VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

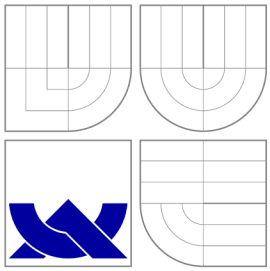
BOOSTING A EVOLUČNÍ ALGORITMY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

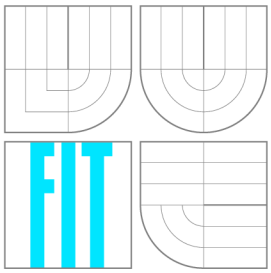
AUTOR PRÁCE
AUTHOR

MICHAL MRNUŠTÍK

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

BOOSTING A EVOLUČNÍ ALGORITMY

BOOSTING AND EVOLUTION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MICHAL MRNUŠTÍK
AUTHOR

VEDOUCÍ PRÁCE — Ing. MICHAL HRADIŠ
SUPERVISOR

BRNO 2008

Abstrakt

Tato práce představuje kombinaci AdaBoostu a evolučního algoritmu. Evoluční algoritmus je použit pro hledání lineární kombinace Haarových příznaků. Z té je vytvořen slabý klasifikátor pro AdaBoost. Jsou zde popsány základy klasifikace, Haarovy příznaky a Adaboost. Uvedeny jsou také základní informace o evolučních algoritmech. Dále obsahuje teoretický popis spojení AdaBoostu a evolučního algoritmu, doplněný o některé implementační detaily. Implementace je testována na obrazových datech jako součást systému pro detekci obličeje. Výsledky jsou porovnány se samostatnými Haarovými příznaky.

Klíčová slova

boosting, adaboost, evoluční algoritmy, rozpoznávání vzorů, haarovy příznaky

Abstract

This thesis introduces combination of the AdaBoost and the evolutionary algorithm. The evolutionary algorithm is used to find linear combination of Haar features. This linear combination creates the feature to train weak classifier for AdaBoost. There are described basics of classification, Haar features and the AdaBoost. Next there are basic information about evolutionary algorithms. Theoretical description of combination of the AdaBoost and the evolutionary algorithm is included too. Some implementation details are added too. Implementation is tested on the images as part of the system for face recognition. Results are compared with Haar features.

Keywords

boosting, adaboost, evolutionary algorithms, pattern recognition, haar features

Citace

Michal Mrnuštík: Boosting a evoluční algoritmy, bakalářská práce, Brno, FIT VUT v Brně, 2008

Boosting a evoluční algoritmy

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Michala Hradiše.

.....
Michal Mrnuštík
12. května 2008

Poděkování

Především bych chtěl poděkovat vedoucímu práce Ing. Michalu Hradišovi za kvalitní vedení a podporu.

Obsah

1 Úvod **2**

2 Klasifikace, rozpoznávání vzorů a strojové učení **3**
2.1 Klasifikace . . . 3
2.2 Učení klasifikátorů . . . 3
2.3 AdaBoost . . . 4
2.4 Haarovy příznaky . . . 4
2.5 Kaskáda klasifikátorů . . . 6

3 Evoluční algoritmy **8**
3.1 Křížení . . . 9
3.2 Mutace . . . 10
3.3 Výběr . . . 10

4 Spojení AdaBoostu a evolučních algoritmů **12**
4.1 Genom a příznaky . . . 12
4.2 Fitness funkce . . . 13
4.3 Průběh evolučního algoritmu . . . 13

5 Implementace **15**
5.1 Třída `TGAHaarFeatures` . . . 15
5.2 Evoluční algoritmus . . . 15
5.3 Vyrovnávací paměť . . . 16

6 Testy **17**
6.1 Parametry evolučního algoritmu . . . 17
6.2 Trénovací a testovací data . . . 20
6.3 ROC křivka . . . 20
6.4 Rychlost klasifikátoru . . . 21
6.5 Výsledky . . . 21

7 Závěr **23**
7.1 Genom, mutace a křížení . . . 23
7.2 Fitness . . . 24
7.3 Shrnutí . . . 24

Kapitola 1

Úvod

Člověk se odmalička učí rozeznávat věci okolo sebe pomocí všech svých smyslů. S rozvojem umělé inteligence se také stroje stávají schopnými učit se a poznávat své okolí. Tímto se zabývá strojové učení a rozpoznávání vzorů. Při získávání informací a komunikaci s okolím jsou nejdůležitějšími smysly zrak a sluch. Bylo vymyšleno mnoho metod jak zařídit, aby stroj slyšel nebo viděl a byl schopen tuto informaci dále zpracovat. Ty bývají většinou specializované na jeden konkrétní problém. Existují tak systémy pro rozpoznávání obličejů, automobilů či dalších objektů v obraze.

Základní informace o rozpoznávání vzorů a strojovém učení jsou uvedeny v kapitole 2. Dále tato kapitola popisuje algoritmus pro strojové učení AdaBoost, získávání informace z obrazu pomocí Haarových příznaků a spojení více klasifikátorů pomocí tzv. kaskády.

U strojového učení je často potřeba vybrat vhodné řešení z velkého počtu možností. Pokud není k dispozici vhodný matematický postup, lze použít evoluční algoritmy. Ty jsou založeny na simulaci přirozeného výběru, který probíhá v přírodě. Schopnost přežít je vyjádřena schopností řešit určitý problém. Evolučním algoritmům se věnuje kapitola 3. Problematika evolučních algoritmů je velmi rozsáhlá, proto jsou popsány jen principy a informace nutné k pochopení dalších kapitol.

Cílem této práce je vytvořit funkční spojení AdaBoostu a evolučního algoritmu. V ideálním případě bychom tak chtěli dosáhnout lepších výsledků, než poskytují stávající metody. Tato fáze byla řešena v rámci semestrálního projektu. Prezentované řešení bylo rozšířeno o použití Haarových příznaků, proto je ve stávající podobě použitelná jen pro zpracování obrazu. Teoretické řešení je popsáno v kapitole 4.

Toto spojení bylo implementováno a přidáno do již existujícího projektu, který je vyvíjen na Fakultě informačních technologií Vysokého učení technického v Brně. Jedná se o systém sloužící k výzkumu klasifikátorů pro detekci objektů v obraze. Podrobně je popsán v článku [3]. Vybrané implementační detaily jsou uvedeny v kapitole 5. Zdrojové kódy se nacházejí na přiloženém CD.

Výsledný systém byl trénován a testován na obrazových datech pro detekci obličeje. Kapitola 6 pojednává o konfiguraci, jenž byla použita, a provedených testech. Obsahuje informace o trénovacích a testovacích datech. Výsledky testů jsou zhodnoceny a porovnány s výsledky, které byly získány za použití samostatných Haarových příznaků.

Při vytváření této práce bylo odhaleno mnoho nedostatků v evolučním algoritmu i celkovém návrhu metody. V závěrečné kapitole 7 jsou uvedeny jak možnosti řešení těchto nedostatků, tak i jiná vylepšení, jenž by byla realizovatelná při dalším vývoji.

Kapitola 2

Klasifikace, rozpoznávání vzorů a strojové učení

V této kapitole se dozvíte o rozpoznávání vzorů (viz [1]) a další informace s tím spojené. Systémy pro rozpoznávání vzorů se skládají z více částí. Nejdříve je nutno načíst vstupní data a pokud nebudeme pracovat přímo s nimi, tak z nich spočítat příznaky. Potom přijde na řadu klasifikátor, který určí třídu dat. Na té závisí výstup systému.

2.1 Klasifikace

Klasifikace je přiřazování vzorků dat do tříd. Vzorkem dat může být cokoli s čím pracujeme, ale většinou se jedná o vektor čísel. Třídou je oblast, do které může vzorek dat patřit. Klasifikátor je pravidlo, které přiřadí vzorek dat do třídy. Pokud se například snažíme o rozpoznání řeči, tak vzorek dat je zvukový signál. Třídami jsou jednotlivá slova nebo znaky. Při detekci obličeje je vzorkem dat oblast obrazu. Jednou třídou jsou obličeje, druhou vše ostatní (tzv. pozadí).

Vstupem klasifikátoru mohou být přímo data, která ale bývají mnohorozměrná (např. obraz má tolik rozměrů kolik pixelů), a tak se z nich častěji extrahují tzv. příznaky (features). Tím snížíme počet rozměrů dat, nejlépe na jeden. Bylo vypracováno mnoho metod jak toho dosáhnout. Pro obecná data lze použít například PCA (Principal components analysis) nebo LDA (Linear discriminant analysis). Ty jsou založeny na linární kombinaci a statistických metodách. Pro obrazová data se velmi osvědčily Haarovy příznaky (viz 2.4).

Nejjednodušším příkladem klasifikátoru je mezní hodnota (tzv. treshold), ta tvoří hranici mezi třídami. Všechny vzorky menší než mez jsou přiřazeny do jedné třídy, větší do druhé. Pokud je třeba rozlišovat více tříd, použijeme více mezních hodnot a třídy budou ohraničeny ze dvou stran.

