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

ROZPOZNÁNÍ RUČNĚ PSANÝCH ČÍSLIC

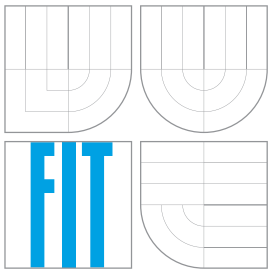
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MARTIN DOBROVOLNÝ

BRNO 2009

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

ROZPOZNÁNÍ RUČNĚ PSANÝCH ČÍSLIC

RECOGNITION OF HANDWRITTEN DIGITS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MARTIN DOBROVOLNÝ
AUTHOR

VEDOUCÍ PRÁCE — Ing. ADAM HEROUT, Ph.D.
SUPERVISOR

BRNO 2009

Abstrakt

Rozpoznávání ručně psaných číslic je součástí oboru počítačovho vidění a také problematikou, se kterou se v dnešní době nelze vypořádat na 100 %. Tato práce popisuje metodu rozpoznávání ručně psaných číslic založenou na charakteristických rysech a rozhodovacích stromech. Metody takto založené jsou známé jejich dlouhou dobou strojového učení a vysokou rychlostí při vyhodnocování. Tato metoda užívá relativních úhlů mezi charakteristickými oblastmi, což jí umožňuje jistou míru volnosti při porovnávání podobných vzorků a také schopnost odolávat vůči nelineárním deformacím písma.

Abstract

Recognition of handwritten digits is one of computer vision problematics that can not be solved with 100 % success these days. This document describes a method for handwritten digits recognizing based on shape features and randomized tree classifiers. These methods are known for their long time machine learning and quick characters recognizing. This method is due to use of relative angles among key locations and is nearly invariant to substantial affine and nonlinear defarmations.

Klíčová slova

Rozpoznání ručně psaných číslic, tag, tagy, rozhodovací stromy, tagovací stromy, MNIST, matice záměn.

Keywords

Handwritten digits recognition, tag, tags, randomized tree, tag tree, MNIST, confusion matrix.

Citace

Martin Dobrovolný: Rozpoznání ručně psaných číslic, bakalářská práce, Brno, FIT VUT v Brně, 2009

Rozpoznání ručně psaných číslic

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Adama Herouta Ph.D.. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Martin Dobrovolný
19.5

Poděkování

Děkuji vedoucímu práce Ing. A. Heroutovi Ph.D. za správné vedení, dobré rady, trpělivost, ochotu, zpřístupnění výpočetního clusteru a mnohé další.

Obsah

Kapitola 1

Úvod

Počítačové rozpoznávání je neodmyslitelnou součástí oboru počítačové grafiky, přesněji je součástí odvětví počítačového vidění. V dnešním moderním světě, který je plný ručně psaných informací, stoupá poptávka po automatizaci jejich zpracování. Ale dříve, než je možné tyto informace zpracovat, je nutné je do počítače nějakým způsobem vložit. Ruční opisování je sice nejjednodušší cestou k cíli, avšak jistě ne tou nejrychlejší a tudíž i nejlevnější.

Tato práce se zabývá rozpoznáváním ručně psaných číslic. Využití tohoto specializovaného typu rozpoznávání není velké, ale tam, kde si najde uplatnění, dokáže značně urychlit práci, nebo usnadnit rutinní činnost. Nejvhodnější využití bude tam, kde se používají vyplňovací formuláře. Toto by se dalo využít například pro scanovací formuláře na testy, kde žák vyplňuje svoje osobní číslo. Jako další využití si lze představit stroj na poště, který třídí dopisní obálky podle PSČ. V dnešní době se také rozmáhají mobilní aplikace využívající fotoaparátu zabudovaného v zařízení. Tímto směrem se otevíra další oblast využití. Můžeme si představit aplikaci, která pomocí fotoaparátu přečte číslo z vizitky, papíru a nabídne uživateli jeho využití.

Práce je založena na článku [2]. Článek abstraktně popisuje jeden konkrétní algoritmus, proto se zde budu zabývat výhradně tímto jediným algoritmem. Mým úkolem bylo tento algoritmus prozkoumat, implementovat a otestovat. Teoretickým popisem implementovaného algoritmu se budeme zabývat v kapitole 2 a v následujících podkapitolách. Dále si rozebereme implementaci v kapitole 3. Nakonec popisuji metodiku testování a dosažené výsledky v kapitole 4.

Dnešní komerčně využívané aplikace pro rozpoznávání znaků se zakládají na principu neuronových sítí, Markovských modelů, nebo stromových klasifikátorů. Mnou implementovaný algoritmus využívá stromových klasifikátorů. Ty jsou ve srovnání s ostatními metodami časově náročné při trénování, avšak dosahují vysoké rychlosti při rozpoznávání.

Počítačové rozpoznávání je problémem, který nelze vyřešit na 100%. Každý člověk má jiný rukopis a každý člověk může jeden znak interpretovat různým způsobem. O tomto problému se více dozvíme v sekci 4.1. Obecně lze říci, že počítač rychlostí rozpoznávání ručně psaných číslic snadno předčí člověka a svou chybovostí se mu může hodně přiblížit.

Kapitola 2

Rozpoznávání ručně psaných číslic pomocí rozhodovacích stromů

Úkolem rozpoznávání ručně psaných číslic je klasifikace testovaného vzorku na základě statistických informací. Implementovaný algoritmus tyto informace získává ve fázi trénování, tato fáze je časově velmi náročná. Trénování může trvat několik hodin, ale i dní. Záleží na výkonu a možnostech použitého počítače. Konkrétní čísla naleznete v kapitole 4 - Testy. Výhodou tohoto algoritmu je rychlost rozpoznávání a univerzálnost. Rychlost rozpoznávání je vysoká, vzhledem k tomu, že při testování není potřeba a ani možnost program dále učit. Algoritmus je univerzální vzhledem k možnostem rozpoznávat nejen ručně psaná čísla. Je možné natrénovat například písmena, geometrické tvary a podobně. Efektivita bude ovšem záviset na kvalitě trénovací databáze a úpravách pro daný úkol. Dle [2] lze algoritmus využít i k rozpoznávání 3D objektů.

Nyní se podívejme, jakým způsobem algoritmus funguje. Nejprve jsou získány informace o každém trénovaném vzorku a to tzv. význačné body (viz. sekce 2.1). Tyto body nám poskytují informaci o tvaru trénovaného vzorku. Význačné body jsou dále roztříděny tagovacími stromy (viz. sekce 2.2). Tagovací stromy význačné body rozdělí do skupin vzájemně podobných význačných bodů. Od této fáze jsou význačné body nazývány tagy. Následuje vlastní trénování rozhodovacích stromů. Rozhodovací stromy se budují pomocí informací získaných z význačných bodů a na základě podobnosti mezi vzorky z trénovací databáze (viz. sekce 2.3). Rozhodovací stromy nám poskytují statistické informace a skupinu pravidel pro vyhodnocení testovaných vzorků. Rozpoznávání ručně psaných čísel (viz. sekce 2.4) se provádí na základě získaných informací z více rozhodovacích stromů současně, což mimo jiné ve výsledku umožňuje dosáhnout statisticky lepších výsledků.

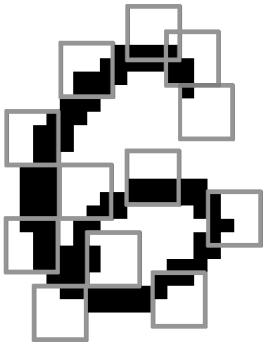
Informace o algoritmu popisovaném v této kapitole byly čerpány ve zdroji [2].

2.1 Hledání význačných bodů

Prvním krokem popisovaného algoritmu je získáni tzv. význačných bodů. Každý tento bod nám poskytuje částečnou informaci o tvaru daného vzorku. Význačný bod je malá oblast v obrázku, která má nějakou vypovídací hodnotu. Tyto oblasti se tedy nacházejí na okrajích čar.

2.1.1 Jak najít význačné body

Aby měl význačný bod požadovanou vypovídací hodnotu, je nutné aby v jeho malé oblasti docházelo ke změně barvy. Vzhledem k tomu že pracujeme s dvoubarevnou, černobílou databází, požadujeme, aby alespoň jeden pixel význačného bodu měl jinou barvu než ostatní pixely v této oblasti. Ilustrace na obrázku 2.1.



Obrázek 2.1: Příklad nalezených význačných bodů

Rozměry této malé oblasti mohou být různé. Závisí na rozlišení obrázku, avšak na jednom obrázku jsou vždy stejné. Pro naše účely byly zvoleny rozměry význačného bodu 4 x 4 px. Tyto rozměry byly ve zdroji [2] považovány za optimální.

2.1.2 Počty nalezených význačných bodů

Počty nalezených význačných bodů se liší. Jak již bylo řečeno záleží na rozlišení, ale také na typu číslice a tloušťce čáry. V tabulce 2.1 vidíme průměrný počet nalezených význačných bodů v jednom vzorku. Informace byly získány nad trénovací databází MNIST (více o databázi v sekci 4.1).

Typ číslice	0	1	2	3	4	5	6	7	8	9
Počet význ. bodů	299	145	270	266	231	249	251	205	261	222

Tabulka 2.1: Tabuka průměrného počtu nalezených význačných bodů daných číslicích

2.2 Tagovací stromy

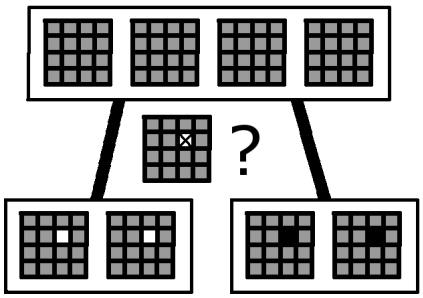
Po získání dostatečného počtu význačných bodů je potřebujeme rozdělit, potažmo sloučit do skupin, které budeme dále nazývat třídy. Zařazení význačného bodu do patřičné třídy nám ve výsledku poskytuje informaci o podobnosti význačných bodů v konkrétní třídě. Význačný bod zařazený do třídy se poté nazývá tag. Ve skutečnosti je význačný bod součástí více tříd současně. Tento princip a jak se vytvářejí tagovací stromy popisuje následující sekce.

2.2.1 Tvorba tagovacích stromů

Tagovací strom je binární strom. Každý uzel obsahující tagy má 2 syny. Synové obsahují každý část tagů svého otcovského uzlu. Jeden z potomků nemusí obsahovat žádný tag,

avšak tagy obou potomků dohromady vždy tvoří množinu tagů stejnou jako jejich otcovský uzel. Syn uzlu, neobsahující žádný tag, se tak stává prázdným listem stromu.

Podívejme se, jakým způsobem se tagovací strom tvoří. Nejprve vezmeme všechny význačné body a vložíme je do kořenového uzlu tagovacího stromu. Nyní potřebujeme tyto význačné body rozdělit co nejrovnoměrněji do potomků uzlu. To provedeme podle barvy jednoho z pixelů význačných bodů (ilustrace na obrázku 2.2). Tento pixel je vybrán tak, aby rozdělil význačné body otcovského uzlu co nejrovnoměrněji. Jestliže rozměry význačného bodu jsou 4 x 4 px, máme zde na výběr z celkem šestnácti pixelů.



Obrázek 2.2: Ilustrace rozdělení význačných bodů do synů v tagovacím stromu

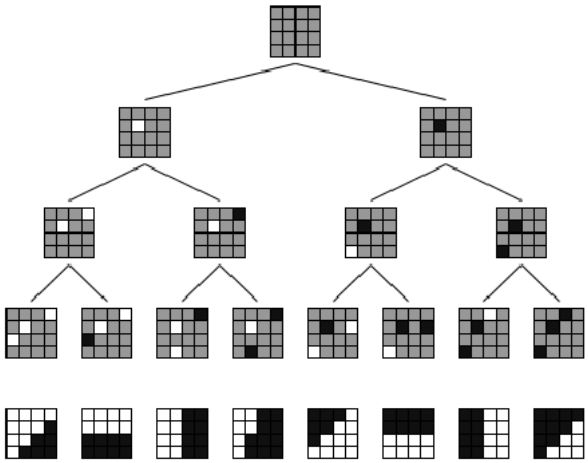
Na obrázku vidíme dělení význačných bodů podle pixelu [2,1] (číslováno od [0,0], označeno křížkem). Šedé pixely jsou pixely o kterých v této hloubce nemáme žádnou informaci. Nevíme zda se jedná o černé, nebo bílé pixely. V levém synovi vidíme význačné body které obsahovaly bílý pixel na pozici dělícího pixelu. V pravém synovi vidíme význačné body obsahující černý pixel na pozici dělícího pixelu.

