VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

GENETICKÉ PROGRAMOVÁNÍ V ÚLOHÁCH PREDIKCE

GENETIC PROGRAMMING IN PREDICTION TASKS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **MICHAL MACHAČ**
AUTHOR

VEDOUCÍ PRÁCE **prof. Ing. LUKÁŠ SEKANINA, Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Machač Michal**
Program: Informační technologie
Název: **Genetické programování v úlohách predikce**
Genetic Programming in Prediction Tasks
Kategorie: Umělá inteligence

Zadání:

1. Seznamte se se základními metodami strojového učení, které se používají v úlohách predikce. Zaměřte se na genetické programování.
2. Zvolte si softwarovou knihovnu implementující predikční metody a seznamte se s jejím používáním.
3. Navrhněte a implementujte systém pro automatizovaný návrh prediktorů, který bude založen na genetickém programování.
4. Na alespoň dvou úlohách z oblasti predikce ověřte funkčnost prediktorů navržených pomocí vytvořeného systému.
5. Porovnejte dosažené výsledky s výsledky získanými za pomoci knihovny vybrané v bodě 2 zadání.
6. Zhodnoťte dosažené výsledky.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění bodů 1 a 2 zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Sekanina Lukáš, prof. Ing., Ph.D.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 25. října 2019

Abstrakt

V této práci jsou představeny různé metody strojového učení, jež lze využít v úlohách predikce založených na regresi. Detailněji je popsáno stromové a lineární genetické programování. S vybranými algoritmy strojového učení (lineární regrese, náhodný les, vícevrstvý perceptron a stromové genetické programování) jsou provedeny experimenty na volně dostupných datových sadách za využití knihoven scikit-learn a gplearn, a získané výsledky jsou porovnány z pohledu kvality predikce. Stěžejní částí této práce byla implementace systému lineárního genetického programování v programovacím jazyce C++, která byla nejprve testována na umělých úlohách symbolické regrese, a následně na reálných datových sadách. Výsledky získané pomocí vytvořené implementace jsou porovnány vůči výsledkům získaným pomocí knihovny gplearn.

Abstract

This thesis introduces various machine learning algorithms which can be used in prediction tasks based on regression. Tree genetic programming and linear genetic programming are explained more thoroughly. Selected machine learning algorithms (linear regression, random forest, multilayer perceptron and tree genetic programming) are compared on publicly available datasets with the use of scikit-learn and gplearn libraries. A core part of this project is a new implementation of linear genetic programming which was developed in C++, tested on common symbolic regression problems and then evaluated on real datasets. Results obtained with the proposed system are compared with the results obtained with gplearn.

Klíčová slova

genetické programování, lineární genetické programování, strojové učení, lineární regrese, náhodný les, vícevrstvý perceptron, regrese, Python, C++, scikit-learn, gplearn

Keywords

genetic programming, linear genetic programming, machine learning, linear regression, random forest, multilayer perceptron, regression, Python, C++, scikit-learn, gplearn

Citace

MACHAČ, Michal. *Genetické programování v úlohách predikce*. Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Lukáš Sekanina, Ph.D.

Genetické programování v úlohách predikce

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana prof. Ing. Lukáše Sekaniny, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Michal Machač
28. května 2020

Poděkování

Děkuji prof. Ing. Lukáši Sekaninovi, Ph.D. za vedení mé bakalářské práce, a dále za jeho přístup a ochotu poradit.

Obsah

Kapitola 1

Úvod

V této práci se zabývám nejen genetickým programováním, ale i ostatními metodami strojového učení, které lze využít pro úlohy predikce. Předmětem zkoumání jednotlivých metod je jejich schopnost aproximovat funkci, která co nejlépe popisuje data. Získat takto aproximovanou funkci je pak užitečné zejména v situacích, kdy skutečnou funkci neznáme, a je velice obtížné či dokonce nemožné skutečnou funkci získat. Úlohy predikce jsou tak v rámci této práce rozuměny jako úlohy regrese.

Strojové učení je oblastí umělé inteligence, která se tyto problémy snaží řešit prostřednictvím „učení“. Učením rozumíme schopnost algoritmů modifikovat se a dosahovat lepších výsledků na základě provádění daného algoritmu nad poskytnutými daty. Do oblasti strojového učení náleží i evoluční algoritmy.

Evoluční algoritmy se inspirují biologickou evolucí. Tyto algoritmy začínají evoluci s počáteční populací jedinců, kde každý jedinec je ohodnocenou zvolenou metrikou. Následně jsou vybraní jedinci modifikováni pomocí operátorů inspirovaných genetikou a je z nich vytvořena nová populace. Tento postup je opakován, dokud není splněna ukončující podmínka.

Genetické programování je jednou z variantou evolučních algoritmů, kdy jednotliví jedinci mají formu počítačových programů. Za hlavního průkopníka genetického programování je považován John Koza, a to díky jeho práci [9].

Cíl této práce je seznámit se se základními metodami strojového učení, které lze využít pro úlohy predikce, zaměřit se na genetické programování, a následně vybrané metody porovnat z hlediska kvality predikce. Dalším cílem je implementace systému lineárního genetického programování, a na vybraných úlohách z oblasti predikce implementovaný systém porovnat vůči knihovní implementaci.

Obsah této práce je do kapitol rozdělen následovně:

- V kapitole 2 jsou představeny základní principy strojové učení. Dále jsou popsány vybrané metody, které jsou dále v této práci předmětem porovnání.
- V kapitole 3 jsou vysvětleny principy stromového genetického programování.
- V kapitole 4 jsou nejprve popsány kroky, které jsou obvykle nutné provést před trénováním modelu. Následně jsou provedeny experimenty s vybranými algoritmy na vybraných datových sadách, a získané výsledky jsou porovnány.
- V kapitole 5 je popsáno lineární genetické programování.

- V kapitole 6 je popsána implementace systému založeném na lineárním genetickém programování. Detailněji jsou popsány části systému, se kterými uživatel přímo manipuluje. Ostatní části systému jsou popsány stručněji. V poslední podkapitole je předvedena ukázka práce s navrženým systémem.
- V kapitole 7 je systém otestován na benchmarkových polynomech, a následně také na čtyřech reálných datasetech. Výsledky na reálných datasetech získané s implementovaným systémem jsou porovnány s výsledky získanými pomocí knihovny gplearn.

Kapitola 2

Strojové učení

V této kapitole budou přestaveny metody strojového učení vybrané pro tuto práci. Ty budou následně porovnány s genetickým programováním v kapitole 4.

2.1 Úvod do strojového učení

Strojové učení je proces, ve kterém je počítač schopný vykonávat mu zadaný úkol lépe a lépe. Tento přístup je zvlášť ceněný u problémů, pro které vhodný algoritmus neexistuje, ale zase máme k danému problému spoustu dat. Algoritmy strojového učení lze rozdělit do tří základních skupin dle typu učení, a to na učení s učitelem, učení bez učitele, a posilované učení. Tato podkapitola čerpá z knihy [12] a [15].

2.1.1 Učení s učitelem

Při učení s učitelem je cílem najít takovou funkci, která je schopna na základě vstupních nezávislých proměnných predikovat hodnotu závislé proměnné. Při trénování je k dispozici učitel, který pro každý trénovací vstup zná správný výstup, a pro každý predikovaný výstup tedy umí říci, zda se jedná o výstup špatný či správný, případně jak moc špatný nebo dobrý. Trénovací data mají podobu dvojic vstupních a výstupních hodnot $(x_0, y_0), (x_1, y_1), \ldots, (x_N, y_N)$. Pokud y_i nabývá hodnot z konečné množiny možností, pak se jedná o klasifikaci. V případě, že y_i nabývá hodnot z nekonečné množiny, pak se jedná o regresi. Vybraný model je nejprve natrénován na ukázkových datech, a po natrénování by měl být schopný predikovat na nových (neviděných) datech.

Mezi běžný příklad klasifikace patří filtrování příchozích emailů do dvou skupin - na žádané a nevyžádané. Problém řešený pomocí regrese je např. predikce ceny diamantu na základě jeho parametrů. Také lze predikovat budoucí hodnoty časových řad na základě hodnot předchozích.

Do kategorie učení s učitelem patří např. tyto algoritmy:

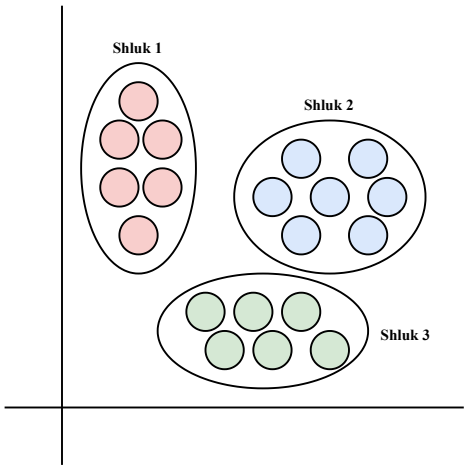
- **K-nejbližších sousedů:** Algoritmus často využívaný pro klasifikaci. Funguje na principu klasifikace instance do té třídy, do které spadá jeho k-nejbližších sousedů.
- **Lineární regrese:** Funguje na bázi proložení grafu přímkou. Často využíváno pro regresi.

- **Metoda podpůrných vektorů:** Tato metoda je založena na lineárním dělení vstupních příznaků do N skupin, kde každá skupina je asociována s patřičnou výstupní třídou.

2.1.2 Učení bez učitele

V učení bez učitele mají trénovací data podobu $x_0, x_1, \ldots, x_N$. Vstupy tedy nejsou nijak asociovány s jakoukoli výstupní kategorií či numerickou hodnotou, jako je tomu při učení s učitelem. Cílem učení bez učitele často bývá transformovat vstupní data z podoby, ve které pro lidské vnímání nemají žádnou výpovědní hodnotu či je nelze vhodně vizualizovat, do podoby, ve které již člověkem interpretovatelná jsou.

Typickým příkladem učení bez učitele je shlukování (clustering) a redukce dimenzionality (dimensionality reduction). Pří shlukování jsou jednotlivé prvky, které jsou si vzájemně dle nějaké metriky podobné, řazeny do stejné skupiny, viz obr. 2.1.



Obrázek 2.1: Příklad shlukování. Jednotlivé vzorky byly rozděleny do třech různých skupin.

Při redukci dimenzionality je cílem snížit počet nezávislých proměnných, které je třeba brát v potaz. Nezávislé proměnné, které jsou výrazně vzájemně korelované, lze zkombinovat do jedné, a tím výrazně snížit dimenzionalitu datasetu.

2.1.3 Posilované učení

Typickým příkladem posilovaného učení je počítač snažící se naučit hrát nějakou počítačovou hru či robot snažící se naučit pohybovat se v prostředí.

V posilovaném učení probíhá učení prostřednictvím agentů pohybujících se v prostředí (např. šachovnice). Agentem se rozumí někdo, kdo provádí rozhodnutí (tzv. decision maker, může se jednat o člověka či počítač). Prostředí se nachází v jednom z M možných stavů. Po provedení vybraného rozhodnutí se stav prostředí změní, a agent je nucen opět vybrat a provést jedno z možných rozhodnutí. Po provedení posloupnosti N rozhodnutí, kterými se agent dostal do koncového stavu (např. konec hry), je agent odměněn. Velikost odměny závisí na tom, jak kvalitní byla sekvence provedených rozhodnutí agentem.

2.2 Lineární regrese

Pomocí lineární regrese je možné vyjádřit vztah mezi závislou proměnnou, a jednou či více nezávislými proměnnými. Tento vztah je možné definovat jako 2.1, kde Y_i je výstupní závislá proměnná, X_i jsou jednotlivé nezávislé proměnné, a β_n jsou váhové koeficienty.

$$Y_i = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \ldots + \beta_n X_{in} \tag{2.1}$$

Každá proměnná X_i je vynásobena koeficientem β_n, a následně jsou všechny hodnoty sečteny. Tento součet je pak predikovanou hodnotou Y_i. Koeficienty β_n se počítají metodou nejmenších čtverců.

Mezi výhody lineární regrese patří jednoduchost této metody, a také snadná interpretovatelnost získaných výsledků. Dalším kladem jsou nízké nároky na výpočetní zdroje. Nevýhodou je vysoká senzitivita na odlehlé hodnoty, kdy hodnota, která se příliš odlišuje od ostatních, je schopna získané výsledky značně negativně ovlivnit. Tato podkapitola čerpá z knihy [6].

2.3 Náhodný les

Náhodný les (random forest), viz obr. 2.2, je metoda patřící do souborového učení (ensemble learning), která při učení vytváří více rozhodovacích stromů (decision tree), které pracují souběžně. Výsledkem je pak nejčastěji predikovaná třída v případě klasifikace, a průměrná hodnota v případě regrese získaná z těchto stromů.

Rozhodovací stromy fungují na principu rekurzivního dělení vstupního prostoru na podprostory. To se děje na jednotlivých podprostorech tak dlouho, dokud není dosaženo koncových listů.

Rozhodovací strom má tvar binárního stromu. V nelistových uzlech probíhá vyhodnocení podmínky. Na základě toho, zda byla daná podmínka splněna či nikoli, se dále pokračuje s vyhodnocováním levého či pravého uzlu, a tomu se děje tak dlouho, dokud algoritmus nenarazí na koncový list. Koncový list je takový, kterému je přiřazena některá z výstupních tříd či patřičná numerická hodnota. Tato podkapitola čerpá z [4].