U klasifikátoru sledujeme několik hodnot. Celková chyba je poměr špatně určených vzorků k celkovému počtu. Dále ve vztahu k jedné třídě sledujeme počet prvků, které byly do třídy chybně přiřazeny (false positive) a těch, co do třídy patří, ale byly přiřazeny jinam (false negative).

2.2 Učení klasifikátorů

Pokud má klasifikátor dobře určovat příslušnost k třídě, musí se to naučit. K tomu potřebujeme trénovací množinu dat. Jestli chceme zjistit nakolik bylo trénování úspěšné, je

třeba mít navíc testovací množinu dat. Data v obou množinách se musí lišit, ale musí být podobná reálnému nasazení klasifikátoru. To znamená, že pokud bude systém používán v určitém prostoru (laboratoř, tovární hala), měly by data pocházet odtud. V podstatě stačí, pokud z daného prostoru budou pocházet vzorky pozadí (obraz haly, rachot strojů). Vzorky detekované třídy (lidský hlas, obrazy obličeje) mohou být pořízeny jinde.

Klasifikátor se snaží dosáhnout co nejlepší úspěšnosti na trénovacích datech. Na testovací množině zkoušíme, jak je klasifikátor použitelný i pro jiná data než ta, na kterých byl trénován. Se zvyšováním přesnosti na trénovacích datech může začít růst chyba na tetsovacích datech. Dochází k tzv. přetrénování. Příčinou je přílišné přizpůsobení trénovacím datům, které nenechává žádný prostor pro odlišnosti testovacích dat. Proto je nutné trénování včas ukončit.

Existují dva způsoby učení. Učení s učitelem (supervised learning) a učení bez učitele (unsupervised learning). Při učení s učitelem trénovací data obsahují informaci o třídě, do které patří. Díky tomu může klasifikátor sledovat, jak moc se mu daří určit správnou třídu a podle toho se přizpůsobovat. U učení bez učitele klasifikátor žádnou informaci o třídě nedostává. Snaží se v datech najít podobnost a podle toho je rozdělit do předem daného počtu shluků (ten odpovídá počtu tříd). Svou úspěšnost hodnotí podle toho, jak se mu to podařilo. Nevýhodné je, že algoritmus nepozná, zda třídy rozdělil dobře, ani která třída je která. Proto se při učení klasifikátorů většinou využívá učení s učitelem.

2.3 AdaBoost

AdaBoost je algoritmus, který využívá lineární kombinace slabých klasifikátorů k dosažení lepšího výsledku. U slabého klasifikátoru nám stačí, když má chybu menší než 50 %. Na začátku je každému vzorku přiřazena váha, která se časem mění. Pokud je třída vzorku určena špatně, jeho váha stoupá, pokud dobře, tak klesá. To znamená, že se klade větší důraz na špatně klasifikované vzorky. To zaručuje, že chyba postupně klesá a ve většině případů nedochází k přetrénování. Na druhou stranu je tím způsobna citlivost vůči šumu. Více o AdaBoostu lze najít v [2] a [8], z kterých vychází algoritmus 2.1.

2.4 Haarovy příznaky

Haarovy příznaky dobře uchovávají informace o obrazu, a proto se používají pro obrazová data. Lze je rychle spočítat a pokud je dobře vybereme (viz níže), závisí jen na té části obrazu, která je pro klasifikaci podstatná. Díky tomu je lze úspěšně použít pro tvorbu klasifikátorů.

Na obrázku 2.2 jsou dvojbarevné obdélníky. To jsou příklady tvarů, kterých mohou Haarovy příznaky nabývat. Jejich velikost se může libovolně měnit, ale poměr bílé a černé části musí být zachován. Pro obraz 24×24 pixelů je příznaků více než 180 000. Z tohoto počtu je nutno vybrat ty, kterými jednotlivé třídy nejlépe rozlišíme. Klasifikátoru většinou nestačí jeden příznak, ale je nutno použít jejich kombinaci. V [7] je pro slabý klasifikátor použit pouze jeden příznak, což je kompenzováno použitím více slabých klasifikátorů (viz 2.3).

Obraz, který vstupuje do systému, musí být převeden do stupňů šedi. Hodnotou bodu je intenzita barvy v tomto bodě. Příznak spočítáme tak, že přiložíme obdélník na obraz, sečteme hodnoty pixelů v bílé oblasti a odečteme od součtu pixelů v černé oblasti. Pokud bychom měli při výpočtu procházet všechny pixely dané oblasti, trval by příliš dlouho.

1. Na vstupu jsou trénovací data $(x_1, y_1) \ldots (x_m, y_m)$, kde $x_i \in X$, $y_i \in Y = \{-1, +1\}$. X je množina vstupních dat. Y je množina tříd.

2. Každému vzorku dat i přiřaď počáteční váhu $D_1(i) = 1/m$.

3. T je počet prvků (slabých klasifikátorů) výsledného silného klasifikátoru.
   Pro $t = 1, \ldots T$:

   - Pomocí vah D_t, vyber slabý klasifikátor $k_t \colon X \to \{-1, +1\}$ s nejmenší chybou (součtem vah vzorků, které slabý klasifikátor špatně určil) $\epsilon_t = \sum_{i:k_t \neq y_i}^{m} D_t(i)$.
   - $\alpha_t = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
   - Přepočítej váhy vzorků:

$$\begin{aligned} D_{t+1}(i) &= \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{když } k_t(x_i) = y_i \\ e^{\alpha_t} & \text{když } k_t(x_i) \neq y_i \end{cases} \\ D_{t+1}(i) &= \frac{D_t(i) \exp\left(-\alpha_t y_i k_t(x_i)\right)}{Z_t} \end{aligned}$$

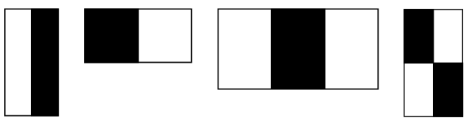
     Z_t je normalizační faktor, zajišťující, že D_t má vlastnosti distribuční funkce (integrál je roven jedné) a spočítáme jej:

$$Z_t = \sum_{i=1}^{m} D_t(i) \exp\left(-\alpha_t y_i k_t(x_i)\right)$$

4. Výsledný silný klasifikátor je:

$$K(x) = \begin{cases} +1 & \text{když } \sum_{t=1}^{T} \alpha_t k_t(x) \geq 0 \\ -1 & \text{když } \sum_{t=1}^{T} \alpha_t k_t(x) < 0 \end{cases}$$

Algoritmus 2.1: AdaBoost podle [2] a [8]



Obrázek 2.2: Příklady haarových příznaků

Tento problém řeší tzv. integrální obraz. U integrálního obrazu je hodnota v jednom bodu součtem intenzit ve všech bodech od tohoto bodu nahoru a doleva. Z toho lze již jednoduše spočítat součet intenzit v obdélníkové oblasti.

Pokud chceme spočítat hodnotu oblasti v integrálním obraze, podíváme se nejdříve na

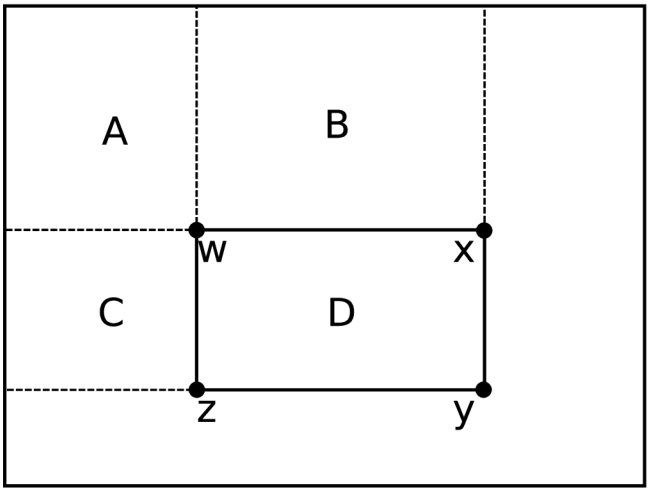
obrázek 2.3. Je na něm znázorněn integrální obraz, body a oblasti v něm. Naším cílem je zjistit hodnotu oblasti D. Body w, x, y, z musí být voleny jako na obrázku 2.3. A, B, C, D jsou součty intenzit v jednotlivých oblastech. Z rovnic

$$\begin{aligned} w &= A \\ x &= A + B \\ y &= A + B + C + D \\ z &= A + C \end{aligned}$$

potřebujeme vyjádřit D pomocí w, x, y, z. Abychom toho dosáhli, vynásobíme některé rovnice -1:

$$\begin{aligned} w &= A \\ -x &= -A - B \\ y &= A + B + C + D \\ -z &= -A - C \end{aligned}$$

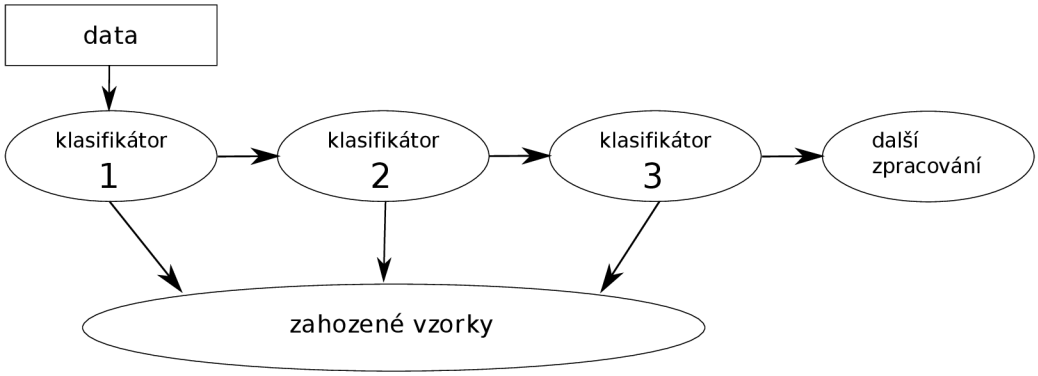
Z toho získáme $D = w - x + y - z$.