Tímto způsobem dělíme význačné body dokud každá větev stromu nekončí listem. Uzel stromu se stává listem pokud neobsahuje žádný význačný bod, nebo větev dosahuje požadované hloubky. Omezení hloubkou je zde proto, aby si tagy v listech byly vzájemně podobné, avšak byla mezi jejich tvarem jistá míra tolerance. Míru podobnosti tagů ve třídě nám tedy udává hloubka tagovacího stromu. Pro lepší představu je zde ukázka na obrázku 2.3. Vidíme na něm jako postupně s hloubkou stromu jsou si tagy v jednom uzlu podobnější. V hlooubce 3 tak obsahují 3 stejné tagy. Na obrázku je také ukázka jak může tag v dané třídě vypadat.

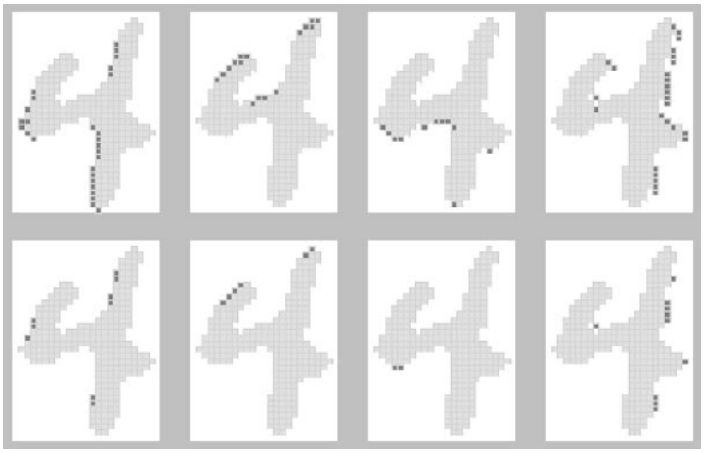
Dle zdroje [2] je vhodná hloubka tagovacího stromu 6. V listech takového stromu už se nacházejí tagy, které jsou si velmi podobné, ale současně je tu jistá míra tolerance vzhledu tagu. Takový strom obsahuje 63 tříd.

2.2.2 Třídy

Jak už napovídá obrázek 2.3 tagy v jedné třídě budou vytvářet v konkrétním vzorku jisté shluky, které se tvoří na okraji čáry vzorku. Toto ilustruje obrázek 2.4. Na obrázku jsou vidět tagy nalezené v různých třídách, ve stejné hloubce tagovacího stromu a ve stejném vzorku. Lze tedy říct že v určité třídě se shromáždí význačné body např. z horních okrajů čar, nebo levých dolních okrajů čar a podobně.



Obrázek 2.3: Budování tagovacího stromu a tagy které se objevují v listech, převzato z [2]



Obrázek 2.4: Tagy nalezené v hloubce 3 tagovacího stromu, převzato z [2]

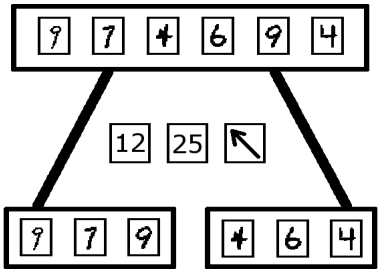
2.3 Rozhodovací stromy

Rozhodovací stromy jak už z názvu vyplívá jsou stromy, které nám v konečné fázi pomáhají rozhodovat o výsledku rozpoznávání. Jedná se tedy o klíčovou součást algoritmu. Rozhodovací strom nám poskytuje dva typy informací. Pravidla a statistické informace.

Pro tvorbu rozhodovacího stromu jsou potřeba tagy, tedy význačné body rozdělené do tříd, a všechny trénované vzorky. Strom je binární a pro snadnější pochopení si jeho syny pojmenujeme Ano-uzel a Ne-uzel. Toto bude vysvětleno dále.

2.3.1 Tvorba rozhodovacích stromů

Rozhodovací strom se vytváří následovně. Nejprve vložíme do kořenového uzlu všechny trénované vzorky. Syny uzlu vytvoříme tak, že vybereme pravidlo (pravidla budou popsána níže) podle kterého vzorky otcovského uzlu rozdělíme do synů (ilustrace na obrázku 2.5). Tento postup opakujeme dokud na koncích všech větví stromu nejsou listy.



Obrázek 2.5: Dělení vzorků do synů podle pravidla

Listy rozhodovacího stromu nám poskytují požadovanou statistickou informaci. Jedná se o kvantitativní rozložení jednotlivých typů číslic v listu, dále jen rozložení.
Uzel se stává listem pokud splňuje alespoň jednu z následujících podmínek:

1. hloubka stromu v daném místě dosahuje maxima
2. uzel obsahuje malý počet vzorků
3. jeden z typů číslic se stal početně dominantním oproti ostatním

Pozn. informace o vlivu nastavení těchto parametrů naleznete v kapitole 4 - Testy.

Všechny tyto parametry se snaží vytvořit strom co nejnižší a současně se snaží co nejvíce oddělit různé typy číslic od sebe. V ideálním případě by rozhodovací strom obsahoval taková pravidla, která by vedla k absolutně čistému oddělení jednotlivých typů číslic od sebe (tj. každý list bude obsahovat jen jeden typ číslice). Ideální rozhodovací strom by také měl co nejmenší možnou hloubku.

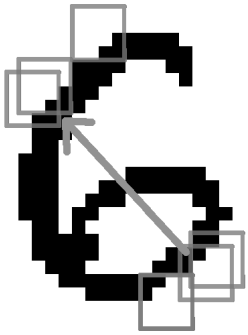
2.3.2 Pravidla

Nyní o pravidlech která rozdělují vzorky otcovského uzlu do synů.
Pravidlo má následující tvar:

[identifikátor počáteční třídy][identifikátor koncové třídy][směr]

Jestliže má pravidlo tento tvar, je jednoduché rozhodnout, zda jej vzorek obsahuje, či nikoliv. Zda vzorek splňuje pravidlo, se můžeme ptát tímto způsobem: „Obsahuje daný vzorek takové dva tagy, aby jeden z nich náležel počáteční třídě, druhý z tagů náležel koncové třídě a současně mezi sebou oba tagy mají daný směr?“. Pokud ano, vzorek může postoupit dále do Ano-uzlu, pokud ne, vzorek postupuje do Ne-uzlu. Toto ilustruje obrázek 2.6.

Pravidlo je tedy vzájemným vztahem dvou tagů. Poslední částí pravidla, o které ještě nic nevíme, je směr. Směr je jeden vybraný směr z osmi možných. Celých 360° jsme si rozdělili na 8 stejných částí. Orientace těchto částí se dá pro lepší představu popsat směry světových stran (S, SV, V, JV, J, JZ, Z, SZ). Rozdělení směrů do osmi kategorií na místo jeho přesného určování nám umožňuje jistou volnost vztahu mezi dvěma tagy. Díky tomu, že vzdálenost mezi tagy nehraje žádnou roli, nám zde vzniká prostor pro podobnost mezi vzorky.



Obrázek 2.6: Obrázek znázorňuje pravidlo (vztah dvou tagů)

2.3.3 Metrika

Nyní k výběru pravidla. Na výběr z pravidel se nám nabízí celkem 62 x 62 x 8 = 30752 pravidel. Pravidla vybíráme podle hodnoty metriky. Při takovémto množství pravidel není možné vypočítat metriku pro všechny. Časová náročnost trénování by byla příliš vysoká.

Metrika pro výběr nejlepšího pravidla:

$$M = K_1 * N * \left(x_\phi - \frac{P}{2N}\right)^2 - K_2 * \left(\sum_{i=0}^{9} (x_i - x_\phi)^2\right) \tag{2.1}$$

Vysvětlivky:

1. N - počet druhů číslic
2. K_1 - konstanta pro vyvažování stromu
3. x_ϕ - průměrný počet vzorků jednotlivých číslic splňující konkrétní pravidlo
4. P - počet všech vzorků v uzlu
5. K_2 - konstanta pro oddělování jednoho typu číslice
6. x_i - počet vzorků i-tého typu číslice splňující pravidlo

Vzorec 2.1 byl vytvořen spoleně s vedoucím práce. V původním článku není o metrice žádná zmínka. V tomto článku [1] je možné se dočíst o postupu tvorby vzorce pro náš účel. Vzorec v článku má ve výsledku velmi podobný tvar k našemu vzorci 2.1. Podobá se první části našeho vzorce, tudíž bere ohled pouze na vyvážení rozhodovacího stromu. Navrhnutá metrika 2.1 se skládá ze dvou částí v závorkách s váhovými koeficienty K_1 a K_2, oddělených znaménkem mínus. Levá část slouží k vytváření vyvážených stromů. Pravá část slouží k oddělování jednoho, či co nejméně typů číslic od ostatních vzorků. Ideální hodnota této metriky je co nejnižší číslo. Testy parametrů lze najít v kapitole 4 - Testy.

2.4 Rozpoznávání ručně psaných číslic

Poslední fází algoritmu je vlastní rozpoznávání ručně psaných číslic. V této fázi bychom již měli mít natrénované rozhodovací stromy, které budeme potřebovat pro vyhodnocení rozpoznávání, a k nim také potřebujeme tagovací stromy, pomocí kterých byly rozhodovací stromy vytvořeny.

2.4.1 Nalezení význačných bodů

Postup pro testování vzorku je velmi podobný trénovací fázi algoritmu. Proto již nebudu zmiňovat dříve vysvětlené aspekty algoritmu. Nejprve je nutné na testovaném vzorku najít význačné body. První testy ukázaly, že je vhodné najít všechny význačné body.

2.4.2 Otagování význačných bodů

Význačné body testovaného vzorku potřebujeme nyní otagovat. Zde je odlišnost v postupu oproti fázi trénování. Nyní už tagovací strom nevytváříme, použijeme strom již vytvořený. Bodem našeho zájmu na již vytvořeném tagovacím stromu jsou pixely které rozdělují význačné body z otcovského uzlu do synovských uzlů. Postup tagování význačných bodů je v tomto případě zřejmý. Vložíme význačné body do kořenového uzlu již vytvořeného tagovacího stromu. Význačné body rozdělíme do synovských uzlů podle pixelu který rozhodoval v daném uzlu už při trénování. Tento postup se opakuje dokud všechny nové tagy nedosáhnou listů stromu. Tímto postupem opět získáme význačné body rozdělené do tříd. Těchto tříd bude stejný počet jako počet tříd vzniklých při trénování. Některé třídy samozřejmě nemusí obsahovat žádný tag, tzn. jsou prázdné.

2.4.3 Rozpoznávání ručně psaných číslic

Vlastní rozpoznávací proces je velmi jednoduchý. Představme si náš testovaný vzorek. Vložíme jej do kořenového uzlu rozhodovacího stromu. Podíváme se na pravidlo, které dělí obrázky do synů. Náš testovaný vzorek má tagy rozděleny do tříd. Dokonce byly tagy rozděleny do tříd stejným způsobem jako při trénovacím procesu. Tedy nemáme žádný problém rozhodnout o tom, zda daný vzorek vyhovuje pravidlu či nikoliv. Vzpomeňme si ještě na poslední část pravidla - směr. Zde se právě uplatní jistá forma volnosti v pravidle ohledně vzájemném směru mezi tagy. Takto můžeme strom procházet až dokud nedosáhneme jeho listu.

Nyní je pro nás zajímavé kolik jakých typů číslic skončilo v tomto listu při trénování. Podívejme se na příklad v tabulce 2.2.

Typ číslice	0	1	2	3	4	5	6	7	8	9
Počet číslic v uzlu	0	35	2	3	2	0	1	5	1	1
Procentuelní podíl	0 %	70 %	4 %	6 %	4 %	0 %	2 %	10 %	2 %	2 %

Tabulka 2.2: Tabuka počtu číslic v listu

V tabulce 2.2 vidíme že 70 % číslic které skončily v tomto uzlu byly jedničky. Jestliže testovaný vzorek splňuje stejná pravidla jako trénované vzorky které skončily v tomto listu, tak můžeme například říci, že se 70 % pravděpodobností je náš testovaný vzorek jednička.

2.4.4 Více rozhodovacích stromů

Navzdory tomu, že 70 % v tabulce 2.2 se může zdát jako dostatečně přesvědčivá pravděpodobnost, je zde také 30 % možnost chyby. Navíc ne vždy je rozložení číslic v listu takto jednoznačné. Nejpočetnější číslice by mohla také mít jen 20 %. Což už není tolik přesvědčivý argument.