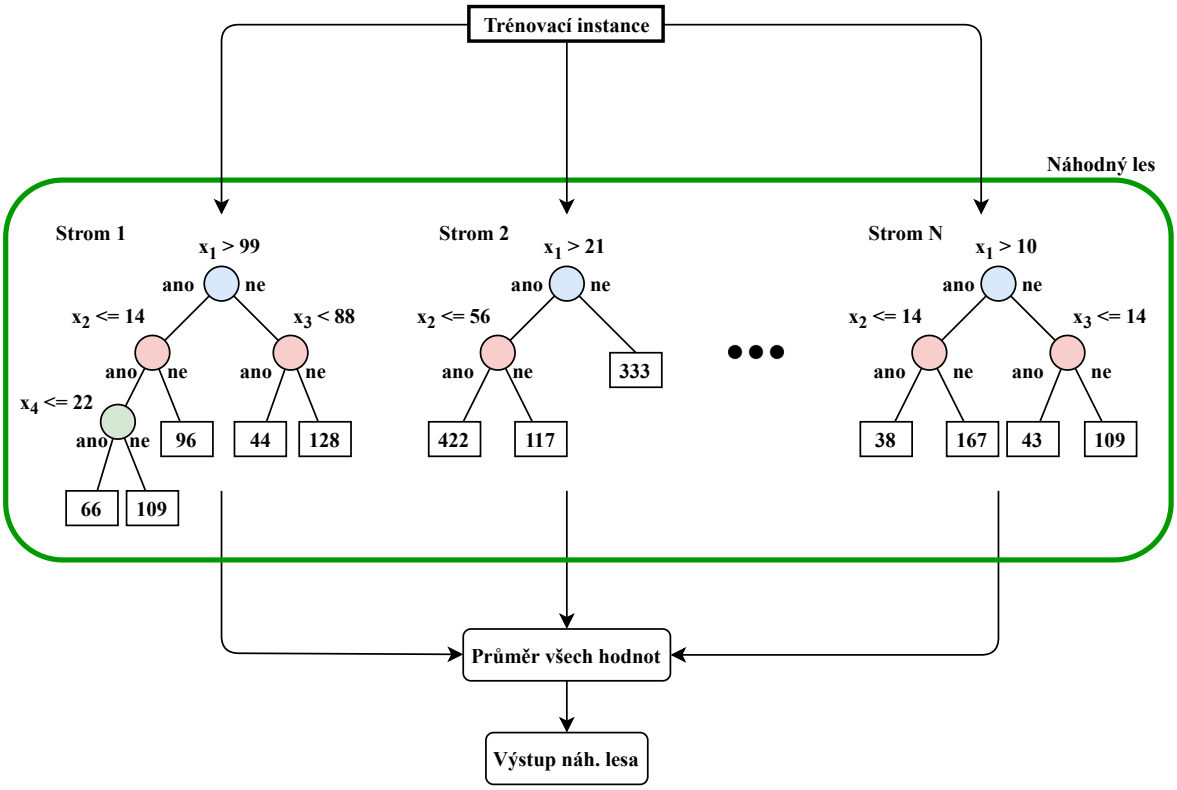
2.4 Vícevrstvý perceptron

Perceptron (obr. 2.3) je model patřící do kategorie umělých neuronových sítí. Inspirací tohoto modelu je lidský mozek. Perceptron je základní jednotkou, která přijímá vstupy buď z prostředí, či z ostatních perceptronů. Každý z vstupů, $x_1, ..., x_n$, do perceptronu je ovlivněn jemu přiřazenou synaptickou vahou $w_1, ..., w_n$. y je výstup perceptronu, který je vypočten dle vztahu 2.2. w_0 je práh (bias), který říká, o kolik musí být vážený součet větší než 0, aby se perceptron aktivoval.

$$y = sigmoid(\sum_{i=1}^{n} w_i x_i + w_0) \tag{2.2}$$

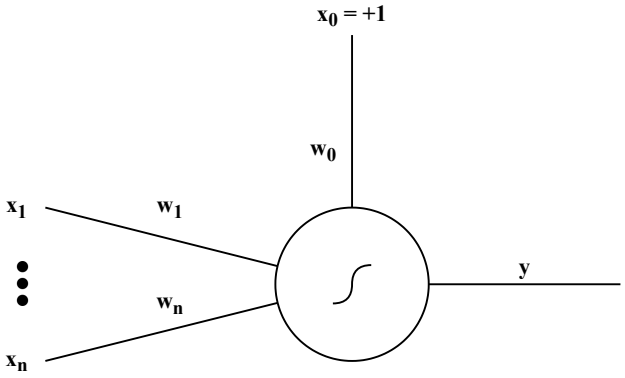
Perceptron pouze s jednou vrstvou (input layer) není sám o sobě schopný aproximovat nelineární funkce. Aproximaci nelineárních funkcí umožňují modely obsahující skryté vrstvy (hidden layer).

Model, jež obsahuje jednu či více skrytých vrstev, se nazývá vícevrstvý perceptron (obr. 2.4). Takovýto model je již schopný provádět aproximaci nelineárních funkcí. Ve vícevrstvém

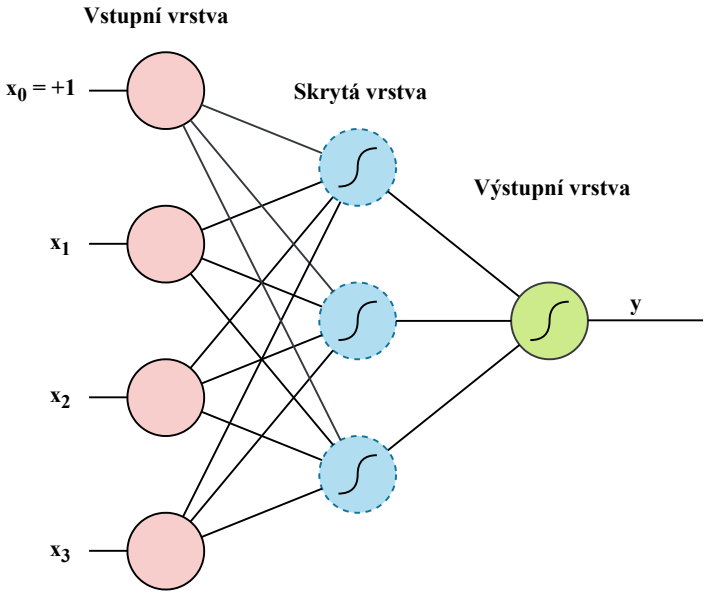


Obrázek 2.2: Příklad náhodného lesa, který provádí predikci

perceptronu vstupní vrstva přímo odpovídá vstupním hodnotám, a následně je spočítána hodnota v jednotlivých skrytých vrstvách, které sestávají z perceptronů. Výstupní vrstvu opět tvoří perceptron(y). Jedná se o plně propojenou dopřednou síť. Učení neuronových sítí může probíhat buď offline (trénování je provedeno najednou na celém datasetu), nebo online, kdy je neuronová síť trénována postupně (po jednotlivých vzorcích). Při trénování jsou počáteční váhy neuronové sítě inicializovány náhodně, a poté jsou každou iterací tyto váhy upraveny. Tato podkapitola čerpá z knihy [1].



Obrázek 2.3: Model perceptronu



Obrázek 2.4: Vícevrstvý perceptron s jednou skrytou vrstvou

Kapitola 3

Genetické programování

Genetické programování (pseudokód viz alg. 1) je variantou evolučních algoritmů, jež iterativně vyvíjí počítačové programy. Algoritmus začíná s počáteční populací náhodně vygenerovaných jedinců, a s každou iterací na selektované jedince aplikuje různé genetické operátory s cílem získat nové, a ideálně lepší jedince. Kvalitu jedince určuje hodnota jeho zdatnosti (fitness). Hlavními genetickými operátory jsou křížení a mutace, které jsou aplikovány na selektované jedince. Tato kapitola 3 čerpá z knihy [13].

- **Křížení:** Nový jedinec je vytvořen z částí náhodně vybraných z rodičovských programů.
- **Mutace:** Nový jedinec je vytvořen náhodnou změnou v náhodně vybrané části rodičovského programu.

Algorithm 1: Genetické programování v pseudokódu

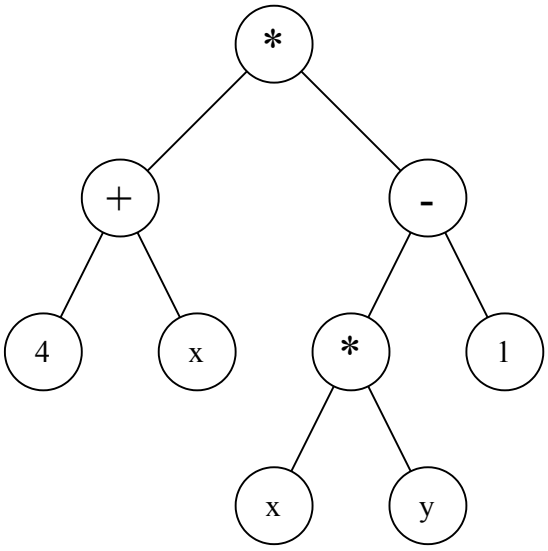
```
náhodně inicializuj počáteční populaci a ohodnoť každý program
while ukončovací podmínka nebyla splněna do
    vyber n programů z populace pro aplikaci genetických operátorů
    vytvoř nové programy aplikací genetických operátorů dle jejich
     pravděpodobností, čímž vznikne další generace
    ohodnoť každý program v nově vzniklé generaci
end
```

3.1 Reprezentace programů

V genetickém programování existuje mnoho různých reprezentací programů. Tato práce se částečně věnuje nejběžnější formě genetického programování založeném na syntaktických stromech (obr. 3.1), takzvanému stromovému genetickému programování. Mnohem více pozornosti je věnováno lineárnímu genetickému programování, které reprezentuje programy jako posloupnost instrukcí imperativního programovacího jazyka. Lineární genetické programování bude detailněji popsáno v kapitole 5.

Ve stromovém genetickém programování jsou listy stromu buď proměnné nebo konstanty, nazývané terminály. Mezi terminály patří:

- **Externí vstupy programu:** Většinou ve formě pojmenovaných proměnných, např. x či y.



Obrázek 3.1: Reprezentace výrazu (4+x)*((x*y)-1) ve stromovém GP

- **Funkce bez argumentů:** Funkce typu rand() či funkce s vedlejšími efekty (side effect). Funkce s vedlejšími efekty mohou například změnit nějakou globální strukturu či něco vykreslit na zobrazovací zařízení.
- **Konstanty:** Může se jednat o předem určené či náhodně generované konstanty.

Aritmetické operátory, jako je násobení či sčítání, jsou nazývány funkce. Množina funkcí a terminálů se dohromady nazývá množina primitiv (primitive set).

Pro bezproblémové fungování genetického programování je nutné, aby množina funkcí měla vlastnost zvanou uzávěr (closure). Tato vlastnost garantuje to, že vyhodnocením libovolného podstromu obdržíme takovou hodnotu, kterou lze využít jako argument libovolné funkce z množiny funkcí.

3.2 Inicializace populace

Počáteční populace je inicializována buď náhodně, tak jak je u evolučních algoritmů zvykem, nebo pomocí vhodné heuristiky v případě, že existují vhodná kandidátní řešení (programy, u kterých se předpokládá, že budou dobrým základem pro řešení problému). Čím větší diverzita jedinců v počáteční populaci, tím více různého genetického materiálu si jedinci mezi sebou mohou vyměnit. Základními metodami inicializace počáteční populace jsou metody „full“, „grow“, a jejich kombinace „Ramped half-and-half“. Všechny tyto metody začínají generováním kořenového uzlu, který obsahuje náhodně vybranou operaci z množiny funkcí, a liší se v tom, z jaké množiny se dále náhodně vybírají operace do vytvářených uzlů.

- **Full:** Tato metoda generuje plné stromy, což znamená, že všechny listy stromu mají stejnou, předem určenou hloubku. Neznamená to ale, že všechny stromy mají stejnou velikost (počet uzlů). To, že všechny generované uzly jsou stejně velké, nastává pouze v případě, kdy všechny funkce z množiny primitiv mají stejnou aritu (počet parametrů jež daná funkce vyžaduje).

- **Grow:** Metoda grow na rozdíl od metody předcházející negeneruje pouze plné stromy (nabízí tedy větší rozmanitost ve velikostech a tvarech stromů). Při konstrukci stromů u této metody se využívá množina funkcí i terminálů.
- **Ramped half-and-half:** Jedná se o kombinaci dvou výše zmíněných metod, kdy jedna polovina populace je inicializovaná jednou metodou, a druhá polovina metodou druhou. Pokud je maximální dovolená hloubka stromu h, potom se generují stromy s hloubkami 2,3 ... h a pro každou hloubku se v polovíně případů použije metoda Grow a ve druhé polovině metoda Full. Hlavní výhodou této metody je mnohem větší rozmanitost velikostí a tvarů jedinců v počáteční populaci.

3.3 Selekce

Genetické operátory jsou aplikovány na jedince v závislosti na jejich hodnotě zdatnosti. Čím je jedinec zdatnější, tím větší má šanci že bude vybrán. Nejpoužívanější metodou pro výběr jedinců je turnajová selekce, ale existuje jich více a je možné použít libovolnou z nich. V turnajové selekci je náhodně vybrán libovolný počet (k, $k > 1$) jedinců z populace, a tito jedinci jsou mezi sebou porovnáni. Nejlepší z nich je vybrán jako rodič. V případě, že má být provedeno křížení, je nutno provést takové turnaje dva.

3.4 Křížení a mutace

Křížení a mutace se provádí na kopiích rodičovských programů z důvodu jejich zachování pro situaci, kdy je rodič vybrán pro variaci vícekrát.

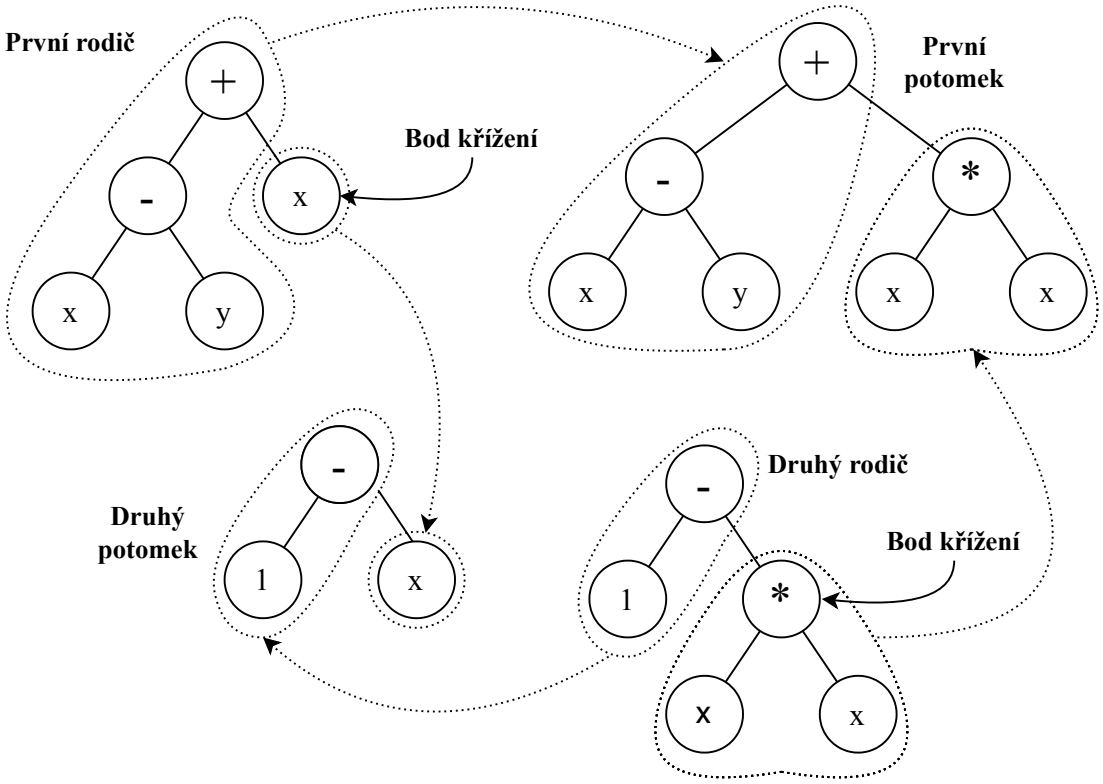
Nejčastěji využívaným typem křížení je křížení podstromu (subtree crossover) (obr. 3.2), kdy u obou rodičů se náhodně vybere bod křížení (crossover point). Bodem křížení je některý z uzlů stromu. V bodě křížení prvního rodiče je původní podstrom nahrazen podstromem z bodu křížení druhého rodiče. Výsledkem křížení být buď jeden nový jedinec, či dva (obr. 3.2). Zda bude vytvořen jeden jedinec či dva, závisí na implementaci.

