VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

EVOLUČNÍ NÁVRH KLASIFIKÁTORU OBRAZŮ

EVOLUTIONARY DESIGN OF IMAGE CLASSIFIER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **MARTIN KOČI**
AUTHOR

VEDOUCÍ PRÁCE **Ing. MICHAELA DRAHOŠOVÁ, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Koči Martin**
Program: Informační technologie
Název: **Evoluční návrh klasifikátoru obrazů**
Evolutionary Design of Image Classifier
Kategorie: Umělá inteligence

Zadání:

1. Seznamte se se základními metodami strojového učení, které se používají v úlohách klasifikace. Zaměřte se na genetické programování.
2. Navrhněte program umožňující provádět evoluční návrh klasifikátoru obrazů pomocí genetického programování.
3. Program z bodu 2 implementujte a ověřte jeho funkčnost na zadaných úlohách.
4. Porovnejte dosažené výsledky s jinými přístupy.
5. Zhodnoťte dosažené výsledky.

Literatura:

- Dle pokynů vedoucí práce.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 a 2 zadání.

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

Vedoucí práce: **Drahošová Michaela, Ing., Ph.D.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 30. července 2021
Datum schválení: 30. října 2020

Abstrakt

Táto práca sa zaoberá evolučným návrhom klasifikátora obrazov pomocou genetického programovania, konkrétne kartézskeho genetického programovania. Práca popisuje teoretické základy strojového učenia, evolučných algoritmov a genetického programovania. Súčasťou práce je popísaný návrh programu a jeho implementácia. Ďalej sú vykonané experimenty na dvoch riešených úlohách pre klasifikáciu ručne písaných číslic a klasifikáciu obrázkov kociek pomocou, ktorých sa dá určiť miera demencie pri Parkinsonovej chorobe. Najlepšie navrhnuté riešenie pre čísla má AUC 0.95 a pre kocky 0.86. Navrhnuté riešenia sú porovnané s inými metódami, konkrétne konvolučnými neurónovými sieťami (CNN) a metódou podporných vektorov (SVM). Výsledná AUC pre klasifikáciu číslic, je pre obe CNN a aj SVM 0.99 pre kocky mala CNN výslednú AUC 0.81 a SVM 0.69. Kocky sú následne porovnané z existujúcim riešením, pri ktorom bola výsledná AUC 0.70, takže na základe výsledkov experimentov je vidieť zlepšenie pri použitej metóde v tejto práci.

Abstract

This thesis deals with evolutionary design of image classifier with help of genetic programming, specifically with cartesian genetic programming. Thesis discribes teoretical basics of machine learing, evolutionary algorithms and genetic programming. Part of this thesis is described design of the program and its implementation. Futhermore, experiments are performed on two solved tasks for the classification of handwritten digits and the classification of cube drawings, which can be used to determine the rate of dementia in Parkinson's disease. The best designed solution for digits is with AUC of 0.95 and for cubes 0.86. Designed solutions are compared by other methods, namely convolutional neural networks (CNN) and the support vector machines (SVM). The resulting AUC for the classification of digits for both CNN and SVM is 0.99, for cubes CNN has a final AUC 0.81 and SVM 0.69. The cubes are then compared with existing solution, which resulted in AUC 0.70, so that the results of the experiments show an improvement in the method used in this thesis.

Kľúčové slová

strojové učenie, klasifikácia, evolučné algoritmy, genetické programovanie, stromové genetické programovanie, kartézské genetické programovanie, konvolučné neuronové siete, metóda podporných vektorov, klasifikácia obrázkov

Keywords

machine learning, evolutionary algorithms, genetic programming, cartesian genetic programming, tree-based geneting programming, convolutional neural networks, support vector machines, image classification

Citácia

KOČI, Martin. *Evoluční návrh klasifikátoru obrazů*. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michaela Drahošová, Ph.D.

Evoluční návrh klasifikátoru obrazů

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pani Ing. Michaeli Drahošovej Ph.D. Uviedol som všetky literárne zdroje, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Martin Koči
30. júla 2021

Poďakovanie

Rád by som sa poďakoval Ing. Michaeli Drahošovej Ph.D., za odborné vedie, cenné rady a konzultácie k vypracovaniu tejto bakalárskej práce.

Obsah

Kapitola 1

Úvod

Dnes sa s umelou inteligenciou stretávame všade okolo nás napr. v inteligentných mobiloch, hodinkách, televízoroch, autách, prakticky vo všetkých veciach, ktoré deň čo deň používame. V dnešnej dobe máme veľa zložitých problémov, ktoré sa nedajú riešiť klasickými metódami, lebo sú veľmi výpočetne náročné (ich výpočet by mohol trvať aj niekoľko rokov). V tento moment prichádzajú na rad rôzne metódy evolučných algoritmov, neurónových sieti, celulárnych automatov alebo rôzne metódy inšpirované prírodou. V tejto práci sa používa metóda evolučných algoritmov a to práve pre ich jednoduchosť a schopnosť navrhnúť jednoduché optimálne riešenia.

Úlohou tejto práce je navrhnúť klasifikátor obrázkov, ktorý dokáže správne klasifikovať určité objekty z rôznych uhlov pohľadu. Najpoužívanejšie metódy v tejto oblasti sú napríklad konvolučné neuronové siete, ktoré majú dobré výsledky pri klasifikácii obrazov. Zároveň prinášajú aj určitú nevýhodu a to takú, že pre klasifikáciu jednotlivých objektov vyžadujú veľkú výpočetnú náročnosť a tým pádom táto metóda nieje moc vhodná pre malé zariadenia z nízkym výkonom. Preto sa väčšinou v tejto oblasti musia vstupné dáta posielať na vzdialené servery a vyhodnotenie klasifikácie poslať naspäť na dané zariadenie, alebo sa dá použiť iná metóda prívetivejšia pre hardware ako je napríklad kartézske genetické programovanie (CGP).

V tejto práci sa používa návrh klasifikátoru pomocou CGP na dvoch úlohách. Prvou úlohou je navrhnúť klasifikátor pre ručne písané číslice, ktorý ich bude správne vyhodnocovať a výsledné riešenie bude aj výpočetne prijateľnejšie pre hardware. Druhou úlohou je navrhnúť klasifikátor pre kocky, ktoré sa používajú pri teste vizuo-priestorových schopností. Tento test sa môže používať napríklad pri určovaní mieri demencie pri Parkinsonovej chorobe.

V kapitole 2 je stručne vysvetlená oblasť strojového učenia a klasifikácie. Popísané sú základné typy učenia, dve metódy strojového učenia (konvolučné neuronové siete a metódy podporných vektorov) a ako posledné metódy vyhodnocovania binárnych klasifikátorov. Základný princíp evolučných algoritmov je popísaný v kapitole 3, kde sú vysvetlené základné pojmy, ich princíp, interná reprezentácia, fitness funkcia a rôzne genetické operácie, ktoré sa v evolučných algoritmoch používajú. Špecialná pozornosť je venovaná aj kapitole 4, kde je popísané genetické programovanie, ktoré patrí pod evolučné algoritmy a konkrétne sa tam popisujú dve metódy genetického programovania (stromové genetické programovanie a kartézské genetické programovanie). Riešené úlohy, pre ktoré sa vytvára návrh klasifikátoru sú popísané v kapitole 5. Návrh riešenia klasifikácie obrazov pomocou kartézskeho genetického programovania a implementácia k tomuto návrhu sú popísané v kapitolách 6 a 7. Následne v kapitole 8 je overenie funkčnosti implementácie návrhu a sú vykonané rôzne experimenty,

pre nájdenie najlepšieho klasifikátoru a výsledky sú porovnané z inými prístupmi. V záverečnej kapitole 9 je zhrnutá odvedená práca, výsledky experimentov a možné vylepšenia práce.

Kapitola 2

Strojové učenie a klasifikácia

Strojové učenie je špecializácia počítačovej vedy, ktorá sa zaoberá vytváraním algoritmov, ktoré aby boli užitočné, sa spoliehajú na kolekciu dát. Kolekcie dát môžu pochádzať z prírody, byť vytvorené ľudmi alebo inými algoritmami. Strojové učenie má za úlohu nájsť riešenie pre danú kolekciu dát a následne dokázať predikovať a klasifikovať nové dáta.

Klasifikácia je problém, kde sa automaticky určuje označenie triedy pre neoznačenú vzorku. V strojovom učení je klasifikačný problém riešený pomocou učiacich sa klasifikačných algoritmov, ktoré berú kolekciu označených vzoriek ako vstup a vytvoria model, ktorý vie zobrať ako vstup neoznačenú vzorku a vie jej priradiť označenie. V klasifikačných problémoch je označenie definované ako konečná množina tried. Ak je veľkosť množiny 2 jedná sa o binárnu klasifikáciu, ak je ich množstvo väčšie ako 2 jedná sa o viac triednu klasifikáciu.

Táto kapitola obsahuje najskôr základné typy učenia pri strojovom učení. Následne je stručne vysvetlené čo sú konvolučné neurónové siete a algoritmy podporných vektorov, ktoré sú neskôr v práci použité ako metódy pre porovnanie výsledkov z inými prístupmi. A ako posledná čas tejto kapitoly sú spôsoby vyhodnotenia binárneho klasifikátoru, kde sa píše o matici zámen, základných spôsoboch vyhodnotenia a o pokročilejšej metóde vyhodnotenia ako je plocha pod krivkou ROC (AUC). Informácie v tejto kapitole sú čerpané z Bukov [1], Narkhede [6], [7] a James [3].

2.1 Typy učenia

Pri strojovom učení máme 4 základné typy učenia, ktorými sú: učenie s učiteľom, učenie bez učiteľa. Kombinácia učenia s učiteľom a bez učiteľa a posilňovacie učenie. Všetky informácie v tejto podkapitole sú čerpané z Burkov [1].

Učenie s učiteľom

Učenie s učiteľom je založené na tom, že algoritmus pri učení má k dispozícii dáta, ktoré obsahujú páry vstupných vektorov a očakávaných tried. Úlohou učenia s učiteľom je použiť dáta pre vytvorenie modelu, ktorý berie vstupné vektory ako vstup a očakávaný výstup používa na dedukciu pri predikcii triedy pre vstupný vektor.

Učenie bez učiteľa

V učení bez učiteľa sa používajú dáta, obsahujúce neoznačené vstupné vektory. Cieľom učenia bez učiteľa je vytvoriť model, ktorý berie vstupné vektory ako vstup a buď ich

transformuje na iný vektor alebo hodnotu, ktorá môže byť použitá na vyriešenie daného problému.

Kombinácia učenia s učiteľom a bez učiteľa

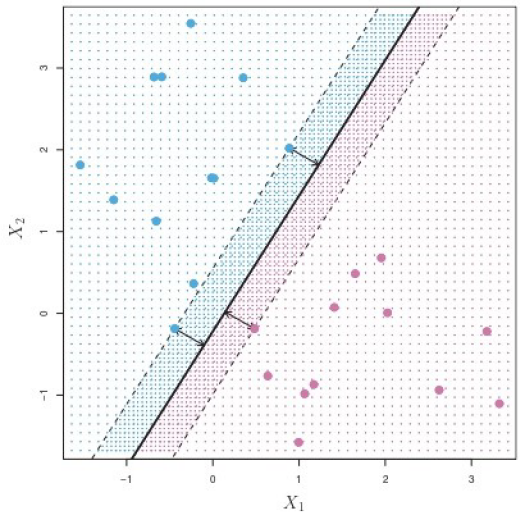
Pri kombinácii dvoch predchádzajúcich typov učenia sa používajú neoznačené vektory a malý počet označených vektorov. Cieľ učenia je rovnaký ako pri učení s učiteľom s rozdielom, že pri väčšom počte neoznačených vektorov sa dúfa, že sa nájde lepší model.

Posilňovacie učenie

Posilňovacie učenie je podmnožina strojového učenia, kde algoritmus je súčasťou prostredia a je schopný uchovávať tento stav ako vektor príznakov. Algoritmus je schopný vykonávať akcie v každom stave, čo prináša rozličné odmeny a môžu posunúť algoritmus do iného stavu prostredia. Cieľom posilňovacieho učenia je naučiť sa funkciu, pre rôzne stavy prostredia a vygenerovať optimálnu akciu, ktorá keď sa vykoná má maximalizovať priemernú odmenu.

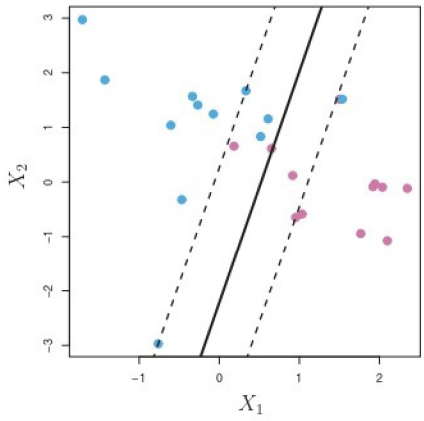
2.2 Algoritmy podporných vektorov

Algoritmy podporných vektorov (anglicky SVM – support vector machine) je generalizáciou jednoduchého intuitívneho klasifikátoru nazývaného klasifikátor maximálneho rozpätia, ktorý je vidieť na Obrázku 2.1.



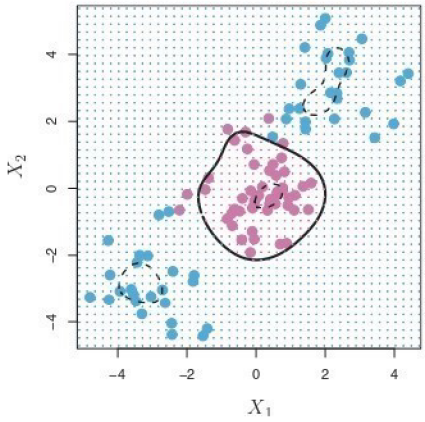
Obr. 2.1: Na tomto obrázku je vidieť príklad klasifikátoru maximálneho rozpätia. Prevzaté z [3]

Aj keď klasifikátor maximálneho rozpätia je veľmi elegantný a jednoduchý nedokáže sa aplikovať na väčšinu dátových sád a to kvôli tomu, že vyžaduje, aby boli triedy od seba oddelené lineárnou rozhodovacou hranicou. Preto vznikol klasifikátor podporných vektorov (anglicky SVC – support vector classifier) je to rozšírenie klasifikátoru maximálneho rozpätia, ktorý dokáže od seba oddeliť triedy pomocou lineárnej rozhodovacej hranice. Rozdiel medzi nimi je, že povoľuje falošne negatívne a falošne pozitívne predikované dáta v určitom rozpätí. SVC je vidieť na Obrázku 2.2.



Obr. 2.2: Na tomto obrázku je príklad SVC prevzaté z [3]

Žiaľ, ale nie všetky dátové sady sa dajú rozdeliť pomocou SVC kvôli tomu, že niektoré rozhodovacie hranice by potrebovali byť nelineárne, preto vznikol algoritmus podporných vektorov, čo je ďalšie rozšírenie od SVC a dokáže spraviť už aj nelineárne rozhodovacie hranice.



Obr. 2.3: Na tomto obrázku príklad SVM s nelineárnou rozhodovacou hranicou prevzaté z [3]

2.3 Dopredná neurónová sieť

Dopredná neurónová sieť (anglicky FFNN – feed-forward neural network) je najjednoduchší typ neurónovej siete. Dopredná neurónová sieť sa dá reprezentovať ako acyklický graf, pri ktorej informácie zo vstupu idú cez jednotlivé vrstvy priamo na jej výstup [1].

2.4 Konvolučné neuronové siete (CNNs)

Konvolučné neurónové siete (CNNs) sú špeciálny typ doprednej neurónovej siete (FFNN), ktorá výrazne znižuje počet parametrov v hlbokej neurónovej sieti bez veľkej straty na kvalite modelu. CNNs sa často používajú na klasifikáciu textu alebo obrázkov. Keď sa zamyslíme nad obrázkami, tak väčšina zo susediacich pixlov nesie rovnakú informáciu. Teda

až na výnimku ak na obrázku sú dva objekty a na ich rozmedzí, kde sa tieto dva objekty dotýkajú, jednotlivé pixle nosia inú informáciu. Tak ak dokážeme natrénovať neurónovú sieť, rozpoznať rovnaké informácie, ako aj hrany jednotlivých objektov, tak tým povolíme neurónovej sieti predikovať objekt, ktorý je reprezentovaný na obrázku. Pri CNNs sa používa metóda posuvného okna. Pritom sa môže naraz trénovať viac malých regresných modelov. Každý malý regresný model dostane ako vstup malú štvorcovú oblasť. Úlohou malých regresných modelov je naučiť sa detektovať špecifický vzor vo vstupných dát [1].