Algoritmus proto využívá pro vyhodnocení jednoho vzorku výsledky z více rozhodovacích stromů současně. Provádí to způsobem, že sečte procentelní podíl jednoho typu číslice ve všech stromech. Toto provede pro všechny typy číslic. Tím získá rozložení číslic, které je podobné průměru výsledných rozložené ze všech rozhodovacích stromů. V tomto výsledném rozložení algoritmus najde typ číslice s nejvyšším podílem a prohlásí ji za výsledek rozpoznávání.

Tato metoda využívající více rozhodovacích stromů má jednu zásadní výhodu. Každý rozhodovací strom je trochu odlišný z důvodu nalezení jiných tagů a použití jiných pravidel pro jeho budování (nelze vybírat ze všech pravidel). Z toho vyplívá že některé stromy mohou mít méně kvalitní větve ohledně úspěšnosti rozpoznávání. Díky užití více stromů, tak dojde k potlačení extrémních stromů. Sestava stromů se tedy ve výsledku chová jako průměrný strom bez extrémních, méně kvalitních větví.

Kapitola 3

Implementace

Tato kapitola popisuje způsob implementace problematiky popsané v předchozí kapitole 2. Pro snadnější orientaci je kapitola rozdělena na části stejně jako rozdělení teoreticky zaměřené předcházející kapitoly.

Vzhledem k autorovým zkušenostem a znalostem byly programy implementovány v jazyce C a C++. Nebylo zde použito objektového přístupu, což se v pozdějších fázích implementace projevilo jako méně vhodný přístup.

Programy byly implementovány a testovány pod operačním systémem Linux. Testy probíhaly na verzi Kubuntu 8.04 a dále na CentOS 5.3 64bit na školním výpočetním clusteru.

Teoretickou část algoritmu je možné rozdělit do dvou logicky úplných celků. Jedná se o část trénování a vytváření rozhodovacích stromů a druhým celkem je samotné rozhodování. Na základě této úvahy byly vytvořeny dva programy. *generator*, program sloužící k vygenerování rozhodovacího stromu dle zadaných parametrů a *reader* program, který slouží k vyhodnocení jednoho vzorku, nebo celé testovací sady.

3.0.1 OpenCV

Jelikož program pracuje s grafickými objekty, byla použita pro jazyk C nestandartní knihovna OpenCV. OpenCV je knihovna vyvinutá společností Intel nyní pod BSD licencí. Tato knihovna poskytuje několik funkcí pro práci s obrázky, mimo jiné obsahuje i maticové operace a další užitečné funkce. Knihovna je schopna pracovat s obrázky několika typů a díky tomu i mnou vytvořený program podporuje různé typy obrázků.

3.1 Hledání význačných bodů

Jak již bylo řečeno v části 2.1, význačné body nám poskytují klíčové a prakticky veškeré potřebné informace o vzorcích. Hledání význačných bodů je pro trénování a testování implementováno mírně odlišně. V této sekci se budeme zabývat hledáním význačných bodů při trénování.

První fází je načtení všech obrázků trénovací databáze. Toto nám zprostředkovává knihovna OpenCV. Tato knihovna nám umožňuje načíst obrázek do rozumé struktury a poté přistupovat k jednotlivým pixelům, nebo subpixelům obrázku. Obrázky testovací databáze jsou uloženy ve složce *Testing*.

Veškeré operace spojené s hledáním význačných bodů se nacházejí v knihovně *tag*. Program postupně prochází jednotlivé vzorky, nalézá na nich význačné body a ty pak všechny ukládá do společného pole.

Vzhledem k tomu, že databáze obsahuje i prázdná místa, jsou jednotlivé vzorky nejprve testovány, zda se na nich nachází, nebo nenachází nějaký vzorek. Tento test je prováděn jednoduchou metodou. Zjistí se barva prvního pixelu ve vzorku a kontroluje se zda se v něm najde nějaký pixel který má jinou barvu. Pokud se takový pixel najde, je vzorek považován za platný a je s ním tak dále zacházeno. Pokud je vzorek vyhodnocen jako neplatný, je přeskočen a program se zabývá následujícím vzorkem v databázi.

Význačné body jsou hledány následujícím způsobem. Jsou vygenerována náhodná čísla tak aby tato čísla mohla být souřadnicemi význačného bodu v právě prohledávaném vzorku. Poté je kontrolováno, zda tyto souřadnice neobsahují význačný bod, který už byl zpracován. Nejdůležitější je kontrola, zda se na daných souřadnicích nachází význačný bod. To znamená, že oblast 4 x 4 px směrem doprava a dolů od souřadnic musí obsahovat více barev. Jednobarevná oblast není význačným bodem (význačné body se nacházejí na hranách čar) a proto se tyto souřadnice zahazují a celý proces hledání bodu se opakuje. Pokud dané souřadnice vyhovují požadavkům na význačný bod, je tento bod uložen do pole význačných bodů pro tento vzorek.

Předcházející postup se opakuje tak dlouho, dokud není na daném vzorku nalezen požadovaný počet význačných bodů. Na některých vzorcích samozřejmě nelze najít požadovaný počet význačných bodů, pokud jsou požadavky příliš vysoké. To je ošetřeno tak, že počet pokusů pro najití požadovanho počtu význačných bodů je omezen. Požadovaný počet je zadáván jako parametr. Snížení počtu význačných bodů se kterými program pracuje značně zvyšuje rychlost jeho práce. Experimentálně bylo zjištěno, že není třeba hledat všechny význačné body v trénovaných vzorcích. Více v kapitole 4 - Testy.

Význačný bod je struktura obsahující tyto informace:

1. typ číslice na které byl tag nalezen - toto je také identifikátor souboru s testovací databází
2. X souřadnice vzorku v obrázku databáze
3. Y souřadnice vzorku v obrázku databáze
4. X souřadnice tagu ve vzorku
5. Y souřadnice tagu ve vzorku
6. třída tagu - tento parametr se jako jediný ukládá až v pozdější fázi

Po dokončení prohledávání vzorku jsou jeho význačné body přidány do pole společného pro všechny trénované vzorky. Toto pole je poté vráceno do hlavní funkce programu a je dále zpracováváno tagovacími stormy.

3.2 Tagovací stromy

Další fáze programu se týká tagovacích stromů. Tagovací stromy jsou tvořeny pomocí pole nalezených význačných bodů. Výsledkem jsou význačné body rozdělené do tříd, tedy tagy.

3.2.1 Tvorba tagovacího stromu

Implementace tagovacího stromu a související funkce se nacházejí v knihovně *tagTree*. Strom je vytvářen rekurzivně. Přesněji je nejprve vytvořen kořenový uzel a až poté jseou rekurzivně vytvářeni jeho synové.

Uzel je tedy tvořen takto. Nejdříve je zkontrolována hloubka v jaké se uzel nachází. Tagovací strom je totiž omezen svojí hloubkou. Dále se najde pixel podle kterého se budou tagy rozdělovat do synů. Při velikosti výynačného bodu 4 x 4 px máme na výběr z 16 možností. V ideálním případě by se význačné body do synů rozdělili půl na půl. Známe počet význačných bodů v uzlu, tudíž známe i ideální počet význačných bodů kteřé budou mít daný pixel černý. Proto vyzkoušíme rozdělit pole význačných bodů v uzlu podle každého ze šestnácti pixelů a vybereme ten pixel, který pole rozdělí tak, aby počet význačných bodů s černým pixelem byl co nejblíže ideálnímu počtu.

Po rozdělení význačných bodů dojde k rekurzivnímu volání funkce pro tvorbu uzlu pro levého (bílého) a pravého (černého) syna.

Poté je strom procházen a do struktury význačných bodů je vyplňována poslední položka ze seznamu v předchozí sekci a to třída tagu.

Nakonec je strom uložen do souboru. Na začátek souboru je uložen soubor settings.h. Tento soubor obsahuje základní nastavení se kterým byl strom vygenerován.

Do souboru pro uložení tagovacího stromu jsou uloženy pouze nejnutnější informace. Ukázka ze souboru je pod tímto odstavcem. Vytvořil jsem co nejjednodušší syntaxi zápisu. Zápis vlastního tagovacího stromu začíná značkou „<tagTree>“. Dále je na každý řádek uložen jeden uzel stromu. Počáteční značka uzlu je „<node>“, bezprostředně za ní následuje číslo uzlu ukončené středníkem. Číslo uzlu je jednoznačný identifikátor uzlu sloužící k rekonstrukci stromu. Lze podle něj identifikovat otcovské a synovské uzly. Levý, bílý syn má vždy dvojnásobné číslo uzlu než jeho otec. Pravý, černý syn má číslo uzlu dvojnásobně větší než otec inkrementované o jedničku. Pokud uzel není listem následuje ještě ploška označená „<SplitPx>“. Pod touto položkou se nacházejí souřadnice pixelu, který rozděluje význačné body v daném uzlu do synů.

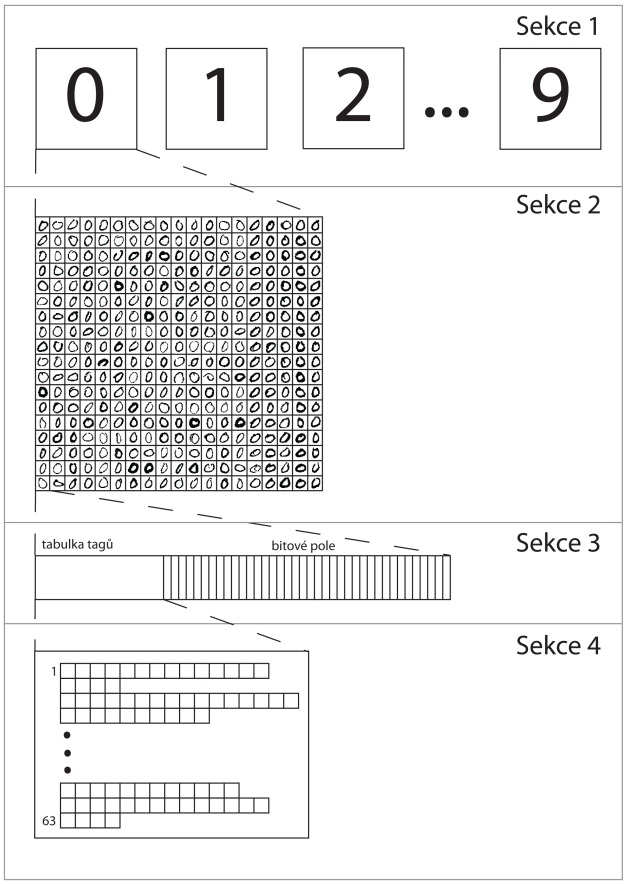
```
<tagTree>
<node>1;<SplitPx>3;3;
<node>2;<SplitPx>1;0;
<node>4;<SplitPx>0;3;
<node>8;<SplitPx>3;0;
<node>16;<SplitPx>0;1;
<node>32;
```

3.2.2 Tabulka tříd

Tagovací strom nám poměrně rychle rozdělí význačné body do tříd a vytvoří tak tagy. Nicméně pro efektivní a rychlý přístup k tagům a jejich informacím bylo zapotřebí vytvořit promyšlenou strukturu která nahradí tagovací strom a bude vyžadovat co nejméně sekvenčního přístupu.

Na obrázku 3.1 vidíme strukturu, která je vytvářena a užívána pomocí funkcí v knihovně *classTable*. Na obrázku je vidět, že struktura je hierarchicky uspořádaná. Obrázek byl rozdělen na sekce pro snadnější popis. Samotný tag se nachází až úplně nejníže v sekci 4, tudíž pro získání informace o konkrétním tagu je zapotřebí znát poměrně dost informací pro indexování. Nicméně nedostatek informací pro indexaci konkrétního tagu nemusí nutně vést ke zdlouhavému prohledávání ve velkém množství tagů.

V sekci 1 na obrázku 3.1 vidíme, že jsou tagy rozděleny do deseti skupin podle typu číslice ze které pocházejí. Sekce 2 představuje dvourozměrné pole dělící tagy podle souřadnic vzorku v daném obrázku testovací databéze. Sekce 3 je struktura v programu nazývaná



Obrázek 3.1: Obrázek znázorňující strukturu tabulky tříd

rowTableOfClasses, tedy řádek tabulky tříd. Řádek tabulky tříd obsahuje dvě informace. Bitové pole, které říká, zda vzorek kterému patří řádek tabulky obsahuje tag v konkrétní třídě. Druhou informací je vlastní dvourozměrné pole tagů zobrazené v sekci 4. Dvourozměrné pole tagů v sekci 4 je indexováno podle třídy tagu a dále už jen pořadím tagu ve třídě. Tagy ve třídě nejsou nijak seřazeny.