Mutace bývá obvykle realizována pomocí mutace podstromu (subtree mutation), kdy je náhodně vybrán bod mutace (některý uzel ve stromu), a v tomto bodě je původní podstrom nahrazen novým, náhodně vygenerovaným, podstromem. K mutaci podstromu je též možno přistupovat způsobem, kdy v bodu mutace není vygenerován pouze nový podstrom, ale celý program. Tento přístup se nazývá „bezhlavá slepice“ [2].

Hojně využívanou formou mutace je také bodová mutace (point mutation). V tomto případě je náhodně vybrán uzel, a ten je nahrazen jiným primitivem z množiny primitiv se stejnou aritou.

Mutace podstromu obvykle nahrazuje pouze jeden podstrom, ovšem u bodové mutace je obvykle vybráno více uzlů. To, který genetický operátor bude aplikován, záleží na pravděpodobnosti aplikace operátorů (operator rates). Pravděpodobnost křížení je doporučována 90% a více, kdežto doporučená pravděpodobnost na provedení mutace se pohybuje kolem jednoho procenta.

Pokud je součet pravděpodobností na křížení a na mutaci menší než 100%, zavádí se operátor zvaný reprodukce (reproduction). Tento operátor pouze zkopíruje vítěze turnaje do další generace.



Obrázek 3.2: Ukázka křížení podstromů

3.5 Fitness funkce

Vždy, když vznikne nový jedinec (ať už je to při generování počáteční populace či aplikací některého z genetických operátorů), jeho hodnota fitness je neznámá. Úkolem funkce zdatnosti (fitness funkce) je ohodnotit jedince hodnotou fitness. Fitness funkce může vypadat libovolně, ovšem odvíjí se od řešeného problému. Typickou aplikací genetického programování je symbolická regrese. Cílem symbolické regrese je najít matematický model, který co nejlépe aproximuje daný dataset. U symbolické regrese může být fitness funkcí např. průměrná absolutní chyba 3.1.

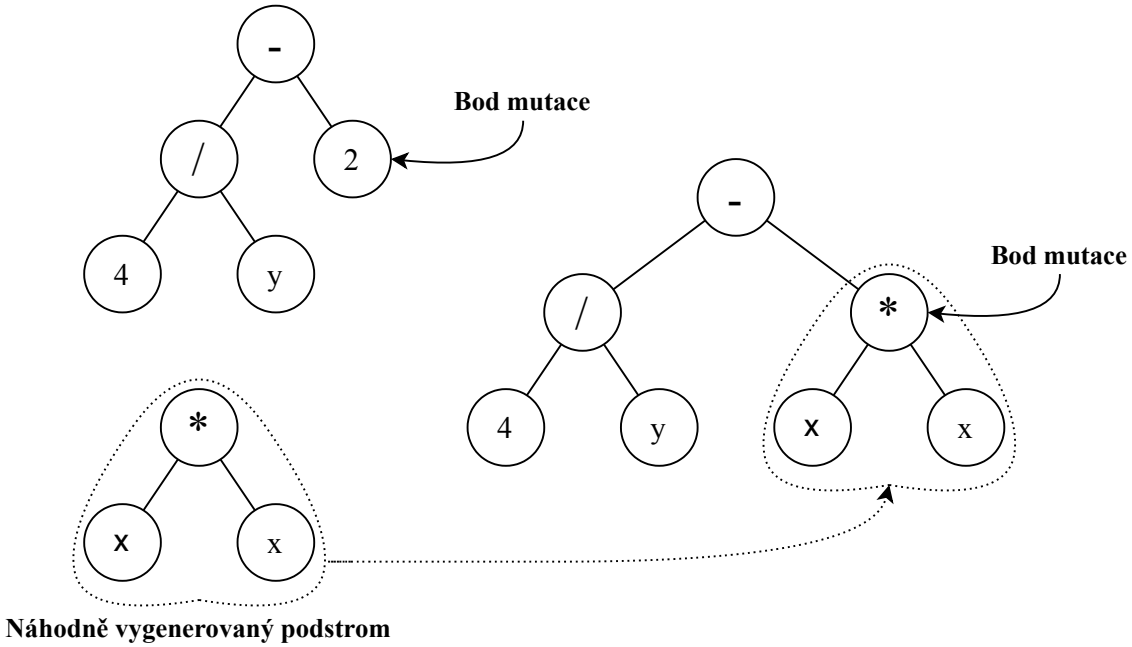
$$MAE = \frac{1}{n}\sum_{i=1}^{n}(gp(x_i) - y_i) \tag{3.1}$$

Průměrná absolutní chyba umožňuje vyjádřit míru chyb mezi dvěma vzorky, tj. v tomto případě mezi predikovanou hodnotou $gp(x_i)$, a hodnotou skutečnou (y_i), kde n je počet datových vzorků.

3.6 Použití genetického programování

Před tím, než je možné aplikovat genetické programování na vybranou úlohu, je potřeba provést několik kroků. Je nezbytné rozdělit dataset na data trénovací a testovací. Obvyklým přístupem je vyhradit 80% dat pro trénování a 20% dat pro testování.

Dále je nezbytné vybrat množinu funkcí a terminálů, které mají být při konstrukci jedinců používány. Při symbolické regresi jsou téměř vždy využívány základní matematické



Obrázek 3.3: Ukázka mutace podstromů

operace ($+$, $-$, $\times$, $/$), a následně lze volit další operace, o kterých usuzujeme, že budou při řešení problému nápomocné.

Následně je nutné definovat způsob vytvoření počáteční populace, ukončující podmínku, fitness funkci, a zvolit parametry běhů genetického programování jako jsou např. velikost populace či rozsah využívaných konstant. Je nezbytné provést několik běhů (v řádech desítek) a otestovat systém s různými hodnotami různých parametrů s cílem vybrat ty, které vedou na nejlepší výsledky.

Kapitola 4

Experimenty s knihovnou gplearn a scikit-learn

V této kapitole jsou popsány experimenty, které byly provedeny na různých datových sadách za využití knihoven gplearn[1] a scikit-learn[2]. Cílem je porovnat využitelnost jednotlivých algoritmů pro regresi.

V úvodu této kapitoly jsou krátce představeny využité knihovny. V podkapitole 4.1 se nachází přehled kroků při zpracování datových sad (dataset), které je obvykle nutné vykonat před trénováním modelu. Řešení těchto problémů je prezentováno z pohledu knihovny pandas[3], která umožňuje snadnou a rychlou manipulaci s datasety. Podkapitola 4.2 popisuje jednotlivé datasety, na kterých budou vybrané modely trénovány. V podkapitole 4.3 jsou získané výsledky a jejich porovnání.

Gplearn

Knihovna gplearn implementuje genetické programování. Pomocí této knihovny lze řešit problémy binární klasifikace (klasifikace do dvou tříd), symbolické transformace (automatická konstrukce nových nezávislých proměnných z dostupných nezávislých proměnných) a symbolickou regresi (snaha najít matematický model, který co nejlépe popisuje daná data).

Scikit-learn

Scikit-learn je volně dostupná knihovna implementující různé modely strojového učení. Tato knihovna je napsaná převážně v jazyce Python[4]. Některé interní části knihovny jsou implementovány v jazyce Cython[5] pro vylepšení výkonu. Pro výpočetně náročné operace využívá knihovny NumPy[6].

[1] https://gplearn.readthedocs.io/en/stable/
[2] https://scikit-learn.org/stable/
[3] https://pandas.pydata.org/
[4] https://www.python.org/
[5] https://cython.org/
[6] https://numpy.org/

	carat	cut	color	clarity	depth	table	price	x	y	z
1	0,23	Ideal	E	SI2	61,5	55,0	326	3,95	3,98	2,43
2	0,21	Premium	E	SI1	59,8	61,0	326	3,89	3,84	2,31
3	0,23	Good	E	VS1	56,9	65,0	327	4,05	4,07	2,31
4	0,29	Premium	I	VS2	62,4	58,0	334	4,20	4,23	2,63
5	0,31	Good	J	SI2	63,3	58,0	335	4,34	4,35	2,75

Obrázek 4.1: Ukázka dataframe z knihovny pandas

Pandas

Knihovna pandas umožňuje načíst datasety ze souborů typu (nejen) .csv (comma separated values). Načtená data ukládá jako řádky x sloupce do tzv. datových rámců (obr. 4.1) (dataframe), kde řádek přestavuje představuje jedno pozorování (observation či instance), tj. množinu vlastností, které nás o dané věci zajímají. Sloupce představují nezávislé proměnné v datasetu.

4.1 Předzpracování dat

Aby byl dataset využitelný pro trénování modelu, musí být všechna data v něm v numerické podobě. Datasety, které obsahují nenumerická, chybějící, či jinak nevhodná data, jsou tedy pro trénování naprosto nevyužitelné. Je nutné je tedy převést do podoby číselné. Tato problematika se nazývá předzpracování dat (data preprocessing). V podkapitole 4.1 je čerpáno z [5] a [7].

Chybějící data

Chybějícími daty se rozumí chybějící hodnota na řádku pro některou z nezávislých proměnných. Nejsnadnějším řešením tohoto problému je odebrat řádek s chybějící hodnotou. Ovšem tento přístup se snadno může ukázat jako poměrně nevhodný. V případě, že dataset obsahuje hodně řádků s chybějícími hodnotami, a tyto řádky se zahodí, můžeme velice snadno přijít o velkou část datasetu.

Problému zahození příliš mnoha řádků se lze vyhnout pomocí metody imputace (imputation). Imputace takové řádky nezahazuje, a namísto toho chybějící hodnoty nahradí zvolenou metodou. Chybějící hodnoty tak mohou být nahrazeny např. průměrem či mediánem ostatních hodnot, hodnotou s nejčastějším výskytem či zvolenou konstantou (často se volí hodnota 0).

Škálování dat

Dalším důležitým krokem před trénováním modelu je škálování dat (feature scaling). Neškálovaná data nabývají hodnot z různých intervalů, např. věk zaměstnanců se může pohybovat v rozsahu přibližně dvaceti až šedesáti let, kdežto výplata se pohybuje v desetitisících. Pokud nejsou data naškálovaná, pak se může stát, že některá nezávislá proměnná ovlivňuje výsledný model víc než by měla. Také jsou problémem data v jiných jednotkách.

Ty je nutné převést do stejného řádu. Dvěma nejběžnějšími metodami škálování jsou normalizace a standardizace.

Normalizace je poměrně přímočará metoda – hodnoty jsou naškálovány do intervalu <0,1> dle vztahu 4.1, kde x' je naškálovaná hodnota, x je původní hodnota, a $min(x)$ a $max(x)$ je minimální, respektive maximální hodnota, jež x nabývá.

$$x' = \frac{x - min(x)}{max(x) - min(x)} \tag{4.1}$$

Standardizace je další populární metodou pro škálování. Standardizace 4.2 nahrazuje jednotlivé hodnoty jejich z-score. Z-score udává, jak moc se x liší od průměru. V rovnici 4.2 je z z-score, x je původní hodnota, $\bar{x}$ je průměr všech hodnot x, a σ je standardní odchylka.

$$z = \frac{x - \bar{x}}{\sigma} \tag{4.2}$$

Kódování řetězcových dat

Jak již bylo výše zmíněno, algoritmy strojového učení vyžadují pouze numerická data, a data ve formě řetězců je tedy nutno zakódovat do numerické podoby. Pro jednoduché kódování lze využít metodu One hot encoding. Tato metoda zakóduje kategorické cílové proměnné do vektoru jedniček a nul. Sloupec s nulou znamená že se nejedná o danou kategorii, jednička znamená že se jedná o danou kategorii. Pro případ, kdy je potřeba přiřadit konkrétním řetězcům konkrétní hodnoty, např. při nutnosti zachování hierarchického významu jednotlivých tříd, lze využít metodu `map` z knihovny Pandas.

4.2 Využité datasety

4.2.1 Diamanty

Tento dataset obsahuje cenu v amerických dolarech a vlastnosti (jakou je např. čirost nebo počet karátů) téměř 54 000 diamantů. Jedná se tedy o poměrně rozsáhlý dataset. Tento dataset je dostupný na webu kaggle[7]. Obsahuje 9 nezávislých proměnných (vlastnosti diamantu) a 1 závislou (cenu diamantu), z toho 3 nezávislé proměnné nabývají nenumerických hodnot, které bylo potřeba převést do číselné podoby.

4.2.2 Výroba elektřiny

Druhým datasetem, na kterém budou jednotlivé algoritmy testovány, je dataset paroplynové elektrárny. Dataset obsahuje 4 nezávislé proměnné (teplota, okolní tlak, relativní vlhkost a množství upuštěného vakua). Predikovaným atributem je produkce elektrické energie za hodinu v megawattech. Tento dataset má necelých 10 000 instancí. Všechna data jsou v numerické formě, není tedy nutné tento dataset dále upravovat. Dataset je dostupný na webu kaggle[8].

