statistics. Nevertheless, the reader will find easy descriptions and will not have to dig through endless formalisms and equations. The book can be used as:

- introduction to evolutionary computation,
- introduction to machine learning, and
- guide to machine learning for evolutionary computation.

The book gives an introduction to problem solving and modeling, to the corresponding machine learning methods, and overviews of the most important related work, and short depictions of experimental analyses.

1.5 Benchmark Problems

The analysis of algorithms in the intersection of machine learning and evolutionary computation are most often based on computer experiments. Computer experiments are the main analytic tool of this work. In blackbox optimization, such experimental analyses are focused on particular benchmark functions with characteristic properties, on which the analyzed algorithms should prove to succeed. Various benchmark function have been proposed in literature. In this work, we only use a small set of benchmark problems, as the main focus is to demonstrate potential hybridizations. Of course, experimental analyses are always limited to some extend. On the one hand, the concentration on particular benchmark problems is the only possible way of experimentalism as it will never be possible to analyze the algorithmic behavior on all kinds of problems. On the other hand, the concentration can lead to a bias towards a restricted set of problems.

The focus of this book to a small set of known benchmark problem is motivated by the fact that the employed machine learning algorithms have already proven to be valid and efficient methods in various data mining and machine learning tasks. The task of this book is to demonstrate, how they can be integrated into evolutionary search and to illustrate their behaviors on few benchmark functions.

Computer experiments are the main methodological tool to evaluate the algorithms and hybridizations introduced in this work. As most algorithms employ randomized and stochastic components, each experiment has to be repeated multiple times. The resulting mean value, median and the corresponding standard deviation give an impression of the algorithmic performance. Statistical tests help to evaluate, if the superiority of any algorithm is significant. As the results of most EA-based experiments are not Gaussian distributed, the standard student T-test can usually not be employed. Instead, the rank-based tests like the Wilcoxon test allow reasonable conclusions. The Wilcoxon signed-rank test is the analogue to the T-test. The test makes use of the null hypothesis, which assumes that the median difference between pairs of observations is zero. It ranks the absolute value of the differences between observations from the smallest (rank 1) to the largest. The idea is to add the ranks of all differences in both directions, while the smaller sum is the output of the test.

1.6 Overview

This book is structured in ten chapters. The following list gives an overview of the chapters and their contributions:

Chapter 2

In Chap. 2, we introduce the basic concepts of optimization with evolutionary algorithms. It gives an introduction to evolutionary computation and nature-inspired heuristics for optimization problems. The chapter illustrates the main concepts of translating evolutionary principles into an algorithmic framework for optimization. The (1+1)-ES with Rechenberg's mutation strength control serves as basis of most later chapters.

Chapter 3

Covariance matrix estimation can improve ES based on Gaussian mutations. In Chap. 3, we integrate the Ledoit-Wolf covariance matrix estimation method into the Gaussian mutation of the (1+1)-ES and compare to variants based on empirical maximum likelihood estimation.

Chapter 4

In Chap. 4, we sketch the main tasks and problem classes in machine learning. We give an introduction to supervised learning and pay special attention to the topics model selection, overfitting, and the curse of dimensionality. This chapter is an important

introduction for readers, who are not familiar with machine learning modeling and machine learning algorithms.

Chapter 5

Chapter 5 gives an introduction to SCIKIT-LEARN, a machine learning library for PYTHON. With a strong variety of efficient flexible methods, SCIKIT-LEARN implements algorithms for pre-processing, model selection, model evaluation, supervised learning, unsupervised learning, and many more.

Chapter 6

In Chap. 6, we show that a reduction of the number of fitness function evaluations of a (1+1)-ES is possible with a combination of a k-nearest neighbor (kNN) regression model, a local training set of fitness function evaluations, and a convenient meta-model management. We analyze the reduction of fitness function evaluations on a small set of benchmark functions.