Pro získání přístupu k informaci o x souřadnici tagu ve vzorku indexujeme např. takto:

```
(*classTable)[number][row][col].tagArray[class][k]->x
```

Vysvětlivky:

1. number - typ číslice
2. row - řádek v obrázku trénovací databíze
3. col - sloupec v obrázku trénovací databíze
4. class - třída
5. k - pořadí vzorku v třídě
6. x - x souřadnice tagu ve vzorku

Struktura byla navržena na základě požadavků přístupu k informacím tagů. K informacím je nejčastěji přistupováno při výpočtu metriky. Původní řešení a to sekvenční vyhledávání tagů v tagovacím stromu bylo velmi pomalé. Užití této struktury oproti původnímu řešení zrychlilo generování rozhodovacích stromů v řádově v desetinásobcích až stonásobcích.

3.3 Rozhodovací stromy

Rozhodovací stromy jsou tvořeny rekurzivně podobně jako tagovací stromy. Avšak je tu jeden podstatný rozdíl. Rozdíl spočívá v tom, že rozhodovací stromy jsou vytvářeny vícevláknově. Toto na víceprocesorových, nebo vícejaderných počítačích značně zrychluje generování rozhodovacích stromů. Rozhodovací stromy jsou opět ukládány pro další použití do souboru.

3.3.1 Tvorba rozhodovacích stromů

Implementace tvorby rozhodovacího stromu a další funkce pro práci s ním se nacházejí v knihovně *randomizedTree*.

Uzel je tvořen z obrázků vzorků z trénovací sady a pravidla, které tyto obrázky dělí do synů uzlu. Tvorba stromu je implementována rekurzivně a proto začíná ověřováním zda je daný uzel listem. Za jakých podmínek se uzel stává listem je popsáno v zde 2.3.1. Pokud uzel není listem, pokračuje se dále vyhledáním vhodného pravidla pro rozdělení vzorků v listu. Tímto se zabývá následující sekce. Poté co je pravidlo nalezeno, dochází k rozdělení vzorků do synů rekurzivním voláním funkce pro tvorbu uzlu.

V tomto místě také dochází k rozdělení tvorby stromu do více vláken. Rozdělení do vláken probíhá až do zvolené hloubky. Levého Ano-syna vytváří nové vlákno, kdežto pravého Ne-syna vlákno původní. Z tohoto principu a toho, že můžeme zvolit do jaké hloubky se mohou vlákna dělit vyplívá, že počet vláken ve kterém bude strom vytvářen je mocnina čísla 2.

3.3.2 Výběr pravidla

Výběr pravidla je pravděpodobně časově nejnáročnější částí celého trénování. Proto byl mírně vylepšen oproti původnímu návrhu. Výběr pravidla je implementován v knihovně „arrangementTable“.

Původní způsob výběru pravidla spočíval v tom, že byla náhodně generována pravidla, uložena do tabulky pravidel, pravidla byla ohodnocena metrikou, která je popsána zde v části 2.3.3. Tato metrika využívala pro výpočet všech vzorků v uzlu. Na základě jejího hodnocení bylo vybráno a vráceno nejlépe vyhovující pravidlo. Tato metoda umožňovala ošem v každém uzlu vybírat pouze z velmi omezeného počtu pravidel.

Proto byl výběr pravidla upraven na „dvoufázový“. Tento způsob je modifikací předchozího popsaného. Rozdíl je ve způsobu výběru pravidel, která se ukládají do tabulky pravidel. Této tabulce teď budeme říkat hlavní tabulka pravidel. Do ní se již nevkládají náhodně generovaná pravidla, nýbrž pravidla, která byla vyhodnocena jako nejlepší z tzv. vedlejší tabulky pravidel.

Vedlejší tabulka pravidel je tvořena jistým počtem náhodně generovaných pravidel. Tato pravidla jsou ohodnocena metrikou, ale pro hodnocení se neberou všechy vzorky z daného uzlu, ale pouze malá reprezentativní skupina náhodně vybraných. Po ohodnocení všech

pravidel ve vedlejší tabulce je vybráno několik nejlépe hodnocených pravidel a ta jsou přesunuta do hlavní tabulky pravidel. Takto se pokračuje vytvořením několika vedlejších tabulek pravidel dokud se hlavní tabulka pravidel nenaplní. Následuje ohodnocení pravidel v hlavní tabulce pravidel a výběr nejlepšího z nich.

Tento způsob dovoluje vybírat za stejný čas z mnohem většího množství pravidel.

3.3.3 Uložení do souboru

Hotový strom je nakonec uložen do souboru. Toto je nutné aby mohl být natrénovaný strom použit vícekrát a nemuselo se při každém použití programu nejprve trénovat.

Soubor je tvořen podobně jako soubor pro tagovací strom. Tedy na začátek je uložen soubor settings.h. Následuje značka pro začátek rozhodovacího stromu „<randomizedTree>“ (ukázka souboru uloženého tagovacího stromu následuje pod tímto odstavcem). Následuje zápis údajů o uzlech. Jeden uzel je zapsán na jeden řádek. Řádek začíná značkou „<node>“. Za touto značkou následuje jedinečný identifikátor čísla uzlu zakončený středníkem. Číslování je opět podobné jakou u tagovacího stromu. Levý syn má dvakrát větší číslo než otcovský uzel, číslo pravého syna je dvakrát větší a inkrementované o jedničku oproti otcovskému uzlu. Následuje informace o dělícím pravidle v uzlu, nebo informace o rozložení jednotlivých typů číslic v listu. Pro dělící pravido je to značka „<arrangement>“ následovaná třemi čísly oddělenými středníky. Tato čísla označují čísla tříd a směr ve formátu jaký je uveden v části 2.3.2. Pro list a tedy rozložení následuje značka „<distribution>“ následovaná čísly oddělenými středníky. Tato čísla určují podíl typu číslice v daném listu. Pořadí typů číslic je 0,1,2,...9.

```
<randomizedTree>
<node>1;<arrangement>40;27;7;
<node>2;<arrangement>39;23;3;
<node>4;<arrangement>42;25;3;
.
.
.
<node>16777216;<distribution>0.879565;0.000000;0.007382;0.001943;0.000000;
0.001554;0.032246;0.000000;0.069153;0.008159;
<node>16777217;<distribution>0.000000;0.000000;0.000000;0.000000;0.000000;
0.000000;0.000000;0.000000;1.000000;0.000000;
```

Touto částí programu prakticky končí celé generování rozhodovacích stromů. Dále už je pouze uvolňována naalokovaná paměť. Výsledkem programu *generator* jsou tedy dva soubory s uloženým tagovacím a rozhodovacím stromem.

Generátor rozhodovacích stromů byl poměrně značně optimalizován pro rychlost, což je vidět např. na komplikované struktuře tabulky tříd. Generátor také zvládá zadávání parametrů, pro generování stromu, z příkazové řádky. To umožňuje generování mnoha stromů s různými nastaveními pro rozsáhlé testování.

3.4 Rozpoznávání ručně psaných číslic

Druhou aplikací je *reader*. Tento program slouží k přečtení a rekonstrukci zadaných rozhodovacích a tagovacích stromů a následně vyhodnotí zadané vystupy. Tento program je

primárně určen k vyhodnocení testů na databázi MNIST o které je možné se dočíst v sekci 4.1.2. Vstupem programu jsou parametry příkazové řádky, soubory s uloženými rozhodovacími a tagovacími stromy a soubor s požadavky na vyhodnocení. Výstupem programu jsou soubory s maticí záměn a úspěšností rozpoznávání. O matici záměn se lze dočíst v části 4.2.

3.4.1 Rekonstrukce stromů

První fází programu reader je rekonstrukce rozhodovacích stromů ze souborů. Implementace tohoto se nachází v knihovně *readRandomizedTree*. Rekonstruovány jsou postupně všechy rozhodovací stromy které jsou uvedeny jako parametr příkazové řádky. Díky jednoduché syntaxi souborů s uloženými rozhodovacími stromy je jejich rekonstrukce poměrně snadná. Nejprve je přečten celý soubor. Poté je jsou vytvořeny všechy uzly stromu v paměti. Uzly jsou naplněny daty ze souboru. Tedy rozhodovacími pravidly a rozložením jednotlivých typů číslic v listech. Nakonec jsou vzájemně propojeny jejich vztahy otec – syn pomocí ukazatelů.

Následuje načtení obrázků u kterých je požadováno vyhodnocení. Seznam těchto obrázků je uložen v souboru. Tento soubor obsahuje i další informace a to zda obrázek obsahuje celou testovací sadu, nebo pouze jeden vzorek pro vyhodnocení.

Dále jsou v knihovně *readTagTree* načteny a rekonstruovány kostry tagovacích stromů. Jak již bylo naznačenov části 2.4.2 z tagovacích stromů se rekonstruují pouze vztahy uzlů (otec – syn) a dělící pixely. Postup rekonstrukce tagovacích stromů je podobný rekonstrukci rozhodovacích stromů. Tedy nejprve přečten je celý soubor se stromem. Následuje vytvoření jednotlivých uzlů, naplnění uzlů příslušnými daty a nakonec spojení vztahů ve stromu pomocí ukazatelů.

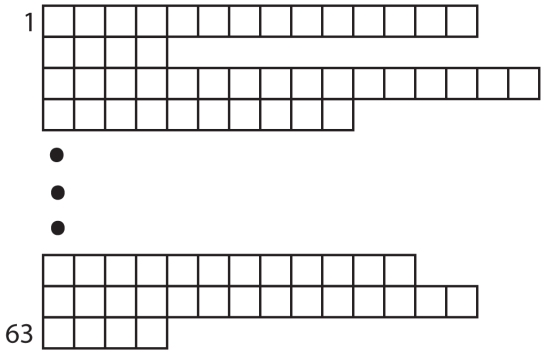
3.4.2 Příprava na rozpornávání

Další postup v programu se mírně liší podle typu zadané úlohy pro program. Buď je vyhodnocován jeden vzorek, nebo celá testovací sada. Dále popíši vyhodnocení celé sady, vyhodnocení jednoho vzorku je v tomto zahrnuto. Vyhodnocení probíhá v knihovně *recognize*.

Testovací sada je rozdělena na jednotlivé vzorky. Tyto vzorky jsou vyhodnocovány postupně. Na každém vzorku je nutné nejprve najít význačné body. Zde jsou hledány všechny význačné body. Vzorek je tedy procházen postupně pixel po pixelu a každý takový pixel je vyhodnocován zda je, či není význačným bodem. Po získání všech význačných bodů tyto body rozdělíme do tříd pomocí předpřipravených koster tagovacích stromů. Rozhodovací pixely již máme, tedy dochází pouze k „prosetí“ těchto bodů tagovacími stromy. Tím získáme tagy rozdělené do tříd a vytváří se zde jednoduchá tabulka tříd reprezentovaná dvourozměrným polem tagů. Pro názornost na obázku 3.2. Jednotlivé čtverečky znázorňují tagy.

3.4.3 Vyhodnocení rozhodování

Nyní je vše připraveno pro vlastní průchod rozhodovacími stromy. Vzorek postupně prochází všemi zadanými rozhodovacími stromy až do jejich listů. V listech jsou posbírány statistické informace o rozložení jednotlivých typů číslic z doby trénování. Rozložení z jednotlivých rozhodovacích stromů jsou umocňována a sčítána po typech číslic. Toto umocnění je drobné vylepšení, které v původním článku není popsáno. Umožňuje dosáhnout lepších statistických



Obrázek 3.2: Jednoduchá tabulka tříd

výsledků při použití více rozhodovacích stromů. Nakonec je rozhodnuto o výsledku podle nejvyššího podílu číslice v celkovém přepočítaném rozložení.

Takto je vyhodnocena celá testovací sada a výsledky rozhodování jsou průběžně ukládány pro následné vytvoření matice záměn. Matice záměn je nakonec uložena do souboru. Jednotlivé hodnoty jsou v souboru odděleny tabulátory, díky tomuto je s maticí snadné pracovat např. v Excelu. Dále je uložen do dalšího souboru výsledek úspěšnosti rozpoznávání.

Program končí uvolněním naalokované paměti. Výstupem jsou dva soubory. Reader nebyl nijak výrazně optimalizován pro rychlost a ani neobsahje grafické uživatelské rozhraní, ale obsahuje poměrně propracovaný systém příkazů načítaných z příkazové řádky pro umožnění testů pomocí skriptů.

Kapitola 4

Testování

Rozpoznávání ručně psaných číslic je netriviální problematikou. V celém algoritmu je mnoho prostoru pro vlastní úpravy, ale i pro nastavení parametrů. Proto se tato práce neobejde bez většího množství testů.

Vzhledem k vysokému množství parametrů které lze nastavit a relativně dlouhé době po kterou test trvá, byly vyloučeny metody testování při kterých se testuje více proměnných současně. Tento způsob by byl časově velmi náročný, nebo nepřesný.