4.2.3 Odolnost betonu

Třetí dataset obsahuje různé parametry betonu. Cílem je předpovědět odolnost betonu na základě těchto parametrů. Obsahuje 8 nezávislých proměnných (např. podíl cementu

[7] https://www.kaggle.com/shivam2503/diamonds
[8] https://www.kaggle.com/gova26/airpressure

Tabulka 4.1: Parametry datasetů

Dataset	Trénovací	Testovací	Celkem	Počet nezáv. prom.
Cena diamantů	43152	10788	53940	9
Pevnost betonu	824	206	1030	8
Výroba elektřiny	7654	1914	9564	4
Kritická teplota supravodičů	17010	4253	21263	82

či obsah vody), a závislou proměnnou, kterou je odolnost betonu (v MPa). Dataset je dostupný na webu UCI[9]. Autorem tohoto datasetu je I-Cheng Yeh [16]. Všechna data jsou v numerické formě, není tedy nutné tento dataset dále upravovat. Jedná se poměrně o malý dataset čítající pouze něco přes 1000 instancí.

4.2.4 Kritická teplota supravodičů

Posledním datasetetem je dataset obsahující různé vlastnosti polovodičů. Cílem je předpovědět kritickou teplotu supravodiče na základě jeho vlastností. Tento dataset obsahuje 81 nezávislých proměnných, 1 závislou proměnnou (kritickou teplotu daného supravodiče), a tvoří ho téměř 22 000 instancí. 81 nezávislých proměnných z tohoto datasetu činí poměrně složitý dataset. Všechna data jsou v numerické formě, není tedy nutné tento dataset dále upravovat. Jedná se o poměrně rozsáhlý dataset. Tento dataset je dostupný na webu UCI[10]. Autorem tohoto datasetu je Kam Hamidieh [8].

4.3 Experimenty

Hodnotící kritérium při porovnávání jednotlivých modelů je jejich koeficient determinace na daném datasetu. Koeficient determinace je dán vztahem 4.3, kde y_i je skutečná hodnota, $\hat{y}_i$ je predikovaná hodnota, a $\bar{y}_i$ je průměr skutečných hodnot.

$$R^2 = 1 - \frac{RSS}{TSS} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y}_i)^2} \tag{4.3}$$

Čitatel pak vyjadřuje reziduální součet čtverců (RSS), který vyjadřuje rozdíl mezi skutečnou hodnotou a predikovanou. Jmenovatel vyjadřuje absolutní součet čtverců (TSS), který vyjadřuje rozdíl mezi skutečnými hodnotami a jejich průměrem [14].

4.3.1 Genetické programování s knihovnou gplearn

Datasety jsou rozděleny v poměru 80% dat trénovacích, a 20% dat testovacích. Jednotlivé instance jsou do těchto dvou skupin rozděleny náhodně. Na každém z vybraných datasetů byl model natrénován celkem padesátkrát s parametry uvedenými v tabulce 4.2. Zvolené parametry byly vybrány na základě prvotních zkušebních běhů, které jsem provedl. Na obr. 4.2 je graf zobrazující získané výsledky za padesát běhů (vyjma běhů s negativním R^2 skóre) na testovacích datech. Pro sloupec s nejlepším R^2 skóre byla použita největší hodnota ze všech 50 běhů. Pro průměrnou a minimální hodnotu byly brány v potaz pouze nezáporné hodnoty.

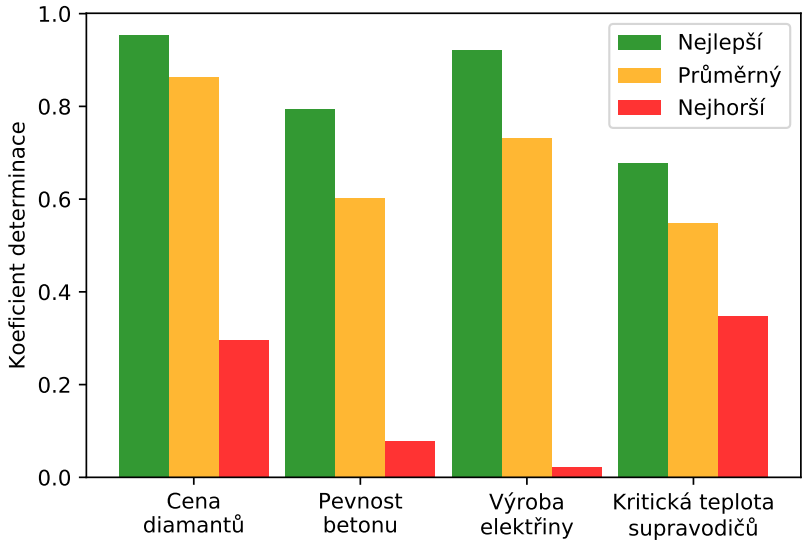
[9] https://archive.ics.uci.edu/ml/datasets/Concrete+Compressive+Strength
[10] https://archive.ics.uci.edu/ml/datasets/Superconductivty+Data

Pro dataset diamantů byl koeficient determinace kladný ve 49 z 50 případů, pro dataset pevnosti betonu ve všech 50 případech, a stejného výsledku bylo dosaženo i u datasetu kritické teploty supravodičů. Pro dataset produkce elektřiny bylo koeficient determinace kladný pouze v 36 případech z 50.

Schopnost modelu se dostatečně natrénovat byla nejvyšší u datasetu diamantů, což lze vysvětlit vysokým počtem trénovacích instancí, a poměrně nízkým, ovšem ne příliš nízkým, počtem nezávislých proměnných. Podobných, ale o něco horších výsledků bylo dosaženo u datasetu výroby elektřiny. Překvapivě pozitivních výsledků bylo dosaženo u datasetu pevnosti betonu, kdy i přes nízký počet trénovacích instancí bylo dosaženo dobrého výsledku. Nejhorších (v porovnání s ostatními datasety) výsledků bylo dosaženo u datasetu vlastností supravodičů. Důvodem bude pravděpodobně vysoká dimenzionalita tohoto datasetu.

Tabulka 4.2: Parametry

Cíl:	najít program, který najde co nejlepší vztah mezi nezávislými proměnnými a závislou proměnnou
Množina funkcí:	+, -, *, /, sqrt, abs, log
Množina terminálů:	jednotlivé nezávislé proměnné
Fitness funkce:	absolutní chyba
Selekce:	turnajová
Inicializace populace:	ramped half and half
Parametry:	70% křížení, 30% mutace podstromu
Velikost populace:	100
Velikost turnaje:	25
Počet generací:	50
Rozsah generovaných konstant:	-1.0 – 1.0
Ukončení:	dosažený maximální počet generací

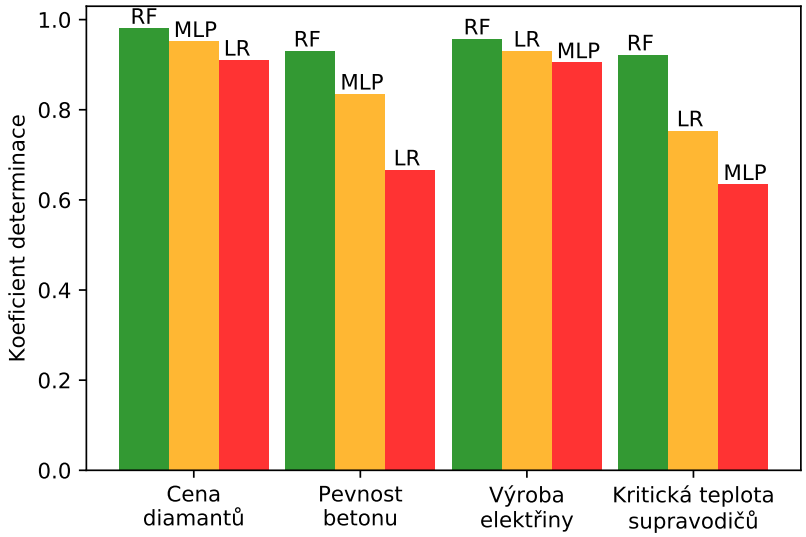


Obrázek 4.2: Hodnoty koeficientu determinace na testovacích datech s GP

4.3.2 Ostatní metody s knihovnou scikit-learn

Hodnoty koeficientu determinace získané pomocí lineární regrese, náhodného lesa a vícevrstvého perceptronu byly vzaty z jednoho trénování, neb na základě předchozích experimentů se výsledky v jednotlivých bězích lišily pouze minimálně, a brát průměr z několika běhů by tedy bylo zbytečné. Tyto algoritmy jsou spouštěny s defaultními parametry, které jsou uvedeny v dokumentaci knihovny scikit-learn.

Z těchto metod si na vybraných datasetech nejlépe vede náhodný les (RF na obr. 4.3). U datasetu diamantů a výroby elektřiny bylo dosaženo obdobných výsledků jako u genetického programování. S metodou náhodného lesa bylo dosaženo o něco lepších výsledků než s genetickým programováním na datasetu pevnosti betonu. Mnohem lépe si vedla metoda náhodného lesa u kritické teploty supravodičů. Zdá se, že náhodný les je schopný provádět dobré predikce i přes vysoký počet nezávislých proměnných. Lineární regrese a vícevrstvý perceptron vykazují podobné výsledky u datasetu diamantů a datasetu výroby elektřiny. U datasetu pevnosti betonu vítězí vícevrstvý perceptron oproti lineární regresi, a u kritické teploty supravodičů je tomu obráceně.



Obrázek 4.3: Hodnoty koeficientu determinace na testovacích datech jednotlivými metodami

Tabulka 4.3: Nejlepší dosažené R^2 koeficienty jednotlivých metod

	GP	LR	RF	MLP
Cena diamantů	0.95	0.91	0.98	0.95
Odolnost betonu	0.79	0.66	0.93	0.83
Výroba elektřiny	0.92	0.93	0.96	0.90
Krit. tep. supravodičů	0.67	0.75	0.92	0.63

Kapitola 5

Lineární genetické programování

Lineární genetické programování vyvíjí programy reprezentované jako sekvence instrukcí nějakého imperativního programovacího jazyka, a nebo jazyka symbolických instrukcí. Tato práce se věnuje té první možnosti. V rámci této práce se instrukcemi rozumí různé matematické operace a podmíněné skoky. Tyto instrukce akceptují jako operandy konstanty či proměnné, a ukládají výsledek této operace do jiné proměnné. Tyto proměnné jsou uloženy v registrech. Některé registry jsou vyhrazeny pro vstupní data, a výstup je očekáván v definovaném registru či registrech. Dalším typem registrů jsou konstantní registry. Ty jsou inicializovány náhodnou konstantní hodnotou z předem definovaného rozsahu. Posledním typem registrů jsou výpočetní registry. S obsahem těchto registrů je na rozdíl od ostatních manipulováno v průběhu výpočtu. Tato kapitola čerpá z knihy [3].

5.1 Reprezentace programů

V lineárním genetickém programování jsou programy tvořeny instrukcemi vybraného imperativního programovacího jazyka. Instrukce se skládá z operace provedené nad dvěma zdrojovými registry, a přiřazením výsledku této operace do registru cílového. Příklad takového programu je ve výpisu 5.1.

```
double r[5] // pole simulujici registry;
r[3] = sin(r[1]);
r[1] = r[2] / r[1];
r[4] = r[2] / r[3];
r[2] = sqrt(r[4]);
if (r[2] < r[1])
    r[0] = r[3] + r[2];
r[3] = r[2] - r[1];
r[0] = r[3] + r[0];
r[3] = r[0] + r[1];
```

Výpis 5.1: Příklad kandidátního programu

5.2 Kódování instrukcí

Do instrukce je nutné zakódovat čtyři informace - operaci, která má být provedena, registr, do kterého má být výsledek operace uložen, a dva zdrojové registry, které slouží jako operandy pro prováděnou operaci.

Každá instrukce může být tedy zakódována jako pole čtyř indexů - jeden index do pole instrukcí, jeden index do pole s cílovými registry a po jednom indexu do pole se zdrojovými registry pro každý z obou zdrojových registrů. Instrukce pro sečtení dvou čísel uložených v registrech r_1 a r_2 a uložení výsledku této operace do registru r_3, tedy $r_3 = r_1 + r_2$, pak může být zakódovaná do pole hodnot typu integer jako `int instrukce[4] = {1, 3, 1, 2}`, za předpokladu, že číslo 1 na první pozici reprezentuje sčítání, číslo 3 na druhé pozici reprezentuje index cílového registru, a čísla 1 a 2 reprezentují indexy zdrojových registrů.

5.3 Inicializace populace

Počáteční populace programů je inicializována, jak je u evolučních algoritmů zvykem, náhodně, avšak do jisté míry lze počáteční populaci ovlivnit. Společným prvkem všech inicializačních metod je předem daná maximální délka programu, tj. jeho celkový počet instrukcí. Standardním přístupem je volná inicializace, kdy je počáteční populace inicializována zcela náhodně. Existují ovšem i další přístupy:

- **Plně efektivní inicializace:** Programy v počáteční populaci jsou tvořeny pouze efektivním (tj. bez intronů) kódem. Tato inicializační metoda generuje programy od poslední instrukce po první.
- **Maximální inicializace:** Všechny programy v počáteční populaci mají délku rovnající se maximální délce programu.
- **Proměnlivá inicializace:** Každému programu v počáteční populaci je náhodně dlouhý z předem daného rozsahu.
- **Konstantní inicializace:** Každý program v počáteční populaci má stejnou délku.

Délka programů by měla odpovídat řešenému problému. Např. pro nalezení programu aproximující polynom $x^4 + x^3 + x^2 + x$ je zbytečně neefektivní hledat toto řešení u programů s délkou v řádu stovek instrukcí, když můžeme intuitivně očekávat vhodné řešení v řádu desítek instrukcí.

5.4 Evoluce programů

Existují dva základní přístupy k evoluci programů. Prvním je generační, kdy nová generace se skládá z `N` potomků, kde `N` je velikost populace, a novou populaci tvoří pouze tito potomci. Druhý přístup je tzv. steady-state populace, kdy potomci nahrazují jejich rodiče v původní populaci. Při tomto přístupu se pak definují ekvivalenty generací, často jako počet evaluací. Jedním generačním ekvivalentem pak může být počet evaluací roven velikosti populace.