Obrázek 2.3: Výpočet obdélníkové oblasti pomocí integrálního obrazu

2.5 Kaskáda klasifikátorů

V [7] je kromě AdaBoostu a Haarových příznaků uveden i postup jak výrazně urychlit klasifikaci. Je zobrazen na obrázku 2.4 a spočívá ve zřetězení klasifikátorů. Ty postupně zpracovávají vzorky dat a rozhodují, zda patří do detekované třídy. Pokud kterýkoli klasifikátor rozhodne, že tam vzorek nepatří, je zahozen a k dalšímu zpracování už nedochází. Klasifikátor je nutno trénovat tak, aby zahodil co nejméně prvků, které do třídy patří (co nejmenší false negative). Na konci zůstanou vzorky, co do třídy patří. Může to být i za cenu více vzorků do třídy chybně zařazených (vyšší false positive).



Obrázek 2.4: Kaskáda klasifikátorů

Kapitola 3

Evoluční algoritmy

Příroda a její zákony se ukázaly jako dobrá inspirace při vývoji výpočetních metod. Základy pro evoluční algoritmy položil Charles Darwin svou teorií přirozeného výběru. Druhy obývající naši planetu se již po miliony let vyvíjely z jednobuněčných organismů. Přežívají ty, které se dokázaly přizpůsobit okolním podmínkám lépe než ostatní.

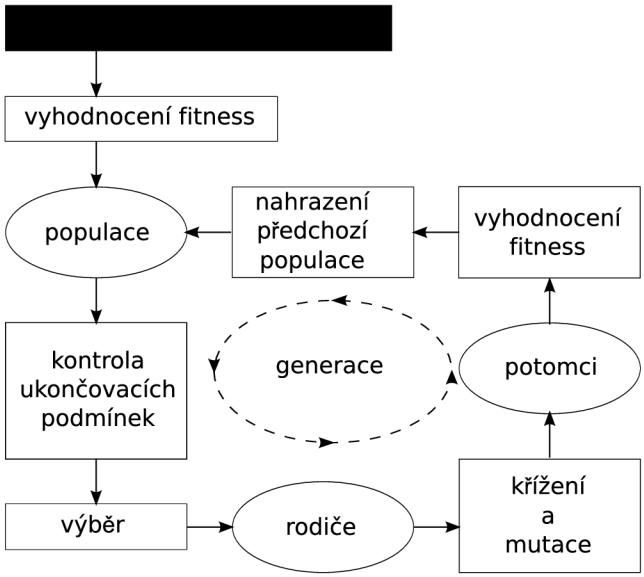
Evoluční algoritmy se používají k řešení složitých optimalizačních problémů, kde neexistuje jiný postup, nebo je časově příliš náročný. Pracují na podobném principu jako evoluce v přírodě. Místo jednotlivých organismů máme jedince představující řešení problému. Jedinec bývá definován svou vnitřní reprezentací (tzv. genomem) a hodnotou fitness, která představuje číselně vyjádřenou kvalitu řešení, které jedinec poskytuje. Genom mívá různou reprezentaci. Nejčastější je binární nebo reálná, ale lze použít cokoli, co bude vhodné (znaky, stromy, atd.). Fitness funkce je klíčovým bodem při návrhu evolučního algoritmu. Měla by být výpočetně co nejméně náročná. Počet vyhodnocení se pohybuje v řádech tisíců, až miliónů. Dále musí platit, že čím vyšší (nebo nižší, pokud hledáme minimum) je hodnota fitness, tím kvalitnější řešení jedinec poskytuje.

Podle [4] rozlišujeme evoluční algoritmy především na genetické algoritmy, evoluční strategie a genetické programování.

- Genetické algoritmy mají binárně kódované jedince. Mutace i křížení se projevuje na úrovni jednotlivých bitů.
- Evoluční strategie reprezentuje jedince jako vektor reálných čísel. Pro křížení a mutaci lze použít jak metod používaných pro genetické algoritmy, tak různých vektorových operací.
- Genetické programování je vytváření algoritmů pomocí evolučního procesu.

Na obrázku 3.1 jsou jednotlivé fáze evolučního algoritmu. Schéma odpovídá implementaci popisované v kapitole 5 i průběhu popisovanému v kapitole 4.3. Na začátku je náhodně vygenerována populace jedinců. Je vyhodnocena jejich fitness. Následně se kontroluje, zda nebyla splněna některá ukončovací podmínka (např. dosažení určité hodnoty fitness, maximálního počtu generací atd.). Pak jsou vybráni rodiče následující populace (viz 3.3). Křížením a mutací z nich získáme potomky a vyhodnotíme jejich fitness. Výběrem z rodičů a potomků nahradíme stávající populaci. Tak algoritmus pokračuje, dokud není splněna ukončovací podmínka.

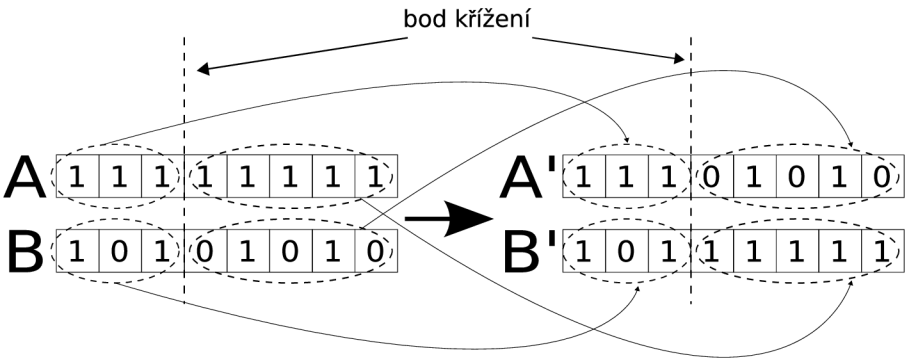
Ve zbytku této kapitoly jsou příklady postupů pro jednotlivé kroky evolučního algoritmu. Kterýkoli z nich lze nahradit jiným (vhodnějším) postupem pro konkrétní problém.



Obrázek 3.1: Schéma evolučního algoritmu podle [5]

3.1 Křížení

Křížením se kombinují vlastnosti dvou jedinců (v některých případech lze použít jedinců i více, ale většinou k tomu není důvod). Cílem je vznik jedince lepšího než kterýkoli z rodičů. Postupy mohou být velmi rozmanité a lze je různě kombinovat. Mezi nejpoužívanější patří segmentové křížení, které je znázorněno na obrázku 3.2 a popsáno v algoritmu 3.3.



Obrázek 3.2: Segmentové křížení

Existují i varianty, kde se generuje více bodů křížení. Například u dvou bodů vzniknou tři části A,B a C. Potomci vzniknou výměnou částí B.

Toto křížení lze použít i na genom reprezentovaný vektorem reálných čísel, jen místo bitů jsou složky vektoru. Další použitelnou metodou je křížení průměrem. Ze složek rodičovských vektorů r_1 a r_2 vznikne jediný potomek s odpovídající složkou $p = \frac{r_1+r_2}{2}$.

Odlišný přístup je křížení lineární kombinací rodičů. Pro každou složku r_1 a r_2

- Vyber dva rodiče A, B.
- Náhodně zvol bod křížení.
- A' vznikne z té části A, která leží před bodem křížení a z části B ležící za bodem křížení.
- B' vznikne z té části B, která leží před bodem křížení a z části A ležící za bodem křížení.