Zvolená metoda testování podrobuje testu vždy jen jednu proměnnou. Toto snižuje nutnost generovat velké množství rozhodovacích stromů a současně vede k poměrně přesným výsledkům testů. Po otestování proměnné dochází k jejímu přenastavení na ideální hodnotu a poté je testována další proměnná. Po otestování všech potřebných proměnných dochází k další iteraci a proměnné se testují znovu. Toto vede k postupnému vyhledání ideálního nastavení programu. Nevýhoda této metody spočívá ve vzájemném ovlivňování některých proměnných. Změnou nastavení jedné proměnné může dojít k posunu ideálního nastavení jiné proměnné. Toto v nejhorším případě vede k nekonečnému cyklu.

První testy – orientační byly prováděny na notebooku, jejich výsledky byly poměrně nepřesné, byly využity pouze k upřesnění oblasti prohledávané dalšími testy.

Na notebooku nebylo možné si dovolit test vyžadující několik dnů strojového času. Proto byly další testy prováděny za pomocí vedoucího projektu na školním výpočetním clusteru. Díky velikosti výpočetního výkonu tohoto zařízení bylo možné vidět výsledky každého testu za maximálně několik hodin. Mnoho výpočetních jednotek, procesorů a jader umožnilo generovat rozhodovací stromy s požadovaným nastavením souběžně. Dokončení testů, vyhodnocení pomocí *readeru*, posbírání výsledků a následné vyhodnocení pak už probíhalo opět na notebooku. Tato část testů už byla velmi rychlá.

Spuštění a testování velkého množství parametrů a jejich nastavení si vyžádalo aby byly programy schopny přijímat nastavení z příkazových řádek. Dalším krokem byla automatizace testů pomocí skriptů. Toto vedlo ke skriptu *testGenerator.sh* který generuje samotné testy *tests.sh* a *testsA.sh* podle zadaných parametrů. *tests.sh* slouží k vygenerování rozhodovacích stromů podle zadaných parametrů. *testsA.sh* slouží k zpracování vygenerovaných rozhodovacích stromů programem *reader*. Poslední skript *collector.sh* slouží ke sbírání výsledků testů a jejich shromažďování do jednoho souboru. Tento sobor pak slouží k vytvoření grafu a následnému vyhodnocení testu.

4.1 Testovací databáze

Základním stavebním kamenem pro učení rozpoznávacích algoritmů, nebo jejich testování je databáze. Databáze může mít různé podoby, různé formáty a různý původ.

4.1.1 NIST

Nejznámější a také nejpoužívanější databáze pro srovnávání rozpoznávacích algoritmů ohledně ručně psaných čísel vytvořil Národní Institut Standardů a Technologií (National Institute of Standards and Technology) v USA, dále jen NIST. Tyto databáze se jmenují NIST Test Data 1 a NIST Special Database 3. V první ze jmenovaných databází se nachází písmo amerických středoškolských žáků. Druhá databáze obsahuje písmo získané z formulářů pro sčítání lidu. NIST zorganizoval soutěž v rozpoznávání ručně psaných číslic. Jako trénovací databáze byla použita NIST Special Database 3 a jako testovací databáze NIST Test Data 1. Soutežící byli nešťastní když po testování na části sady NIST Special Database 3 dosahovali chybovosti menší než 1%. Ale poté při konečných testech na testovací sadě NIST Test Data 1 dosahovali mnohem horších výsledků. Důsledkem byl rozdíl rukopisů v obou databázích. Nicméně i přes tento známý problém jsou tyto databáze v této formě stále používány. Nyní jsou obě databáze zahrnuty v NIST Special Database 19.

Výsledky uvedené v článku [2] jsou prováděny nad touto databází, avšak kvůli její zastaralosti a sloučení do jiné databáze se mi ji nepodařilo opatřit. Více o vzniku databází ručně psaných čísel zde [3].

4.1.2 MNIST

Další hodně známou a používanou databází je MNIST database of handwritten digits, tedy MNIST databáze ručně psaných čísel, dále už jen MNIST databáze. Autoři databáze MNIST si byli vědomi problémů s databází NIST a současně v ní viděli velký potenciál. NIST databáze je dostatečně velká a vzorky jsou od poměrně širokého spektra pisatelů. Proto se autoři MNIST rozhodli NIST databázi vytvořit spojením, promícháním a rozdělením NIST Special Database 3 a NIST Test Data 1 databází.

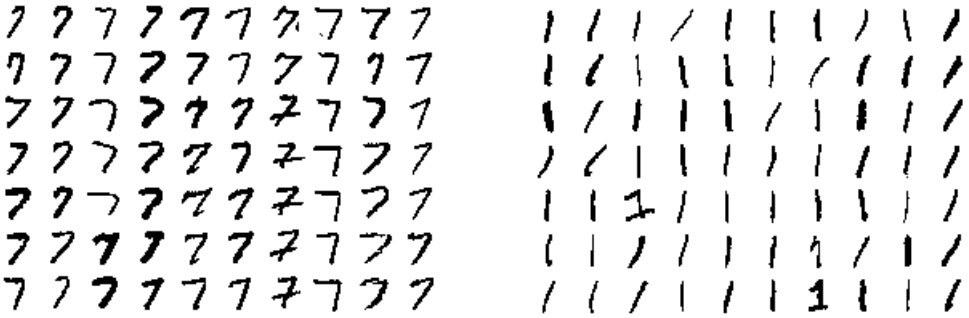
Autoři nejprve rozdělili databázi NIST Test Data 1 na prvních 250 pisatelů, což tvoří cca 30 000 vzorků a vložili je do trénovací databáze. Druhou polovinu NIST Test Data 1 vložili do testovací databáze. Trénovací databáze byla poté doplněna z NIST Special Database 3 na celkem 60 000 vzorků. Poté přidali 5 000 vzorků z NIST Special Database 3 do testovací databáze, která poté měla 35 000 vzorků.

Autoři této databáze poskytují na svých stránkách celou trénovací sadu (60 000 vzorků) a 10 000 vzorků z databáze testovací (5000 z NIST Special Database 3 a 5000 z NIST Test Data 1). Více informací k databázi a její stažení je možné zde [5]. Studie o algoritmech a jejich úspěšnosti na této databázi se nachází zde [3].

MNIST je tedy databázi kterou jsem se rozhodl použít. Jedná se o databázi amerického písma. Americky psaná čísla se v některých směrech mírně liší od evropského vzoru. Na ukázku obrázek z MNIST databáze 4.1. Na levé straně vidíme sedmičky, na pravé straně vidíme jedničky. Zatímco nám evropanům můžou některé znaky vlevo připadat jako jedničky, američan je jednoznačně označí jako sedmičky. Napravo vidíme několik jedniček. Američané nejsou zvyklí dělat u jedniček horní zobáček jako my evropané.

Toto byla malá ukázka rozdílu mezi americkým a evropským stylem písma. Nicméně je zřejmé, že po trénování programu na americké databázi může mít program vysokou

úspěšnost při testování na americké databázi. Nicméně úspěšnost rozpoznávání na česky psaných číslech by byla mnohem nižší.



Obrázek 4.1: Ukázka databáze MNIST. Vlevo výřez z sedmiček, vpravo výřez jedniček.

Na obrázku 4.1 je vidět ukázka databáze se kterou můj program pracuje. Velikost jednoho vzorku je 28 x 28 px. Pro své účely jsem si databázi nejprve mírně upravil. Původní databáze obsahuje 256 odstínů šedi. Článek popisující mnou implementovaný algoritmus počítá pouze s dvoubarevnou databází. Proto jsem si databázi upravil z 256 odstínů šedi na dvoubarevnou pomocí prahování. Práh byl nastaven na hodnotu 128. Vzhledem k čistotě a minimu šumu ve vzorcích, v databázi by při této hodnotě prahu mělo dojít k minimální ztrátě informace o tvaru číslice. Tuto úpravu jsem provedl v programu GIMP, není tedy součástí implementovaného programu.

4.2 Matice záměn

Pro podrobné vyhodnocení výsledků testů nám nestačí pouze úspěšnost rozpoznávání, ale zajímají nás i další údaje. Jako například jaká číslice bývá nejčastěji chybně vyhodnocena, nebo s jakou číslicí se konkrétní číslice nejvíce plete?

K tomuto účelu slouží pomůcka česky nazvaná „matice záměn“ (anglicky confusion matrix, nebo také table of confusion). Tato matice je nástrojem používaným v oboru umělé inteligence pro asistované učení. Tato vizualizační pomůcka nám pomáhá odpovědět na naše otázky.

Jedná se o tabulku. V této tabulce sloupce znázorňují aktuální, skutečný výsledek. Řádky znázorňují předpovídaný výsledek. Tedy výsledek rozhodování. Vše si ukážeme v následující tabulce 4.1.

V tabulce, nebo přesněji v matici záměn 4.1 vidíme výsledek jednoho z testů. Tabulka nám říká kolik vzorků bylo vyhodnoceno jako jaký vzorek. Z výsledků můžeme odvodit že například v testu kde byly samé jedničky bylo v 95,98 % případů vyhodnoceno jako jednička. Z toho samého sloupce vidíme, že jednička je nejčastěji milně vyhodnocena jako sedmička a to v 0,92 % případů. Dále vidíme, že osmička je nejčastěji milně vyhodnocena jako nula a to v 8 % případů. Další výsledky je možné si odvodit podobným způsobem. Myslím že tímto je vysvětlen i název „matice záměn“.

Matici záměn lze rozdělit na 4 logické části. Rozdělení je v tabulce 4.2.
Vysvětlivky:

- pravda, pozitivní - správný výsledek rozpoznávání

výsledek rozhodování	9	2.44	0.00	4.13	0.74	4.76	2.34	2.06	1.83	6.09	80.82
	8	3.57	0.15	6.57	3.52	1.52	6.06	6.76	0.00	67.80	3.19
	7	0.56	0.92	1.69	2.96	1.33	0.97	0.00	86.63	0.19	2.25
	6	2.81	0.15	2.44	0.00	3.42	3.71	80.07	0.00	3.23	1.87
	5	1.31	0.30	2.63	5.93	0.76	71.23	2.81	1.83	4.19	1.31
	4	0.56	0.61	1.31	0.55	83.61	0.39	1.12	3.29	1.14	3.57
	3	0.75	0.00	6.01	76.80	0.19	9.19	0.18	2.01	2.66	0.75
	2	0.93	1.85	72.74	8.34	2.09	3.13	0.56	2.38	5.52	1.50
	1	0.00	95.98	0.18	0.18	1.71	1.56	1.31	1.83	1.14	1.12
	0	87.03	0.00	2.25	0.92	0.57	1.36	5.07	0.18	8.00	3.57
		0	1	2	3	4	5	6	7	8	9
	Skutečný výsledek										

Tabulka 4.1: Matice záměn

	Pozitivní příklady	Negativní příklady
Rozhodnuto pozitivně	pravda, pozitivní	nepravda, pozitivní
Rozhodnuto negativně	nepravda, negativní	pravda, negativní

Tabulka 4.2: Matice záměn - rozklad na části

- pravda, negativní - správné zamítnutí tohoto jako výsledek
- nepravda, pozitivní - špatný výsledek rozpoznávání (někdy označováno jako chyba typu 1).
- nepravda, negativní - špatné zamítnutí tohoto jako výsledek (někdy označováno jako chyba typu 2)
- pozitivní, negativní - vztahuje se k výsledku rozhodování (pozitivní znamená: „Ano, je to tato číslice“, negativní znamená: „Ne, tato číslice to není“)
- pravda, nepravda - vztahuje se ke správnosti vyhodnocení výsldku

Podíváme se ještě jednou na tabulku 4.1. Představíme si, že vyhodnocujeme vzorek na kterém je zobrazena číslice jednička. Potom je pro nás celý sloupec 1 „pozitivní“. V případě, že je vzorek vyhodnocen jako jednička, tedy správně, nacházíme se tedy na řádku 1 s hodnotou „pravda“. jestliže bude vzorek vyhodnocen jinak než jako jednička, bude pro nás mít hodnotu „false“. Všechny ostatní sloupce mají pro náš příklad hodnotu negative.

Chyba typu 1 nastává v pozitivní oblasti matice, kdy byl určitý vzorek vyhodnocen chybně. Chyba typu 2 nastává v negativní oblasti matice a to v případě, že byl určitý vzorek vyhodnocen chybně. Podrobněji se celou touto látkou zabývá tento materiál [4].