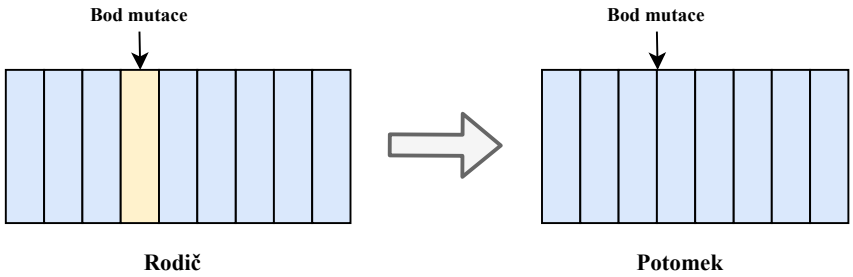
Pro aplikaci genetických operátorů jako křížení je nejprve nutné vybrat vhodné rodiče. Rodiče lze opět vybrat pomocí turnajů, jako u stromového genetického programování 3.3. Genetické operátory mutace a křížení v lineárním genetickém programování jsou si v principu podobné s těmi ve stromovém genetickém programování.

Tyto operátory lze dále rozdělit na makro a mikro. Makro operátory provádí změny na úrovni instrukcí - instrukce jsou nejmenší jednotky, které lze měnit. U mutace se tak může jednat např. o nahrazení původní instrukce nově vygenerovanou instrukcí či přidáním zcela nové instrukce. Naproti tomu mikro operátory provádí změny uvnitř instrukcí - nejmenší jednotky, které lze měnit, jsou indexy registrů. To je prakticky realizovatelné pouze mutací, kdy např. operátor sčítání je nahrazen operátorem součinu.

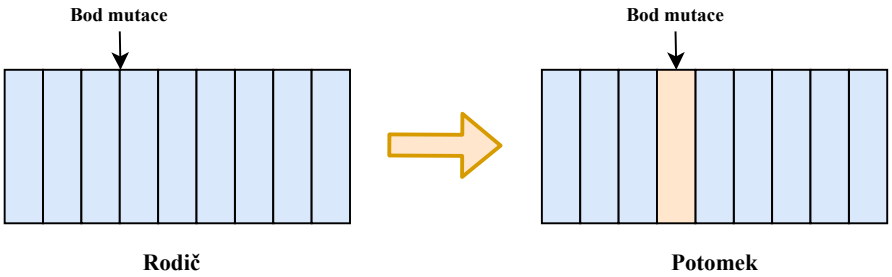
5.4.1 Mutace

V rámci této práce bude brán zřetel na následující typy (makro) mutací:

- **Destruktivní mutace:** Instrukce na náhodné pozici je z programu odstraněna (obr. 5.1.
- **Vkládací mutace:** Na náhodnou pozici v programu je vložena náhodně vygenerovaná instrukce (obr. 5.2).
- **Měnící mutace:** Instrukce na náhodné pozici je nahrazena jinou, náhodně vygenerovanou instrukcí (obr. 5.3).



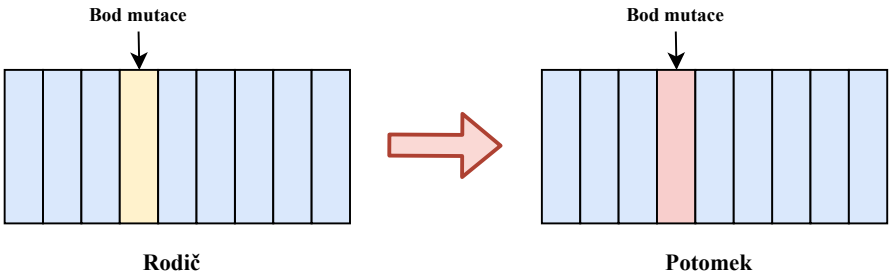
Obrázek 5.1: Destruktivní mutace



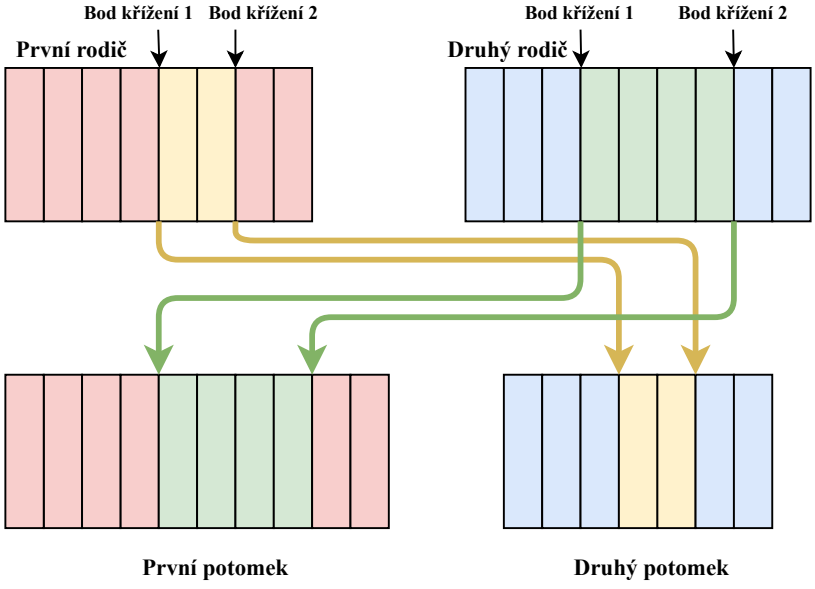
Obrázek 5.2: Vkládací mutace

5.4.2 Křížení

Křížení (obr. 5.4) je typickým makro operátorem - dva jedinci si mezi sebou vymění blok instrukcí o náhodně určené délce. Výsledkem opět může být jeden či dva jedinci. V této práci je brána v potaz ta druhá možnost.



Obrázek 5.3: Měnící mutace



Obrázek 5.4: Křížení v lineárním genetickém programování

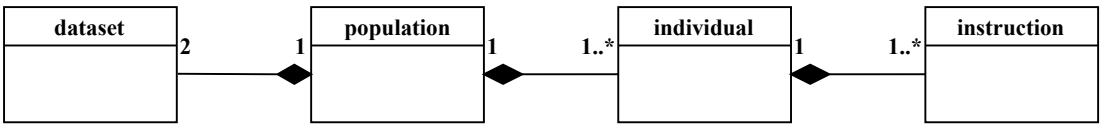
Kapitola 6

Navržená implementace LGP

V této kapitole budou popsány jednotlivé části systému, jenž byl v rámci této práce vytvořen. Jedná se o systém lineárního genetického programování, který využívá principů popsaných v předcházející kapitole. Tento systém byl vytvořen v jazyce C++[1]. Detailněji budou popsány metody třídy `population`, které uživatel přímo používá. Ostatní třídy jsou popsány jen stručně.

6.1 Návrh programu

Systém (viz obr. 6.1) tvoří několik tříd. Třída population uchovává všechny parametry pro běh, zastřešuje různé operace prováděné na jedincích a řídí evoluci. Také poskytuje různé další metody. Třída datataset slouží pro uchovávání trénovacích a testovacích dat. Třída individual slouží k reprezentaci jedince. Třída instruction představuje jednotlivé instrukce, ze kterých se jedinci skládají.



Obrázek 6.1: Třídní diagram implementovaného systému

6.2 Třída population

V této podkapitole bude popsán detailněji konstruktor této třídy a metoda `evolve`. Ostatní metody, se kterýma uživatel může též manipulovat, jsou také stručně popsány.

6.2.1 Konstruktor

Při inicializaci objektu této třídy je nutno zadat cestu k trénovacím a testovacím datům. Dále má uživatel možnost např. zvolit počty různých registrů či ovlivnit minimální a maximální délku programu. Signatura konstruktoru viz výpis 6.1.

```
population(std::string train, std::string test, int feature_index,
           int RWregs, int const_regs, std::pair <int, int> prog_len,
```

[1] http://www.cplusplus.com/

```
int size, std::vector <int> &function_set,
std::pair <int, int> const_range);
```

Výpis 6.1: Konstruktor třídy population

- `std::string train`: Cesta k souboru obsahující data pro trénování.
- `std::string test`: Cesta k souboru obsahující data pro testování.
- `int feature_index`: Číslo sloupce závislé proměnné.
- `int RWregs`: Počet výpočetních registrů.
- `int const_regs`: Počet registrů, které se mají incializovat konstatní hodnotou.
- `std::pair <int, int> prog_len`: Minimální a maximální délka programu.
- `int size`: Počet jedinců v populaci.
- `std::vector <int> &function_set`: Množina funkcí, ze kterých lze konstruovat programy.
- `const_range`: Interval, ze kterého se generují konstanty.

6.2.2 Metoda evolve

Voláním metody `evolve` (výpis 6.2) se spouští evoluce. U mutace lze určit, kolik instrukcí má být nahrazeno/přidáno/odstraněno, čímž lze určit, jak velká část daného programu se má změnit, a tedy jak rychle bude evoluce probíhat. V této metodě se nejprve instanciují objekty třídy `rng` (tato třída je popsána blíže v kapitole 6.5), které jsou využívány pro generování náhodných čísel pro různé části systému, a následně probíhá evoluce. Ta probíhá tak, že nejprve se vygeneruje pseudonáhodné číslo, a dle toho, do jakého intervalu náleží, se aplikuje na vítěze turnaje patřičný genetický operátor.

```
void evolve(int generations, std::pair <int, int> p_changeM,
          std::pair <int, int> p_addM, std::pair <int, int> p_removeM,
          int p_crossover, int t_size);
```

Výpis 6.2: Metoda evolve

- `int generations`: Počet generací, jež se má vykonat.
- `std::pair <int, int> p_changeM`: Pravděpodobnost aplikace měnící mutace na výherce turnaje, a kolik instrukcí má být nahrazeno.
- `std::pair <int, int> p_addM`: Pravděpodobnost aplikace vkládací mutace na výherce turnaje, a kolik instrukcí má být vloženo.
- `std::pair <int, int> p_removeM`: Pravděpodobnost aplikace destruktivní mutace na výherce turnaje, a kolik instrukcí má být odstraněno.
- `int p_crossover`: Pravděpodobnost aplikace křížení na vítěze turnaje.
- `int t_size`: Velikost turnaje.

6.2.3 Metoda predict

Pomocí metody `predict` (výpis 6.3) lze po natrénování modelu predikovat závislou proměnnou. Funkce přijímá jednu instanci a index závislé proměnné, a vrací dvojici predikované a skutečné hodnoty.

```
std::pair <double, double> predict(std::vector <double> instance,
                                   int feature_index);
```

Výpis 6.3: Metoda predict

6.2.4 Metoda score

Metoda `score` spočítá koeficient determinace na trénovacích datech.

```
double score();
```

Výpis 6.4: Metoda score

6.3 Třída individual

Třída `individual` reprezentuje jedince v populaci. Každý jedinec se skládá z daného počtu instrukcí, zná svou délku (počet instrukcí), a svou hodnotu zdatnosti. Signatura konstruktoru viz výpis 6.5. Tento konstruktor přijímá jako parametry i objekty `rng`, které jsou popsány v podkapitole 6.5.

```
individual(rng &rng_opcode_indices, rng &rng_d, rng &rng_s1, rng &rng_s2,
           rng &rng_len, const dataset &dtst, std::vector <double> &registers,
           std::vector <int> &function_set, int RWregs);
```

Výpis 6.5: Konstruktor třídy individual

Metoda `eval` (výpis 6.6) spočítá hodnotu zdatnosti jedince. Tato metoda je volána po iniciliazaci každého jedince, a po aplikaci každého z genetických operátorů, kdy je třeba zjistit, zda má potomek lepší či horší hodnotu zdatnosti než jeho rodič.

```
void eval(const dataset &dtst, std::vector <double> &registers,
          std::vector <int> &function_set, int RWregs);
```

Výpis 6.6: Metoda eval

6.4 Třída instruction

Třída `instruction` reprezentuje jednu instrukci. Instrukce se skládá z indexu do pole operačních kódů, indexu do pole cílových registrů, a dvou indexů do pole zdrojových registrů. Konstruktor této třídy (signatura viz výpis 6.7) přijímá čtyři různé objekty třídy `rng`, které slouží pro generování indexů do patřičných polí.

```
instruction(rng &rng_o, rng &rng_d, rng &rng_s1, rng &rng_s2);
```

Výpis 6.7: Konstruktor třídy instruction

6.5 Třída rng

Tato třída využívá pro generování náhodných čísel třídu `std::mt199937`[2]. Tato třída potřebuje pro inicilizaci tzv. semínko (seed). Semínko je dáno uplynulým časem od počátku Unix Epochy[3] `std::chrono::system_clock::now().time_since_epoch().count();` Defaultně je využíván konstruktor s náhodně vygenerovaným semínkem (výpis 6.8), ovšem lze využít i konstruktor třídy rng s předem daným semínkem, viz výpis 6.9.

```
rng(int a, int b);
```

Výpis 6.8: První konstruktor třídy rng

```
rng(int a, int b, int seed);
```

Výpis 6.9: Druhý konstruktor třídy rng

6.6 Třída dataset

Třída `dataset` reprezentuje dataset. Načítá data ze souborů typu .csv[4]. Konstruktor této třídy přijímá dva parametry, a to cestu k souboru s daty, a index závislé proměnné (výpis 6.10).

```
dataset(std::string filename, int label_index);
```

Výpis 6.10: Konstruktor třídy rng

6.7 Předvedení práce se systémem

Pro zahájení evoluce stačí instanciovat třídu `population` a následně zavolat její metodu `evolve`. Následným voláním metody `score` lze natrénovaný model na testovacích datech otestovat, a získat tím koeficient determinace na daných trénovacích datech.

```
int main() {
 /* 1:addition, 2:subtraction, 3:multiplication, 4:safe division
    5:sqrt 6:pow 7:sin 8:cos 9:tan
    10:< 11:> 12:log 13: absolute value 14:neg */
    std::vector<int> func_set {1,2,3,4,5,12,13,14};
    std::string train = R"(path/to/train/file)";
    std::string test = R"(path/to/test/file)";
    int label_index = 7;
    int POP_SIZE = 100;
    std::pair <int, int> prog_len(25, 100);
    std::pair <int, int> const_range(0.0, 10.0);
    int RW_registers = 15;
    int CONST_registers = 2;
    // init population
    population populace(train, test, label_index, RW_registers,
```

[2] http://www.cplusplus.com/reference/random/mt19937/
[3] https://www.unixtimestamp.com/index.php
[4] https://opendata.gov.cz/standardy:csv

```
    CONST_registers, prog_len, POP_SIZE, func_set, const_range);
    int n_generations = 10;
    std::pair <int, int> change_m (30, 10);
    std::pair <int, int> insert_m (15, 10);
    std::pair <int, int> delete_m (5, 10);
    int crossover = 50;
    int t_size = 10;
    // start evolution
    populace.evolve(n_generations, change_m, insert_m, delete_m, crossover, t_size);
    std::cout << populace.score() << std::endl;
    return 0;
}
```

Výpis 6.11: Ukázka práce se systémem

Kapitola 7

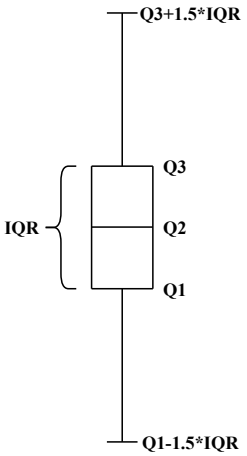
Experimenty se systémem

Oblast genetického programování je nechvalně proslulá svým nedostatkem benchmarkových úloh, a to nejen pro symbolickou regresi. Některé benchmarkové úlohy či datasety jsou sice o něco populárnější než jiné, ovšem problémem zůstává porovnatelnost jednotlivých výsledků v různých pracích z důvodu chybějící standarizace vyhodnocovacího procesu[11].

