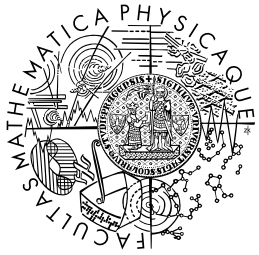

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Gabriela Suchopárová

Evolutionary optimization of machine learning workflows

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the bachelor thesis:	Mgr. Roman Neruda, CSc.
Study programme:	Computer Science
Study branch:	General Computer Science

Prague 2019

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ..... signature of the author

Dedication.

Title: Evolutionary optimization of machine learning workflows

Author: Gabriela Suchopárová

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: Mgr. Roman Neruda, CSc., Department of Theoretical Computer Science and Mathematical Logic

Abstract: Abstract.

Keywords: Machine learning Evolutionary computing Meta-learning Workflows

Contents

Introduction

1. Preliminaries

What we will talk about, theory

1.1 Machine learning

The field of machine learning encompasses a broad range of algorithms and statistical methods for data processing. In his book on machine learning, Flach provides the following general definition:

> Machine learning is the systematic study of algorithms and systems that improve their knowledge or performance with experience. (Flach [2012])

The knowledge of a system is gained through learning from *experience*. This procedure is referred to as the *training* phase. In this process, the algorithm adjusts its parameters according to the nature of training data. The result of the process is a prediction function, which depends on learned parameters. By applying this function on previously unseen data, denominated as the *testing* data, we obtain the output of the algorithm. The result is then evaluated to determine the performance of the method. (Bishop [2006]) The character of the learning process varies with different machine learning problems. There are three main classes of tasks: *supervised*, *unsupervised* and *reinforcement* learning.

In case of supervised learning, the training data is a set of labelled examples and the task is to predict labels of previously unseen data. The field is further subdivided into two groups. If the labels are elements of a finite number of discrete categories, the problem is called *classification*. The continuous case is then called *regression*.

In contrast to supervised learning, in unsupervised learning the training data is unlabelled. The key task is therefore to divide the data into groups of similar examples.

The task of reinforcement learning is to find suitable actions as to maximize a reward. The training data is typically some history of previous actions and corresponding rewards.

It is important to note that good performance on training data does not ensure just as good performance on new data. Sometimes the model performs exceptionally well on training data, but fares much worse on testing data. This behaviour is called *overfitting* and usually occurs when unnecessarily subtle details of the data are learned. The opposite concept is called *generalisation*, which is the ability to perform well on different types of testing data.

A related term is the so-called '*bias-variance dilemma*'. A low-complexity model will not overfit, but a lot of errors will be made, thus introducing a certain bias from the correct output. On the other hand, by increasing the number of model parameters, it will highly depend on training data. Then, with small changes in data there will be a high variance in output. A balance can rarely be achieved in practice, hence it is often necessary to choose the side which is less harmful to the task.

1.1.1 Model ensembles

Model ensembles are powerful learning techniques that combine simpler models to achieve better results.

Shapire proved that a model is weakly learnable if and only if it is PAC-learnable. Schapire [1990]

1.2 Metalearning

1.3 Evolutionary computing

Evolutionary computing is a heuristic method of optimization inspired by Charles Darwin's theory of *natural selection*. Darwin [1859] In a population, individuals with the best characteristics are most likely to reproduce, thus passing the traits to the offspring. As the evolution is repeated over several generations, the most advantageous traits predominate. This phenomenon is also called 'survival of the fittest'.

In an evolutionary algorithm, the goal is to find the "best" solution to the given problem. The term 'population' refers to a set of solutions encoded as chromosomes which represent the defining features of a particular solution. The 'natural selection' can be then understood as a stochastic search through the space of possible chromosome values. (Engelbrecht [2007])

As can be seen in algorithm 1, a genetic algorithm should define a suitable *selection* method, *mutation* and/or *crossover* operators and the *fitness function*. The algorithm terminates when some *stopping condition* is met. Some commonly used criteria, as listed by Engelbrecht, are for example a limit on the number of generations, a fitness threshold or termination after no improvement is observed. ([Engelbrecht, 2007])

Algorithm 1: Evolutionary algorithm

Data: population size n, stopping condition c, crossover probability p_{cx} and mutation probability p_{mut}
Result: evolved individuals