2.5 Spôsoby vyhodnotenia binárneho klasifikátoru

Klasifikácia je zložitý problém, pri ktorom potrebujeme zistiť ako je model vhodný pre riešenie daného problému. Pri tomto klasifikačnom probléme môžu pomôcť rôzne spôsoby vyhodnotenia modelov. V tejto podkapitole sa nachádzajú základné spôsoby vyhodnotenia modelov (Precíznosť, Presnosť, Citlivosť, Špecifickosť) ale aj zložitejšie metódy ako je napr. krivka ROC a plocha pod krivkou ROC (AUC). Všetky informácie v tejto podkapitole sú čerpané z Burkov [1], Narkhede [6] a [7].

Pojmy

Pre pochopenie základných a zložitejších spôsobov vyhodnotenia modelov treba pochopiť pojmy, ktoré sa budú používať v tejto podkapitole.

- **TP** – model klasifikoval vzorky dát správne ako pozitívne.
- **TN** – model klasifikoval vzorky dát správne ako negatívne.
- **FP** – model klasifikoval vzorky dát nesprávne ako pozitívne.
- **FN** – model klasifikoval vzorky dát nesprávne ako negatívne.

Matica zámen

Po získaní dátovej sady, jej vyčistení, predspracovaní a rozdelení na trénovaciu a testovaciu sadu, sa trénovacia sada poskytne algoritmu pre vytvorenie modelu. Následne testovacou sadou sa zistí schopnosť generalizácie (dokázať správne klasifikovať doteraz nevidené dáta) modelu. V tom dokáže pomôcť matica zámen. Matica zámen sumarizuje ako je klasifikačný model úspešný pri predikovaní testovacích dát, ktoré patria do jednotlivých tried. Jedna osa matici zámen sú triedy, ktoré model predikoval a na druhej osy sú triedy, ktoré sú skutočné pre danú testovaciu sadu. Tabuľka 2.1 zobrazuje výsledky klasifikácie modelu pre binárny klasifikačný problém rozpoznania jabĺk. Tabuľka obsahuje 200 testovacích dát z toho 100 sú jablká a 100 hrušky. Z tabuľky je vidieť, že model lepšie klasifikuje jablká ako hrušky a zároveň hodnoty, ktoré nám tabuľka poskytuje pomôžu pri výpočte nasledujúcich spôsobov vyhodnotenia klasifikačných modelov.

	Jablko (skutočná)	Hruška (skutočná)
Jablko (predikovaná)	90 (TP)	16 (FP)
Hruška (predikovaná)	10 (FN)	84 (TN)

Tabuľka 2.1: Matica zámen modelu pre klasifikáciu obrázkov zobrazujúca hrušky a jablká

Precíznosť

Precíznosť (anglicky precision alebo PPV – positive predictive value) ukazuje koľko pozitívne predikovaných testovacích dát je skutočne pozitívnych. Dá sa vypočítať nasledovne:

$$Precíznosť = \frac{TP}{TP + FP} \tag{2.1}$$

Pomocou tohoto vzorca sa dá z príkladu v Tabuľke 2.1 vypočítať, že precíznosť je 0.85.

Citlivosť

Citlivosť (anglicky recall alebo TPR – true positive rate) ukazuje koľko pozitívne predikovaných testovacích dát bolo predikovaných správne. Dá sa vypočítať pomocou vzorca:

$$Citlivosť = \frac{TP}{TP + FN} \tag{2.2}$$

Teraz vieme určiť, že citlivosť z výsledkov klasifikácie modelu v Tabuľke 2.1 je 0.9.

Presnosť

Presnosť (anglicky ako ACC – accuracy) určuje koľko z pozitívne a negatívne predikovaných dát boli predikované správne. Dá sa vypočítať nasledovne:

$$Presnosť = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.3}$$

Z Tabuľky 2.1 sa dá vypočítať, že presnosť je 0.87.

Špecifickosť

Špecifickosť (anglicky specificity alebo TNR – true negative rate) určuje koľko z negatívne predikovaných testovacích dát bolo skutočne predikovaných správne. Dá sa vypočítať ako:

$$Špecifickosť = \frac{TN}{TN + FP} \tag{2.4}$$

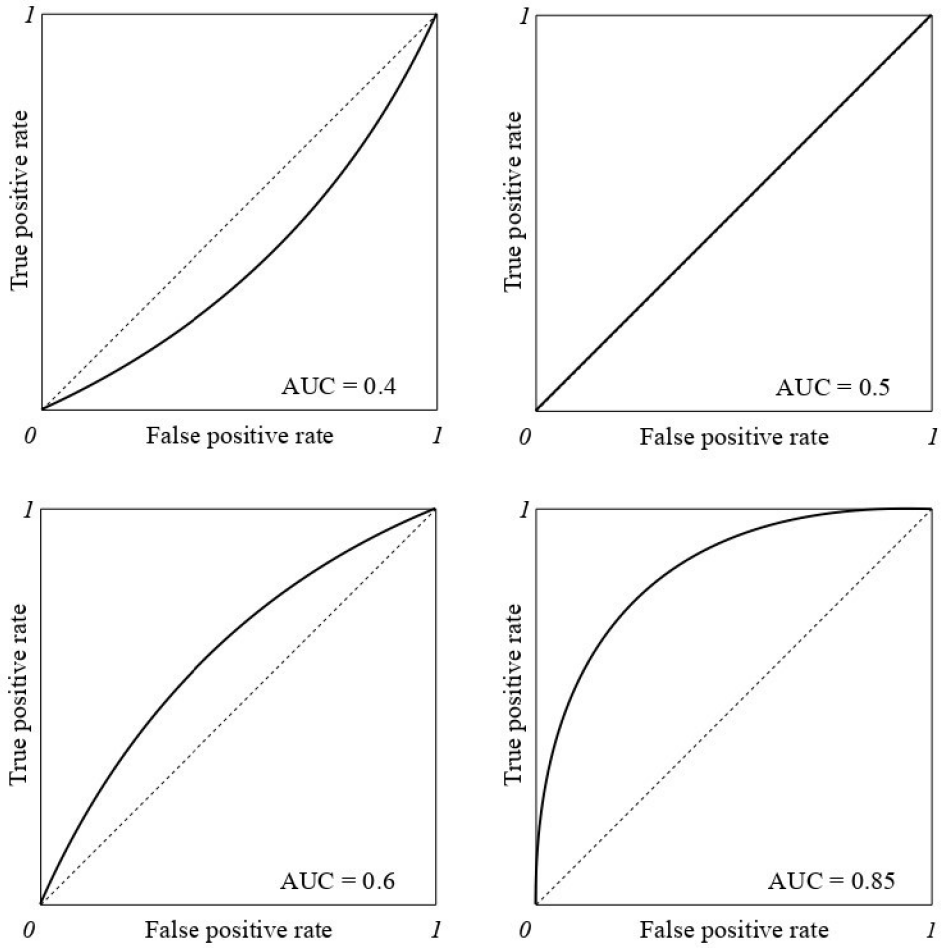
Z nasledujúceho vzorca sa dá vypočítať, že špecifickosť pre výsledky klasifikácie modelu z Tabuľky 2.1 je 0.84.

Plocha pod krivkou ROC (AUC)

ROC krivka (anglicky receiver operating characteristics, prevádzkové charakteristiky prijímača) je metóda popisu kvality klasifikátoru pri rôznych rozhodovacích prahoch. ROC je pravdepodobnostná krivka a AUC reprezentuje oddeliteľnosť medzi jednotlivými triedami. ROC krivka je vykreslená ako kombinácia TPR (citlivosti) a FPR (miera falošne pozitívnych, anglicky false positive rate), kde na ose y je znázornená TPR a na ose x FPR. Krivka je vytvorená pomocou počítania TPR a FPR pri rôznom určení rozhodovacieho prahu. FPR sa dá vypočítať ako:

$$FPR = 1 - Špecifickosť = \frac{FP}{TN + FP} \tag{2.5}$$

Na Obrázku 2.4 je vidieť viacero ROC kriviek s ich príslušnou AUC (anglicky area under curve. Z obrázku je vidieť, že ak máme rozhodovací prah na 0 všetky testovacie dáta budú predikované ako pozitívne, takže TPR a FPR budú obe 1. Na druhú stranu ak rozhodovaciu hranicu budeme mať na 1, tak všetky testovacie dáta budú vyhodnotené ako negatívne, čo znamená, že TPR a FPR budú obe 0.



Obr. 2.4: Plocha pod krivkou ROC prevzaté z [1]

.

Plocha pod krivkou ROC (AUC – anglicky area under curve, plocha pod krivkou) je reprezentované ako jedno reálne číslo, ktoré môže nadobúdať hodnoty medzi 0 až 1 vrátane. Čím je AUC väčšia tým je klasifikátor lepší. Perfektný klasifikátor by mal mať AUC rovnú 1. Klasifikátor, ktorý má AUC väčšiu ako 0.5 je lepší ako náhodný klasifikátor. Klasifikátor, ktorý má AUC nižšiu ako 0.5 väčšinou znamená, že je niečo zle s modelom lebo klasifikuje triedy opačne. Rozhodovací prah je vhodné vyberať maximalizáciou TRP a zároveň minimalizáciou FPR.

ROC pre vyhodnotenie klasifikácie viacero tried

Pri modeloch s väčším množstvom tried vieme vytvoriť N ROC kriviek pre N tried pričom každá bude reprezentovaná pomocou metódy jedna trieda proti všetkým ostatným triedam.

Kapitola 3

Evolučné algoritmy

Jedna varianta strojového učenia sú taktiež aj evolučné algoritmy. Evolučné algoritmy pochádzajú z Darwinovej evolučnej teórii, ktorá zahŕňa populáciu jedincov, ktorý sa dynamicky menia po danom čase a vo výberovom procese prežijú vždy len tí najlepší jedinci.

Evolučné algoritmy boli vytvorené pre ich prirodzený spôsob hľadania riešení pre náročné výpočtové problémy. Preto sú evolučné algoritmy inšpirované schopnosťami prírody, ale nie sú nimi obmedzované.

V tejto kapitole je popísaný základný princíp evolučných algoritmov, vnútorná reprezentácia evolučných algoritmov, vysvetlenie fitness funkcie a základné genetické operátory ako sú mutácia, kríženie a selekcia. Informácie v tejto kapitole boli prevzaté z Sekanina [8] a De Jong [2]

3.1 Pojmy

Pre dobré pochopenie problematiky je vhodné si definovať pojmy, ktoré sa často používajú v evolučných algoritmoch a sú adoptované z biológie.

- **Gen** – je základný element chromozómu a nositeľ informácie. Nadobúda hodnotu z konečnej množiny definovanej abecedy.
- **Chromozóm, jedinec, genotyp** – je zoskupenie génov, ktoré reprezentuje jeden stav stavového priestoru.
- **Fenotyp, kandidátne riešenie** – Potenciálne riešenie zadaného problému. Je zostavené na základe genotypu.
- **Populácia** – je štruktúra chromozómov, ktorá obsahuje jeden alebo viacej chromozómov.
- **Generácia** – je jedna iterácia evolučného algoritmu.

3.2 Princíp evolučných algoritmov

Základný princíp evolučných algoritmov je vytvorenie a udržovanie populácie jedincov, ktorý reprezentujú potenciálne kandidátne riešenie daného problému. Jednotlivý jedinci majú v sebe zaznamenanú ich fitness hodnotu, ktorá reprezentuje kvalitu potenciálneho kandidátneho riešenia, ktoré daný jedinec reprezentuje. Jedinci sú vyberaný, aby sa stali

rodičmi a vytvorili potomkov, to sú nové potenciálne kandidátne riešenia, ktoré sú podobné rodičovi ale nie sú rovnaké ako on. Následne sú vybraný jedinci, ktorý zomrú (budú zmazaný z populácie). Ak jeden z evolučných algoritmov je spustený po niekoľko generácií, selekcia na základe fitness pomaly zaistí zvýšenie fitness celej populácie, zaisťujúce silný zmysel pre optimalizáciu fitness. Z tohto popisu vieme vytvoriť jednoduchý obecný evolučný algoritmus:

1. Náhodne vygeneruj počiatočnú populáciu o veľkosti M
2. Opakuj nasledujúce operácie pokiaľ nejaké kritérium nieje splnené:
   (a) Vyber nejakého jedinca, ktorý sa stane rodičom pre potomkov.
   (b) Vyber nejakých jedincov, ktorý zomrú.
3. Vráť ako kandidátne riešenie jedinca, ktorý má najvyššiu pozorovanú fitness.

Pri návrhu nasledujúceho algoritmu nie je presne definovaný spôsob uloženia jedincov, veľkosť populácie, spôsob vyberania rodiča, koľko potomkov by malo byť vytvorených z rodiča, spôsob akým budú vytvorený potomkovia a ako budú vybraný jedinci čo prežijú. Všetky tieto vlastnosti algoritmu sa líšia pri každom z rozličných spôsobov implementácie evolučných algoritmov (evolučné stratégie, evolučné programovanie, genetické algoritmy, atď.). Pri evolučnej stratégii je tradičný prístup mať kandidátne riešenie reprezentované ako n-dimenzionálny vektor parametrov a obsahuje malé množstvo rodičov (napr. $M < 10$). Každý člen populácie rodičov vygeneruje 1 alebo viac potomkov klonovaním rodiča a mutáciou niektorých parametrov potomka. Spojené populácie rodičov a potomkov sa zoradia podľa fitness a prvých M prežije do novej generácie.

3.3 Interná reprezentácia

Možnosť internej reprezentácie kandidátnych riešení spadá do dvoch biológiou inšpirovaných kategórii: reprezentácia genotypom alebo reprezentácia fenotypom. Reprezentácia genotypom zakóduje kandidátne riešenia interne analogickým spôsobom ako biologický genetický kód. Čo znamená, že kandidátne riešenia sú reprezentované ako reťazec formovaný univerzálnou abecedou (napr. binárny reťazec). Genetické operátory sú definované pre manipuláciu s univerzálnou abecedou, čo zvyšuje pravdepodobnosť využitia evolučného algoritmu aj pre iné aplikácie. Pri takýchto evolučných algoritmoch je pre nový problém treba implementovať len patričný kodér/dekodér. Fenotypická reprezentácia predstavuje kandidátne riešenie priamo bez pokročilého kódovania. Genetické operácie sú definované pre manipuláciu priamo s kandidátnymi riešeniami. Čo znamená, že genetické operácie musia byť priamo upravené na použitie pre špecifický problém.

3.4 Fitness funkcia

Fitness funkcia slúži na výpočet hodnoty fitness kandidátneho riešenia, ktorá nám určuje jeho kvalitu. Pri evolučných algoritmoch a pre každého jedinca populácie sa fitness funkcia počíta v každej generácii, to znamená že je to proces, ktorý pri evolučnom algoritme zaberie veľa času a preto musí byť ideálne čo najrýchlejší. Pri rôznych problémoch sa hodnota fitness maximalizuje alebo minimalizuje. Hodnota fitness sa používa pri selektívnych mechanizmoch a napomáha ku nájdeniu globálneho optima.

3.5 Genetické operátory

Veľmi dôležitá schopnosť genetických operátorov je použiť existujúce kandidátne riešenia ako východisko a vedieť z nich vytvoriť nové zaujímavé riešenia. Z biologického prístupu máme dva možné spôsoby ako vykonávať genetické operácie a tie sú asexuálne a sexuálne. Asexuálne genetické operácie zahrňujú klonovanie jedného rodiča a aplikovanie operátoru mutácie. Táto stratégia je používaná v tradičných evolučných stratégiách a evolučnom programovaní. Sexuálne genetické operácie zahŕňajú kombináciu rôznych parametrov z viac ako jedného rodiča s malým množstvom mutácie pre vytvorenie potomka, ktorý zdedí niečo z každého rodiča. Táto metóda sa väčšinou používa v genetických algoritmoch, kde sa potomkovia tvoria z dvoch rodičov.