Algoritmus 3.3: Segmentové křížení

rodičovských vektorů a číslo x v intervalu $\langle 0, 1\rangle$ jsou vypočítány složky potomků:

$$\begin{aligned} p_1 &= xr_1 + (1-x)r_2 \\ p_2 &= (x-1)r_1 + xr_2 \end{aligned}$$

3.2 Mutace

V průběhu evolučního algoritmu je vhodné, když se v populaci objeví prvky, kterých nelze dosáhnout křížením. To zajišťuje právě mutace. Algoritmus 3.4 popisuje základní postup mutace pro binární genom. U reálných genomů se neprovádí negace, ale například součet s náhodně generovaným číslem v předem daném rozsahu.

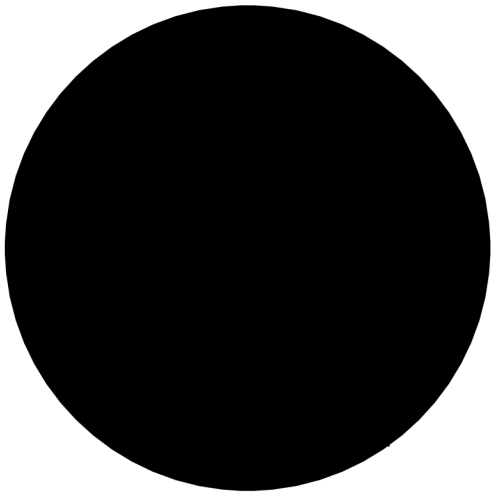
- Je dána pravděpodobnost mutace $p_m \in \langle 0, 1\rangle$.
- Pro každý bit b v genomu:
  - Náhodně generuj $x \in \langle 0, 1\rangle$.
  - Pokud $x \leq p_m$ neguj b.

Algoritmus 3.4: Mutace binárního genomu

3.3 Výběr

Existují různé postupy jak vybrat jedince, kteří se stanou rodiči následující populace. Většinou jsou založeny na náhodném výběru, kde jedinci s vyšší fitness mají větší šanci stát se rodiči. Takovým je i ruleta (roulete wheel, fitness proportionate selection).

Princip je velmi jednoduchý. Na obrázku 3.5 vidíte kruh rozdělený na části. Písmeny jsou označeny jedinci. Velikost políčka je určena hodnotou fitness. Výběr se provádí tak, že se „roztočí ručička“ a prvek, na který ukáže, je zařazen mezi rodiče. Pravděpodobnost, že bude jedinec i vybrán je: $p_i = \frac{f_i}{\sum_{j=1}^{N} f_j}$, kde N je velikost populace a f je hodnota fitness. Výběr probíhá tolikrát, kolik potřebujeme rodičů (jedinec může být vybrán i víckrát).



Obrázek 3.5: Ruleta

Kapitola 4

Spojení AdaBoostu a evolučních algoritmů

Adaboost i evoluční algoritmy jsou samy o sobě použitelné pro učení klasifikátorů. Každý přístup má své výhody a mohou se vzájemně doplňovat. To je dobře ukázáno v [6], kde je evolučního algoritmu použito k prohledávání prostoru Haarových příznaků. Nejlepší příznak slouží k vytvoření slabého klasifikátoru pro AdaBoost.

Jak už bylo uvedeno v kapitole 2.4, Haarovy příznaky jsou vhodné pro zpracování obrazu. Jejich spojení s AdaBoostem má však jeden malý nedostatek. V pozdějších fázích AdaBoostu už je velmi obtížné nalézt Haarův příznak, který by přinášel nějakou novou informaci. To lze řešit více způsoby. Jednou z možností, jak se s tím vyrovnat, je použít více Haarových příznaků pro jeden slabý klasifikátor. Spojení může tvořit například lineární kombinace (viz 4.1). Evoluční algoritmus je pak možno využít k hledání vhodné linární kombinace příznaků pro slabý klasifikátor, jako se v [6] hledají jen příznaky.

4.1 Genom a příznaky

Genom je tvořen vektorem reálných čísel $\overrightarrow{v} = a_1, b_1, a_2, b_2, \ldots, a_m, b_m$, kde $a_m, b_m \in \langle -1, 1\rangle$. Reálné číslo b_m se při vyhodnocení musí převést na celé číslo, které je identifikačním číslem konkrétního Haarova příznaku. Při převodu je nejdříve nutno spočítat $i_m = |b_m| \cdot N$, kde N je celkový počet Haarových příznaků snížený o jedna, a zaokrouhlit i. Potom pro vzorek dat x získáme hodnotu příznaku $H(x) = a_1 h_{i_1}(x) + a_2 h_{i_2} + \cdots + a_m h_{i_m}$, kde $h_{i_m}(x)$ je hodnota Haarova příznaku s identifikačním číslem i_m.

Výsledná hodnota příznaku, který je použit v slabém klasifikátoru, je:

$$H(x) = a_1 h_1(x) + a_2 h_2(x) + \cdots + a_n h_n(x)$$

Kde $a_n \in \mathcal{R}$, $h_n(x)$ je hodnota Haarova příznaku h_n pro vzorek dat x.

Aby se daly příznaky dále zpracovávat, je třeba omezit jejich hodnotu. V našem případě se hodnota příznaku $H(x)$ nachází v intervalu $\langle -4, 4\rangle$. Toho je dosaženo normalizací složek $a_1, a_2, \ldots, a_n$ vektoru $\overrightarrow{v}$ tak, aby délka byla 1. Hodnoty Haarových příznaků jsou v intervalu $\langle -2, 2\rangle$. Abychom nepřekročili rozsah $\langle -4, 4\rangle$, je nutné hodnotu každého Haarova příznaku vydělit 2. Hodnota příznaku získaného pomocí evolučního algoritmu včetně normalizace je:

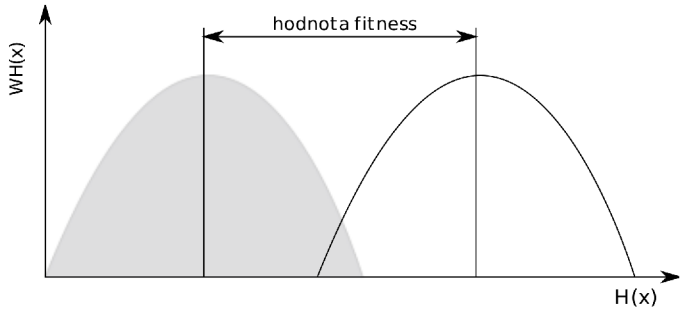
$$H(x) = \frac{a_1}{\sum_{j=1}^{m} a_j} \cdot \frac{h_{i_1}(x)}{2} + \cdots + \frac{a_m}{\sum_{j=1}^{m} a_j} \cdot \frac{h_{i_m}(x)}{2}$$

4.2 Fitness funkce

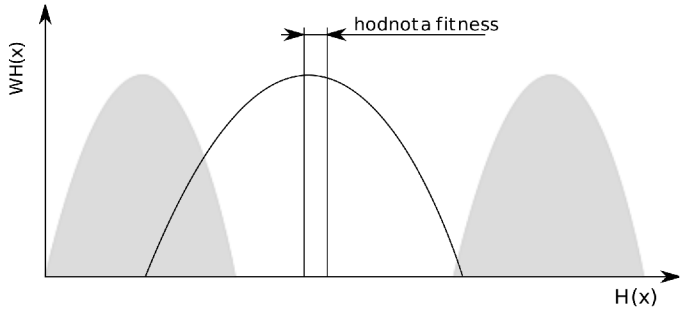
Fitness funkce se snaží vyjádřit kvalitu slabého klasifikátoru, který lze s daným příznakem vytvořit. To lze jednoduše vyjádřit vzdáleností průměrných hodnot obou tříd. Ta je počítána z trénovacích vzorků. Při výpočtu je nutno vzít v úvahu váhy trénovacích vzorků, které určuje AdaBoost. Hodnota příznaku $H(x)$ je pak $WH(x) = H(x) \cdot w(x)$, kde $w(x)$ je váha vzorku x. Celková hodnota fitness při počtech trénovacích vzorků v jednotlivých třídách X_A a X_B se pak spočítá:

$$F = \left| \sum_{x_A=1}^{X_A} \frac{H(x_A)}{WH(x_A)} - \sum_{x_B=1}^{X_B} \frac{H(x_B)}{WH(x_B)} \right|$$

Vyšší fitness tak mají příznaky, které vytvářejí rozložení hodnot podobná obrázku 4.1. Vážené průměry obou tříd jsou dost rozdílné a fitness takovýchto příznaků bude vysoká. Dobrý klasifikátor lze však vytvořit i z rozložení na obrázku 4.2, ale průměrné hodnoty se příliš neliší. Příznaky vytvářející taková rozložení mají nízkou fitness a s největší pravděpodobností zaniknou.



Obrázek 4.1: Znázornění vhodného rozložení hodnot pro fitness



Obrázek 4.2: Znázornění nevhodného rozložení hodnot pro fitness

4.3 Průběh evolučního algoritmu

Počáteční generace je generována náhodně. Je vhodné složit počáteční populaci z více částí, které jsou vytvářeny různým způsobem. Například tak, že část bude obsahovat

vektory jen s kladnými složkami a druhá se zápornými. Díky tomuto postupu je větší pravděpodobnost, že se využijí dobré vlastnosti z obou rozsahů.