Chapter 7

In the line of research on fitness function surrogates, the idea of meta-modeling the constraint boundary is the next step we analyze in Chap. 7. We employ support vector machines (SVMs) to learn the constraint boundary as binary classification problem. A new candidate solution is first evaluated on the SVM-based constraint surrogate. The constraint function is only evaluated, if the solution is predicted to be feasible.

Chapter 8

In Chap. 8, we employ the concept of bloat by optimizing in a higher-dimensional solution space that is mapped to the real-solution space with dimensionality reduction, more specifically, with principal component analysis (PCA). The search in a space that employs a larger dimensionality than the original solution space may be easier. The solutions are evaluated and the best w.r.t. the fitness in the original space are inherited to the next generation.

Chapter 9

The visualization of evolutionary blackbox optimization runs is important to understand evolutionary processes that may require the interaction with or intervention by the practitioner. In Chap. 9, we map high-dimensional evolutionary runs to a two-dimensional space easy to plot with isometric mapping (ISOMAP). The fitness of embedded points is interpolated and visualized with `matplotlib` methods.

Chapter 10

In multimodal optimization, the task is to detect most global and local optima in solution space. Evolutionary niching is a technique to search in multiple parts of the solution space simultaneously. In Chap. 10, we present a niching approach based on an explorative phase of uniform sampling, selecting the best solutions, and applying clustering to detect niches.

Chapter 11

In Chap. 11, we summarize the most important findings of this work. We give a short overview to evolutionary search in machine learning and give insights into prospective future work.

1.7 Previous Work

Parts of this book built upon previous work that has been published in peer-reviewed conferences. An overview of previous work is the following:

- The covariance matrix estimation approach of Chap. 3 based on Ledoit-Wolf estimation has been introduced on the Congress on Evolutionary Computation (CEC) 2015 [5] in Sendai, Japan.
- The nearest neighbor meta-model approach for fitness function evaluations presented in Chap. 6 is based on a paper presented at the EvoApplications conference (as part of EvoStar 2016) [6] in Porto, Portugal.
- The PCA-based dimensionality reduction approach that optimizes in high-dimensional solution spaces presented in Chap. 8 has been introduced on the International Joint Conference on Neural Networks (IJCNN) 2015 [7] in Killarney, Ireland.
- The ISOMAP-based visualization approach of Chap. 9 has also been introduced on the Congress on Evolutionary Computation (CEC) 2015 [8] in Japan.

Parts of this work are consistent depictions of published research results presenting various extended results and descriptions. The book is written in a scientific style with the use of "we" rather than "I".

1.8 Notations

We use the following notations. Vectors use small bold latin letters like $\mathbf{x}$, scalars small plain Latin or Greek letters like σ. In optimization scenarios, we use the variable $\mathbf{x} = (x_1, \ldots, x_d)^T \in \mathbb{R}^d$ for objective variables that have to be optimized w.r.t. a fitness function f. In machine learning, concepts often use the same notation. Patterns are vectors $\mathbf{x} = (x_1, \ldots, x_d)^T$ of attributes or features x_j with $j = 1, \ldots, d$ for machine learning models f. This is reasonable as candidate solutions in optimization are often treated as patterns in this book. Patterns lie in an d-dimensional data space and are usually indexed from 1 to N, i.e., a data set consists of patterns $\mathbf{x}_1, \ldots, \mathbf{x}_N$. They may carry labels $y_1, \ldots, y_N$ resulting in pattern-label pairs

$$\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}.$$

When the dimensionality of objective variables or patterns differs from d (i.e., from the dimensionality, where the search actually takes place), we employ the notation $\hat{\mathbf{x}}$. In Chap. 8, $\hat{\mathbf{x}}$ represents an abstract solution of higher dimensionality, in Chap. 9 it represents a visualizable low-dimensional pendant of solution $\mathbf{x}$.