Operátor mutácie

Operátor mutácie vytvára z vybraného jedinca mutovaného jedinca. Ak sa jedná o binárnu reprezentáciu chromozómu tak sa jedná väčšinou o operáciu negácie nad jednotlivými bitmi. Pravdepodobnosť mutácie určuje s akou pravdepodobnosťou sa vykoná mutácia daného bitu v chromozóme, s pravidla sa určuje veľmi malá. Nízkou pravdepodobnosťou mutácie sa zaisťuje, že potenciálne kandidátne riešenie sa bude meniť pomaly po malých krokoch. Pri celočíselnej reprezentácii chromozómu, kde sú jednotlivé gény obmedzené na určité celočíselné intervaly. Treba zaistiť, že pri mutácii budú mať všetky gény hodnoty z daných intervalov.

R: 0 1 **0** 1 1 **0** **1** 0

P: 0 1 1 1 1 1 0 0

(a) Binárna reprezentácia chromozómu

R: 30 50 **43** 21 70 **63** **99** 22

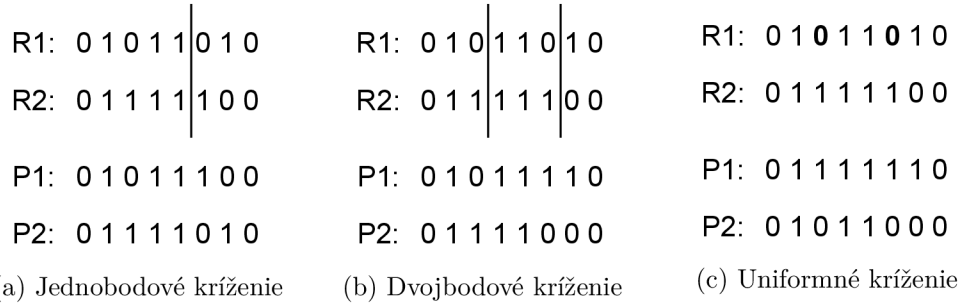
P: 30 50 70 21 70 20 10 22

(b) Reálna reprezentácia chromozómu

Obr. 3.1: Tento obrázok znázorňuje a) binárnu reprezentáciu chromozómu a ako funguje mutácia v tomto prípade len negácia vybratých bitov, b) ukazuje reálnu reprezentáciu chromozómu kde jednotlivé gény mutujú v rámci intervalu [0, 100].

Operátor kríženia

Pri operácii kríženia sa jedná o operáciu, kde sa vymieňajú informácie medzi dvomi alebo viacerými rodičmi (chromozómami). Máme viac druhou kríženia (napr. jednobodové kríženie, viacbodové kríženie a uniformné kríženie). Jednobodové kríženie pri binárnej reprezentácii chromozómov dochádza takým spôsobom, že sa vyberie jeden bod v chromozóme a od tohoto bodu sa všetky bity medzi dvomi rodičmi navzájom vymenia. Viacbodové kríženie pridáva možnosť vybratia dvoch bodov a všetky bity, ktoré sú medzi týmito dvomi bodmi si rodičia vymenia. A ako posledná možnosť je uniformné kríženie, pri ktorom sa vyberajú konkrétne bity, ktoré budú medzi chromozómami zamenené. Na Obrázku 3.2 môžeme vidieť príklad všetkých týchto variant. Pri každej z týchto variant dochádza vždy k vytvoreniu dvoch potomkov. Napríklad pri genetických algoritmoch je pravdepodobnosť kríženia rodičov možné nastaviť na 0.7 čo znamená, že 70 % populácie budú tvoriť novo vytvorený jedinci krížením a zbytok budú kópie rodičov prípadne ich zmutované varianty.



(a) Jednobodové kríženie (b) Dvojbodové kríženie (c) Uniformné kríženie

Obr. 3.2: Na tomto obrázku môžeme vidieť jednotlivé typy kríženia. R1 a R2 reprezentujú rodičov a P1 a P2 vzniknutých potomkov. Na obrázku a) môžeme vidieť jednobodové kríženie, b) dvojbodové kríženie a na obrázku c) uniformné kríženie.

Selekčné mechanizmy

Pri evolučných algoritmoch máme dve príležitosti pre selekciu: keď vyberáme rodičov a keď vyberáme jedincov, ktorý prežijú do novej generácie. Obe príležitosti nám dávajú možnosť ovplyvňovať výber pomocou hodnoty fitness. Pri selekčných mechanizmoch je veľmi dôležitý proces vedieť ako rýchlo sa chceme dostať ku lokálnemu optimu. Pri vyberaní vždy tých najlepších jedincov sa môžeme veľmi rýchlo ocitnúť v lokálnom optime a nemusí to byť zrovna najlepšie riešenie. Pri niektorých problémoch je rýchle konvergovanie do lokálneho optima adekvátne pri iných zase potrebujeme pomalý proces, pri ktorom zabránime algoritmu, aby sa veľmi rýchlo dostal do lokálneho optima. Takže pri evolučných algoritmoch je veľmi dôležité vybrať pre konkrétny problém správne množstvo selektívneho tlaku. Ak pomocou fitness hodnoty vyberáme rodičov (z ktorých sa vygenerujú potomkovia) a aj jedincov, ktorý prežijú do novej generácie tak veľmi rýchlo budeme konvergovať do lokálneho optima a výsledkom je neoptimálne riešenie. Preto sa selektívny mechanizmus pomocou fitness hodnoty vykonáva len pri jednom z týchto procesov selekcie a druhý sa vykonáva nejakým iným spôsobom. Pri selektívnom procese výberu pomocou fitness máme viacero metód, ktoré môžeme použiť a sú zoradené zostupne podľa selektívneho tlaku:

- **Deterministická selekcia** – vyber N najlepších jedincov podľa fitness hodnoty
- **Turnajová selekcia** – opakovane vyber náhodne K jedincov a z K vybraných jednocov ponechaj len tých najlepších
- **Selekcia podľa poradia** – Vyberanie jedincov proporcionálne podľa ich poradia fitness v populácii.
- **Proporcionálna selekcia** – Vybranie jedincov proporcionálne podľa hodnoty ich fitness

Proporcionálna a turnajová selekcia sú rovnaké ako v prirodzených evolučných biologických systémoch. Deterministická selekcia a selekcia podľa poradia sú podobné stratégiám, ktoré sa používajú vo farmárskych chovoch. Pri evolučných algoritmoch dosť záleží na tom či medzi sebou rodičia a potomkovia súťažia o prežitie alebo nie. Z matematického modelu v evolučných systémoch sa robí predpoklad, že len potomkovia prežijú. Tento predpoklad primárne uľahčuje matematickú analýzu. Z perspektívy počítačovej vedy, rozhodnutie či spolu rodičia a potomkovia budú súťažiť bude ovplyvňovať rýchlosť konvergovania v evolučných algoritmoch. Keď spolu rodičia a potomkovia súťažia tak to vytvára rýchlejšiu

konvergenciu ako keď spolu nesúťažia. Jedna z posledných vecí, na ktoré si treba dávať pozor pri evolučných algoritmoch je dosiahnuť efektívnu rovnováhu medzi prehľadávaním a vykorisťovaním. Tradičné evolučné algoritmy majú veľmi silnú selekciu na prežitie, ale kompenzujú si to agresívnymi mutáciami. Na rozdiel od evolučných algoritmov, genetické algoritmy majú menší selektívny tlak a majú miernejšiu formu genetických operácii.

Kapitola 4

Genetické programovanie

Genetické programovanie je evolučný prístup, ktorý rozširuje genetické algoritmy o prehľadávanie priestoru v počítačových programoch. Tak isto ako aj pri iných evolučných algoritmoch, genetické programovanie pracuje na definovaní cieľa pomocou kvalitatívneho kritéria (fitness) a následne používa toto kritérium na evolúciu množiny (populácii) kandidátnych riešení (jedincov), napodobňovaním základných princípov Darwinovej evolúcie. Genetické programovanie plodí riešenia na problémy pomocou iteračného procesu, ktorý zahŕňa pravdepodobnostnú selekciu najlepších riešení a ich variant vytvorených pomocou genetických operátorov, obvykle kríženia a mutácie.

Rozdiel medzi genetickými algoritmami a genetickým programovaním je, že v prvom sú riešenia reprezentované ako reťazce pevnej dĺžky, na druhú stranu pri druhom sú reprezentované ako hierarchická štruktúra variabilnej veľkosti, ktorá reprezentuje počítačový program. Pri genetických algoritmoch, ktoré riešenia reprezentovali ako reťazce pevnej dĺžky vznikal problém, že boli neprirodzené a veľmi obmedzujúce pre veľké množstvo aplikácii, tento problém vyriešilo genetické programovanie.

Táto kapitola sa zaoberá dvomi variantami genetického programovania využívajúce reprezentáciu pomocou stromovej štruktúry a kartézskej mriežky. Všetky informácie v tejto kapitole sú prevzaté z Miller [5], Vanneschi [10] a Sekanina [8].

4.1 Stromové genetické programovanie

Stromové genetické programovanie je najstaršou a zároveň aj najpoužívanejšou variantou genetického programovania, ktoré využíva hierarchické stromové štruktúry sploštené do reťazca.

Princíp stromového genetického programovania

Vzor genetického programovania funguje vykonávaním nasledujúcich krokov:

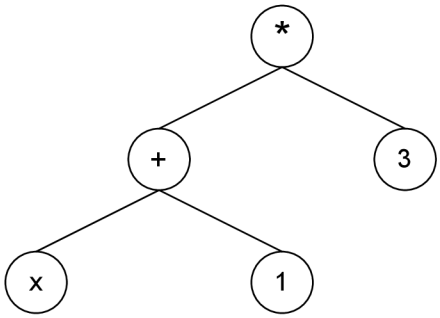
1. Vygeneruj počiatočnú populáciu jedincov.
2. Vykonávaj nasledujúce kroky pokým nie je uspokojená ukončovacia podmienka:
   (a) Ohodnoť každého jedinca v populácii.
   (b) Vytvor novú populáciu použitím nasledujúcich krokov:
      i. Pravdepodobnostne vyber množinu jedincov pre párenie na základe ich fitness.

ii. Skopíruj niektorých z vybraných jedincov (bez toho aby si ich modifikoval) do novej populácie.
iii. Vytvor nových jedincov pomocou kríženia medzi dvomi vybranými rodičmi.
iv. Vytvor nových jedincov pomocou mutácie náhodne vybraného jedinca.

3. Najlepší jedinec, ktorý sa objavil v ľubovolnej generácii je navrhnutý ako výsledok procesu genetického programovania. Tento výsledok môže byť riešenie, alebo to môže byť aproximácia riešenia daného problému.

Reprezentácia jedincov

Množina všetkých možných štruktúr zahŕňa všetky stromy, ktoré dokážu byť vytvorené rekurzívne z množiny funkčných a terminálnych symbolov. Každá funkcia má pevný počet vstupov (arita). Funkcie môžu obsahovať matematické funkcie, aritmetické, booleovské, podmienené, iteračné alebo iné operácie, ktoré budú definované presne pre daný problém. Všetky terminály sú buď premenné alebo konštanty.



Obr. 4.1: Príklad stromu s množinou funkcii $\mathscr{F} = \{+, *\}$ a terminálov $\mathscr{T} = \{x, 1, 3\}$

Je dobrou praktikou vyberať množiny funkcií a terminálov také, ktoré uspokoja dve podmienky: uzavretosti a dostatočnosti. Podmienka uzavretosti požaduje, aby každá funkcia z množiny funkcií bola definovaná pre všetky kombinácie hodnôt, ktoré môže dostať ako vstupné hodnoty. Táto podmienka je dôležitá, lebo potrebujeme vedieť jednotlivé programy (jedincov) spustiť a priradiť im ich hodnotu fitness. Ak by sa nepodarilo spustiť program, tak by to viedlo buď k zlyhaniu, alebo by priviedlo celý systém do nepredvídateľných výsledkov. Podmienka dostatočnosti požaduje, aby množiny terminálov a funkcií dokázali vyjadriť riešenie k danému problému.

Inicializácia populácie

Inicializácia populácie je prvý krok v evolučnom procese. Zahŕňa vytvorenie štruktúr programu, ktoré sa budú počas jeho behu vyvíjať. Najčastejšie používané metódy, ktoré sa používajú pri stromovom programovaní sú grow, full a ramped half-and-half metóda. Tieto metódy sú všetky kontrolované maximálnou povolenou hĺbkou d stromu.

Grow metóda sa dá popísať nasledujúcim algoritmom:

1. Pomocou uniformnej pravdepodobnosti je vybraný jeden náhodný symbol z množiny funkcií, ktorý je použitý ako koreň stromu; nech n je arita vybraného symbolu funkcie.

2. Uniformnou pravdepodobnosťou je vybraných n uzlov z prieniku množiny funkcií a terminálov, aby boli deťmi koreňa.

3. Pre každý symbol funkcie z n vybraných uzlov je grow metóda rekurzívne volaná, ich deti sú vyberané z prieniku množín funkcií a terminálov dokým hĺbka stromu nieje rovná $d-1$ v tom prípade sa listy vyberajú už len z množiny terminálov.

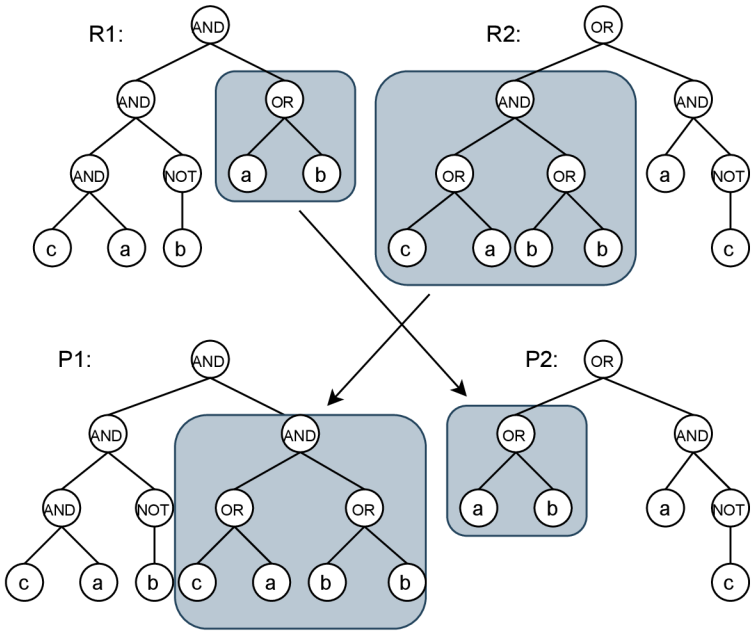
Pri tejto metóde treba podotknúť že tvar a veľkosť počiatočných stromov závisí na pomere funkcií a terminálov. Ak máme viacej funkcií, tak stromy môžu pravdepodobnejšie dosiahnuť svojej maximálnej hĺbky, ako keď bude funkcií menej a terminálov viac.

Full metóda na rozdiel od grow metódy vyberá uzli len z množiny funkcií, pokiaľ nieje dosiahnutá hĺbka $d-1$, následne vyberá už len z množiny terminálov. Výsledkom tejto metódy je, že všetky počiatočné stromy majú maximálnu hĺbku.

Ramped half-and-half metóda sa snaží o čo najväčšiu diverzitu medzi počiatočnými stromami. S touto metódou je časť $\frac{1}{d}$ populácie inicializovaná s maximálnou hĺbkou 1, časť $\frac{1}{d}$ z hĺbkou 2 atď. Pre každú skupinu, je polovica stromov inicializovaná pomocou grow metódy a polovica pomocou full metódy.

Operátor kríženia

Operátor kríženia vytvára variáciu v populácii produkovaním potomka, ktorý je zložený z genetického materiálu pochádzajúceho z rodiča. Rodičia R_1 a R_2 sú vybraný pomocou selekčnej metódy. Štandardné kríženie v genetickom programovaní začína výberom náhodného uzlu v každom rodičovi (bod kríženia). Väčšinou sa ako body kríženia vyberajú s väčšou pravdepodobnosťou vnútorné uzly, než terminály a listy stromu. Potomci sú vytvorený odobraním podstromu z rodiča R_1 (resp. R_2) a pridaním ho na bod kríženia do rodiča R_2 (resp. R_1).