Z predchozích řádků je vidět, že matice záměn je sofistikovaným nástrojem pro reprezentaci statistických informací o výsledcích rozpoznávání. Avšak pro vyhodnocení a porovnání úspěšnosti rozpoznávání je tento nástroj příliš komplikovaný.

$$tpr = TP/pos \tag{4.1}$$

$$fpr = FP/neg \tag{4.2}$$

1. TP - pravda, pozitivní (true, positive)
2. FP - nepravda, pozitivní (false, positive)
3. pos - pozitivní rozpoznávání
4. neg - negativní rozpoznávání

Po porovnání úspěšnosti, nebo chybovosti rozpoznávání se používají pojmy „true positive rate“ a „false positive rate“. Pro tyto dva pojmy budu i nadále používat česká označení „úspěšnost rozpoznávání“ a „chybovost“. Tyto vztahy jsou popsány v rovnicích 4.1 a 4.2.

Tyto dva pojmy jsou používaně pro porovnávání mezi různými algoritmy, a také pro vyhodnocení nastavení programu. Mnou prováděné testy obsahují výsledky uváděné pomocí úspěšnosti čtení. Úspěšnost čtení pro více číslic se spočítá jako průměr úspěšností čtení jednotlivých číslic. Toto je ukázáno pro deset číslic ve vzorci 4.3. Pro upřesnění, čísla ze kteých se počítá průměr se nacházejí na diagonále matice záměn začínající v bodě [0,0].

$$tpr_phi = \sum_{i=0}^{i=9} \frac{tpr_i}{10} \tag{4.3}$$

4.3 Výsledky testů

Zde se dostáváme k výsledkům testů. Pro lepší přehlednost byly testy zpracovány do grafů. Svislá osa zobrazených grafů vždy představuje úspěšnost rozhodování, jednotkou je %. V některých případech zobrazuje strojový čas generování jednoho rozhodovacího stromu v minutách. Čas byl měřen na notebooku Asus F5RL. Procesor Intel Core 2 Duo T5550 na frekvenci 1,8 GHz. Vodorovná osa zobrazuje hodnoty testovaného parametru.

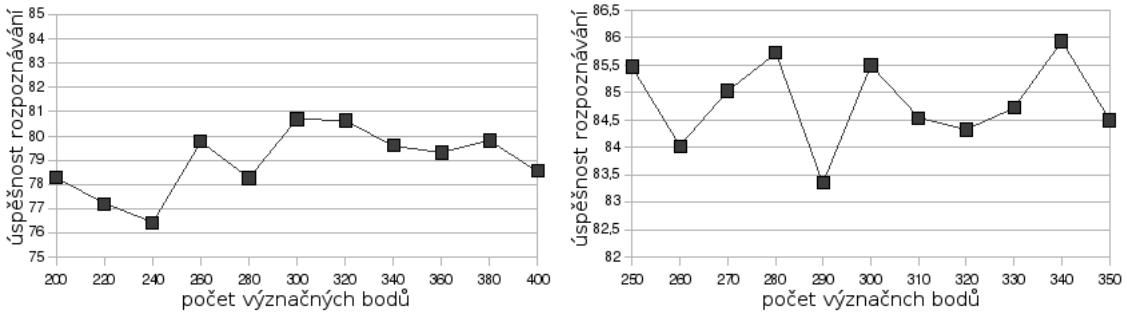
Pro každou hodnotu v testu byly vygenerovány tři rozhodovací stromy. Tři proto, aby se snížil vliv odchylek v hodnocení některými stromy. Tři stromy také proto, že generování tohoto počtu stromů je přijatelně časově náročné. Některé testy, a na tyto bude upozorněno, byly prováděny s jiným počtem stromů, například pro zvýšení přesnosti výsledků.

Testovací databáze byla rozdělena na dvě části. Prvních 90 % sloužilo k trénování, zbylých 10 % bylo použito jako kontrolní množina vzorků. Nakonec po nastavení všech parametrů byl proveden test na testovací databázi.

4.3.1 Počet hledaných význačných bodů

Počet hledaných význačných bodů značně ovlivňuje čas generování rozhodovacích stromů. Proto byl proveden test, který zkoumá, zda je nutné hledat všechny význačné body, nebo kolik význačných bodů je optimální vyhledávat.

Na levém grafu obrázku 4.2 vidíme graf začínající na hodnotě 200. Tato hodnota byla určena z prvotních testů jako nutné minimum pro vyšší úspěšnost rozpoznávání. Levý graf napovídá, že nemá smysl hledat méně než 250 význačných bodů, nebo také více než 350 výzančných bodů. Z důvodů vysokých odchylek v prvním grafu, byl druhý test (pravý graf) proveden s pěti stromy na jednu hodnotu. Toto mírně snížilo odchylku, nicméně náhodné prvky v algoritmu generátoru zde mají významný vliv na výsledky testů. Jako ideální hodnota byl zvolen počet 300.

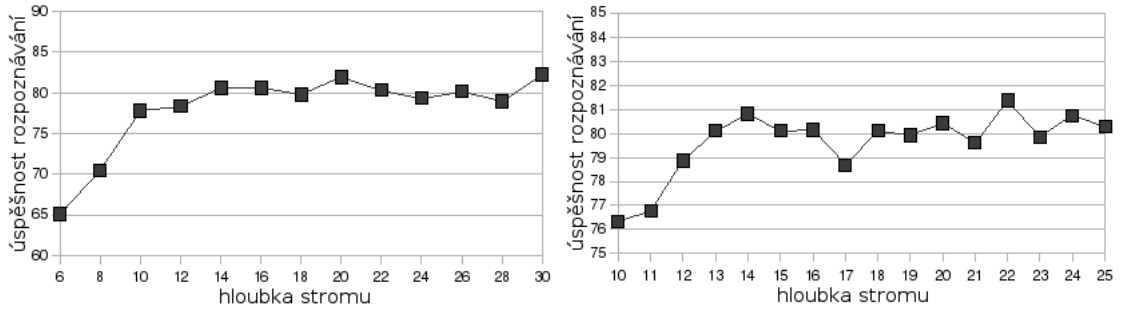


Obrázek 4.2: Grafy vlivu počtu hledaných význačných bodů

4.3.2 Omezení hloubky rozhodovacího stromů

Rozhodovací strom by měl být omezen svojí hloubkou z několika důvodů. Prvním je urychlení generování rozhodovacího stromu. Druhým důvodem je možnost vytvoření takové kombinace vzorků v uzlu rozhodovacího stromu že by pravidla pro vytvoření listu z tohoto uzlu nezareagovala, nebo reagovala až ve velké hloubce stromu. To by způsobovalo buď velmi dlouhý čas generování stromů, pád programu vlivem nedostatku paměti počítače, nebo pomalejší vyhodnocení rozpoznávání.

Z levého grafu na obrázku 4.3 lze vypozorovat, že nemá velký smysl generovat rozhodovací stromy hlubší než je hloubka 14. Druhý graf byl pořízen pro kontrolu s menším rozestupem hodnot hloubky. S ohledem na druhý graf byla jako ideální hloubka vybrána dvacítka. Rozdíl v čase při generování rozhodovacích stromů s maximální hloubkou 14 a 20 je malý a výsledky testů okolo hloubky 20 vypadají lépe než výsledky okolo hloubky 14.

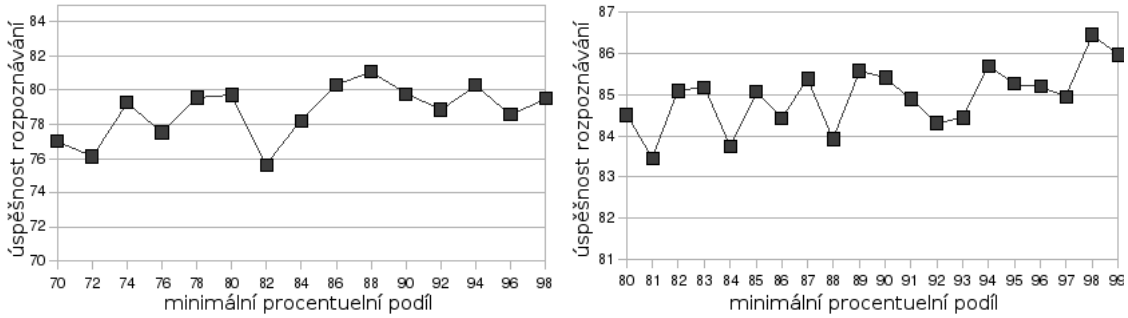


Obrázek 4.3: Grafy vlivu omezení maximální hloubky rozhodovacích stromů

4.3.3 Minimalni podíl dominantního typu číslice v listu

Toto nastavní má v hodnotách nad 0,7 jen minimální vliv na délku generování rozhodovacích stromů. Avšak vliv na výkon je zde patrný.

Na levém grafu obrázku 4.4 pozorujeme mírnou stoupající tendenci u hodnot blížících se číslu 1. Avšak rozptyl hodnot velmi zkresluje výsledky testu. Proto byl druhý test na pravém grafu proveden s pěti stromy na každou hodnotu. To vedlo ke zpřehlednění grafu a je zde jasně vidět stoupající tendence u hodnot blížících se k číslu 1. Proto byla jako ideální hodnota zvoleno číslo 0,98.

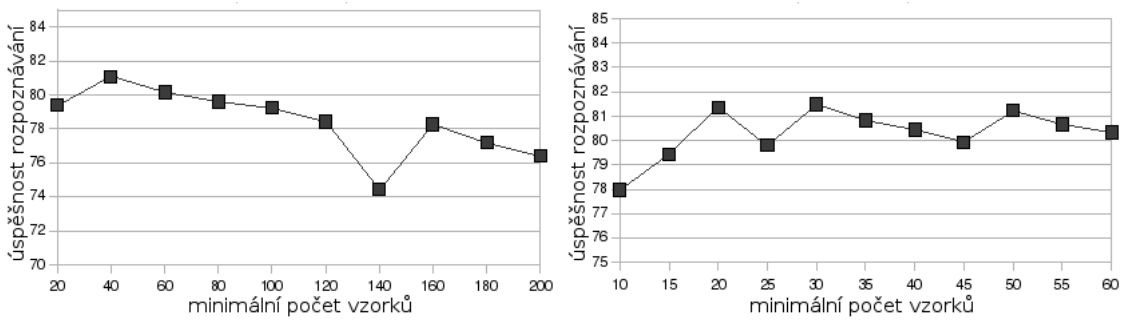


Obrázek 4.4: Grafy vlivu minimálního podílu dominantního typu číslice v listu

4.3.4 Minimum vzorků v uzlu

Minimum vzorků v listu je parametr o kterém se můžeme dočíst zde [1]. Je to parametr, pod který klesne-li počet vzorků v uzlu, tento uzel se stává listem. Tento zajímavý parametr umožňuje hlavně snížit počet uzlů ve stromech. To vede k rychlejšímu generování rozhodovacích stromů a rychlejšímu rozpoznávání.

Levý graf obrázku 4.5 ukazuje že vysoké hodnoty vedou k horšímu rozpoznávání vlivem méně hlubokých stromů. Nedochází totiž k dostatečnémmu oddělení jednotlivých typů čílic od sebe. Pravý graf upřesňuje hodnoty mezi 10 a 60. Jako ideální počet byla zvolena třicítka.

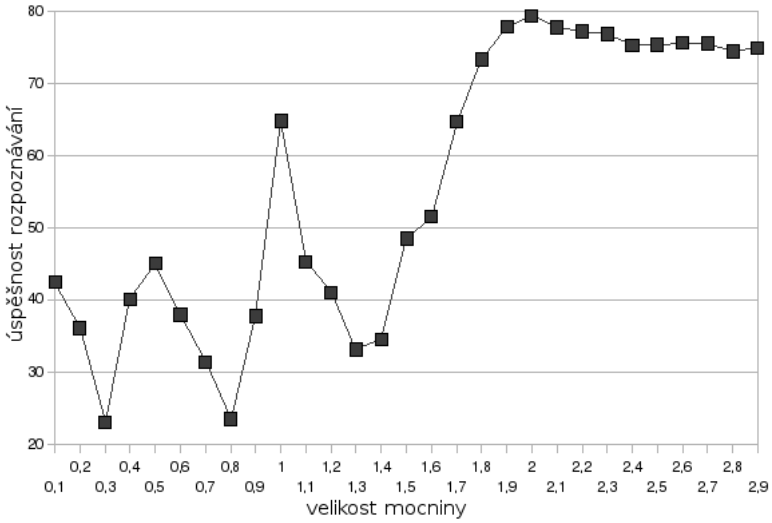


Obrázek 4.5: Grafy vlivu minimálního počtu vzorků v uzlu

4.3.5 Metrika - mocnina pro vyvážení

Metrika se skládá ze dvou částí jak je popsáno v části 2.1. Jako mocniny byly zvoleny druhé mocniny. Nicméně byl proveden pokus zda je toto číslo vyhovující. Vzhledem k tomu, že obě části metriky jsou kvadratickými funkcemi, je pro nás rozhodující poměr jejich mocnin. A proto nebyly prováděny další pokusy s mocninou druhé části metriky.