Výběr rodičů zajišťuje ruleta. Ta je posána v kapitole 3.3. Počet rodičů je stejný jako velikost populace. Díky ruletě se však někteří mohou zařadit mezi rodiče víckrát a jiní vůbec. Záleží na jejich fitness.

Křížení se účastní rodiče po dvou podle toho, jak byli vybráni ruletou. Na pravděpodobnosti křížení závísí, jestli se rodiče budou křížit. V tom případě se náhodně vybere ze způsobů uvedených v seznamu níže . Pokud ke křížení nedojde, rodiče se beze změny zařadí mezi potomky a mohou mutovat. Použity jsou následující typy křížení z kapitoly 3.1:

- Jednobodová varianta segmentového křížení.
- Křížení lineární kombinací rodičů. Pro každou složku rodičovských vektorů r_1 a r_2 je generováno náhodné číslo x s rovnoměrným rozložením v intervalu $\langle 0, 1\rangle$. Potomci pak mají odpovídající složky $p_1 = xr_1 + (1 - x)r_2$ a $p_2 = (x - 1)r_1 + xr_2$.
- Také se jedná o křížení lineární kombinací. Jediný rozdíl je, že x je generováno jen jednou a použito pro všechny složky rodičovských vektorů.

Mutace může probíhat dvěma způsoby. Jeden využívá čísel v rovnoměrném a druhý v normálním rozložení. Každý pokrývá jiný rozsah. To zajišťuje výběr z širokého množství příznaků a zároveň zabraňuje vynechání těch nejbližších, zvlášť pokud má jedinec vysokou fitness, vydrží více generací a jeho potomci podstoupí mutaci víckrát. Pravděpodonost mutace určuje, zda potomek bude mutovat. Pokud k mutaci dojde, tak se náhodně určí, který způsob se použije.

- Uniformní mutace upraví každou složku vektoru o hodnotu generovanou s rovnoměrným rozložením v určitém rozsahu. Měla by sloužit k výběru nejbližších příznaků.
- Normální mutace upraví každou složku vektoru o hodnotu generovanou s gaussovým rozložením a určitou standardní odchylkou. Rozsah mutace by měl být výrazně větší než u předchozího způsobu. Díky většímu rozsahu vznikají jedinci, ke kterým bychom se pomocí prvního způsobu nedostali.

Následující generace se skládá z potomků a předchozí generace. Přežije lepší polovina z rodičů a lepší polovina z potomků. Ostatní zanikají.

Ukončovací podmínky jsou vyhodnocovány před výběrem rodičů. Základní možností je ukončení po dosažení určitého počtu generací. Dále je možno kontrolovat počet vyhodnocení fitness funkce. Algoritmus se ukončí pouze v místě, kde se podmínka kontroluje. To znamená, že fitness může být vyhodnocena vícekrát, aby se dokončila celá generace.

Kapitola 5

Implementace

Evoluční algoritmus popsaný v kapitole 4 je přidán jako další způsob generování příznaků pro AdaBoost do systému pro výzkum klasifikátorů popsaného v [3]. Tento K implementaci je využita knihovna `Evolving Objects` (ke stažení na adrese `http://eodev.sourceforge.net/`), která je dostupná pod licencí GLPL [1]. Zdrojové kódy jsou na přiloženém CD. Jedná se o soubory `GAHaarFeatures.cpp` a `GAHaarFeatures.h` v adresáři `program/src/features/`.

Třída `TGAHaarFeatures` je stěžejním bodem celé implementace. Její instance slouží k trénování klasifikátoru. Ten spustí evoluční algoritmus a z výsledné populace si vybere nejvhodnějšího jedince. Jedinec vybraný klasifikátorem je uložen pomocí instance třídy `TGAHaarFeature` do xml, aby odtud mohl být načten a použit jinou instancí třídy při vlastní klasifikaci.

5.1 Třída `TGAHaarFeatures`

Konstruktor načte konfiguraci z xml souboru a vytvoří Haarovy příznaky.

Metoda `initialize` je volána jednou pro každou iteraci AdaBoostu. Vytvoří vyrovnávací paměť pro příznaky (viz 5.3). Vygeneruje počáteční populaci, normalizuje všechny jedince (viz 4.1), spustí evoluční algoritmus (zavolá metodu `run`). Potom znovu normalizuje všechny jedince. To je nutné z toho důvodu, že v průběhu evoluce se normalizace sice provádí při výpočtu fitness, ale jedinci si zachovávají hodnotu. Na konci jsou k dispozici příznaky pro AdaBoost.

Metoda `evaluate` vrací hodnotu určitého příznaku. Je využívána, když AdaBoost vyhledává vhodný příznak.

Metoda `getFeature` vytvoří instanci třídy `TGAHaarFeature` a vrátí ukazatel na tuto instanci.

5.2 Evoluční algoritmus

Implementace evolučního algoritmu se nachází v metodě `run` třídy `TGAHaarFeatures`. Kvůli struktuře knihovny `Evolving Objects` je fitness funkce nezávislá na třídě

[1] Text GNU Lesser General Public License je dostupný na `http://www.gnu.org/licenses/lgpl.html`

`TGAHaarFeatures` a využívá globálního ukazatele na vyrovnávací paměť (viz 5.3). To by mohlo způsobit problémy při vytvoření více instancí třídy `TGAHaarFeatures`.

Počáteční generace je generována v rámci metody `initialize`. První polovina populace obsahuje vektory složené z čísel v intervalu $\langle 0, 1\rangle$ a druhá v intervalu $\langle -1, 0\rangle$. Hodnoty jsou generovány náhodně s rovnoměrným rozložením. Velikost populace se nemění. Nastavuje se pomocí xml atributu `populationSize`.

Křížení je reprezentováno třemi různými způsoby, které jsou popsány v 4.3. Pravděpodobnost křížení se nastavuje xml atributem `crossoverProbability`. Kříženi lineární kombinací představují třídy `eoSegmentCrossover` a `eoHypercubeCrossover` z knihovny `Evolving Objects`. Každý z těchto způsobů má pravděpodobnost, že bude zvolen 40 %. Zbylých 20 % zbývá na segmentové křížení. V `Evolving Objects` není obsažena reálná varianta tohoto křížení a bylo nutno ji vytvořit. Jedná se o třídu `eo1PtRealSegmentCrossover`.

Mutace je představována třídami `eoUniformMutation` a `eoNormalMutation`. Pravděpodobnost mutace je nastavována xml atributem `mutationProbability`. Pokud k mutaci dojde, vybere se náhodně z těchto možností:

- `eoUniformMutation` změní každou složku jedince o číslo generované s rovnoměrným rozložením v rozsahu $\langle -0.00005, 0.00005\rangle$, což při počtu Haarových příznaků asi 180 000 odpovídá výběru z přibližně 20 příznaků.
- `eoNormalMutation` mění složky jedince o hodnotu generovanou s normálním rozložením. Střed je daná složka vektoru a standardní odchylka je 0.01.

Ukončovací podmínky jsou nastavitelné v konfiguračním xml souboru pomocí atributů:

- `maximalGenerations` určuje maximální počet generací. Pokud není definován, je implicitně nastaven na 1 000.
- `maximalEvaluatedFitness` zastaví evoluční algoritmus po určitém počtu vyhodnocení fitness funkce. V okamžiku, kdy je dosaženo stanoveného počtu, se algoritmus neukončí, ale pokračuje až do kontroly podmínek (viz obrázek 3.1).

Pokud se má kontrolovat jen počet vyhodnocení fitness funkce, je třeba nastavit počet generací na 0. Jinak se použije implicitní hodnota a algoritmus skončí po jejím dosažení.

5.3 Vyrovnávací paměť

Výpočet fitness funkce je zrychlen použitím vyrovnávací paměti. Ta uchovává hodnoty naposledy použitých Haarových příznaků pro všechny trénovací vzorky. Počet uchovávaných příznaků se nastavuje v xml souboru volbou `cacheSize`. Paměťová náročnost je dána:

$$\textit{počet trénovacích vzorků} \times \texttt{cacheSize} \times \texttt{sizeof(double)} + \textit{konstantní režie}$$

Do vyrovnávací paměti se přistupuje pomocí dvou indexů. Jeden index určuje příznak a druhý hodnotu tohoto příznaku pro konkrétní trénovací vzorek. Pokud hodnoty pro požadovaný příznak ve vyrovnávací paměti nejsou, tak se porovná počet uložených příznaků s `cacheSize`. Pokud by další příznak tuto hodnotu překročil, smaže se ten nejstarší. Potom jsou spočítány a uloženy do paměti hodnoty příznaku pro všechny trénovací vzorky.