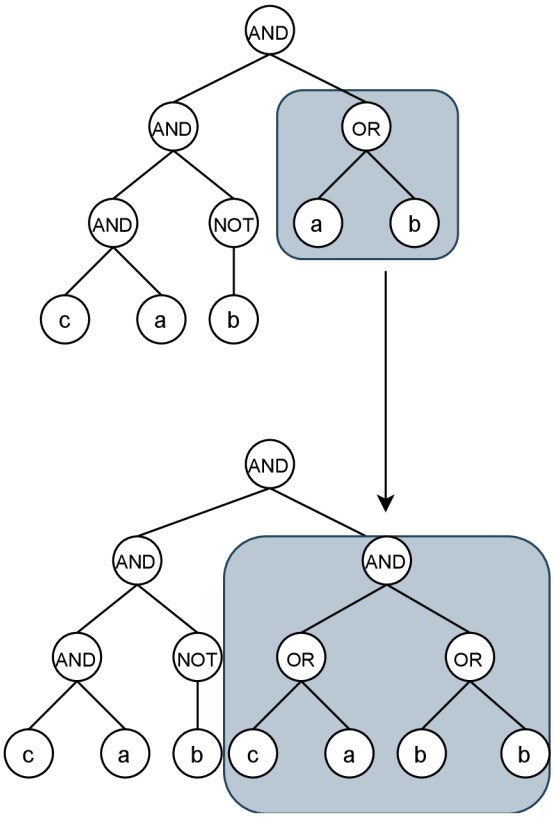
Obr. 4.2: Príklad kríženia stromov. Podstromy, ktoré sa nachádzajú na bodoch kríženia sú označené v sivo vyznačených miestach.

Pri tomto prístupe sa môže stať, že ak bude jeden bod kríženia koreň stromu a v druhom list, tak budú vznikať veľmi veľký potomkovia. Takýmto spôsobom môžu vznikať aj potomkovia, ktorý presahujú maximálnu hĺbku stromu. Takto vzniknutý potomok sa následne buď

zahodí, alebo označí veľmi malou fitness. Tento problém sa dá riešiť viacerými spôsobmi (napr. pridávaním väčšej pravdepodobnosti na kríženie do vnútorných uzlov a nižšej pri koreni a listoch stromu, alebo určiť bod kríženia v oboch rodičoch na rovnakom mieste).

Operátor mutácie

Štandardný operátor mutácie v stromovom genetickom programovaní začína výberom náhodného bodu mutácie vo vybranom jedincovi pomocou uniformnej pravdepodobnosti. Následne podstrom, ktorý sa nachádza na bode mutácie sa zmaže a na jeho miesto je náhodne vygenerovaný nový podstrom.



Obr. 4.3: Príklad štandardnej mutácii pri stromovom genetickom programovaní.

4.2 Kartézské genetické programovanie

Kartézské genetické programovanie je varianta genetického programovania, u ktorej sa kandidátne riešenia reprezentujú pomocou acyklických orientačných grafov.

Princíp kartézského genetického programovania

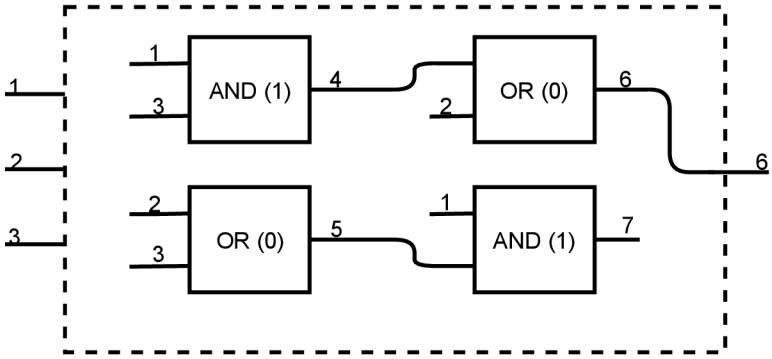
Algoritmus prehľadávania v kartézskom genetickom programovaní je veľmi podobný variante v evolučných stratégiách ES($1 + \lambda$). Populácia je zložená z $1 + \lambda$ jedincov. Nová populácia sa tvorí tak, že sa vyberie najlepší jedinec podľa fitness hodnoty a bude tvoriť novú populáciu spoločne so svojimi λ zmutovanými potomkami (λ je väčšinou určená na 4). Pri výbere najlepšieho jedinca je v kartézskom genetickom programovaní jedno dôležité

pravidlo, pri ktorom platí, že ak je jeden alebo viac jedincov, ktorý majú rovnakú fitness ako rodič, tak sa vyberie jeden z nich ako nový rodič. Týmto krokom sa zaistí väčšia genetická diverzita. Tento algoritmus môžeme popísať nasledujúcimi krokmi:

1. Vygeneruj $1 + \lambda$ náhodných jedincov pri inicializácii populácii.
2. Ohodnoť všetkých jedincov populácie pomocou fitness funkcie.
3. Nájdi najlepšie ohodnoteného jedinca (a kontrola duplicity s predchádzajúcou generáciou).
4. Vygeneruj λ potomkov z najlepšieho jedinca pomocou mutácie.
5. Najlepší jedinec spoločne s jeho λ potomkami tvoria novú populáciu.
6. Pokiaľ nieje splnená ukončovacia podmienka, pokračuje sa krokom 2.

Reprezentácia jedinca

V kartézskom genetickom programovaní sú jedinci reprezentovaný, ako acyklický orientované grafy. Tieto grafy sú reprezentované, ako dvoj-dimenzionálne polia programovateľných uzlov. Gény, ktoré tvoria genotyp sú v kartézskom genetickom programovaní reprezentované ako celočíselné hodnoty, ktoré reprezentujú vstupy odkiaľ si uzol berie dáta, funkciu ktorú daný uzol vykonáva a výstup kde je uložený výsledok. Keď je genotyp dekódovaný, niektoré uzly môžu byť ignorované. Táto skutočnosť je spôsobená tým, že niektoré uzly nie sú potrebné pre výpočet výstupnej hodnoty jedinca. Fenotyp je výsledok dekódovania genotypu. Genotyp má v kartézskom genetickom programovaní pevnú veľkosť, ale fenotyp môže mať veľkosť od 0 do maximálneho počtu uzlov definovaných v genotype. Počet primárnych vstupov n_i a primárnych výstupov n_o obvodu je pevne určený na začiatku evolúcie. Každý uzol má práve jednu funkciu (s n_n vstupmi), pochádzajúcu z množiny funkcii Γ. Nech počet týchto funkcii je $n_f = |\Gamma|$. Vstupy uzlu, ktorý sa nachádza na i-tom stĺpci môžu byť pripojené buď na primárne výstupy, alebo výstupy iných uzlov umiestnených až l stĺpcov pred ním. Pre $l = 1$ sa vstupy uzla v danom stĺpci môžu pripojiť len na primárne uzly alebo na predchádzajúci stĺpec. Nech n_c je počet stĺpcov a n_r je počet riadkov obvodu. Pre $l = n_c$ bude miera prepojenosti v danej kartézkej mriežke maximálna, každý uzol v danom stĺpci môže mať ako vstup ľubovolný predchádzajúci uzol. Primárny výstup môže byť pripojený na ľubovolný uzol alebo ľubovolný primárny vstup.



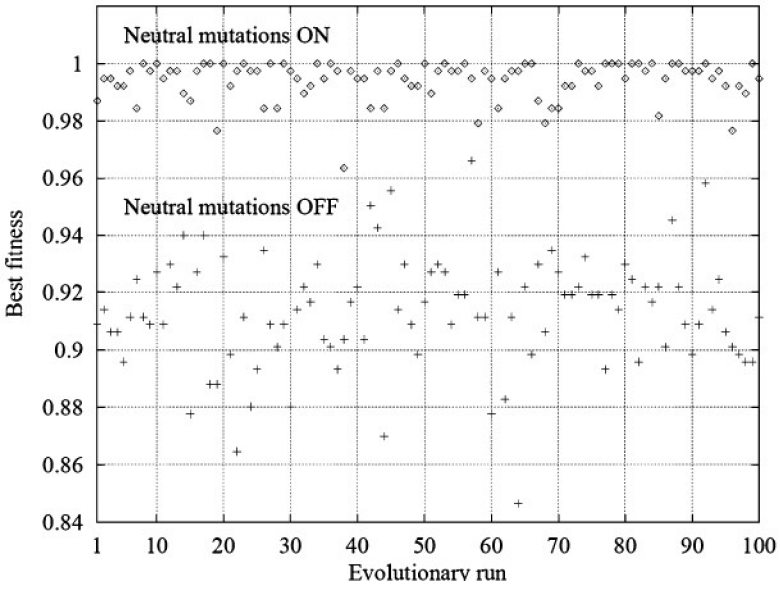
Obr. 4.4: Na tomto obrázku môžeme vidieť príklad kandidátneho riešenia v kartézskom genetickom programovaní s parametrami $n_r = 2$, $n_c = 2$, $n_i = 3$, $n_o = 1$, $n_n = 2$, $n_f = 2$, $l = 1$, $\Gamma = \{OR(0), AND(1)\}$, uzol 5 a 7 niesú vo fenotype využité. Chromozóm: (1,3,1), (2,3,0), (4,2,0), (1,5,1), (6). Posledné číslo určuje zapojenie výstupu.

Dekódovanie genotypu

Dekódovanie genotypu je rekurzívny algoritmus, pri ktorom sa začína od primárneho výstupu genotypu. Keďže genotyp obsahuje veľa neaktívny uzlov, tak by bolo veľmi neefektívne počítať všetky uzly. Preto sa najskôr pozerá, ktoré uzly sú aktívne. Uzly sa pokladajú za aktívne ak od nich existuje cesta ku primárnemu výstupu. Tento algoritmus sa dá vykonať viacerými spôsobmi. Jeden z nich by bol zistiť, ktoré uzly sú aktívne (rekurzívne) a zaznamenať ich pre budúce použitie.

Operátor mutácie

Operátor mutácie je zatiaľ jediný operátor, ktorý sa používa pri genetických operáciach v kartézskom genetickom programovaní. Jedná sa o jedno bodové mutácie, pri ktorých náhodná hodnota génu v chromozóme sa zmení na novú náhodne vygenerovanú správnu hodnotu. Ak je vybraný gén z funkciou, tak je vybraná náhodná správna hodnota z množiny funkcií Γ, resp. ak je vybraný vstupný gén, tak sa vyberá správna adresa výstupného génu z predchádzajúcich stĺpcov, alebo ľubovolná adresa primárneho vstupu. Počet génov, ktoré budú v chromozóme mutovať je definovaný na začiatku evolúcie užívateľom, väčšinou sa jedná o percento z celkového počtu génov v genotype (tento pojem sa nazýva miera mutácie). Mutáciou sa z aktívneho resp. neaktívneho uzlu môže stať neaktívny resp. aktívny uzol. Mutácia v kartézskom genetickom programovaní má veľký vplyv na tvar fenotypu. Mutáciu nazývame neutrálnou ak nemá vplyv na hodnotu fitness. Takáto situácia môže nastať ak sa zmení fenotyp, ale hodnota fitness je rovnaká, alebo pri mutácii len neaktívnych uzly. Ak mutácia nieje neutrálna, tak sa jedná o mutáciu adaptívnu. Ak je vykonaná séria neutrálnych mutácii, ktoré sú nasledované jednou adaptívnou mutáciou môže sa stať, že do teraz neaktívne uzly sa stanú aktívne a fenotyp zmení kompletne svoj tvar. Tento prístup neplní len svoj tradičný účel mierne meniť fenotyp, ale zároveň pridáva aj výrazný posun v prehľadávaní priestoru, čo nahradzuje operátor kríženia v kartézskom genetickom programovaní. Na nasledujúcom Obrázku 4.5 môžeme vidieť vývoj fitness pri používaní a nepoužívaní neutrálnych mutácii.



Obr. 4.5: Tento obrázok znázorňuje vývoj fitness pri zapnutých a vypnutých mutáciách. Je vidieť, že pri 100 nezávislých evolučných behoch po 10 miliónov generáciách sa vývoj fitness pri zapnutých neutrálnych mutáciách výrazne zlepšil. Prevzaté z [10].

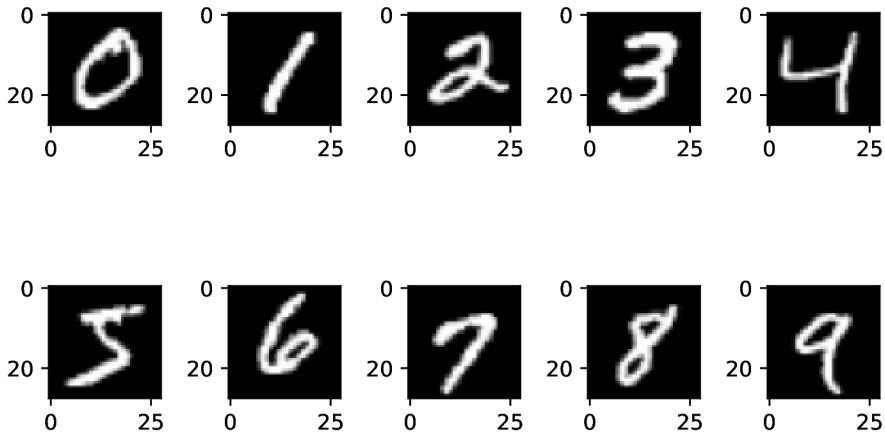
Kapitola 5

Riešené úlohy

Táto kapitola stručne popisuje úlohy, ktoré sú riešené v rámci tejto práce. V prvej sekcii je popísaná úloha MNIST a v druhej sekcii je popísaná úloha kociek.

5.1 MNIST

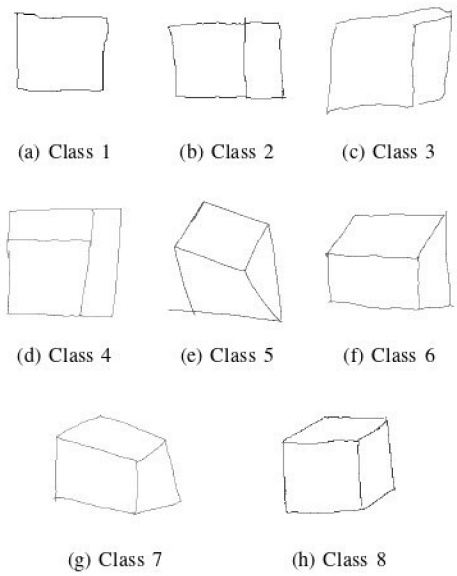
Pri tejto úlohe nám pôjde o nájdenie najlepšieho viac-triedneho klasifikátoru (ktorý je zložený z viac binárnych klasifikátorov) pre ručne písané číslice. Dátová sada MNIST je databáza ručne písaných číslic, ktorá je bežne používaná pre učenie a klasifikáciu v strojovom učení. Je zložená zo šedotónových obrázkov o rozmeroch 28×28 pixlov, ktoré patria do jednej z desiatich tried od 0–9. Databáza obsahuje celkom 70000 obrázkov z toho 60000 sú trénovacie dáta a 10000 testovacie dáta. Časť dátovej sady je vidieť na Obrázku 5.1.



Obr. 5.1: Príklad dátovej sady MNIST

5.2 Kocky

Pri tejto úlohe je cieľ nájsť najlepší viac-triedny klasifikátor pre ručne kreslené kocky. Kreslenie kociek je vlastne jednoduchý vizuo-priestorový test, ktorý sa môže používať pri zisťovaní miery demencie pri pacientoch s Parkinsonovou chorobou. Dátová sada kociek bola získaná od detí v rozmedzí 7 až 11 rokov [9]. Každé dieťa malo niekoľko pokusov na nakreslenie kocky na základe predlohy. Po získaní dát boli jednotlivé kresby kociek ručne klasifikované do tried od 1 po 8, kde trieda 1 je nakreslený štvorec a trieda 8 perfektne nakreslená kocka. Pre triedu 1 neboli zozbierané žiadne dáta, preto sa v tejto práci ani nepoužíva. Dokopy bolo zozbieraných 120 kresieb od 40 subjektov. Dátová sada bola rozdelená na trénovaciu a testovaciu sadu v pomere 1 : 2 v rámci jednotlivých tried ale aj ako celok. Aby sa zabránilo potenciálnej zaujatosti, tak jednotlivé kresby od rovnakých subjektov neboli rozdeľované medzi trénovaciu a testovaciu sadu. Príklad jednotlivých tried vidieť na Obrázku 5.2.