Na grafu obrázku 4.6 je vidět, že i první mocnina má poměrně dobré výsledky. Nicméně druhá mocnina dosahuje mnohem lepších výsledků. Vyšší mocniny už nedosahují takto dobrých výsledků. Proto byla druhá mocnina ve vzorci metriky ponechána.

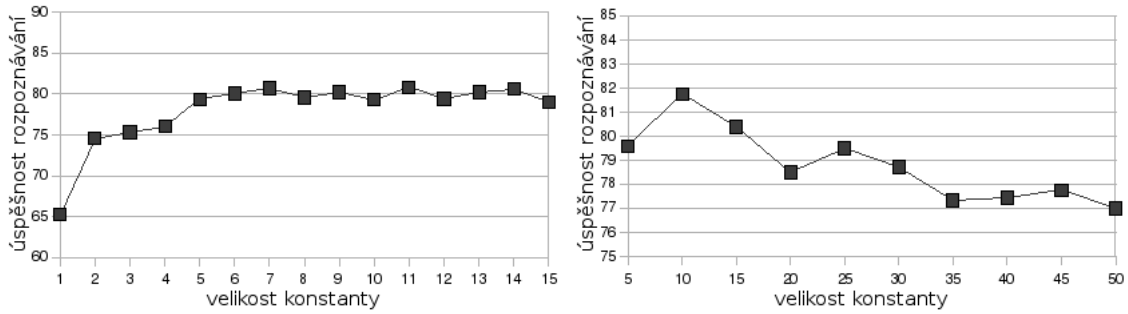


Obrázek 4.6: Graf vlivu mocniny pro vyvážení v metrice

4.3.6 Metrika - konstanta pro vyvážení

Další částí metriky jsou konstanty. Konstanty Jsou opět dvě, podobně jako u mocnin. A opět nás zajímá hlavně poměr těchto konstant. Tedy konstanta pro oddělování podobných vzorků byla nastavena na hodnotu 1. V testu se tedy mění konstanta pro vyvážení.

Levý graf na obrázku 4.7 ukazuje nižší hodnoty, jako ideální konstanta bylo zvoleno číslo 7. Pravý graf pouze ukazuje širší rozsah.



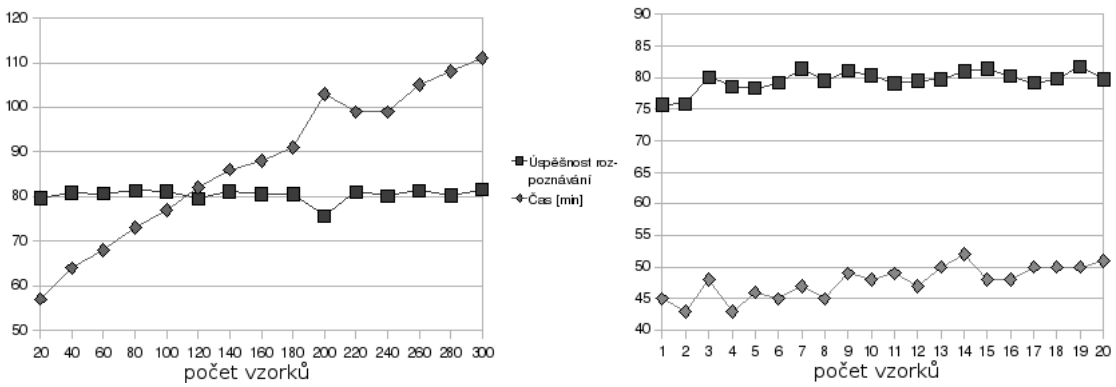
Obrázek 4.7: Grafy konstanty pro vyvážení v metrice

4.3.7 Velikost reprezentačního vzorku pro výběr pravidel

O dvoufázovém výběru pravidla se lze dočíst v sekci 3.3.2. V prví fázi výběru se pravidla testují pouze malým reprezentativním vzorkem vybraných vzorků z uzlu. Počet těchto vzorků je testován právě zde.

Grafy na obrázku 4.8 zobrazují jak úspěšnost čtení, tak dobu generování jednoho rozhodovacího stromu v minutách. Na levém grafu vidíme test s větším rozsahem. Vidíme, že počet reprezentačních vzorků silně ovlivňuje čas generování stromu, ale úspěšnost čtení roste jen minimálně. Na pravém grafu vidíme rozsah 1 až 20. Vidíme že i jeden vzorek může po-

měrně dobře reprezentovat všechny vzorky v uzlu. Nicméně jako optimum byl zvolen počet 20.

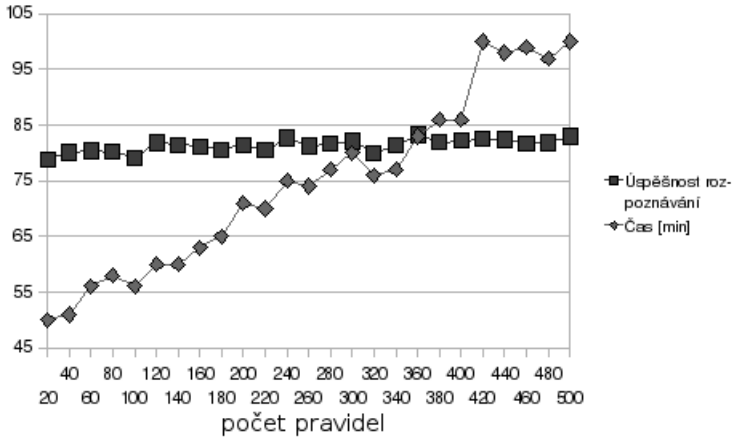


Obrázek 4.8: Graf vlivu počtu reprezentačních vzorků

4.3.8 Velikost vedlejší tabulky pravidel

Velikost vedlejší tabulky pravidel rozhoduje o celkovém počtu pravidel ze kterého se vybírá nejvhodnější pravidlo. Představme si, že hlavní tabulka pravidel má velikost 100 pravidel. Vedlejší tabulka pravidel má velikost 500 pravidel. Z vedlejší tabulky se vybírá 10 nelépe ohodnocených pravidel a ta se předávají dále do hlavní tabulky. Výpočtem 100 / 10 * 500 se dostáváme k celkovému počtu prohledaných pravidel 5000. Za stejnou dobu by se původně metodou bez vedlejší tabulky vyhodnotilo pouze cca 200-300 pravidel.

Z grafu na obrázku 4.9 je vidět že i tento parametr má velký vliv na dobu generování rozhodovacích stromů. Nicméně s rostoucím čase stále mírně roste i úspěšnost čtení. Jako ideální hodnota byl zvolen počet 300 pravidel.

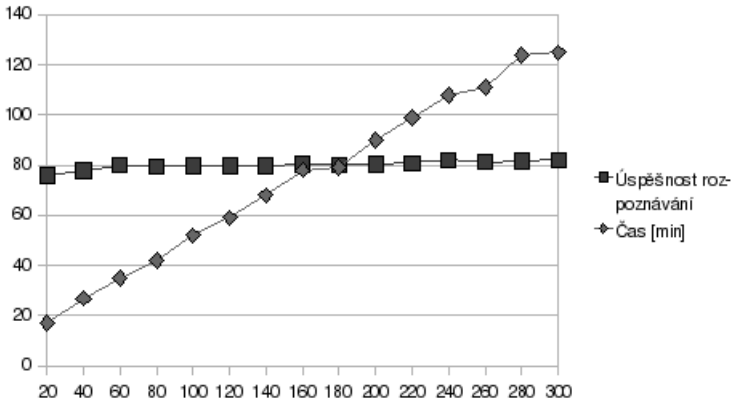


Obrázek 4.9: Graf vlivu velikosti vedlejší tabulky pravidel

4.3.9 Velikost hlavní tabulky pravidel

Hlavní tabulka pravidel obsahuje nejlépe hodnocená pravidla z vedlejších tabulek pravidel. Pravidla v hlavní tabulce jsou hodnocena metrikou za pomocí všech trénovacích vzorků v uzlu. Velikost této tabulky má velmi významný vliv na čas generování rozhodovacích stromů což je vidět i na grafu v obrázku 4.10.

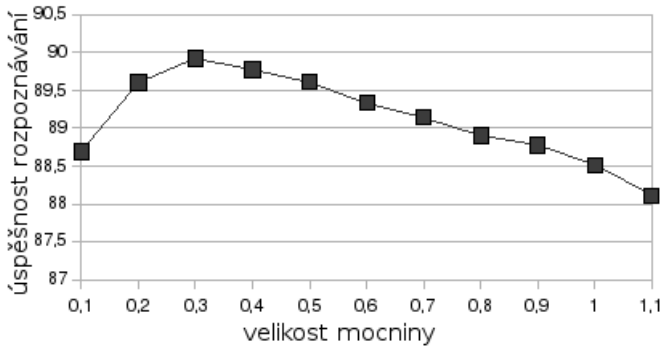
Z grafu je vidět, že čas roste téměř lineárně s množstvím pravidel. Zatímco úspěšnost čtení roste jen minimálně. Jako ideální počet pravidel bylo zvoleno 100 pravidel a to hlavně z důvodů časové náročnosti.



Obrázek 4.10: Graf vlivu velikosti hlavní tabulky pravidel

4.3.10 Mocnina výsledků z rozhodovacích stromů

Výsledná rozložení získaná při rozhodování z listů rozhodovacích stromů měla být původně umocňována. Umocňována proto, aby listy v nichž měla určitá číslice výraznou převahu měly větší váhu. Nicméně testy ukázaly, že toto v praxi nefunguje a ideální je cca třetí odmocnina. To je vidět i na grafu obrázku 4.11.

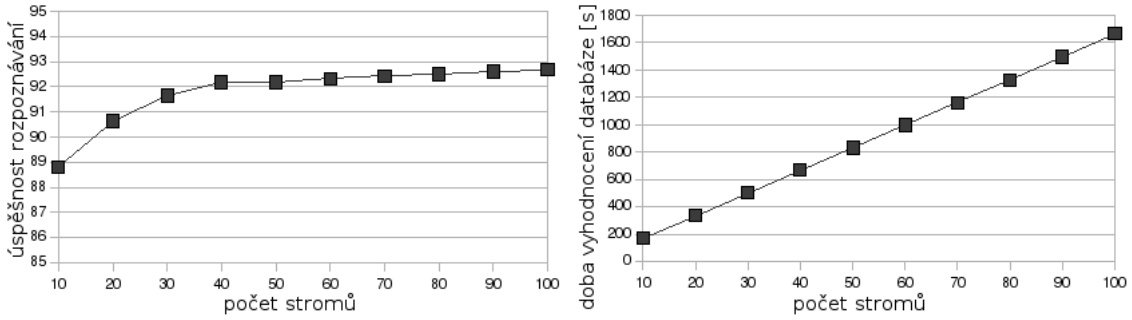


Obrázek 4.11: Graf vlivu mocniny pro sčítání výsledků rozhodovacích stromů

4.3.11 Počet rozhodovacích stromů

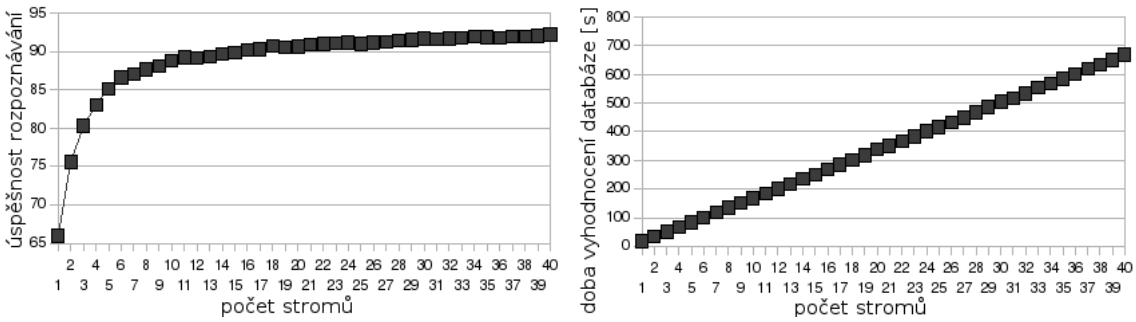
Počet rozhodovacích stromů značně ovlivňuje úspěšnost rozpoznávání a také rychlost vyhodnocování vzorku.