V podkapitole 7.1 je implementovaný systém otestován na několika vybraných problémech z [11]. V podkapitole 7.2 je systém otestován na vybraných datasetech, a získané výsledky jsou porovnány s výsledky získanými pomocí knihovny gplearn.

Boxplot

Výsledky jsou prezentovány pomocí krabicových diagramů (boxplot), viz obr. 7.1. Boxplot zobrazuje dolní kvartil (Q1), medián (Q2), horní kvartil (Q3), a interkvartilní rozsah (IQR = Q3 - Q1), který pokrývá 50% středových dat. IQR je v boxplotu zobrazen pomocí „krabičky“ s čárkou, která reprezentuje medián. Boxplot obsahuje vousy (whiskers), které představují minimální a maximální hodnotu, ovšem ne větší než jedenapůlnásobek IQR. Odlehlé hodnoty (outliers) jsou obvykle zobrazovány kolečkem, které ovšem nemusí být zobrazeny vždy, a nachází se mimo úsek zobrazující minimum a maximum [10].



Obrázek 7.1: Příklad boxplotu

7.1 Vybrané problémy

Parametry, jež mají jednotlivé problémy společné, jsou uvedeny v tabulce 7.1. Parametry, jež se u jednotlivých problémů liší, jsou uvedeny v patřičné tabulce u každého polynomu explicitně. Fitness funkcí, jež jsou jedinci ohodnocování, je průměrná absolutní chyba (MAE), tj. jedná se o přístup, kdy menší hodnota fitness funkce znamená lepšího jedince.

Tabulka 7.1: Společné parametry

Cíl:	aproximace konkrétní funkce
Fitness funkce:	absolutní chyba (MAE)
Selekce:	turnajová
Velikost populace:	100
Ukončení:	dosažený maximální počet generací

7.1.1 Keijzer-1

Cílem této úlohy je aproximovat funkci danou předpisem 7.1. Trénovací data tvoří 21 bodů v intervalu $< -1.0, 1.0 >$ (krok 0.1). Testovací data byla ve stejném rozsahu, ovšem s krokem 0.001. Testovací data tedy obsahují celkem 2001 testovacích případů. Výběrem intervalu generovaných konstant je cílem omezit prohledávaný prostor možných řešení, a urychlit tak proces hledání řešení.

Po 30 bězích byla získána R^2 skóre, která jsou uvedená na obr. 7.3. Na základě získaných výsledků lze usoudit, že bylo nalezeno uspokojivé řešení, a implementovaný systém tento problém zvládl úspěšně vyřešit. Vývoj hodnoty zdatnosti (tj. MAE) viz obr. 7.2.

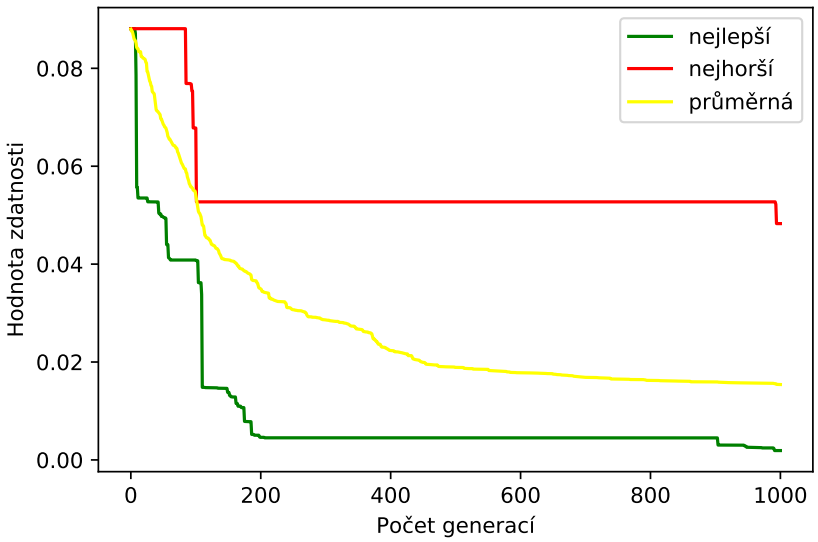
$$y = 0.3 \times sin(2 \times \pi \times x) \tag{7.1}$$

Tabulka 7.2: Parametry pro problém Keijzer-1

Množina funkcí:	+, -, *, /, sqrt, sin, log
Množina záv. prom.:	x
Velikost turnaje:	75
Parametry:	10% křížení, 50% měnící mutace (5), 30% vkládací mutace (10), 10% destruktivní mutace (5)
Počet generací:	1000
Počet výpočetních registrů:	5
Počet konstantních registrů:	3
Rozsah konstant:	0.3 - 3.2
Počáteční délka programů:	25 - 100

7.1.2 Keijzer-6

V této úloze je cílem aproximovat funkci danou předpisem 7.2. Při aproximaci této funkce nebyly využívány žádné konstantně inicializované registry, neboť by proces hledání vhodné aproximace pouze znesnadnily. Trénovací data tvoří 50 bodů v intervalu $< 1, 50 >$ (krok 1), testovací pak 120 bodů v intervalu $< 1, 120 >$ (krok 1).



Obrázek 7.2: Vývoj hodnoty zdatnosti jedinců u problému Keijzer-1

Po 30 bězích byly získány výsledky uvedené v krabicovém grafu na obr. 7.3. U této funkce se podařilo najít perfektní řešení (tj. R^2=1.0). Vývoj hodnoty zdatnosti viz obr. 7.4. Příklad nalezeného evolučního řešení viz výpis A.1

$$y = \sum_{i}^{x} \frac{1}{i} \tag{7.2}$$

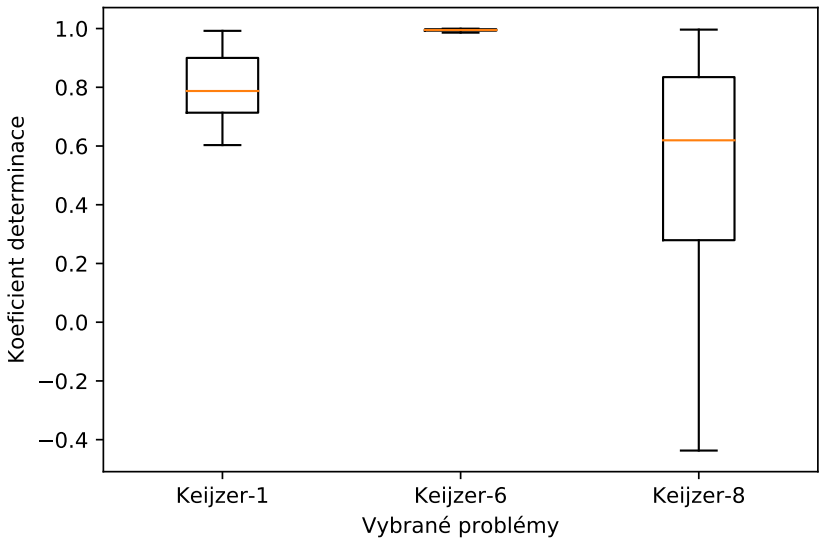
Tabulka 7.3: Parametry pro problém Keijzer-6

Množina funkcí:	+, -, *, /, sqrt, sin, log
Množina záv. prom.:	x
Velikost turnaje:	25
Parametry:	70% křížení, 10% měnící mutace (3), 10% vkládací mutace (5), 10% destruktivní mutace (2)
Počet generací:	100
Počet výpočetních registrů:	1
Počáteční délka programů:	10 - 20

7.1.3 Keijzer-8

V této úloze je cílem aproximovat funkci danou předpisem 7.3. Trénovací data tvoří 100 bodů v intervalu $< 1, 100 >$ (krok 1), testovací pak 1000 bodů v intervalu $< 1, 100 >$ (krok 0.1).

Po 30 bězích byly získány výsledky uvedené v krabicovém grafu na obr. 7.3. U této funkce se podařilo najít téměř perfektní řešení (R^2=0.996). Vývoj hodnoty zdatnosti viz obr. 7.5. Jako nezbytné pro nalezené vhodného řešení se projevily registry inicializované náhodnou konstantní hodnotou. Při hledání optimálních parametrů pro tento problém byl testován



Obrázek 7.3: Hodnoty R^2 skóre na vybraných problémech

nejprve nulový počet takto inicializovaných registrů, ovšem získané výsledky nebyly moc dobré. Postupným zvyšováním množství těchto registrů bylo dosahováno lepších výsledků. Jako vhodný počet konstantně inicializovaných registrů se ukázalo číslo 20.

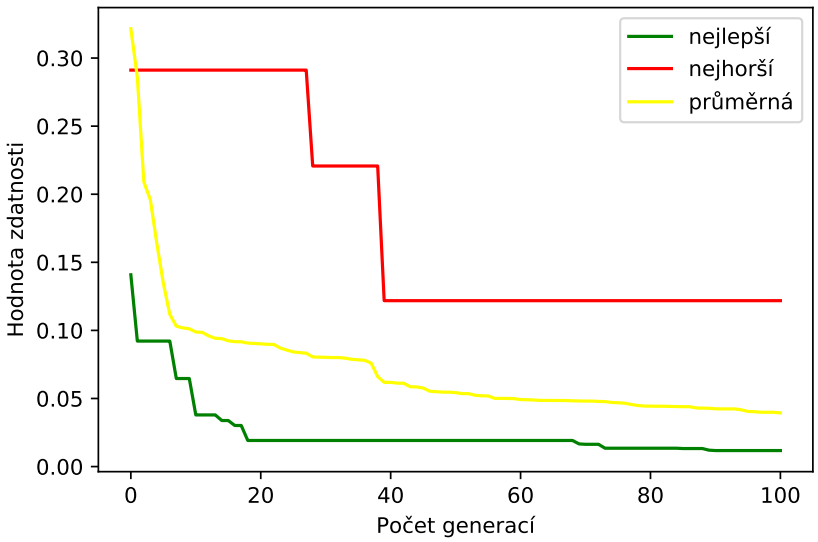
$$y = \sqrt{x} \tag{7.3}$$

Tabulka 7.4: Parametry pro problém Keijzer-8

Cíl:	aproximace konkrétního polynomu
Množina funkcí:	+, -, *, /, pow, log, abs, neg
Množina záv. prom.:	x
Velikost turnaje:	50
Parametry:	34% měnící mutace (15), 66% vkládací mutace (25), 0% destruktivní mutace
Počet generací:	500
Počet výpočetních registrů:	5
Počet konstantních registrů:	20
Rozsah konstant:	-10.0 - 10.0
Počáteční délka programů:	25 - 100

7.1.4 Vliv velikosti populace

Na základě výsledků experimentu s velikostí populace (parametry běhu viz tabulka 7.5, výsledky viz obr. 7.6) byly experimenty prováděny s velikostí populace 100. V tomto obrázku lze vidět, že větší velikost populace neovlivnila medián, a ani nejlepší získanou hodnotu, která byla dosažena pro velikost populace 100. To bude patrně z důvodu, že turnaje se účastní 75 jedinců. V případě velikosti populace o 100 jedincích se jedná o 75% všech



Obrázek 7.4: Vývoj hodnoty zdatnosti jedinců u problému Keijzer-6

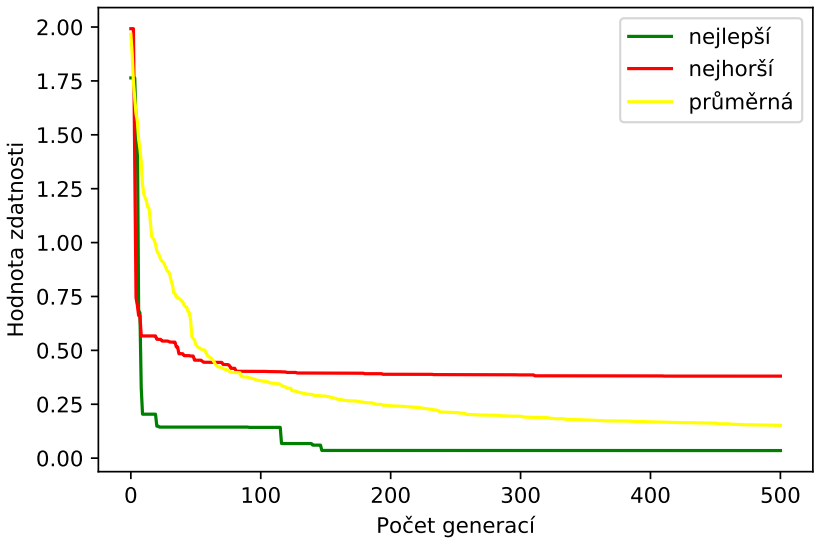
jedinců, u populace o 200 jedincích je to 37,5% všech jedinců, a u populace o velikosti 500 jedinců je to pouze 15% všech jedinců. U populace o velikosti 100 (a velikosti turnaje 75) je tedy větší šance na vybrání lepšího jedince pro aplikaci genetických operátorů.