Obr. 5.2: Príklad jednotlivých tried kociek prevzaté z [9].

Kapitola 6

Návrh

Táto kapitola popisuje evolučný návrh klasifikátoru obrazov pomocou klasického kartézského genetického programovania (CGP), ktorý bude schopný pri poskytnutí trénovacej sady, sa naučiť rozpoznať jednotlivé vzory v obrázkoch a odlíšiť jednotlivé triedy medzi sebou pomocou fitness funkcie AUC a bude vedieť klasifikovať správne obrázky z poskytnutej testovacej sady.

6.1 Predspracovanie dát

Predspracovanie dát v tejto práci slúži prevažne na zlepšenie kvality výsledných kandidátnych riešení, keďže počet primárnych vstupov v kandidátnych riešeniach odpovedá počtu pixlov na daných obrázkoch. V tejto podkapitole sa popisuje predspracovanie dát pre jednotlivé riešené úlohy. Dátové sady MNIST a aj kocky boli rozdelené počas predspracovania na trénovaciu a testovaciu sadu samostatne a to takým spôsobom, že boli vytvorené dva priečinky pre testovanie a trénovanie. Následne do každého priečinka boli vytvorené priečinky s názvami pre každú triedu a do nich boli pridelené jednotlivé obrázky z danou triedou.

MNIST

Dátová sada MNIST ako bolo popísané v sekcii 5.1 sa skladá z obrázkov o veľkosti $28x28$ pixlov, táto dátová sada pre účel experimentov bola podvzorkovaná na veľkosť 14×14 pixlov.

Kocky

Dátová sada pre kocky, ktorá bola popísaná v sekcii 5.2 má obrázky o veľkosti 505x505 pixlov. Tieto obrázky bez predspracovania nie sú vhodné pre použitie v kartézskom genetickom programe. Preto ich treba predspracovať. Ako prvé bolo treba jednotlivé pixle obrázka previesť do odtieňa sivej. Následne bolo treba nájsť kocky a vystrihnúť ich z pôvodného obrázku. Vystrihnuté obrázky boli podvzorkované na tri rôzne velikosti: 8x8, 14x14 a 28x28. Na tieto jednotlivé veľkosti bol aplikovaný filter zvýraznenia hrán kociek a ako posledná úprava bola aplikovaná operácia invertovania všetkých farieb v obrázku na opačné.

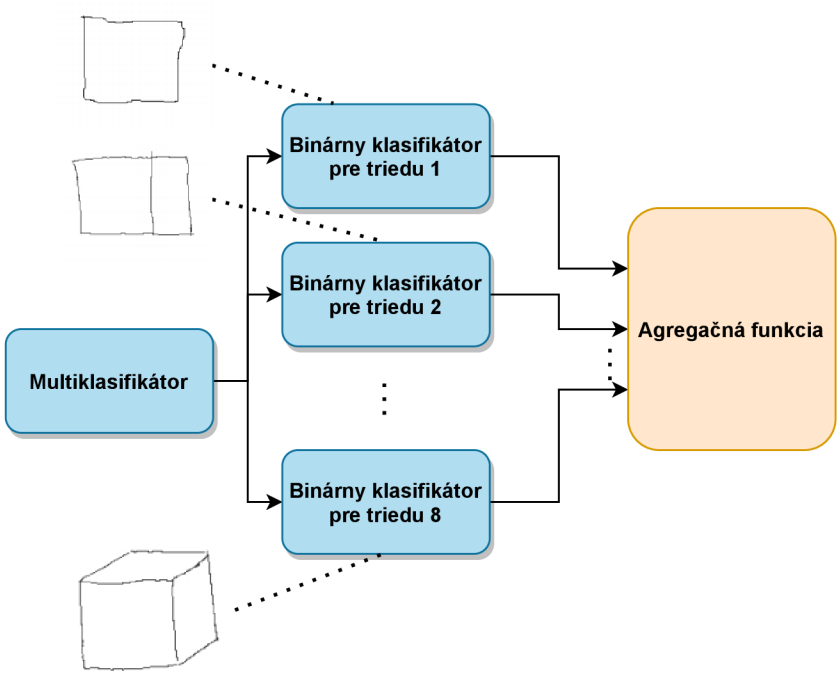
6.2 Model klasifikátoru

K evolučnému návrhu klasifikátoru obrazov bolo využité genetické programovanie, konkrétne kartézské genetické programovanie. Výsledný viac-triedny klasifikátor sa skladá z viac binárnych klasifikátorov. Vstupné dáta budú na vstup binárneho klasifikátora podávané ako vektory o veľkosti obrázkov dátovej sady. Výstupom jedného binárneho klasifikátoru je jedna hodnota. Hodnoty zo všetkých binárnych klasifikátorov sa dajú na vstup agregačnej funkcii, ktorá pomocou rozhodovacej hranice jednotlivých binárnych klasifikátorov namapuje výstupnú hodnotu na interval od 0 do 100 a následne sa trieda vstupného vzorku určí podľa najvyššej namapovanej hodnoty. Tento proces je bližšie popísaný v sekcii 6.4. Keďže jednotlivé pixle v obrázkoch majú 8 bitové hodnoty tak dátový typ, ktorý bol použitý pri výpočtoch bol `unit8_t`. Následne boli pri výpočte použité 8 bitové funkcie, ktoré sú popísané v Tabuľke 6.1.

Index	Funkcia	Popis	Index	Funkcia	Popis
0	255	konštanta	6	$max(x, y)$	maximum
1	x	identita	7	$x + y$	súčet
2	$255 - x$	inverzia	8	$x +^s y$	súčet so saturáciou
3	$x >> 1$	delenie 2	9	$(x + y) >> 1$	priemer
4	$x >> 2$	delenie 4	10	$(x > 127)?x : y$	podmienené priradenie
5	$min(x, y)$	minimum	11	$\|x - y\|$	absolútna hodnota

Tabuľka 6.1: Funkcie použité pri výpočtoch prevzaté z [4].

Funkcie s indexom 8, 9 a 11 z Tabuľky 6.1 sú počítané z dátovým typom `int`.



Obr. 6.1: Na tomto obrázku je vidieť model viac-triedneho klasifikátoru pre kocky

6.3 Trénovanie pomocou CGP

Trénovanie pomocou CGP, bude prebiehať pomocou evolučnej stratégie 1 + 4, pri ktorej sa v každej generácii vyberú jednotlivé najlepšie kandidátne riešenia pre jednotlivé triedy, ktoré postupujú s ich zmutovanými variantami do nasledujúcej generácie ako nová populácia.

Fitness funkcia

Po prvej náhodnej inicializácii populácie sa všetky chromozómy v populácii ohodnotia pomocou fitness funkcie, ktorá je daná hodnotou AUC. Pre výpočet hodnoty fitness chromozómu si potrebujeme zaznamenať výsledky pre jednotlivé vzorky dát, ako vektor dvojíc: výslednej hodnoty chromozómu pre danú vzorku a informáciu či je, alebo nie je trieda vzorky zhodná s triedou chromozómu. Následne sa tento vektor dvojíc zoradí vzostupne a nasledovným prechádzaním po ose x pri hodnote P a po ose y pri hodnote N dôjde k vykresleniu krivky ROC. Plocha pod krivkou ROC (AUC) je vypočítaná pomocou lichobežníkovej metódy, ktorá počíta AUC ako sumu súčtov dvoch funkčných hodnôt, ktoré sa podelia dvomi a vynásobia krokom, ktorý je daný podielom 1 a počtu P vo vektore dvojíc.

Mutácia

Po vybraní najlepších chromozómov z populácie pre jednotlivé triedy, nastáva proces mutácie, pri ktorom sa najskôr vyberie počet génov, ktoré sa budú mutovať. Tento počet sa vyberá pomocou miery mutácie a to jej vynásobeným s celkovým počtom génov v chromozóme. Následne sa náhodne vyberajú gény pre mutáciu. Ak sa vyberie na mutáciu vstupný gén, tak sa proporcionálne vyberie buď ľubovolný primárny vstup alebo výstup iného ľubovolného uzlu z predchádzajúcich stĺpcov podľa l-back hodnoty. Ak sa vyberie funkčný gén tak sa náhodne vyberie ľubovolná funkcia z množiny funkcií a ak sa vyberie primárny výstup chromozómu tak sa náhodne vyberie výstup ľubovoľného uzlu z chromozómu.

Voľba rozhodovacieho prahu

Pre každý binárny klasifikátor bude vybraná vlastná rozhodovacia hranica a to tak, že sa po skončení behu vyberú najlepšie binárne klasifikátory pre dané triedy a pomocou trénovacích dát sa nad jednotlivými binárnymi klasifikátormi vykoná ich klasifikácia a výpočet fitness. Pričom následne z daných výsledkov bude nájdený prah na krivke ROC za pomoci maximalizácie TRP a minimalizácie FPR.

6.4 Testovanie navrhnutých kandidátnych riešení

Po dokončení procesu učenia máme navrhnutých viacero kandidátnych riešení, ktoré reprezentujú binárne klasifikátory, z ktorých si musíme vybrať tie najlepšie a následne z nich zložiť viac-triedny klasifikátor, pri ktorom sa jednotlivé dáta budú klasifikovať pomocou agregačnej funkcie.

Výber najlepších kandidátnych riešení pre výsledný klasifikátor

Výber najlepších kandidátnych riešení spočíva v načítaní všetkých kandidátnych riešení (zo súborov) pre jednotlivé triedy, zo všetkých behov programu pre dané nastavenie. Následne každé kandidátne riešenie vykoná klasifikáciu a výpočet fitness na testovacích dátach

a vrámci jednotlivých tried sa porovnajú všetky kandidátne riešenia a vyberú sa najlepšie kandidátne riešenia pre danú triedu.

Agregačná funkcia

Pri viac-triednom klasifikátore zloženom z binárnych klasifikátorov, potrebujeme mať funkciu pomocou ktorej dokážeme klasifikovať, do ktorej triedy daná testovacia vzorka patrí. Na to nám poslúži agregačná funkcia, ktorá si ako vstup berie od každého klasifikátora jeho rozhodovaciu hranicu h (h patrí do intervalu $\langle 0; 255 \rangle$) , ktorú použije na namapovanie na interval reálnych čísel od 0 do 100. Rozhodovacia hranica h sa namapuje na polovicu tohoto intervalu takým spôsobom, že dĺžka intervalu od 0 do 50 je podelená rozhodovacou hranicou $h + 1$. Následne dĺžka intervalu od 50 do 100 je podelená hodnotou $256 - (h + 1)$. Pri vyhodnocovaní jednotlivých testovacích vzorkou nám binárne klasifikátory vrátia výsledok r, ktorý predáme agregačnej funkcii a ona nám povie z akou percentuálnou pravdepodobnosťou sa jedná o danú triedu, ktorú klasifikátor reprezentuje a to nasledovným spôsobom:

$$f(h, r) = \begin{cases} (50/(h+1)) * (r+1), & (r+1) \leq (h+1) \\ 50 + (50/(256-(h+1))) * ((r+1)-(h+1)), & (r+1) > (h+1) \end{cases} \tag{6.1}$$

Kapitola 7

Implementácia

Táto kapitola stručne popisuje implementáciu programu podľa predchádzajúceho návrhu. V kapitole sú popísané použité nástroje a technológie, štruktúra programu, paralelizácia a pomocné skripty.

7.1 Použité nástroje a technológie

Pri implementácii navrhnutého programu bolo treba použiť rôzne nástroje a technológie. Program bol implementovaný pomocou jazyka C++ konkrétne verziou C++20. Pre preklad programu sa použil CMake[1]. Pre verzovanie programu bol použitý nástroj Git vo spojení s webovou službou GitHub. Operačný systém, na ktorom prebiehala implementácia, spúšťanie experimentov a následne aj spracovanie výsledkov bol Ubuntu 20.04. Na implementovanie jednotlivých skriptov pre spúšťanie experimentov a ich vyhodnocovanie bol použitý jazyk Python 3.9. Na implementáciu konvolučnej neurónovej sieti pre MNIST aj kocky bola použitá knižnica TensorFlow[2] a na implementáciu metódy podporných vektorov bola použitá knižnica scikit-learn[3]. Pre paralelizáciu jednotlivých častí programu bola použitá knižnica OpenMP[4] a pre načítanie obrázkov bola použitá knižnica OpenCV[5].

7.2 Štruktúra programu

Výsledný implementovaný program sa skladá z nasledujúcich zdrojových súborov:

- `main` – hlavný zdrojový súbor, ktorý zaisťuje spracovanie argumentov, učenie, testovanie a výpis štatistík.
- `Classifier` – Trieda, ktorá reprezentuje klasifikátor.
- `Chromosom` – Trieda, ktorá reprezentuje chromozóm kandidátneho riešenia.
- `TrainCGP` – Trieda, ktorá zabezpečuje načítanie dát a inicializáciu populácie.
- `TestEvaluator` – Trieda, ktorá testuje navrhnuté kandidátne riešenia.
- `argument_parser` – zdrojový súbor, ktorý spracúva argumenty programu.

[1] https://cmake.org/
[2] https://www.tensorflow.org/
[3] https://scikit-learn.org
[4] https://www.openmp.org/
[5] https://opencv.org/

Spracovanie argumentov

Zdrojový súbor `main` má na starosti spracovanie a kontrolu argumentov príkazovej riadky zavolaním dvoch funkcii (`parse_arguments a check_arguments` zo zdrojového súboru `argument_parser`. Program má dva módy: mód trénovania a mód testovania. Jednotlivé módy sa dajú nastaviť pomocou argumentov programu (`test_dataset` a `train_dataset`) nasledovne:

- V argumentoch sa vyskytuje len `--train_dataset` – vykoná sa len učenie jednotlivých binárnych klasifikátorov.
- V argumentoch sa vyskytuje len `--test_dataset` – vykoná sa testovanie uložených binárných klasifikátorov z jednotlivých behov programu, vyberú sa zvlásť pre každú triedu najlepšie binárne klasifikátori pomocou hodnoty fitness a následne sa otestujú ako celok pomocou agregačnej funkcie.
- V argumentoch sa nachádzajú oba argumenty – vykoná sa učenie aj testovanie.

Trénovacie a testovacie dáta sú poskytnuté ako cesty k priečinkom. Kde každý z priečinkov obsahuje priečinky jednotlivých tried, ktoré obsahujú dáta ku daným triedam.

Prípadne voliteľné argumenty programu, sú ešte parametre učenia CGP a paralelné spracovanie programu.

Inicializácia populácie

Nech n, je veľkosť počiatočnej populácie a nech m, je počet tried v dátach. Inicializácia počiatočnej populácie začína vytváraním n inštancií triedy `Classifier` a následne, každý `Classifier` si vytvorí m inštancií triedy `Chromosom`. Týmto sa zaisťuje trénovanie, pre všetky triedy v dátach naraz.

Výber najlepších kandidátnych riešení z populácie

Výber najlepších kandidátnych riešení je implementovaný výberom najlepších inštancií tried `Chromosom` (podľa fitness) naprieč populáciou (inštancií tried `Classifier`) podľa jednotlivých typov chromozómov (typ – jedna trieda z dátovej sady).

Uloženie kandidátnych riešení

Ukladanie kandidátnych riešení prebieha v každom behu programu do súborov s príponou `.chr`. Jednotlivé kandidátne riešenia sa ukladajú vo formáte: {výška obrázka, šírka obrázka, CGP stĺpce, CGP riadky, L-back, Rozhodovacia hranica, Trieda kandidátneho riešenia} nasleduje genotyp v tvare ([index1]vstup1, vstup2, funkcia)([index2], vstup1, vstup2, funkcia) až dokým sa nedostane po primárny výstup genotypu, ktorý je následne zapísaný ako (výstup ľubovolného uzlu).