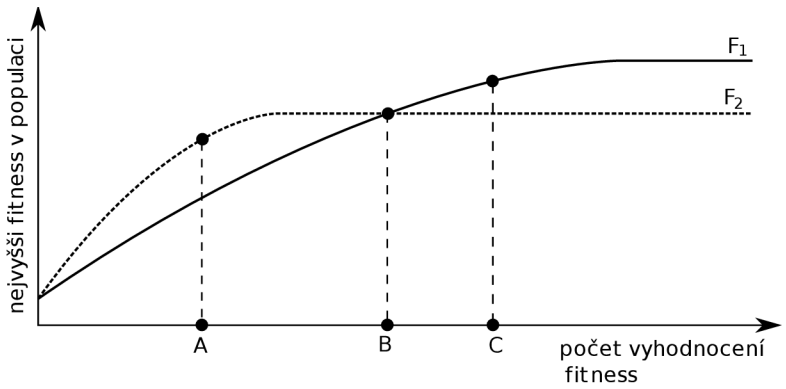
Kapitola 6

Testy

Po dokončení implementace a odladění zdrojového kódu byly provedeny testy. Jejich cílem bylo zjistit vlastnosti lineární kombinace Haarových příznaků a porovnat je se samostatnými Haarovými příznaky. Pro každý z těchto přístupů je natrénován klasifikátor. Poté jsou ověřeny jeho vlastnosti na testovacích datech. Jsou použita stejná trénovací a testovací data pro oba klasifikátory. Prostředky pro trénování a testování již byly zahrnuty v projektu zmiňovaném v kapitolách 1 a 5.

6.1 Parametry evolučního algoritmu

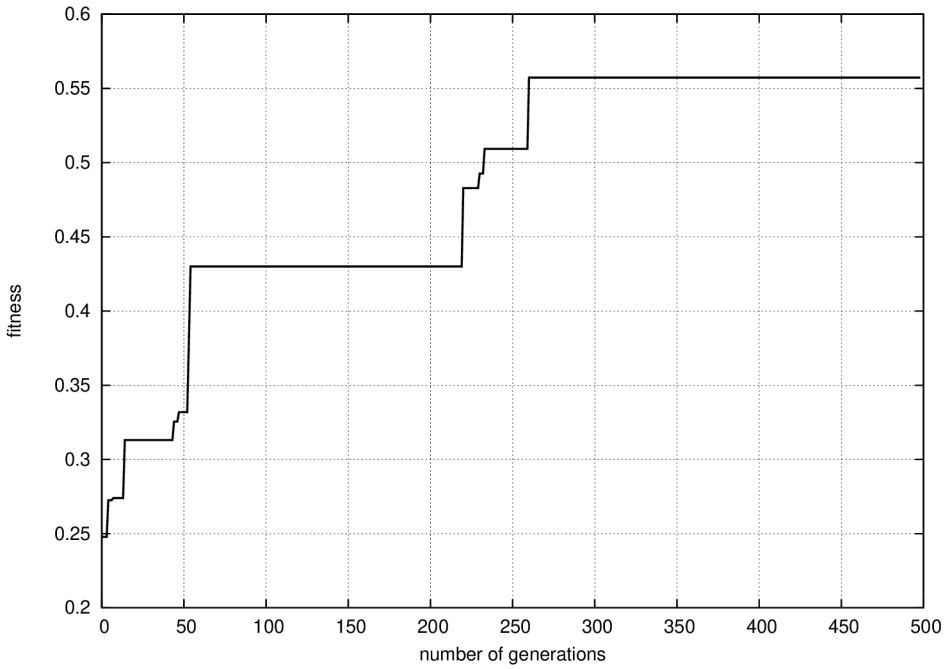
Metrikou evolučních algoritmů je počet vyhodnocení fitness funkce. Růst hodnoty fitness však není tomuto počtu úměrný. Když máme dostatek času a výpočetních prostředků, můžeme nechat fitness funkci vyhodnotit víckrát. Nevadí nám pomalejší růst, pokud je na konci dosaženo lepší hodnoty. Pokud chceme mít výsledek rychle, potřebujeme zpočátku strmější růst, i když by se fitness ustálila na nižší hodnotě než v předchozím případě.



Obrázek 6.1: Příklad průběhu fitness

Na obrázku 6.1 vidíme příklad takovýchto dvou průběhů. V bodě A má vyšší hodnotu průběh F_2 a v bodě C průběh F_1. Pokud potřebujeme výpočet ukončit dříve než v bodě B, použijeme průběh F_2. Když je možno ho ukončit až za bodem B, zvolíme F_1. Požadovanému chování je pak nutno přizpůsobit konfiguraci.

Obrázek 6.1 je jen ilustrační. Reálný průběh fitness funkce vypadá většinou jako na obrázku 6.2, který byl získán při ladění programu. Abychom získali takovéto průběhy, je



Obrázek 6.2: Příklad průběhu fitness

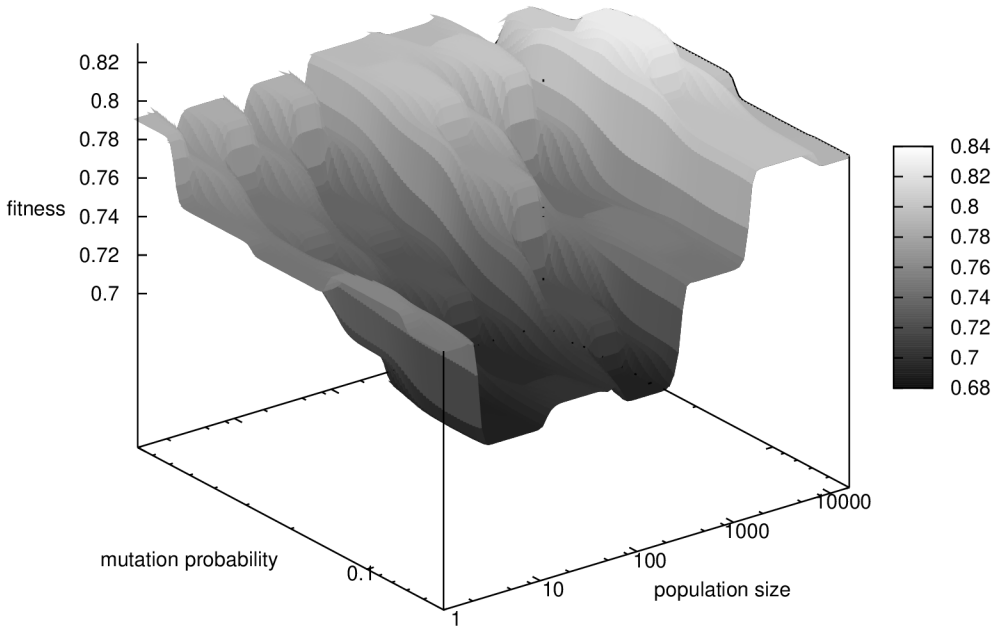
třeba sledovat chování fitness pro různé hodnoty parametrů. V konfiguračním xml souboru lze nastavit následující:

- `mutationProbability` – pravděpodobnost mutace
- `crossoverProbability` – pravděpodobnost křížení
- `populationSize` – velikost populace
- `haarFeaturesCount` – počet Haarových příznaků v lineární kombinaci

Možných kombinací těchto parametrů je příliš mnoho. Během ladění algoritmu se ukázala jako nejvhodnější lineární kombinace dvou Haarových příznaků a pravděpodobnost křížení můžeme zvolit 0.8. Ukázalo se, že nejvíce je růst fitness funkce ovlivněn pravděpodobností mutace a velikostí populace.

Proto jsme sledovali vývoj fitness funkce pro velikosti populace 1, 5, 25, 125, 625, 3 125, 15 625 a pravděpodobnosti mutace 0.05, 0.1, 0.2, 0.4, 0.8. Hodnotu fitness jsme zjišťovali při počtu vyhodnocení fitness funkce 100, 200, 400, 800, 1 600, 3 200, 6 400, 12 800, 25 600, 51 200, 102 400, 204 800, 404 800 a 600 000. Pro každou kombinaci těchto hodnot proběhla evoluce dvanáctkrát, vždy pro první iteraci AdaBoostu. Na obrázku 6.3 je rozložení fitness při počtu vyhodnocení 404 800. Je tam vidět, že se fitness víceméně zvyšuje s větší populací a vyšší pravděpodobností mutace.

Nejvyšší dosažená hodnota fitness byla 0.9141. Z hodnot získaných ze všech dvanácti běhů byly spočítány aritmetické průměry. Nejvyšší průměrné hodnoty fitness pro všechny



Obrázek 6.3: Hodnoty fitness funkce po 404 800 vyhodnoceních fitness funkce

zaznamenané počty vyhodnocení jsou v tabulce 6.1. Interval spolehlivosti (confidence interval) byl spočítán podle [9]. Byla zvolena úroveň věrohodnosti 95 %.