Tabulka 7.5: Parametry pro testování velikosti populace

Cíl:	aproximace funkce (problém Keijzer-1)
Množina funkcí:	+, -, *, /, sqrt, sin, log
Množina záv. prom.:	x
Fitness funkce:	absolutní chyba (MAE)
Selekce:	turnajová
Velikost turnaje:	75
Velikost populace:	100 | 200 | 500
Parametry:	10% křížení, 50% měnící mutace (5), 30% vkládací mutace (10), 10% destruktivní mutace (5)
Počet generací:	100 | 50 | 20
Počet výpočetních registrů:	5
Počet konstantních registrů:	3
Rozsah konstant:	0.3 - 3.2
Počáteční délka programů:	25 - 100
Ukončení:	dosažený maximální počet generací

7.2 Experimenty na reálných datasetech

Byly vybrány 4 různé datasety co do počtu trénovacích instancí a množství závislých proměnných, viz kap. 4.2. Na každém z datasetů byl systém spuštěn celkem padesátkrát. Index



Obrázek 7.5: Vývoj hodnoty zdatnosti jedinců u problému Keijzer-8

závislé proměnné je pro dataset diamantů 7, pro dataset pevnosti betonu 9, pro dataset výroby elektřiny 5, a pro dataset kritické teploty supravodičů 82.

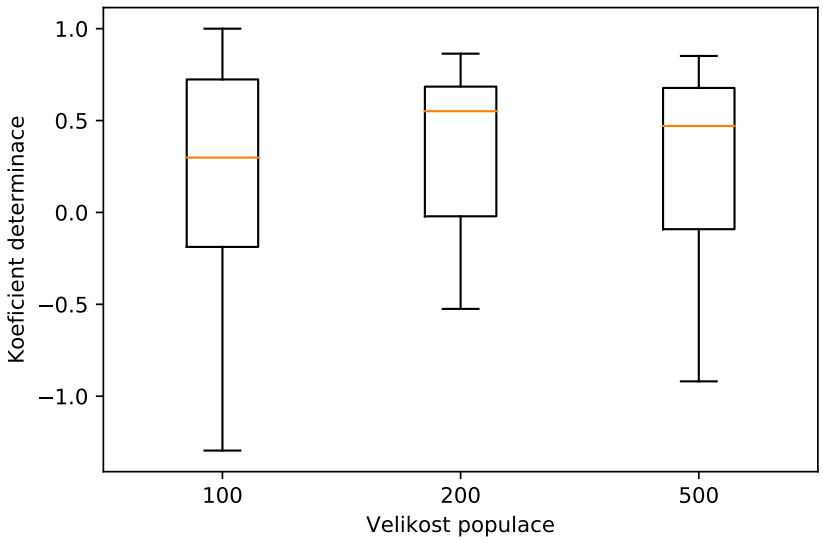
Společné parametry pro tyto datasety jsou uvedeny v tabulce 7.6 (jak pro systém LGP, tak pro stromové GP z knihovny gplearn). V tabulkách 7.7, 7.9, 7.11, 7.13 jsou uvedeny specifické parametry pro každý z běhů LGP, v tabulkách 7.8, 7.10, 7.12, 7.14 pak parametry pro běhy s knihovní implementací stromového GP. Na obr. 7.7, 7.8, 7.9, a 7.10 je uveden vývoj hodnoty zdatnosti jedinců na daném datasetu pro systém LGP.

Tabulka 7.6: Společné parametry

Cíl:	aproximace funkce popisující data
Fitness funkce:	absolutní chyba (MAE)
Selekce:	turnajová
Velikost populace:	100
Ukončení:	dosažený maximální počet generací

Tabulka 7.7: Parametry pro dataset ceny diamantů - LGP

Množina funkcí:	+, -, *, /, sqrt, pow, log
Velikost turnaje:	75
Parametry:	70% měnící mutace (10), 30% vkládací mutace (15)
Počet generací:	100
Počet výpočetních registrů:	4
Počáteční délka programů:	25 - 100



Obrázek 7.6: Vliv velikosti populace na hodnotu koeficientu determinace

Tabulka 7.8: Parametry pro dataset ceny diamantů - stromové GP

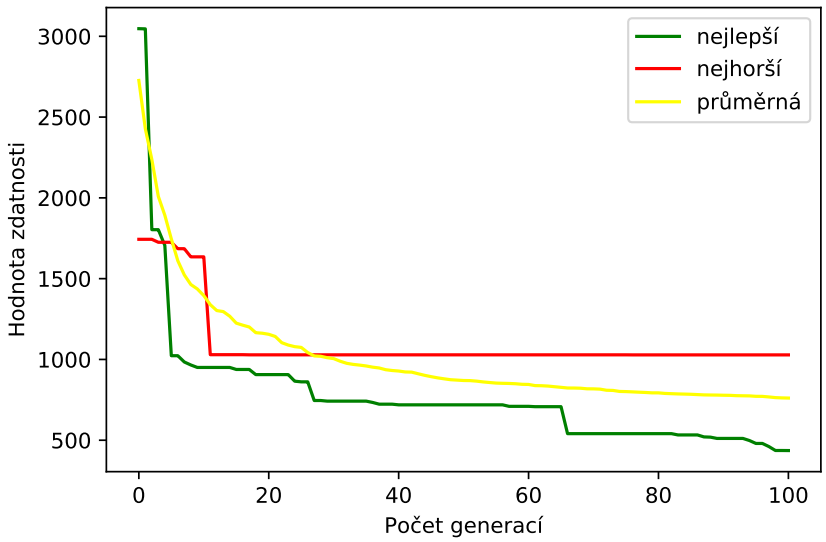
Množina funkcí:	+, -, *, /, sqrt, pow, log
Velikost turnaje:	75
Parametry:	70% mutace podstromu, 30% křížení
Počet generací:	100
Metoda inicializace populace:	half and half

7.3 Zhodnocení výsledků

V grafu na obr. 7.11 jsou uvedeny výsledky (bez outlierů) získané s implementovaným systémem a s knihovnou gplearn. Na obr. 7.12 jsou pak získané výsledky, včetně outlierů.

Jako první si lze všimnout outlierů (obr. 7.12), a to zejména u datasetů výroby elektřiny a kritické teploty supravodičů s knihovnou gplearn. U těchto datasetů outliery nabývají hodnot zcela mimo ostatní data. Jeden outlier se vyskytuje i u datasetu ceny diamantů, ovšem v porovnání s dvěma předchozími datasety se jedná o zanedbatelnou odchylku. Zdá se, že knihovní implementace stromového GP má tendenci občas nalézt extrémně špatná řešení. Implementovaný systém problémem extrémních outlierů nedisponuje.

Na datasetech pevnosti betonu a kritické teploty supravodičů si implementovaný systém oproti knihovní implementaci vede o něco hůře. Medián získaných R^2 skóre s implementovaným systémem je přibližně o dvě desetiny horší oproti knihovně, a nejlepší získané R^2 skóre jsou horší přibližně o 5 setin. O něco lépe si implementovaný systém vedl na datasetu ceny diamantů, kdy medián získaných R^2 skóre je přibližně o jednu desetinu horší než u výsledků získaných s knihovnou. Nejlepší získané R^2 skóre se liší pouze minimálně. Na datasetu výroby elektřiny je vítězem implementovaný systém. Nejlepší získaná hodnota R^2 skóre je stejná (0,91), ovšem medián získaných hodnot s knihovnou je záporný (-1,29), a s implementovaným systémem je 0,62.



Obrázek 7.7: Vývoj hodnoty zdatnosti jedinců u datasetu ceny diamantů

Tabulka 7.9: Parametry pro dataset pevnosti betonu - LGP

Množina funkcí:	+, -, *, /, sqrt, abs
Velikost turnaje:	50
Parametry:	30% křížení, 30% měnící mutace (30),
	30% vkládací mutace (30), 10% destruktivní mutace (15)
Počet generací:	200
Počet výpočetních registrů:	6
Počet konstantních registrů:	2
Rozsah konstant:	-10.0 - 10.0
Počáteční délka programů:	250 - 500

7.4 Doba trénování

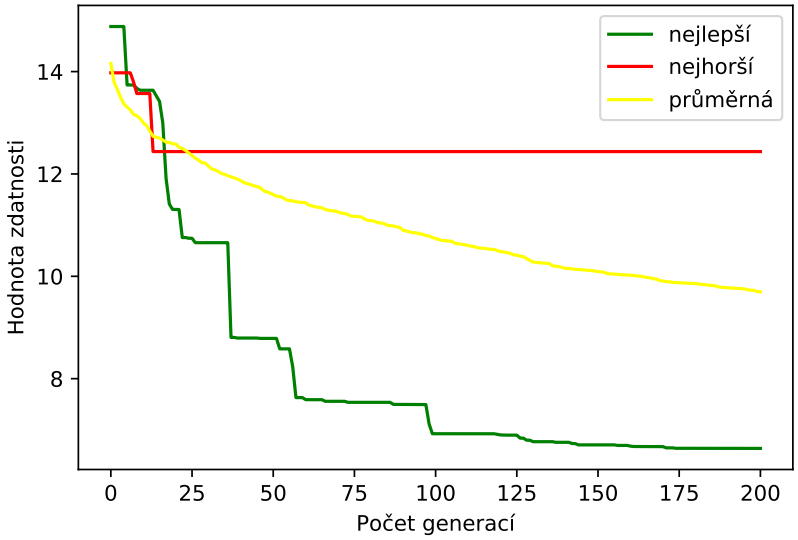
Také byla změřena doba trénování na problému Keijzer-1 (viz kap. 7.1.1) a na datasetu pevnosti betonu (viz kap. 4.2.3). Nejprve proběhlo testování na 1 000 evaluacích, a následně na 10 000 evaluacích (pro problém Keijzer-1 i pro dataset pevnosti betonu). Tento experiment se liší v počtu provedených evaluací, tj. rozdílný počet generací, než jaký je uveden v dále odkazovaných tabulkách (10 a 100 generací při populaci 100). Měření proběhlo na počítači s operačním systémem Linux Mint 19.2 (Tina), procesorem Intel Core i5-7200 s výkonem 2,5 GHz a pamětí RAM typu DDR4, velikosti 8 GB a frekvenci 1064 MHz.

Trénování implementovaného systému u problému Keijzer-1 proběhlo se stejným parametry, jaké jsou uvedeny v tabulce 7.2). Stejně tomu s parametry bylo i pro dataset pevnosti betonu (viz tabulka 7.9).

Trénování modelu s knihovnou gplearn u problému Keijzer-1 proběhlo s parametry uvedenými v tabulkce 7.16. Měření doby běhu pro dataset pevnosti betonu proběhlo se stejnými parametry, které jsou uvedeny v tabulce 7.10.

Tabulka 7.10: Parametry pro dataset pevnosti betonu - stromové GP

Množina funkcí:	+, -, *, /, sqrt, abs
Velikost turnaje:	50
Parametry:	30% křížení, 70% mutace podstromu
Počet generací:	200
Metoda inicializace populace:	half and half
Rozsah konstant:	-10.0 - 10.0



Obrázek 7.8: Vývoj hodnoty zdatnosti jedinců u datasetu pevnosti betonu

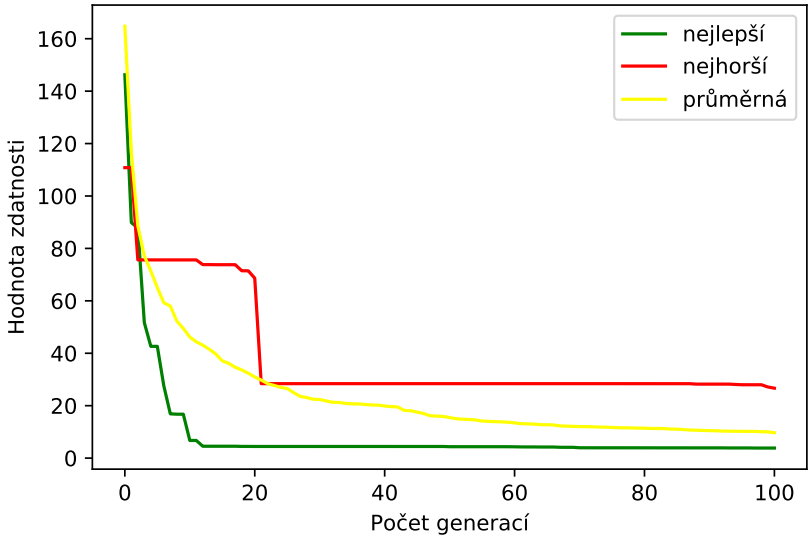
Získané výsledky jsou uvedené v tabulce 7.15. Ač implementovaný systém je výrazně pomalejší oproti knihovní implementaci na datasetu betonu, u problému Keijzer-1 systém vykazuje přibližně trojnásobné zrychlení při 1 000 evaluacích, a při 10 000 evaluacích přibližně zrychlení dvojnásobné.

Tabulka 7.11: Parametry pro dataset výroby elektřiny - LGP

Množina funkcí:	+, -, *, /, sqrt
Velikost turnaje:	50
Parametry:	30% křížení, 30% měnící mutace (10), 30% vkládací mutace (15), 10% destruktivní mutace (3)
Počet generací:	100
Počet výpočetních registrů:	6
Počáteční délka programů:	25 - 100

Tabulka 7.12: Parametry pro dataset výroby elektřiny - stromové GP

Množina funkcí:	+, -, *, /, sqrt
Velikost turnaje:	50
Parametry:	30% křížení, 70% mutace podstromu
Počet generací:	100
Metoda inicializace populace:	half and half



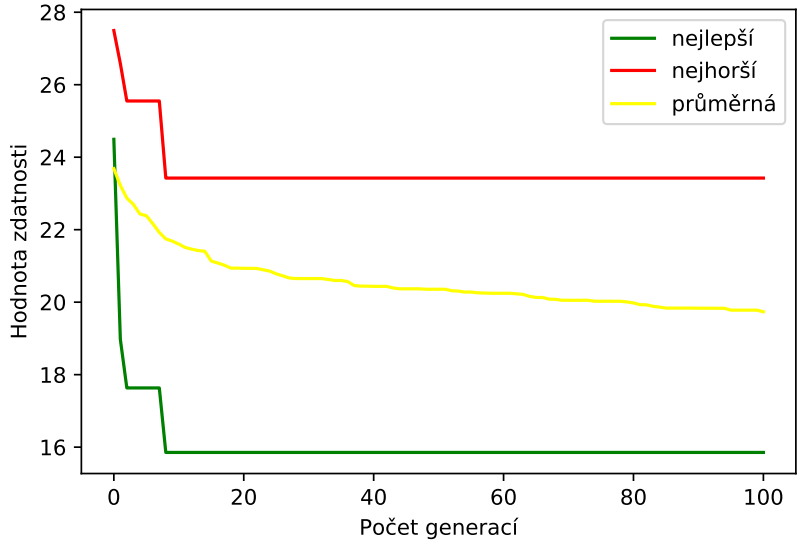
Obrázek 7.9: Vývoj hodnoty zdatnosti jedinců u datasetu výroby elektřiny