7.3 Paralelizácia

Pri programe, niektoré časti kódu sa vykonávajú častejšie ako ostatné. Tieto časti kódu sa oplatí paralelizovať. V tomto implementovanom riešení sa najčastejšie vykonáva vyhodno-

covanie jednotlivých jedincov populácie. Preto vyhodnocovanie jedincov bolo paralelizované pomocou knižnice OpenMP[6].

7.4 Pomocné skripty

Mimo implementácie hlavného programu boli implementované pomocné skripty. Skript `experiments.py` spustí všetky experimenty. Skript `graphs.py`, vytvára z vykonaných experimentov všetky grafy a tabuľky pre experimenty.

7.5 Skripty pre porovnanie s inými metódami

Pre porovnanie s inými metódami boli implementované dva skripty: `cnn.py` a `svm.py`. Prvý skript spustí proces učenia a klasifikácie konvolučnej neurónovej siete na dátovej sade MNIST a kociek, druhý spustí proces učenia a klasifikácie pre metódu podporných vektorov. Pri oboch skriptoch sa nastavuje cesta k dátovej sade, na ktorej sa bude model trénovať a testovať.

[6] https://www.openmp.org/

Kapitola 8

Overenie funkčnosti a experimenty

Po dokončení implementácie navrhnutého programu sa vykonali experimenty. Táto kapitola začína overením funkčnosti implementácie programu na dátovej sade MNIST. Dátová sada MNIST bola rozdelená na dve dátové sady a to trénovaciu a testovaciu. Popis dátovej sady je popísaný v Tabuľke 8.1.

Trieda	Testovacie dáta	Trénovacie dáta
0	980	5923
1	1135	6742
2	1032	5958
3	1010	6131
4	982	5842
5	892	5421
6	958	5918
7	1028	6265
8	974	5851
9	1009	5949

Tabuľka 8.1: Počet dát pre jednotlivé triedy v rámci trénovacej a testovacej dátovej sady MNISTu.

V ďalšej časti tejto kapitoly je riešený problém klasifikácie kociek, ktoré sú popísané v sekcii 5.2. Tento problém bol riešený v nasledujúcej štúdii Smith [9]. Na rozdiel od práce [9] bol tento problém riešení novým spôsobom, ktorý ešte do teraz nebol vyskúšaný. V zmienenej štúdii z dostupnej dátovej sady získali obrysy kociek, ktoré použili pri procese učenia a klasiikácie, pričom využívali variantu CGP: IRCGP. V tejto práci boli kocky použité, ako celé obrázky, ktorých predspracovanie je popísané v sekcii 6.1 a táto práca využíva štandardnú verziu CGP. Ako prvé je vykonané porovnanie rýchlosti učenia a schopnosť klasifikácie na jednotlivých veľkostiach obrázkov. Následne sú porovnané jednotlivé nastavenia kartézskej mriežky. Ďalej je nájdené najlepšie kandidátne riešenie pri vhodnom nastavení CGP a výsledky sú porovnané z inými prístupmi a existujúcim riešením. Dátová sada bola tak isto ako aj MNIST rozdelená na dve dátové sady. Pri rozdeľovaný bola snaha rozdeliť dáta v pomere $1 : 2$, jak v rámci jednotlivých tried tak aj ako celok. Viac obrázkov, ktoré boli poskytnuté rovnakými subjektami neboli rozdelené medzi trénovaciu a testovaciu sadu,

aby sa zabránilo potenciálnej zaujatosti. Popis dátovej sady kociek je popísaný v Tabuľke 8.2.

Trieda	Testovacie dáta	Trénovacie dáta
2	2	4
3	2	3
4	8	11
5	3	6
6	14	25
7	9	19
8	4	10

Tabuľka 8.2: Počet dát pre jednotlivé triedy v rámci trénovacej a testovacej dátovej sady kociek.

Dátová sada	Experimenty
MNIST	1, 6
Kocky	2, 3, 4, 5, 6

Tabuľka 8.3: V tejto tabuľke sú znázornené jednotlivé dátové sady a ich použitie pri jednotlivých experimentoch.

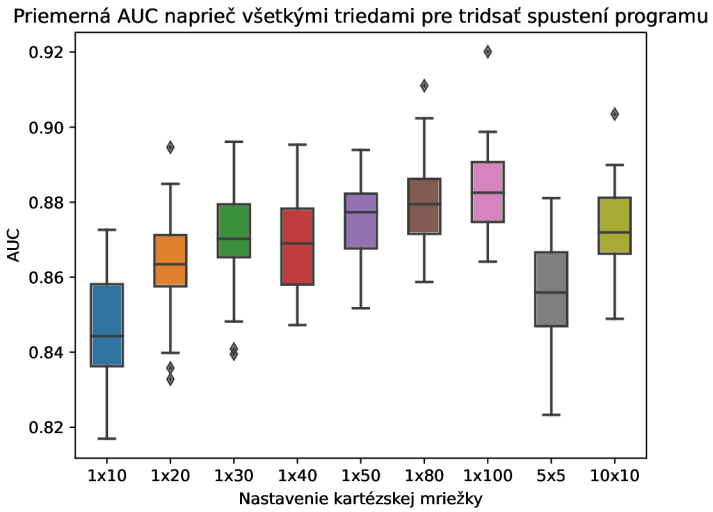
8.1 Experiment 1: Overenie funkčnosti implementácie

Pre overenie funkčnosti implementácie bola použitá dátová sada MNIST. Parametre kartézského genetického programovania boli nastavené na hodnoty popísané v Tabuľke 8.4. Jediný meniaci sa parameter v zvolených nastaveniach je veľkosť kartézskej mriežky. Každé nastavenie bolo spustené tridsať krát. Pričom, každý beh programu navrhol 10 binárnych klasifikátorov, ktoré reprezentujú jednotlivé triedy. Takže pre jedno nastavenie bolo navrhnutých 300 binárnych klasifikátorov a tým pádom pre samostatne každú triedu bolo navrhnutých 30 binárnych klasifikátorov. Výsledky experimentov sú znázornené na Obrázku 8.1

Parametre	Zvolené hodnoty
Tvar kartézskej mriežky	1×10, 1×20, 1×30, 1×40, 1×50, 1×80, 1×100, 5×5, 10×10
L-back	Vždy plné prepojenie
Veľkosť populácie	5
Počet generácií	2000
Miera mutácie	0.05
Veľkosť obrázkov dátovej sady	14×14
Dátový typ použitý pri výpočte	uint8_t

Tabuľka 8.4: Popis parametrov pre nastavenie kartézskeho genetického programovania.

Z Obrázka 8.1 na základe priemernej fitness hodnoty AUC je vidieť, že pri lineárnych a štvorcových nastaveniach platí, čím väčšiu kartézskú mriežku použijeme, tým bude väčšia kvalita fitness. Z nasledujúcej Tabuľky 8.5 môžeme vidieť minimum, priemer a maximum pre všetky nastavenia kartézskej mriežky.



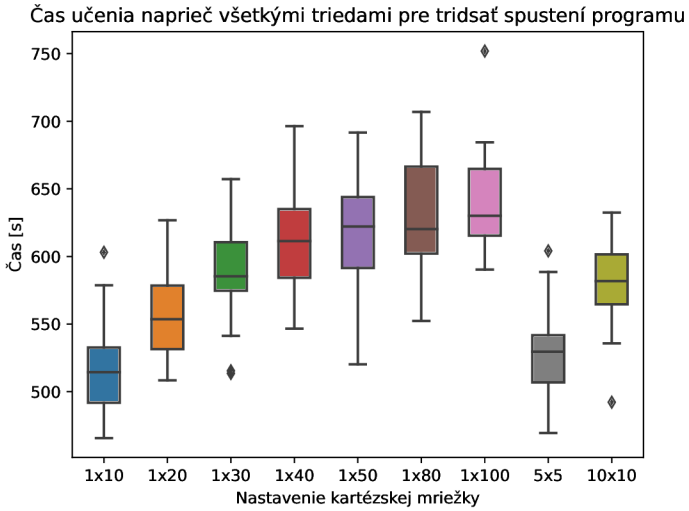
Obr. 8.1: Priemerná fitness hodnota naprieč všetkými klasifikátormy pre jednotlivé nastavenia kartézskej mriežky.

Z Tabuľky 8.5 môžeme vyvodiť, že najlepšie nastavenie kartézskej mriežky je 1×100 s priemernou hodnotou fitness 0.88334. Následne pomocou Mann-Whitney U testu pri hladine pravdepodobnosti 0.05 bolo určené či má najlepšia mriežka podobnosť s niektorým iným nastavením kartézskej mriežky, kde hypotéza $H0$ určuje, že dve distribúcie dávajú podobne kvalitné riešenia a hypotéza $H1$ určuje, že distribúcie nedávajú podobne kvalitné riešenia. Z jednotlivých testov vyšlo, že mriežka 1×100 je podobná z mriežkou 1×80. Z nasledujúceho Obrázku 8.2 je vidieť CPU čas učenia pre jednotlivé kartézské mriežky za daný počet behu programu.

Kartézká mriežka	Hodnota fitness Minimum	Priemer	Maximum
10x10	0.848895	0.873696	0.903477
1x10	0.816923	0.845585	0.872602
1x100	0.864132	0.883341	0.920116
1x20	0.832768	0.863435	0.894612
1x30	0.839483	0.869585	0.896110
1x40	0.847211	0.869759	0.895323
1x50	0.851726	0.875118	0.893908
1x80	0.858708	0.879867	0.911033
5x5	0.823275	0.855900	0.881084

Tabuľka 8.5: Táto tabuľka znázorňuje minimálnu, priemernú a maximálnu hodnotu fitness pre jednotlivé kartézské mriežky.

Na Obrázku 8.2 je vidieť, že CPU čas sa pri zväčšovaní lineárnych aj štvorcových nastavení kartézskej mriežky zvyšuje. V Tabuľke 8.6, sú znázornené jednotlivé minimá, priemery a maximá pre jednotlivé nastavenia kartézskej mriežky.



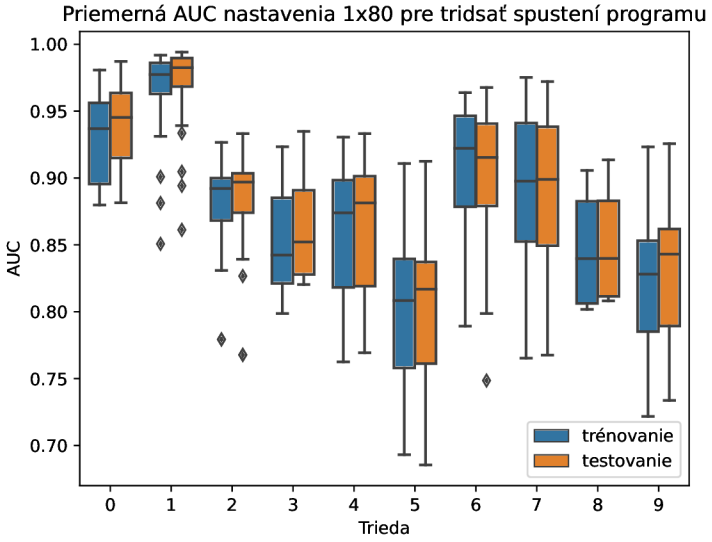
Obr. 8.2: Na tomto obrázku je znázornený CPU čas pre jednotlivé kartézské mriežky za daný počet behov programu.

Z predchádzajúcich výsledkov štatistického testu sa dá určiť, že nastavenia kartézskej mriežky 1×100 a 1×80 dosahujú podobne kvalitné výsledky. Následne z Tabuľky 8.6 sa dalo určiť, že nastevenie kartézkej mriežky 1×100 má priemerný CPU čas učenia 637.4 a mriežka 1×80 má priemerný CPU čas učenia 628.9, čo znamená, že druhé nastavenie je rýchlejšie ako prvé a vzhľadom na to, že majú podobne kvalitné rozloženie bolo toto nastavenie vybrané ako najlepšie a jeho výsledky sú znázornené na Obrázku 8.3 a Tabuľke 8.7.

Kartézká mriežka	CPU čas [s] Minimum	Priemer	Maximum
10x10	492.214324	579.610539	632.501542
1x10	465.673616	516.704741	603.099129
1x100	590.273716	637.408420	751.906100
1x20	508.438202	555.026404	626.809989
1x30	513.508037	589.791083	657.184784
1x40	546.602837	612.554311	696.353259
1x50	520.178895	619.981662	691.597230
1x80	552.293808	628.947006	706.895933
5x5	469.398161	529.387240	604.143888

Tabuľka 8.6: V tejto tabuľke sú zobrazené minimá, priemery, maximálne hodnoty CPU času v sekundách naprieč všetkými triedami pre jednotlivé nastavenia kartézskych mriežok za daný počet behov programu.

Najlepšie kandidátne riešenie pre túto úlohu je zložené z binárnych klasifikátorov, ktoré majú z tridsať behov programu maximálnu hodnotu fitness. Z Obrázku 8.3 a tabuľky 8.7 je vidieť, že najlepšia AUC fitness z tridsať behov programu na testovacích dátach, je pre jednotlivé triedy od 0 po 9 nasledovná: 0.987, 0.994, 0.933, 0.935, 0.933, 0.912, 0.968, 0.972, 0.914, 0.926 čo značí priemernú hodnotu fitness AUC $0,947$.



Obr. 8.3: Obrázok znázorňuje nastavenie kartézskej mriežky 1×80 s L_back 80 a priemernú hodnotu fitness AUC pre jednotlivé triedy naprieč trénovacími a testovacími dátami za daný počet behov programu.

Dátová sada	Hodnota fitness Testovacie dáta			Trénovacie dáta		
	Minimum	Priemer	Maximum	Minimum	Priemer	Maximum
Trieda						
0	0.881481	0.937568	0.987168	0.879770	0.930411	0.980756
1	0.861234	0.969221	0.994092	0.850737	0.964350	0.991777
2	0.767754	0.883412	0.933246	0.779312	0.880634	0.926618
3	0.820297	0.861722	0.934826	0.798721	0.851389	0.923374
4	0.769372	0.869021	0.933199	0.762495	0.864871	0.930551
5	0.685411	0.803233	0.912493	0.693140	0.800174	0.910792
6	0.748574	0.901482	0.967622	0.789248	0.906482	0.963773
7	0.767612	0.893189	0.972035	0.765321	0.895245	0.975247
8	0.808085	0.849328	0.913550	0.801830	0.846021	0.905671
9	0.733695	0.830492	0.925676	0.721725	0.821071	0.923218

Tabuľka 8.7: V tabuľke sú znázornené jednotlivé minimá, priemery, maximá, pri trénovaní a testovaní, pre jednotlivé triedy navrhnutých binárnych klasifikátorov.

Pre overenie funkčnosti implementácie bol najlepší viac-triedny klasifikátor porovnaný s prácou pani Jašíčkovej [4]. Treba podotknúť, že pani Jašíčková vo svojej práci používala inú fitness funkciu, ako je použitá v tejto práci. Konkrétne sa v práci pani Jašíčkovej používala fitness funkcia strednej odchýlky medzi očakávanými a získanými výstupmi pre jednotlivé vstupné dáta. Treba taktiež podotknúť, že pani Jašíčková použila väčší počet generácií pre získanie najlepšieho kandidátneho riešenia. Pri porovnaní matici zámen, ktorú má pani Jašíčková vo svojej práci a matici zámen najlepšieho viac-triedneho klasifikátoru z tejto práci, je vidieť podobnosť medzi výslednými riešeniami a taktiež je vidieť, že viac-triedny klasifikátor v tejto práci má schopnosť správne klasifikovať testovacie dáta pre dátovú sadu MNIST. Týmto je overené, že implementovaný návrh programu funguje správne.

Skutočná trieda	0	1	2	3	4	5	6	7	8	9
Predikovaná trieda										
0	749	0	23	4	2	14	22	1	0	3
1	1	1063	28	12	1	21	18	18	14	4
2	56	21	778	70	6	16	26	11	49	4
3	21	13	43	793	14	316	18	1	210	30
4	13	6	29	38	743	72	44	52	87	239
5	60	4	7	8	6	368	31	8	62	4
6	34	1	25	4	26	6	729	5	3	4
7	16	0	10	13	19	19	16	879	8	107
8	14	24	27	36	18	26	41	12	466	20
9	6	0	13	9	137	27	4	15	60	583
n/a	10	3	49	23	10	7	9	26	15	11

Tabuľka 8.8: Matica zámen pre viac-triedny klasifikátor.