Počet vyhodnocení fitness funkce	Velikost populace	Pravděpodobnost mutace	Fitness	Interval spolehlivosti
100	1	0.2	0.4182	± 0.0554
200	1	0.1	0.4737	± 0.0440
400	1	0.2	0.5160	± 0.0577
800	1	0.2	0.5924	± 0.0245
1 600	1	0.2	0.5968	± 0.0226
3 200	125	0.4	0.6283	± 0.0385
6 400	125	0.4	0.6804	± 0.0430
12 800	125	0.4	0.7021	± 0.0379
25 600	625	0.1	0.7076	± 0.0353
51 200	625	0.4	0.7298	± 0.0336
102 400	3 125	0.05	0.7601	± 0.0261
204 800	3 125	0.2	0.8115	± 0.0273
404 800	3 125	0.2	0.8269	± 0.0271
600 000	3 125	0.2	0.8275	± 0.0271

Tabulka 6.1: Nejlepší hodnoty fitness pro daný počet vyhodnocení fitness funkce

Při testech budeme chtít dosáhnout 90 % maximální dosažené hodnoty fitness funkce:

$$0.9141 \cdot 0.9 = 0.8227$$

Z tabulky 6.1 vybereme nejbližší vyšší hodnotu, což odpovídá počtu vyhodnocení fitness funkce 404 800, pravděpodobnosti mutace 0.2 a velikosti populace 3 125. Tyto hodnoty

nastavíme v konfiguračním xml souboru, který bude použit pro testování klasifikátoru

6.2 Trénovací a testovací data

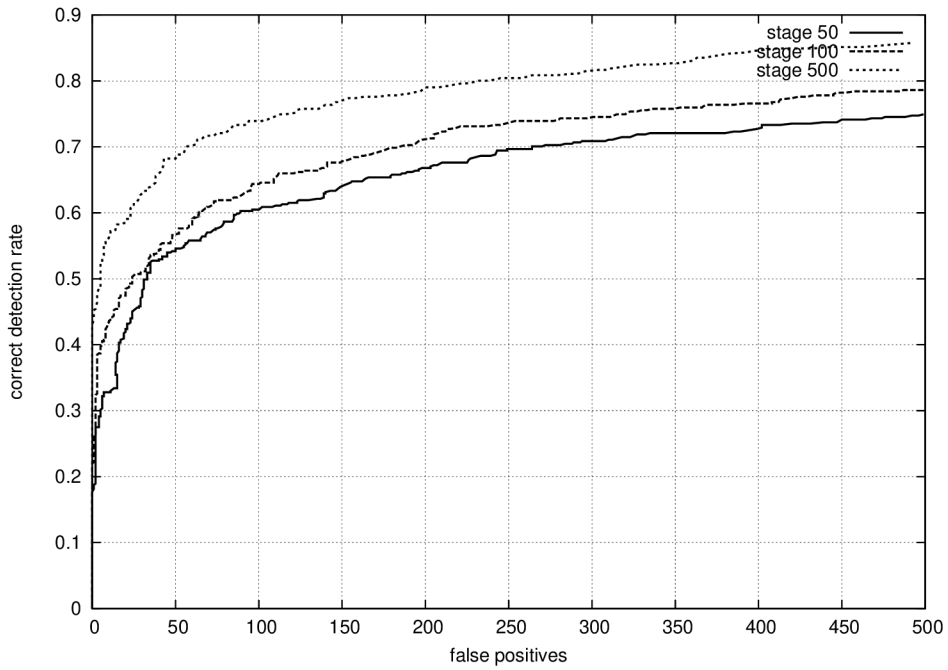
K trénování byly použity obrázky obličejů v rozlišení 24×24 pixelů a stupních šedi. Tyto obrázky byly sesbírány z webových stránek a ručně anotovány. V trénovací sadě se jich nachází 10 000. Obrázky pozadí byly vytvořeny jako náhodné výřezy (také velikosti 24×24 pixelů) z 4 000 obrázků, které neobsahují obličeje. Celkový počet těchto výřezů je 250 000 000.

K testování byla použita sada dat vytvořená na Carnegie Mellon University a Massachusetts Institute of Technology (MIT+CMU dataset). Tato sada obsahuje 114 obrázků, na kterých je přibližně 500 obličejů. Tyto obrázku jsou skenovány po výřezech o velikosti 24×24 pixelů, kterých je celkem 17 000 000.

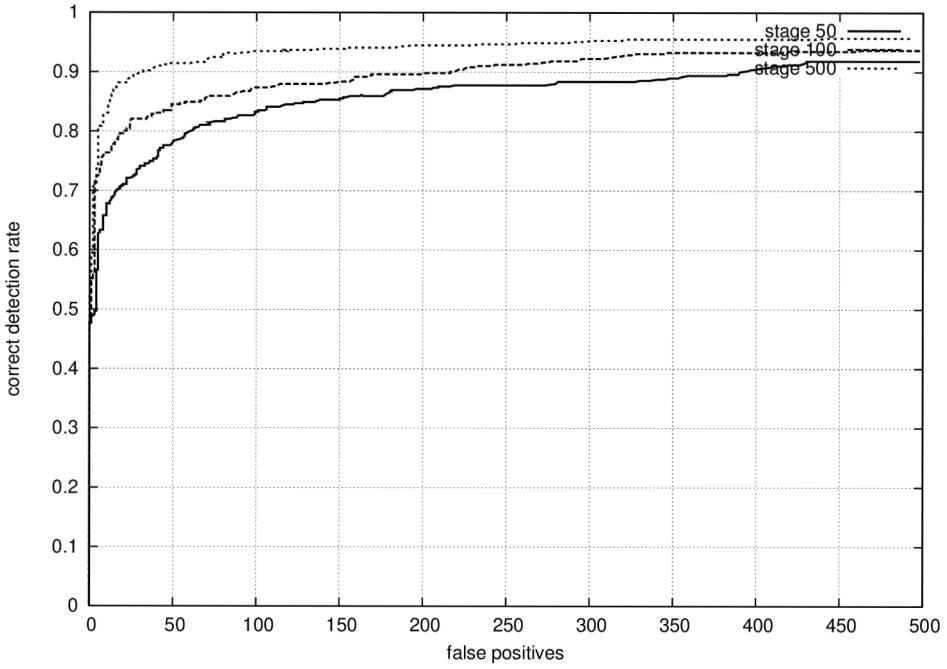
6.3 ROC křivka

Jak již bylo uvedeno v kapitole 2.1, jednou ze sledovaných vlastností klasifikátoru je počet vzorků, které byly do třídy chybně přiřazeny. ROC (receiver operating characteristic) křivka znázorňuje poměr vzorků správně přiřazených do třídy (true positive) vůči všem vzorkům, které byly do třídy přiřazeny (true positive i false positive).

Na obrázku 6.4 jsou ROC křivky pro klasifikátor, který používá lineární kombinaci Haarových příznaků. Na obrázku 6.5 pro samostané Haarovy příznaky. Křivky jsou vždy pro silný klasifikátor složený z 50, 100 a 500 slabých klasifikátorů.



Obrázek 6.4: ROC křivka lineární kombinace Haarových příznaků



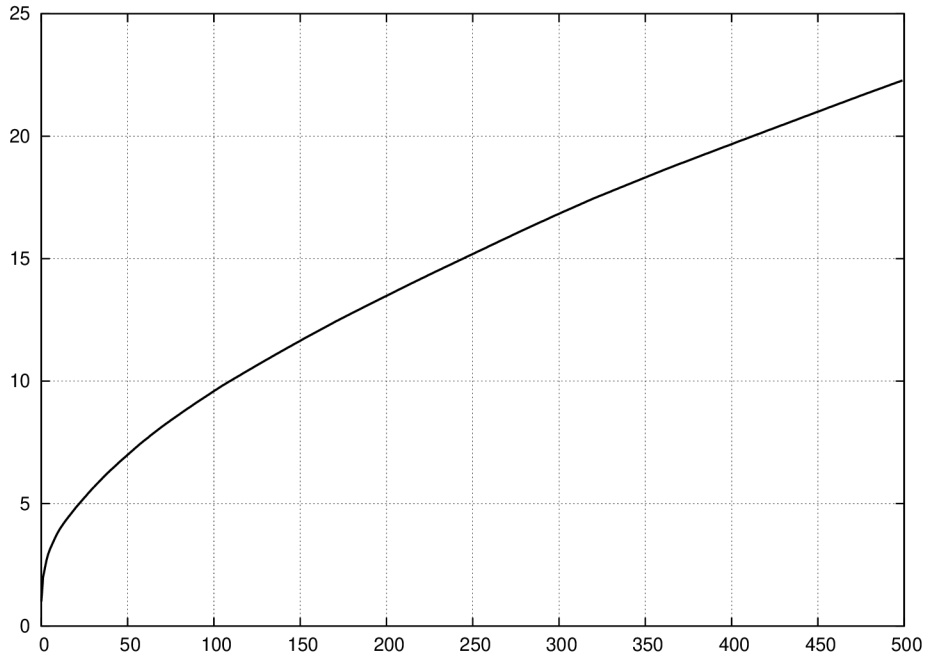
Obrázek 6.5: ROC křivka samostatných Haarových příznaků