Tabulka 7.13: Parametry pro dataset kritické teploty supravodičů - LGP

Množina funkcí:	+, -, *, /, sqrt, pow, log, abs
Velikost turnaje:	75
Parametry:	65% měnící mutace (20), 35% vkládací mutace (20)
Počet generací:	100
Počet výpočetních registrů:	4
Počáteční délka programů:	100

Tabulka 7.14: Parametry pro dataset kritické teploty supravodičů - stromové GP

Množina funkcí:	+, -, *, /, sqrt, pow, log, abs
Velikost turnaje:	75
Parametry:	35% křížení, 65% mutace podstromu
Počet generací:	100
Metoda inicializace populace:	half and half



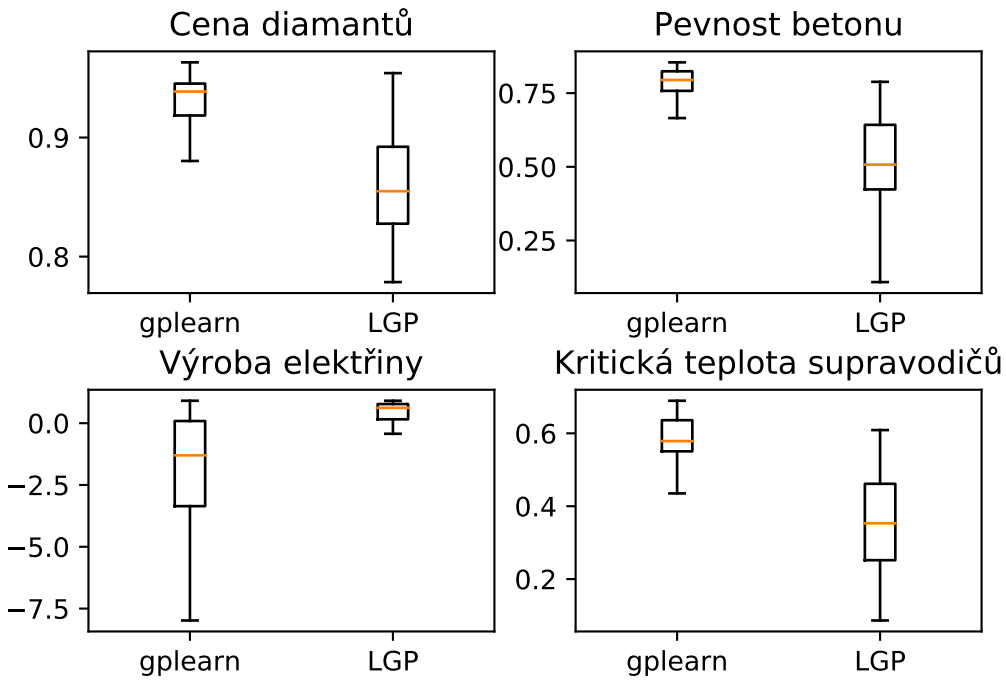
Obrázek 7.10: Vývoj hodnoty zdatnosti jedinců u datasetu kritické teploty supravodičů

Tabulka 7.15: Doba trénování v sekundách na počítači s procesorem Intel Core i5-7200 (2,5 GHz) a pamětí RAM typu DDR4, velikosti 8 GB a frekvenci 1064 MHz

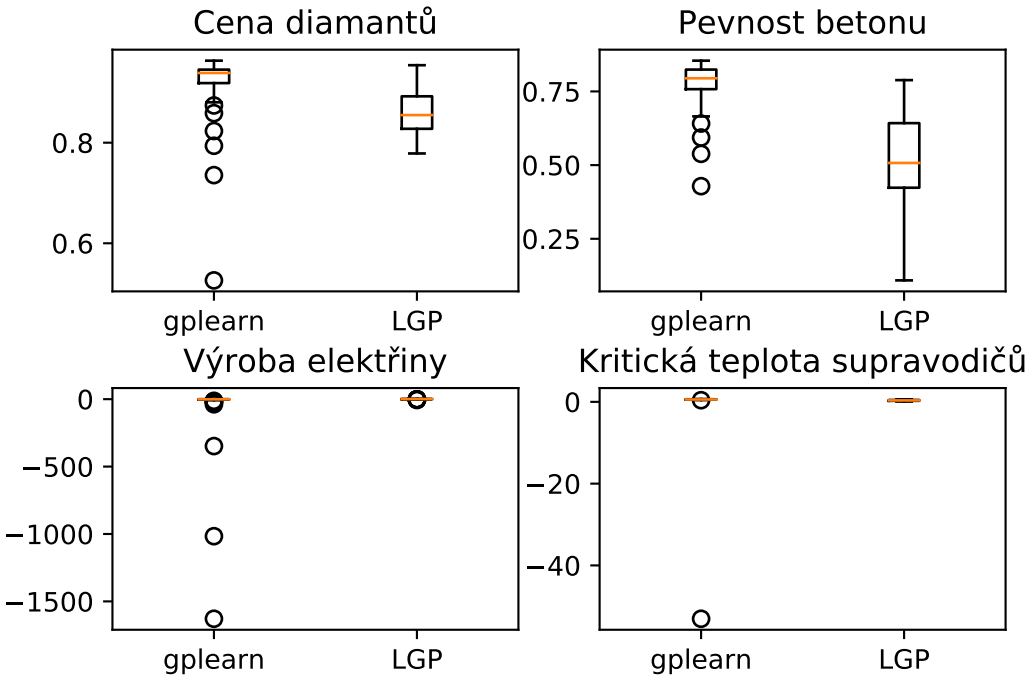
	Počet evaluací	
	1 000	10 000
Keijzer-1 (LGP)	0,4127	4,4711
Pevnost betonu (LGP)	36,704	382,7881
Keijzer-1 (gplearn)	1,3255	7,8604
Pevnost betonu (gplearn)	1,3325	10,5292

Tabulka 7.16: Parametry běhu pro problém Keijzer-1 (gplearn)

Množina funkcí:	+, -, *, /, sqrt, sin, log
Velikost turnaje:	75
Parametry:	70% mutace podstromu, 30% křížení
Počet generací:	10 | 100
Metoda inicializace populace:	half and half



Obrázek 7.11: Získané R^2 skóre (bez odlehlých hodnot)



Obrázek 7.12: Získané R^2 skóre (včetně odlehlých hodnot)

Kapitola 8

Závěr

V rámci této práce byly představeny vybrané metody strojového učení. Dále byly detailněji rozebrány principy stromového genetického programování a následně lineárního genetického programování.

Za využití knihoven scikit-learn a gplearn byly provedeny experimenty na vybraných datasetech a získané výsledky z různých metod byly porovnány. Jako vyhodnocovací metrika byl zvolen koeficient determinace, jež vyjadřuje kvalitu regresního modelu.

V této práci byl také vytvořen systém lineárního genetického programování pro řešení predikce s využitím symbolické regrese. Systém je realizován objektově v programovacím jazyce C++. Užívání tohoto systém je velice snadné - je nutné instanciovat pouze jeden objekt, a následně zavolat patřičnou metodu. Tento systém lze používat jak na operačních systémech Linux, tak na operačních systémech Windows.

Implementovaný systém byl otestován nejprve na různých benchmarkových úlohách, a následně na datasetech z reálného světa. Na zvolených benchmarkových úlohách si systém vedl poměrně dobře. Systém byl schopný najít perfektní, či téměř perfektní řešení. Úspěšnost systému na datasetech se různila. V závislosti na datasetu byl implementovaný systém schopen nalézt horší, podobné, či lepší řešení, než jaké bylo nalezeno pomocí knihovny gplearn.

V práci by bylo možné pokračovat např. rozšířením systému o různé metody inicializace počáteční populace, či třeba vyzkoušet N-bodové křížení. Dalším možným vylepšením je optimalizace systému pro větší počet evaluací.

Literatura

[1] Alpaydin, E. *Introduction to machine learning*. 2nd ed. Cambridge, Mass: MIT Press, 2010. Adaptive computation and machine learning. OCLC: ocn317698631. ISBN 9780262012430.

[2] Angeline, P. J. Subtree Crossover: Building Block Engine or Macromutation? In: Koza, J. R., Deb, K., Dorigo, M., Fogel, D. B., Garzon, M. et al., ed. *Genetic Programming 1997: Proceedings of the Second Annual Conference*. Stanford University, CA, USA: Morgan Kaufmann, 13-16 červenec 1997, s. 9–17. Dostupné z: http://ncra.ucd.ie/COMP41190/SubtreeXoverBuildingBlockorMacromutation_angeline_gp97.ps. ISBN 9781558604834.

[3] Brameier, M. F. a Banzhaf, W. *Linear genetic programming*. 1. vyd. New York: Springer, 2007. Genetic and evolutionary computation series. OCLC: ocm79871927. ISBN 9780387310299 9780387310305.

[4] Breiman, L. *Random Forests*. Kluwer Academic Publishers, říjen 2001. Dostupné z: https://doi.org/10.1023/A:1010933404324.

[5] Chatterjee, A. *Data preprocessing using Scikit-Learn and Pandas*. Prosinec 2018. Dostupné z: https://www.techvariable.com/data-preprocessing-using-scikit-learn-pandas/.

[6] Cohen, J. a Cohen, J., ed. *Applied multiple regression/correlation analysis for the behavioral sciences*. 3rd ed. Mahwah, N.J: L. Erlbaum Associates, 2003. OCLC: ocm49903199. ISBN 9780805822236.

[7] Geron, A. *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. First edition. Beijing ; Boston: O'Reilly Media, 2017. OCLC: ocn953432302. ISBN 9781491962299.

[8] Hamidieh, K. A data-driven statistical model for predicting the critical temperature of a superconductor. *Computational Materials Science*. 1. vyd. listopad 2018, roč. 154, s. 346–354. Dostupné z: https://linkinghub.elsevier.com/retrieve/pii/S0927025618304877. ISSN 09270256.

[9] Koza, J. R. *Genetic programming: on the programming of computers by means of natural selection*. Cambridge, Mass: MIT Press, 1992. Complex adaptive systems. ISBN 9780262111706.

[10] Krzywinski, M. a Altman, N. Visualizing samples with box plots. *Nature Methods*. únor 2014, roč. 11, č. 2, s. 119–120. Dostupné z: http://www.nature.com/articles/nmeth.2813. ISSN 1548-7091, 1548-7105.

[11] McDermott, J., De Jong, K., O'Reilly, U.-M., White, D. R., Luke, S. et al. Genetic programming needs better benchmarks. In: *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference - GECCO '12*. Philadelphia, Pennsylvania, USA: ACM Press, 2012, s. 791. Dostupné z: http://dl.acm.org/citation.cfm?doid=2330163.2330273. ISBN 9781450311779.

[12] Mueller, J. a Massaron, L. *Machine learning for dummies*. 1. vyd. Hoboken, New Jersey: John Wiley & Sons, Inc, 2016. For dummies. OCLC: ocn928487067. ISBN 9781119245513.

[13] Poli, R., Langdon, W. B. a McPhee, N. F. *A field guide to genetic programming*. 1. vyd. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008. (With contributions by J. R. Koza). Dostupné z: http://www.gp-field-guide.org.uk. ISBN 978-1-4092-0073-4.

[14] Renaud, O. a Victoria Feser, M.-P. A robust coefficient of determination for regression. *Journal of Statistical Planning and Inference*. 1. vyd. červenec 2010, roč. 140, č. 7, s. 1852–1862. Dostupné z: https://linkinghub.elsevier.com/retrieve/pii/S0378375810000194. ISSN 03783758.

[15] Russell, S. J., Norvig, P. a Davis, E. *Artificial intelligence: a modern approach*. 3rd ed. Upper Saddle River: Prentice Hall, 2010. Prentice Hall series in artificial intelligence. ISBN 9780136042594.

[16] Yeh, I.-C. Modeling of strength of high-performance concrete using artificial neural networks. *Cement and Concrete Research*. 1. vyd. prosinec 1998, roč. 28, č. 12, s. 1797–1808. Dostupné z: https://linkinghub.elsevier.com/retrieve/pii/S0008884698001653. ISSN 00088846.

Příloha A

Příklad evolučně nalezené řešení u problému Keijzer-6

```
r[0] = r[1] - r[1];
r[1] = log(r[1]);
r[0] = r[2] + r[2];
r[0] = sqrt(r[2]);
r[0] = r[0] / r[2];
r[1] = r[2] + r[2];
r[0] = log(r[1]);
r[1] = log(r[2]);
r[1] = r[1] - r[0];
r[1] = r[2] - r[0];
r[0] = sqrt(r[0]);
r[1] = r[2] / r[2];
r[1] = r[0] / r[1];
r[0] = sqrt(r[2]);
r[0] = log(r[1]);
r[1] = r[1] - r[0];
r[1] = r[1] * r[0];
r[1] = r[1] - r[2];
r[1] = r[1] * r[0];
r[0] = log(r[1]);
r[1] = r[1] * r[0];
r[1] = r[2] - r[0];
r[0] = r[2] + r[2];
r[0] = sqrt(r[0]);
r[0] = sqrt(r[2]);
r[0] = log(r[1]);
r[1] = r[1] - r[0];
r[1] = r[1] * r[0];
r[1] = r[1] - r[2];
r[1] = r[1] * r[0];
r[0] = log(r[1]);
r[1] = r[2] - r[0];
```

```
r[0] = r[2] / r[2];
r[0] = r[1] + r[2];
r[1] = r[2] * r[0];
r[1] = r[0] + r[1];
r[0] = log(r[0]);
r[1] = r[2] * r[0];
```

Výpis A.1: Příklad evolučně nalezeného řešení pro problém Keijzer-6. r[0] je výstupní registr, r[1] je výpočetní registr a r[2] je vstupní registr

Příloha B

Obsah přiloženého paměťového média

Odevzdaný obsah je zabalen v archivu xmacha72.zip. Archiv lze rozbalit pomocí příkazu `unzip xmacha72.zip`.

Písemná zpráva

Písemná zpráva ve formátu pdf je uložena v adresáři BP/. Zdrojový kód zprávy a ostatní závislosti jsou uloženy v adresáři BP/tex_src/.

Zdrojové kódy

Zdrojové kódy programu jsou uloženy v adresáři BP/src/, včetně přeložených souborů a spustitelného programu. V tomto adresáři se také nachází makefile pro přeložení zdrojových kódu. Přeložit lze pomocí příkazu `make`, a následné spuštění dema pomocí příkazu `make run`.

Datasety

Využité datasety se nachází v adresáři BP/data/.