	Číslice									
Třída	0	1	2	3	4	5	6	7	8	9
0	912	0	3	1	0	6	6	2	4	2
1	0	1075	2	0	1	0	3	4	4	5
2	1	2	824	19	1	1	2	12	5	0
3	2	1	5	729	1	5	0	5	8	0
4	2	0	1	0	746	0	12	3	3	6
5	3	1	0	4	1	661	5	0	7	2
6	9	0	2	2	8	12	788	0	8	1
7	2	1	10	7	2	2	0	820	2	6
8	0	2	3	10	3	1	2	4	666	4
9	1	0	2	2	48	8	1	10	5	801
n/a	48	53	180	236	171	196	139	168	262	182

Obr. 8.4: Matica zámen najlepšieho viac-triedny klasifikátor z mriežkou 1×100 a L_back 100 prevzaté z [4].

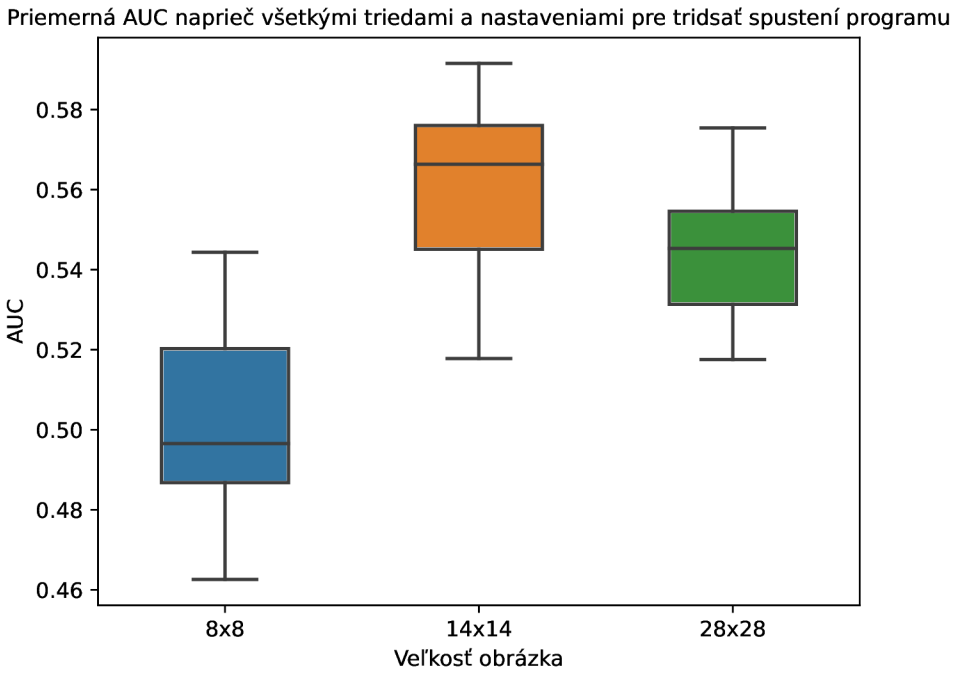
8.2 Experiment 2: Hľadanie najlepšej veľkosti obrázkov

Od tohoto experimentu sa ďalej používa dátová sada pre kocky. Dáta kociek boli predspracované do 3 rôznych dátových sád o rôznych veľkostiach obrázkov (8×8, 14×14, 28×28). Cieľom tohoto experimentu je zistiť vplyv veľkosti obrázka na CPU čas učenia a kvality výsledných riešení. Počiatočné parametre, kartézkého genetického programovania boli nastavené na hodnoty popísané v Tabuľke 8.9. Všetky výsledky experimentov boli získané z tridsať behov programu.

Parametre	Zvolené hodnoty
Tvar kartézskej mriežky	1×10, 1×20, 1×30, 1×40, 1×50, 1×80, 1×100, 5×5, 10×10, 25×25
L-back	Vždy plné prepojenie
Veľkosť populácie	5
Počet generácií	500
Miera mutácie	0.05
Veľkosť obrázkov dátovej sady	8×8, 14×14 , 28×28
Dátový typ použitý pri výpočte	uint8_t

Tabuľka 8.9: Popis počiatočných parametrov pre kartézké genetické programovanie.

Z Obrázku 8.5 sa dalo zistiť, že veľkosť obrázkov 14×14 je najvhodnejšia vzhľadom na to, že dáva najlepšie výsledky. Táto skutočnosť sa dá tiež overiť aj pomocou Tabuľky 8.10. Aj keď je zrejmé, že jednotlivé distribúcie priemernej AUC pre jednotlivé obrázky nemajú medzi sebou žiadnu podobnosť tak sa táto hypotéza overila pomocou Mann-Whitney U testu pri hladine pravdepodobnosti 0.05 a to tak, že sa vybrala distribúcia veľkosti obrázkov 14×14 a porovnala sa z ostatnými veľkosťami obrázkov samostatne, kde bola hypotéza $H0$, ktorá hovorí o podobnosti dvoch distribúcii zamietnutá a bola prijatá hypotéza $H1$, ako pravdivá.

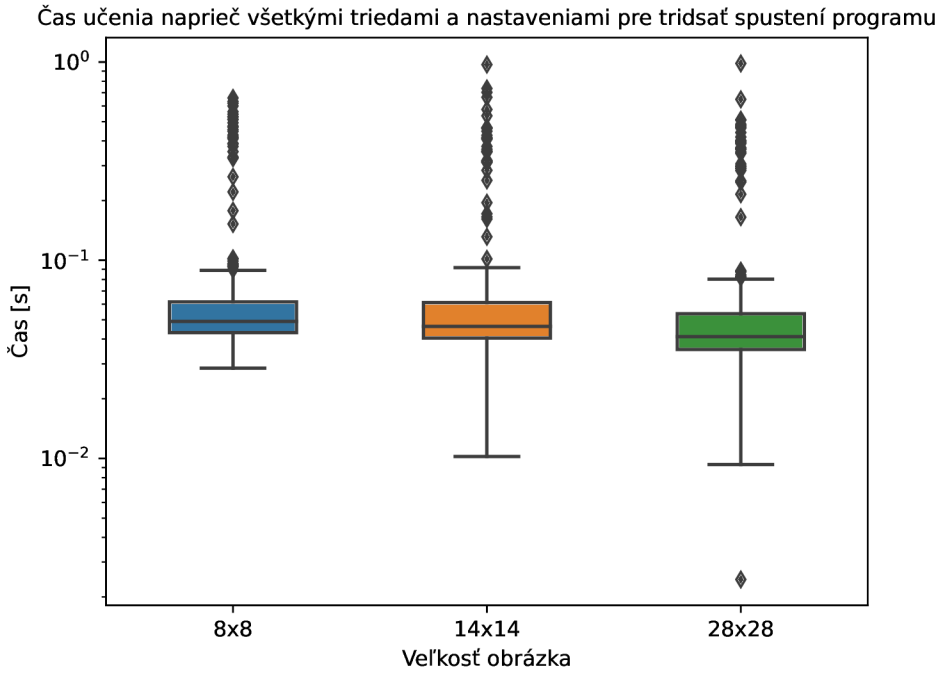


Obr. 8.5: Kvalita jednotlivých veľkostí obrázkov naprieč všetkými nastaveniami a triedami za daný počet behov programu.

	Hodnota fitness		
	Minimum	Priemer	Maximum
Veľkosť obrázkov			
14x14	0.517792	0.561985	0.591535
28x28	0.517566	0.543983	0.575411
8x8	0.462587	0.501962	0.544334

Tabuľka 8.10: Minimum, priemer a maximum hodnoty fitness pre jednotlivé veľkosti obrázkov.

Rozdiely pri čase učenia pre jednotlivé veľkosti obrázkov boli zanedbateľné. Prekvapivo najväčšia veľkosť obrázkov 28×28 trvala v priemere najkratšiu dobu a najmenšia veľkosť obrázkov 8×8 trvala v priemere zase najdlhšiu dobu. Túto skutočnosť si môžeme overiť na Obrázku 8.6 a Tabuľke 8.11



Obr. 8.6: CPU čas v sekundách jednotlivých veľkostí obrázkov naprieč všetkými nastaveniami a triedami za daný počet behov programu.

	CPU čas [s]		
	Minimum	Priemer	Maximum
Veľkosť obrázkov			
14x14	0.010220	0.092030	0.970750
28x28	0.002450	0.082903	0.984001
8x8	0.028532	0.093649	0.659060

Tabuľka 8.11: Minimum, priemer a maximum hodnoty CPU času pre jednotlivé veľkosti obrázkov.

Z následných grafov bolo vyhodnotené, že najvhodnejšia veľkosť obrázkov je 14×14 s priemernou fitness AUC naprieč všetkými triedami a nastaveniami za daný počet behov 0.561985 a z priemerným CPU časom v sekundách 0.09203.

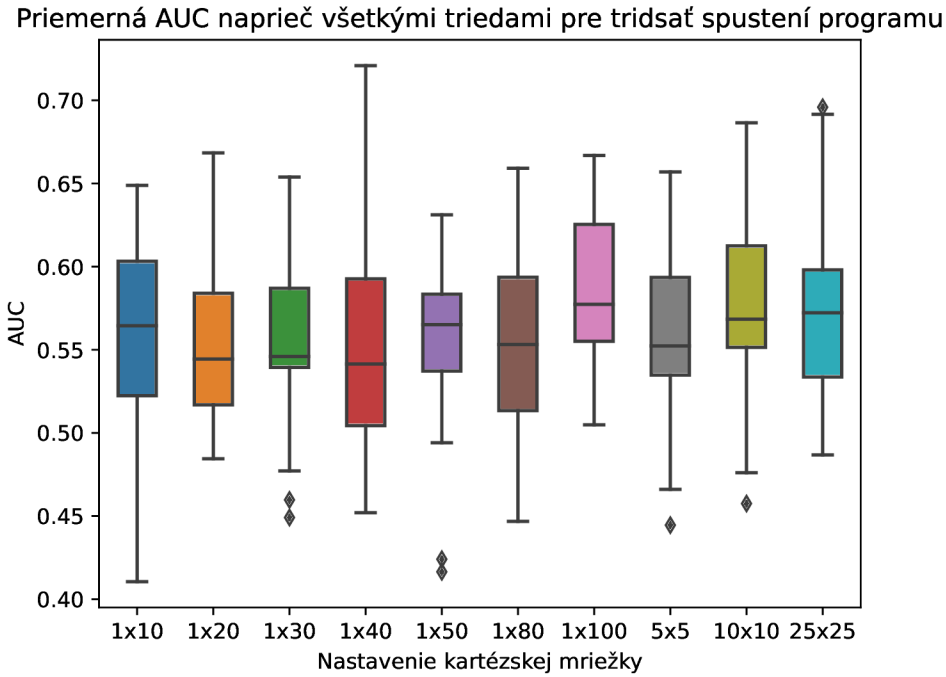
8.3 Experiment 3: Hľadanie najlepšej kartézskej mriežky

Cieľom tohoto experimentu je nájsť najlepšiu kartézsku mriežku pre hľadanie najlepšieho kandidátneho riešenia. Pri tomto experimente sú použité získané dáta zo sekcie 8.2. Parametre pre nastavenie kartézskeho genetického programovania sú popísané v Tabuľke 8.12.

Parametre	Zvolené hodnoty
Tvar kartézskej mriežky	1×10, 1×20, 1×30, 1×40, 1×50, 1×80, 1×100, 5×5, 10×10, 25×25
L-back	Vždy plné prepojenie
Veľkosť populácie	5
Počet generácií	500
Miera mutácie	0.05
Veľkosť obrázkov dátovej sady	14×14
Dátový typ použitý pri výpočte	uint8_t

Tabuľka 8.12: Popis počiatočných parametrov pre kartézké genetické programovanie.

Z obrázka 8.7, sa dá vyčítať, že jednotlivé nastavenia kartézskej mriežky vyzerajú dosť podobne a všetky dosahujú podobných výsledkov. Z tabuľky 8.13, sa vyčítalo, že najlepšiu priemernú AUC má nastavenie 1×100. Pri porovnaní pomocou všetkých zvyšných nastavení kartézskej mriežky samostatne s mriežkou 1×100 pomocou Mann-Whitney U testu z hľadinou pravdepodobnosti 0.05, bolo zistené, že mriežka 1×100 má pravdepodobne rovnakú distribúciu z mriežkami 10×10 a 25×25. Následne boli porovnané rýchlosti jednotlivých vybraných mriežok.

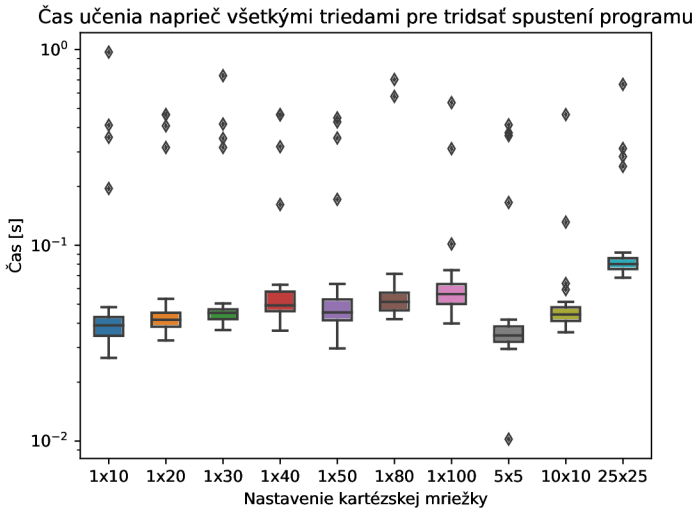


Obr. 8.7: Obrázok znázorňuje priemernú hodnotu fitness pre jednotlivé nastavenia kartézskej mriežky za daný počet behov.

Kartézká mriežka	Hodnota fitness Minimum	Priemer	Maximum
10x10	0.457514	0.572175	0.686490
1x10	0.410523	0.557541	0.648805
1x100	0.504889	0.587533	0.666879
1x20	0.484425	0.554837	0.668357
1x30	0.449111	0.553901	0.653881
1x40	0.452008	0.552717	0.720871
1x50	0.416456	0.554947	0.631096
1x80	0.446849	0.555464	0.659160
25x25	0.486727	0.572571	0.695862
5x5	0.444535	0.558089	0.656923

Tabuľka 8.13: Minimum, priemer a maximum hodnoty fitness pre jednotlivé nastavenia kartézskej mriežky.

Nasledujúci Obrázok 8.8 znázorňuje čas učenia pre jednotlivé kartézske mriežky za daný počet beh programov. Z obrázku sa dá vyčítať, že rýchlosť pri zvyšovaní počtu stĺpcov v kartézskej mriežke pre jednotlivé lineárne nastavenia mriežky sa čas trénovania postupne zvyšuje. To isté platí aj pri zvyšovaní počtu riadku a stĺpcov kartézskej mriežky pre štvorcové nastavenia. Z výsledkov štatistického testu sme zistili, že kartézske mriežky 1×100, 10×10 a 25×25, majú pravdepodobne rovnaké distribúcie. Pričom z tabuľky 8.14 vieme zistiť, že CPU čas v sekundách pre jednotlivé mriežky je nasledovný: $1 \times 100 = 0.083$, $10 \times 10 = 0.063$ a $25 \times 25 = 0.123$. Priemerná hodnota fitness AUC pre jednotlivé nastavenia mriežky je nasledovná: $1 \times 100 = 0.587533$, $10 \times 10 = 0.572175$ a $25 \times 25 = 0.572571$. Všetky hodnoty fitness pre jednotlivé nastavenia sú si veľmi podobné a najrýchlejšie nastavenie je 10×10.



Obr. 8.8: Obrázok znázorňuje priemerný CPU čas [s] pre jednotlivé nastavenia kartézskej mriežky za daný počet behov.