$P(0) \longleftarrow$ population of size n
while *c is not met*:
  for *individual ind*:
    compute fitness $f(ind)$
  for *i in* `range(`$n/2$`):`
    $i_1, i_2 \longleftarrow$ select two individuals
    if p_{cx}:
      perform crossover
    if p_{mut}:
      perform mutation
  $P(n+1) \longleftarrow$ select individuals from $P(n)$

return $P(c)$

The advantage of genetic algorithms is such that there are potentially many different solutions present in every population. With well defined selection and fitness, the algorithm performs a multi-directional search. In comparison with other directed search methods, this proves to be a more robust approach. (Michalewicz [1996], Mitchell [1997])

1.4 Genetic programming

In this section, we present a subfield of evolutionary computing — the genetic programming — where the population is a set of computer programs. The aim of this technique is to evolve programs which provide a good solution to the given problem. There are various approaches in means of how to represent the individuals and what kind of genetic operators to use. The fitness is computed by running the program and comparing the result with the desired output. Poli et al. [2008]

Mention typed GP here

1.4.1 Tree-based genetic programming

The individuals are most frequently represented in the form of *syntax trees*. Inner nodes of the tree are functions, whereas leaves are constants and variables. The initialization step is thus very important, as there are many different ways how to design trees. Also, specialized genetic operators need to be designed.

Initialization During the initialization, both functions and constants are selected from a set of possible nodes which is provided as input to the algorithm. As was mentioned, there are various methods of initialization. We well present two methods which are among the simplest and most used ones — *grow* and *full*.

In both cases, nodes are inserted to the tree up to a certain height limit. The two methods differ only in the way how nodes are selected. The grow method allows to select both functions and terminals before the limit is reached; afterwards, only terminals can be inserted. The full method restricts the selection only to functions on all levels but the last one, thus generating a full tree. Leaves are then chosen from the terminal set like in the previous approach.

The drawback of the full method is that all trees are very similar. On the contrary, the grow method generates a wide range of sizes and shapes, but the number of nodes in a tree might be too small. Because of that, a method called *ramped half-and-half* is often used in practice. It combines both of the presented methods; half of the population is generated using the full method, the other one via grow method. Also, instead of one height limit, a range of values is used to introduce more diversity.

Genetic operators The most common type of crossover is *subtree crossover* of two individuals. A random node — the crossover point — is selected in each individual independently. Then, subtrees corresponding to the points are exchanged between them.

Similarly, the most used mutation technique is *subtree mutation*. Just like in subtree crossover, a mutation point is randomly chosen. Afterwards, the corresponding subtree is entirely replaced by a new randomly generated tree.

1.4.2 Developmental genetic programming

1.5 Workflows

2. Related work

2.1 AutoML

2.2 TPOT

3. Our solution

Conclusion

Bibliography

Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738.

Charles Darwin. *On the origin of species by means of natural selection, or, The preservation of favoured races in the struggle for life*. London, Murray, 1859.

Andries P. Engelbrecht. *Computational Intelligence: An Introduction*. Wiley Publishing, 2nd edition, 2007. ISBN 0470035617.

Peter Flach. *Machine Learning: The Art and Science of Algorithms That Make Sense of Data*. Cambridge University Press, New York, NY, USA, 2012. ISBN 1107422221, 9781107422223.

Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs (3rd Ed.)*. Springer-Verlag, Berlin, Heidelberg, 1996. ISBN 3-540-60676-9.

Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997. ISBN 0070428077, 9780070428072. URL `http://www.cs.cmu.edu/~tom/mlbook.html`.

Riccardo Poli, William B. Langdon, and Nicholas Freitag McPhee. *A Field Guide to Genetic Programming*. Lulu Enterprises, UK Ltd, 2008. ISBN 1409200736, 9781409200734.

Robert E. Schapire. The strength of weak learnability. *Mach. Learn.*, 5(2): 197–227, July 1990. ISSN 0885-6125. doi: 10.1023/A:1022648800760. URL `https://doi.org/10.1023/A:1022648800760`.

List of Algorithms

List of Figures

List of Tables

List of Abbreviations

A. Attachments

A.1 First Attachment