6.4 Rychlost klasifikátoru

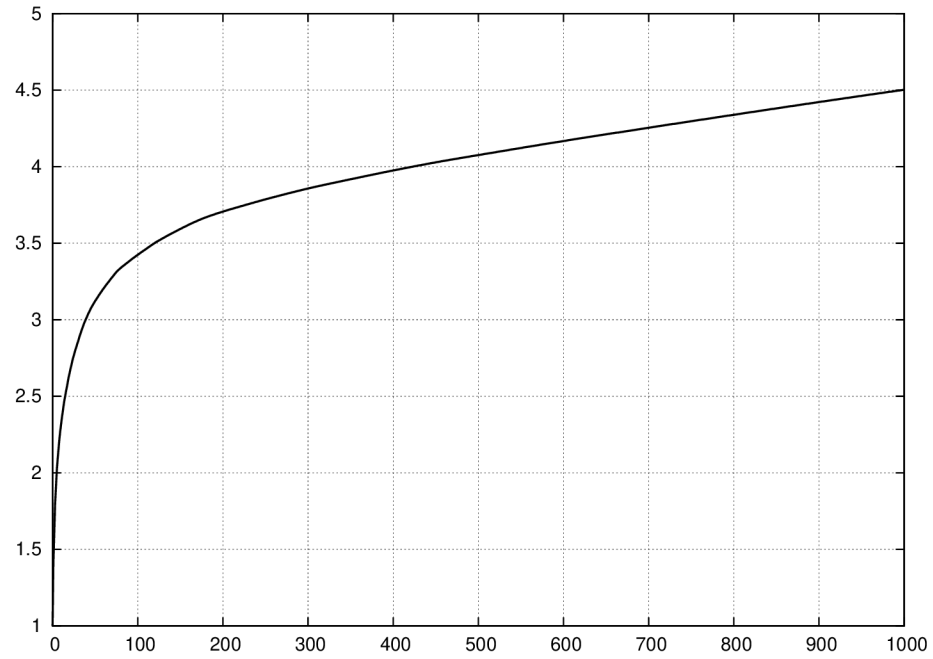
Rychlostí klasifikátoru je myšlen počet prvků kaskády klasifikátorů (viz kapitola 2.5), který je využit při klasifikaci. Celou kaskádou jsou zpracovány jen části obrazu, které patří do detekované třídy. Na obrázcích 6.6 a 6.7 je znázorněn průměrný počet využitých klasifikátorů ve vztahu k celkovému počtu klasifikátorů v kaskádě. Opět je vidět rozdíl mezi lineární kombinací a samostatnými Haarovými příznaky.

6.5 Výsledky

Z grafů popsaných v 6.3 a 6.4 je zřejmé, že lineární kombinace nedosahuje kvalit Haarových příznaků. Příliš mnoho vzorků je chybně zařazeno mezi obličeje. S tím je spojená i nižší průměrná rychlost, protože celou kaskádou prošlo více vzorků, než mělo.



Obrázek 6.6: Průměrná rychlost lineární kombinace Haarových příznaků



Obrázek 6.7: Průměrná rychlost lineární kombinace Haarových příznaků

Kapitola 7

Závěr

Z kapitoly 6 vyplývá, že lineární kombinace Haarových příznaků nesplnila očekávání, která do ní byla vložena, přesto výsledky vypadají docela slibně. Nesmíme zapomínat, že se jedná o rané stádium vývoje. Tento postup má určitý potenciál a pokud se na něm bude dále pracovat, tak snad dosáhneme lepších výsledků. Nyní se postupně podíváme na nedostatky zjištěné při vývoji a testování.

7.1 Genom, mutace a křížení

V současné implementaci jsou Haarovy příznaky generovány a číslovány postupně. Začne se v levém horním rohu obrazu a po řádcích se postupuje až do pravého dolního. Sousední indexy pak mají Haarovy příznaky, které jsou vedle sebe na řádku. Dále sousedí ty, co jsou na konci jednoho řádku s těmi, které jsou na začátku řádku následujícího. Pokud je použito více typů Haarových příznaků (viz obrázek 2.2), dochází k dalšímu problému, protože potom sousedí největší příznak předchozího typu s nejmenším příznakem typu následujícího.

Aby křížení a mutace lépe plnily svůj účel, měly by mít sousední indexy Haarovy příznaky, jenž obsahují podobnou informaci. Tím jsou myšleny ty, které sousedí v řádku nebo sloupci a ty, které jsou o něco větší či menší. Navíc by měly sousedit příznaky stejného typu.

Tyto problémy by bylo možno řešit změnou mechanismů křížení a mutace. Složky genomu, které představují Haarovy příznaky, by byly upravovány odlišným způsobem než zbytek. Také by mohly být reprezentovány celými čísly.

Další nedostatek je způsoben koeficienty $a_1 \dots a_m$ (viz kapitola 4.1). Pokud mají nevhodné hodnoty, mohou způsobit, že i kvalitní kombinace Haarových příznaků zahyne. Tomu by se dalo zabránit úplným vyřazením těchto koeficientů z genomu. Jejich hodnoty pro konkrétní kombinaci Haarových příznaků by pak mohly být vypočítány nějakou statistickou metodou (PCA, LDA).

Existuje mnoho metod pro generování příznaků. Všechny mají své výhody i nevýhody. Nemuselo by se vybírat jen z Haarových příznaků, ale použít i jiné (LDA, NLDA atd.). Každá z těchto metod by však musela mít vlastní postupy pro mutaci. Takovýto systém by pak byl použitelný i pro jiné účely než zpracování obrazu.

7.2 Fitness

Z výsledků testů vyplývá, že příliš mnoho vzorků, které do detekované třídy nepatří, je do této třídy zařazeno. To je pravděpodobně způsobeno tím, že hodnota fitness funkce odpovídá kvalitě slabého klasifikátoru, který je z jedince natrénován jen do určité míry. Řešením by byla fitness funkce, jenž by této kvalitě odpovídala lépe. Nejlépe nahradit stávající fitness funkci úspěšností slabého klasifikátoru (viz kapitola 2.3).

7.3 Shrnutí

Předchozí postřehy se dají shrnout do určitého plánu pro další vývoj. Za vyzkoušení by stály následující úpravy:

- Genom bude tvořen pouze indexy příznaků různých typů.
- Každý typ příznaků bude mít vlastní postup mutace a bude moci mutovat jen v rámci svého typu.
- Počáteční populace bude složena z jedinců, kteří budou obsahovat jen jeden typ příznaků.
- Bude použito jen segmentové křížení. To spolu se složením počáteční populace zajistí kombinaci toho nejlepšího ze všech typů příznaků.
- Koeficienty pro lineární kombinaci budou získávány pomocí PCA nebo LDA.
- Jako fitness funkce bude použita úspěšnost slabého klasifikátoru.

Předpokládám, že po zahrnutí těchto úprav bychom mohli dosáhnout lepších výsledků než pomocí současného postupu. Možná i lepších než za použití Haarových příznaků. Pokud by tato očekávání byla splněna, získali bychom univerzální a rozšiřitelný systém, jenž by byl použitelný pro různé typy dat.

Literatura

[1] Duda, R. O.; Hart, P. E.; Stork, D. G.: *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2000, ISBN 0471056693.

[2] Freund, Y.; Schapire, R.: A short introduction to boosting. 1999, [Online]. URL `http://citeseer.ist.psu.edu/freund99short.html`

[3] Hradiš, M.: Framework for Research on Detection Classifiers. In *Proceedings of Spring Conference on Computer Graphics*, 2008, s. 171–177. URL `http://www.fit.vutbr.cz/research/view_pub.php?id=8608`

[4] Kvasnička, V.; Pospíchal, J.; Tiňo, P.: *Evolučné algoritmy*. STU Bratislava, 2000, ISBN 8022713775.

[5] Schoenauer, M.: Evolving Objects Tutorial. 2002, [Online]. URL `http://eodev.sourceforge.net/eo/tutorial/html/eoTutorial.html`

[6] Treptow, A.; Zell, A.: Combining Adaboost Learning and Evolutionary Search to Select Features for Real-Time Object Detection. 2004. URL `http://citeseer.ist.psu.edu/646131.html`

[7] Viola, P.; Jones, M.: Rapid object detection using a boosted cascade of simple features. 2001, [Online]. URL `http://citeseer.ist.psu.edu/viola01rapid.html`

[8] Wikipedia: AdaBoost — Wikipedia, The Free Encyclopedia. 2007, [Online]. URL `http://en.wikipedia.org/w/index.php?title=AdaBoost&oldid=176678328`

[9] Wikipedia: Student's t-distribution — Wikipedia, The Free Encyclopedia. 2008, [Online]. URL `http://en.wikipedia.org/w/index.php?title=Student%27s_t-distribution&oldid=210886302`