Kartézká mriežka	CPU čas [s] Minimum	Priemer	Maximum
10x10	0.035938	0.062897	0.465260
1x10	0.026570	0.103250	0.970750
1x100	0.039876	0.083317	0.535660
1x20	0.032618	0.094264	0.465530
1x30	0.036913	0.104473	0.735508
1x40	0.036610	0.091993	0.465340
1x50	0.029697	0.090231	0.447580
1x80	0.041905	0.093634	0.703410
25x25	0.068265	0.123010	0.664140
5x5	0.010220	0.074293	0.412490

Tabuľka 8.14: Minimum, priemer a maximum CPU času pre jednotlivé nastavenia kartézskej mriežky.

Z výsledkov experimentu bolo do nasledujúceho experimentu použité najlepšie nastavenie kartézskej mriežky 10×10.

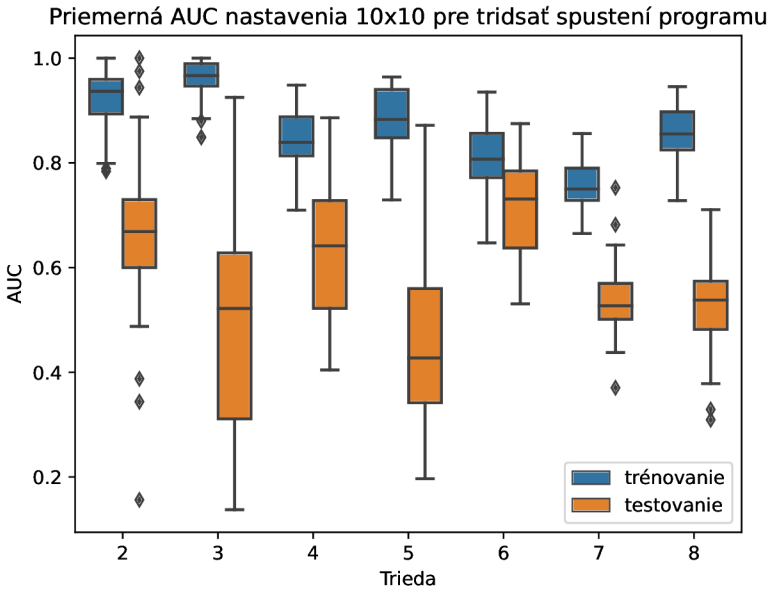
8.4 Experiment 4: Hľadanie najlepšieho kandidátneho riešenia

Cieľom tohoto experimentu je nájsť najlepšie kandidátne riešenie. Pri tomto experimente sú použité získané dáta zo sekcie 8.2. Parametre pre nastavenie kartézskeho genetického programovania sú popísané v Tabuľke 8.15.

Parametre	Zvolené hodnoty
Tvar kartézskej mriežky	10×10
L-back	Vždy plné prepojenie
Veľkosť populácie	5
Počet generácií	500
Miera mutácie	0.05
Veľkosť obrázkov dátovej sady	14×14
Dátový typ použitý pri výpočte	uint8_t

Tabuľka 8.15: Popis počiatočných parametrov pre kartézké genetické programovanie.

Tak isto ako aj pri experimentoch na dátovej sade MNIST aj pri tejto úlohe bolo najlepšie riešenie zložené z tridsať behov programu. Na Obrázku 8.9 sa dá pozorovať, že najlepší klasifikátor pre dané nastavenie mriežky 10×10 je pre triedu 6 s priemernou fitness 0.716 a najhorší klasifikátor je pre triedu 5 s priemernou fitness 0.440. Najlepšie sa zase učil klasifikátor pre triedu 3 z priemernou fitness 0.955 a najhoršie sa učil klasifikátor pre triedu 7 z priemernou fitness 0.757.



Obr. 8.9: Obrázok znázorňuje nastavenie kartézskej mriežky 10×10 s L_back 10 a priemernú hodnotu fitness AUC pre jednotlivé triedy naprieč trénovacími a testovacími dátami za daný počet behov programu.

Z tabuľky 8.16 sa dá pozorovať, že maximálna fitness AUC pri učení na trénovacích dátach bola pre jednotlivé triedy od 2 po 8 nasledovaná: 1, 1, 0.948, 0.964, 0.935, 0.856 a 0.946. Pri klasifikácii na testovacích dátach bola maximálna hodnota fitness pre triedy od 2 po 8 nasledovná: 1, 0.925, 0.886, 0.872, 0.875, 0.752 a 0.710. Pričom tieto hodnoty pri testovacích dátach tvoria najlepšie kandidátne riešenie s priemernou AUC naprieč triedami 0.86.

Datová sada	Hodnota fitness					
	Testovacie dáta			Trénovacie dáta		
	Minimum	Priemer	Maximum	Minimum	Priemer	Maximum
Trieda						
2	0.156250	0.661667	1.000000	0.783784	0.916204	1.000000
3	0.137500	0.497708	0.925000	0.848889	0.955397	1.000000
4	0.404412	0.631434	0.886029	0.709634	0.839746	0.948440
5	0.196581	0.440456	0.871795	0.729167	0.880994	0.964120
6	0.530612	0.716199	0.875000	0.647170	0.809784	0.935094
7	0.370370	0.536925	0.752525	0.665031	0.757327	0.855932
8	0.309211	0.520833	0.710526	0.727941	0.854517	0.945588

Tabuľka 8.16: Minimum, priemer a maximum hodnoty fitness podľa trénovacích a testovacích dát pre jednotlivé nastavenia kartézskej mriežky.

Na nasledujúcom Obrázku 8.17, môžeme vidieť maticu zámen pre najlepší viac-triedny klasifikátor. Ďalej môžeme vidieť, že výsledný viac-triedny klasifikátor má dobré predpoklady na to, že pri väčšej dátovej sade by sa dokázal lepšie naučiť správne klasifikovať

jednotlivé triedy. Je vidieť jasnú snahu o klasifikáciu jednotlivých dát do spravených tried alebo tried jej blízkych.

Skutočná trieda Predikovaná trieda	2	3	4	5	6	7	8
2	2	0	0	0	1	0	0
3	0	1	0	0	1	1	0
4	0	1	7	1	2	1	1
5	0	0	0	1	3	1	0
6	0	0	0	0	3	1	0
7	0	0	0	0	0	3	1
8	0	0	1	0	2	2	2
n/a	0	0	0	1	2	0	0

Tabuľka 8.17: Matica zámen pre najlepšie kandidátne riešenie.

8.5 Experiment 5: Porovnanie fitness s existujúcimi riešeniami

Na začiatku kapitoly bolo popísané riešenie zo štúdie [9]. Pri ich riešení použili variantu CGP ktorá sa volá IRCGP. V tejto práci bola použitá štandardná varianta CGP. Následne v ich práci z jednotlivých obrázkov kociek získali obrysy, na ktorých následne učili IRCGP. Pri tejto práci bol zásadný rozdiel v tom, že sa obrázky brali ako celok v rôznych veľkostiach. Ich výsledky AUC pre najlepší viac-triedny klasifikátor boli pre trénovaciu a testovaciu sadu 0.70. V tejto práci pre najlepší viac-triedy klasifikátor bola AUC pre trénovaciu sadu 0.95 a pre testovaciu 0.86.

8.6 Experiment 6: Porovnanie s inými metódami

Pre porovnanie najlepšieho viac-triedneho klasifikátoru pre dátové sady MNIST a kocky boli zvolené dve metódy: CNN a SVM. CNN sa v tejto práci dá spustiť pomocou skriptu `cnn.py` a SVM pomocou skriptu `svm.py`. Použité technológie pre tieto skripty sú popísané v sekcií 7.1 a jednotlivé skripty sú popísané v sekcii 7.5. Pri oboch metódach bola najskôr načítaná dátová sada a následne bola normalizovaná, čiže jednotlivé pixle obrázkov nadobúdali hodnôt od 0 do 1. CNN sa skladá z nasledujúcich vrstiev: konvolučná vrstva, ktorá obsahuje 32 filtrov o veľkosti 3×3, následne aj druhú konvolučnú vrstvu, ktorá obsahuje 64 filtrov o veľkosti 3×3, vrstvu pre združenie susediacich pixlov z veľkosťou pohybového okna 2×2, poruchovú vrstvu, ktorá pridáva na vstup neurónovej siete 0 z frekvenciou 0.25, vyhladzovaciu vrstvu, aktivačnú vrstvu z počtom jednotiek 128 a aktivačnou funkciou relu, poruchovú vrstvu, ktorá pridáva na vstup neurónovej siete 0 z frekvenciou 0.5 a posledná je aktivačná vrstva z jednotkou počtu tried a aktivačnou funkciou softmax. SVM bolo vyskúšané z lineárnymi a nelineárnymi rozhodovacími hranicami. Z nedostatku času boli tieto dve metódy spustené na daných dátových sadách len raz.

Z Tabuľky 8.18 sa dá vyčítať, že konvolučné neuronové siete a SVM dokázali navrhnúť riešenia s rovnakou kvalitou AUC. Riešenie, ktoré bolo navrhnuté v našej práci má horšiu kvalitu, ale naše riešenie je lepšie optimalizované na hardware. Pri kockách si najlepšie

poradilo riešenie navrhnuté v tejto práci, za nim nasledovalo riešenie navrhnuté konvolučnou neurnovou sieťou a potom SVM.

Metóda Dátová sada	CNN	SVM (lineárne)	SVM (nelineárne)	CGP
MNIST	0.99	0.99	0.99	0.95
Kocky	0.81	0.69	0.65	0.86

Tabuľka 8.18: Porovnanie jednotlivých metód podľa AUC na rôznych testovacích dátových sadách.

Jednou z hlavných výhod klasifikácie pomocou CGP je, že napríklad na rozdiel od CNN vykonáva menší počet operácii na klasifikáciu dát. Pri CNN to môže byť niekoľko stoviek operácii a pri CGP to môže byť niekoľko desiatok čo je výhodnejšie a aj efektívnejšie pre hardware.

Kapitola 9

Záver

Cieľom tejto práce bolo navrhnúť evolučný návrh klasifikátoru obrazov. Návrh využíva metódy genetického programovania, konkrétne sa jedná o kartézské genetické programovanie (CGP). Tento návrh, bol implementovaný a schopnosť učenia a klasifikácie bola overená na dvoch riešených úlohách. Jedna úloha bola klasifikácia ručne písaných číslic a druhá úloha bola klasifikácia kociek, ktoré sa používajú pri teste vizuo-priestorových schopností, ktorý sa používa pri určovaní mieri demencie pri Parkinsonovej chorobe. V experimentoch bola najskôr overená implementácia návrhu. Výsledky boli následne porovnané a overené z prácou [4]. Ďalšia časť experimentov bola venovaná návrhu klasifikátoru pomocou CGP pre kocky. Pri experimentoch sa klasifikátor navrhoval pri rôznych veľkostiach obrázkov a kartézských mriežok. Pri porovnaní rôznych veľkostí obrázkov (8×8, 14×14 a 28×28) došlo k vyhodnoteniu, že najlepšia veľkosť obrázkov pre návrh klasifikátoru je 14×14. Následne bolo hľadané najlepšie nastavenie kartézkej mriežky, z výsledkov tohoto experimentu sa zistilo, že najlepšie nastavenie kartézskej mriežky je 10×10. Z najlepšou veľkosťou obrázkov a kartézskou mriežkou bol navrhnutý najlepší viac-triedny klasifikátor zložený z binárnych klasifikátorov, ktorý mal AUC na testovacej dátovej sade 0.86. Pre čísla bol zase najlepší navrhnutý viac-triedny klasifikátor s AUC pre testovaciu dátovú sadu 0.95. Navrhnuté riešenia boli následne porovnané z inými prístupmi ako sú konvolučné neurónové siete, ktoré pre čísla dosiahli AUC 0.99 a pre kocky 0.81. Ďalej boli riešenia porovnané z metódou podporných vektorov (SVM), ktoré pre čísla pomocou lineárnej a nelineárnej rozhodovacej hranice dosiahli obe navrhnuté riešenia AUC 0.99, pre kocky lineárna rozhodovacia hranica dosiahla AUC 0.69 a nelineárna 0.65. Výsledky boli kvôli nedostatku času vyhodnotené len z jedného spustenia programu pre CNN a SVM s lineárnou a nelineárnou rozhodovacou hranicou, pričom CGP bolo pre jednotlivé nastavenia spustené tridsať krát. Je vidieť, že navrhnuté riešenie pre kocky v tejto práci je výrazne lepšie ako navrhnuté riešenia pre konvolučné neurónové siete a metódu podporných vektorov. Následne bolo najlepšie navrhnuté riešenie pre kocky porovnané s existujúcim riešením [9], ktoré bolo riešené pomocou inej varianty CGP, konkrétne IRCGP a návrh klasifikátoru bol spúšťaný na dátovej sade tvorenej z obrysoch kociek. Výsledná AUC najlepšieho navrhnutého klasifikátoru v danej štúdii bola 0.70. Metóda použitá v tejto práci bola pre riešenú úlohu klasifikácie kociek použitá prvý krát a z výsledkov sa dá pozorovať, že výsledky boli lepšie ako v zmienenej štúdii.

Na základe vykonaných experimentov pri tejto práci, ďalej doporučujem pokračovať doplnením experimentov o rôzne nastavenia maximálneho prepojenia kartézskych mriežok, veľkosti populácie, počtu generácií a miery mutácie. Zaujímavé by mohlo byť vyskúšať poskladať výsledný viac-triedny klasifikátor nie len z najlepších binárnych klasfikátorov získaných z tridsať behov programu pre najlepšie nastavenie, ale skúsiť ho poskladať z naj-

lepších binárnych klasifikátorov z všetkých nastavení, ktoré navrhovali podobne kvalitné výsledné binárne klasifikátory. Ďalej by som doporučil rozšíriť implementáciu o koevolučné algoritmy, ktoré by výrazne urýchlili proces učenia.

Vďaka tejto bakalárskej práci som sa naučil rôzne metódy strojového učenia, ktoré som následne aj implementoval a metódy vyhodnocovania binárnych klasifikátorov, ktoré som pri vyhodnotený aplikoval. Taktiež som sa naučil o evolučných algoritmoch a genetickom programovaní, konkrétne kartézskom genetickom programovaní, ktoré som použil pri implementácii programu pre evolučný návrh klasifikátora obrazov. Táto problematika ma veľmi zaujala a rozhodne by som sa chcel ďalej vzdelávať v oblasti strojového učenia.

Literatúra

[1] Burkov, A. *The Hundred-Page Machine Learning Book*. Quebec, Canada: Andriy Burkov, 2019. ISBN 978-1-9995795-0-0.

[2] De Jong, K. In: *Handbook of Natural Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, kap. Generalized Evolutionary Algorithms, s. 625–635. ISBN 978-3-540-92910-9.

[3] James, G., Witten, D., Hastie, T. a Tibshirani, R. *An Introduction to Statistical Learning: with Applications in R*. 1. vyd. New York, NY: Springer, 2013. Springer Texts in Statistics. ISBN 978-1-4614-7137-0.

[4] Jašíčková, K. *Klasifikace obrazů pomocí genetického programování*. 2018. Diplomová práca. Vysoké učení technické v Brně fakulta informačních technologií.

[5] Miller, J. F. In: *Cartesian Genetic Programming*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, kap. Cartesian Genetic Programming, s. 17–34. ISBN 978-3-642-17310-3.

[6] Narkhede, S. *Understanding AUC – ROC Curve* [online]. 2018. [vid. 18.7.2021]. Dostupné z: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5.

[7] Narkhede, S. *Understanding Confusion Matrix* [online]. 2018. [vid. 18.7.2021]. Dostupné z: https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62.

[8] Sekanina, L. *Evoluční hardware : od automatického generování patentovatelných invencí k sebemodifikujícím se strojům*. Vyd. 1. Praha: Academia, 2009. Gerstner ; sv. 4. ISBN 978-80-200-1729-1.

[9] Smith, S. L. a Lones, M. A. Implicit context representation Cartesian genetic programming for the assessment of visuo-spatial ability. In: IEEE. *2009 IEEE Congress on Evolutionary Computation*. 2009, s. 1072–1078.

[10] Vanneschi, L. a Poli, R. In: *Handbook of Natural Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, kap. Genetic Programming — Introduction, Applications, Theory and Open Issues, s. 709–739. ISBN 978-3-540-92910-9.