While in literature, f stands for a fitness function in optimization and for a supervised model in machine learning, we resolve this overlap in the following. While being used this standard way in the introductory chapters, Chap. 6 uses f for the fitness function and $\hat{f}$ for the machine learning model, which is a surrogate of f. In Chap. 7, f is the fitness function, while g is the constraint function, for which a machine learning surrogate $\hat{g}$ is learned. In Chaps. 8 and 9, f is the fitness function and F is the dimensionality reduction mapping from a space of higher dimensions to a space of lower dimensionality. In Chap. 10, f is again the fitness function, while c delivers the cluster assignments of a pattern.

A matrix is written in bold large letters, e.g., $\mathbf{C}$ for a covariance matrix. Matrix $\mathbf{C}^T$ is the transpose of matrix $\mathbf{C}$. The p-norm will be written as $\|\cdot\|_p p$ with the frequently employed variant of the Euclidean norm written as $\|\cdot\|_2$.

1.9 Python

The algorithms introduced in this book are based on PYTHON and built upon various well-known packages including `Numpy` [9], `SciPy` [10], `Matplotlib` [11], and SCIKIT-LEARN [12]. PYTHON is an attractive programming language that allows functional programming, classic structured programming, and objective-oriented programming. The installation of PYTHON is usually easy. Most Linux, Windows, and Mac OS distributions already contain a native PYTHON version. On Linux systems, PYTHON is usually installed in the folder `/usr/local/bin/python`. If not, there are attractive PYTHON distributions that already contain most packages that are required for fast prototyping machine learning and optimization algorithms. First experiments with PYTHON can easily be conducted when starting the PYTHON interpreter, which is usually already possible, when typing PYTHON in a Unix or Windows shell.

An example for a typical functional list operation that demonstrates the capabilities of PYTHON is:

```
[f(x)*math.pi for x in range(1,10)]
```

This expression generates a list of function values with arguments $1, \ldots, 10$ of function `f(x)`, which must be defined before, multiplied with π.

A very frequently employed package is `Numpy` for scientific computing with PYTHON. It comprises powerful data structures and methods for handling N-dimensional array objects and employs outstanding linear algebra and random number functions. `SciPy` provides efficient numerical routines, e.g., for numerical integration, for optimization, and distance computations. `Matplotlib` is a

PYTHON plotting library that is able to generate high-quality figures. SCIKIT-LEARN is a PYTHON framework for machine learning providing efficient tools for data mining and data analysis tasks. The SCIKIT-LEARN framework will be introduced in more detail in Chap. 4.

References

1. Beyer, H., Schwefel, H.: Evolution strategies—a comprehensive introduction. Nat. Comput. **1**(1), 3–52 (2002)
2. Rechenberg, I.: Evolutionsstrategie—Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Frommann-Holzboog, Stuttgart (1973)
3. Schwefel, H.-P.: Numerische Optimierung von Computer-Modellen mittel der Evolutionsstrategie. Birkhaeuser, Basel (1977)
4. Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor (1975)
5. Kramer, O.: Evolution strategies with ledoit-wolf covariance matrix estimation. In: Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2015, pp. 1712–1716 (2015)
6. Kramer, O.: Local fitness meta-models with nearest neighbor regression. In: Proceedings of the 19th European Conference on Applications of Evolutionary Computation, EvoApplications 2016, pp. 3–10. Porto, Portugal (2016)
7. Kramer, O.: Dimensionality reduction in continuous evolutionary optimization. In: 2015 International Joint Conference on Neural Networks, IJCNN 2015, pp. 1–4. Killarney, Ireland, 12–17 July 2015
8. Kramer, O., Lückehe, D.: Visualization of evolutionary runs with isometric mapping. In: Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2015, pp. 1359–1363. Sendai, Japan, 25–28 May 2015
9. van der Walt, S., Colbert, S.C., Varoquaux, G.: The NumPy array: a structure for efficient numerical computation. Comput. Sci. Eng. **13**(2), 22–30 (2011)
10. Jones, E., Oliphant, T., Peterson, P., et al.: SciPy: open source scientific tools for Python (2001–2014). Accessed 03 Oct 2014
11. Hunter, J.D.: Matplotlib: a 2D graphics environment. Comput. Sci. Eng. **9**(3), 90–95 (2007)
12. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)