Na grafech na obrázku 4.12 vidíme vliv počtu stromů. V levém grafu vidíme, že úspěšnost čtení roste poměrně rychle až do čtyřiceti stromů. Poté začíná přírůstek s každým stromem klesat. Na pravém grafu vidíme dobu vyhodnocování kontrolní sady. Je vidět, že doba vyhodnocování je lineárně závislá na počtu rozhodovacích stromů.



Obrázek 4.12: Grafy vlivu počtu použitých rozhodovacích stromů. Vlevo graf úspěšnosti čtení, vpravo doba vyhodnocování testovací databáze

Na dalším obrázku 4.13 vidíme stejný test jako předchozí, ale s menším rozsahem a užším rozestupem testovaných hodnot. Na levém grafu vidíme, že úspěšnost rozpoznávání stoupá poměrně rychle až do dvanácti stromů. Na pravém grafu opět vidíme lineární závislost doby vyhodnocování kontrolní sady na počtu rozhodovacích stromů. Kontrolní sada byla vyhodnocována rychlostí 301 vzorků za sekundu při užití jednoho rozhodovacího stromu. Rychlost při užití čtyřiceti rozhodovacích stromů byla 8,1 vzorku za vteřinu. Z těchto rychlostí vyplívá, že počet rozhodovacích stromů lze zvolit podle nároků na přesnost, nebo rychlost rozpoznávání. Pro další testy byl zvolen počet 20 rozhodovacích stromů. Tento počet poskytuje rychlost rozpoznávání 16 vzorků za vteřinu a úspěšnost čtení 90,63 %.



Obrázek 4.13: Grafy vlivu počtu použitých rozhodovacích stromů. Vlevo graf úspěšnosti čtení, vpravo doba vyhodnocování testovací databáze

4.4 Vyhodnocení výsledků

Na úplný závěr testů bylo vygenerováno 100 rozhodovacích stromů. Úspěšnost rozpoznávání nejhoršího rozhodovacího stromu byla 59,85 % a nejlepší rozhodovací strom dosáhl 68,97 %. Průměrný strom dosahoval 64,68 %. Střední odchylka činí 1,49 %.

Ztěchto stromů bylo vybráno 20 nejlepších, podle úspěšnosti rozpoznávání. Těchto 20 stromů bylo aplikováno na MNIST testovací databázi s výsledkem 91,51 %. Chybovost tedy činí 8,49 %.

Zde v tabulce 4.3 vidíme matici záměn tohoto závěrečného testu. Z této matice záměn lze odvodit několik závěrů. Nejproblémovější číslicí je pro náš algoritmus číslice 8. Tato číslice dosahuje úspěšnosti rozpoznání pouhých 84,19 %. Tato číslice je často špatně vyhodnocována jako číslice 0, nebo 9. Toto je samozřejmě způsobeno vlivem podobnosti těchto číslic. Nejlépe rozpoznatelnou číslicí je číslice 1. Tato číslice je úspěšně rozpoznávána v 99,26 % případech. Toto je pravděpodobně způsobeno její tvarovou jednoduchostí a malou podobností s ostatními číslicemi.

Výsledek rozhodování	9	1.29	0.00	0.20	0.10	2.15	0.71	0.33	0.10	2.68	86.26
	8	1.82	0.00	1.51	1.14	0.43	2.49	2.11	0.20	84.19	2.60
	7	0.75	0.00	1.61	1.14	0.32	0.00	0.22	93.64	0.32	3.43
	6	1.61	0.09	0.60	0.00	0.96	0.23	90.66	0.00	1.72	0.20
	5	0.64	0.00	0.80	1.87	0.10	90.36	0.88	0.20	1.18	0.41
	4	1.07	0.00	0.20	0.10	95.05	0.00	1.33	0.30	0.53	2.80
	3	0.00	0.00	1.81	91.46	0.00	4.04	0.00	0.70	1.61	0.52
	2	0.32	0.64	91.73	3.22	0.43	1.07	0.22	3.62	1.72	0.83
	1	0.00	99.26	0.20	0.20	0.43	0.59	0.55	1.10	0.21	1.04
	0	92.47	0.00	1.31	0.72	0.10	0.47	3.66	0.10	5.80	1.87
		0	1	2	3	4	5	6	7	8	9
	Skutečný výsledek										

Tabulka 4.3: Matice záměn pro test nad testovací sadou MNIST

4.5 Porovnání výsledků

V této části můžeme porovnat dosažené výsledky s oficiálními výsledky udávanými na webových stránkách databáze MNIST [5].

Tabulka 4.4 uvádí výběr různých metod (klasifikátorů) pro porovnání s mnou dosaženými výsledky. Na webových stránkách MNIST databáze [5] je uvedena kompletní tabulka na které je vidět i časový vývoj jednotlivých metod. V této [3] studii je možné se dočíst podrobněji o konkrétních metodách uvedených v tabulce. Z důvodu nemožnosti úplného překladu některých názvů metod ponechávám výtah z tabulky v originálním jazyce (angličtina).

V tabulce 4.4 vidíme, že různé úpravy obrazu před zpracováním mohou výrazně, pozitivně ovlivnit výkon. Mnou implementované programy nepoužívají žádného preprocesoru. Implementovaný algoritmus například nepočítá s pootočením testovaného vzorku. Z toho vyplívá, že tímto směrem by se mohl vydat další vývoj.

Úspěšnost čtení mnou implementovaného programu na MNIST databázi je 91,51 %, chybovost je tedy 8,49 %. Tento dosažený výsledek je porovnatelný s některými výsledky jiných algoritmů z roku 1998. Toto je také rok ze kterého pochází článek na jehož základě byl můj

Klasifikátor	**Preprocesory**	**Chybovost** [%]	**Autor**
linear classifier (1-layer NN)	-	12.0	LeCun et al. 1998
linear classifier (1-layer NN)	deskewing	8.4	LeCun et al 1998
K-nearest-neighbors, Euclidean (L2)	-	3.09	Kenneth Wilder, U. Chicago
K-nearest-neighbors, Euclidean (L2)	deskewing, noise removal, blurring	1.8	Kenneth Wilder, U. Chicago
K-NN with non-linear deformation (IDM)	shiftable edges	0.54	Keysers et al. 2007
Virtual SVM, deg-9 poly, 1-pixel jittered	-	0.68	DeCoste and Scholkopf, 2002
Convolutional net LeNet-5, (no distortions)	-	0.95	LeCun et al. 1998

Tabulka 4.4: Výtah z oficiální tabulky výsledků na databázi MNIST

program implementován. Autoři původního článku uvádějí úpěšnost čtení na NIST Special Database 3 úspěšnost čtení až 99,5 %. Tato hodnota odpovídá několikaletému výzkumu a množství vylepšení základního algoritmu. Mezi tyto vylepšení patří preprocesor, který vzpřímí otočené testovací vzorky, nebo mechanismus, který odmítne vyhodnotit vzorek, v případě že, jsou výsledky z rozhodovacích stromů sporné.

Kapitola 5

Závěr

Cílem této práce bylo experimentovat s rozpoznáváním ručně psaných číslic, zejména s metodou rozhodovacích stromů. Těchto cílů práce bylo dosaženo, výsledkem je dvojice programů. Jeden pro rozpoznání význačných bodů, vytvoření tagovacích stromů a natrénování rozhodovacích stromů. Druhý program pro vyhodnocení úspěšnosti použité metody nad testovací datovou sadou.

V průběhu práce jsem získal bohaté zkušenosti se studiem odborného textu, se strojovým učením, návrhem a implementací složitého systému, s rozsáhlým testováním a paralelizací úloh. Dále jsem se seznámil s knihovnou OpenCV, prohloubil si znalosti v oboru počítačového vidění, rozpoznávání a v mnoha dalších oborech.

Bylo trénováno a testováno na standartní databází MNIST. To umožnilo porovnání s jinými metodami rozpoznávání. Bylo dosaženo úpěšnosti rozpoznávání 91,51 %. Tento výsledek je porovnatelný s metodami používanými v roce 1998, kdy vyšel článek [2], podle kterého jsem postupoval.

Z matic záměn bylo zjišteno, že algoritmus výborně funguje při rozpoznávání číslice jedna. Hůře však rozpoznává číslice osm a devět. Toto je způsobeno vzájemnou podobností číslic.

Časová náročnost generátoru stromů je vysoká. Program pro generování rozhodovacích stromů byl značně optimalizován pro dosažení co nejvyšší rychlosti. Avšak i poté bylo nutné stromy pro testování generovat na školním výpočetním clusteru. Toto umožnilo vytvořit velké množství testů, jejichž grafy jsou v této práci uvedeny.

Paměťová náročnost generátoru rozhodovacích stromů je také poměrně vysoká. Program za běhu potřebuje průměrně okolo 750 MB, ve krátkodobé špičce 1,5 GB RAM. Tyto hodnoty jsou však akceptovatelné velkou částí dnešních počítačů, navíc rozhodovací stromy stačí vygenerovat jednou a dále je užívat na mnohem méně výkoných počítačích. Spotřeba paměti vyhodnocovacího programu se pohybuje okolo 150 MB při testování. To je způsobeno slabší optimalizací a také uložením všech testovacích informací v paměti po celou dobu testu.

Výběr pravidla z omezeného počtu snižuje výkon programu. Vybírat ze všech možných pravidel by bylo možné, pokud by se testovala pouze malým reprezentativním výběrem vzorků z uzlu. Toto potvrzují i testy. Dále by bylo příjemné, kdyby aplikace pro testování a rozpoznávání měla ještě jiné, konkrétní využití a od toho by se odvíjelo její grafické uživatelské rozhraní. A v neposlední řadě by bylo vhodnější programy tohoto rozsahu implementovat v objektově orientovaném jazyce. Zvýšení úpěšnosti čtení by bylo možné dosáhnout i pomocí preprocesoru.

Rozhodovací stromy mají velmi široké spektrum možností využití. Rozpoznávání ručně psaných číslic je také jejich možným využitím s dobrými výsledky. Počítačové rozpoznávání

prozatím není možné využívat v plné míře z důvodů relativně nízké spolehlivosti. S rostoucím výkonem počítačů roste i spolehlivost rozpoznávání a proto věřím, že v následujících letech budou tyto techniky mnohem více využívány.

Literatura

[1] Amit, Y.; Geman, D.: Shape quantization and recognition with randomized trees. *Neural Comput.*, ročník 9, č. 7, 1997: s. 1545–1588, ISSN 0899-7667.

[2] Amit, Y.; Geman, D.; Wilder, K.: Joint Induction of Shape Features and Tree Classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, ročník 19, č. 11, 1997: s. 1300–1305, ISSN 0162-8828.

[3] Bottou, L.; Cortes, C.; Denker, J. S.: Comparison of classifier methods: A case study in handwritten digit recognition. In *Proceedings of the 12th International Conference on Pattern Recognition, Los Alamitos, CA.*, ročník 2, 1994, s. 77–82.

[4] Flach, P. A.: Tutorial on The Many Faces of ROC Analysis in Machine Learning. Citováno dne 24.4.2009. Jako poznámky dostupné na URL: http://www.cs.bris.ac.uk/~flach/ICML04tutorial/index.html.

[5] Yann LeCun, C. C.: THE MNIST DATABASE of handwritten digits. Citováno dne 24.4.2009. Dostupné na URL: http://yann.lecun.com/exdb/mnist/.

Dodatek A

Obsah CD

Přiložené CD je rozděleno na 5 částí:

Bin - Adresář obsahuje z příkazové řádky spustitelné soubory *reader* a *generator*. Jsou zde připraveny dva skripty. *tests.sh* skript spustí *generator* a vytvoří rozhodovací a tagovací stromy s kořenem jména „final0-0-100“. Soubory se budou nacházet ve složce *Trees*. Toto generování trvá cca 160 minut strojového času na notebooku Asus F5RL - Intel C2D 1,8GHz. Druhý skript *testsA.sh* spustí program *reader* a provede test s dvaceti rozhodovacími stromy. Budou vytvořeny soubory s maticí záměn a úspěšností čtení, s kořenem jména „test0-101“. Tyto soubory se budou nacházet v adresáři *ConfusionMatrixes*. Tento test tvá cca 10 minut strojového času. Binární soubory byly přeloženy a úspěšně spuštěny na studentském serveru Merlin s operačním systémem CentOS 5.3 64bit Linux. Dále se zde nacházejí adresáře a soubory nutné pro funkci binárních souborů.

Data - V tomto adresáři se nacházejí data vzniklá při testování programů.

- *ConfusionMatrixes* - v tomto adresáři se nacházejí matice záměn a soubory s úspěšností rozpoznávání jednotlivých testů.
- *Testy* - adresář obsahuje skripty pro generování testů a vygenerované testovací soubory
- *Trees* - zde se nacházejí soubory tagovacích a rozhodovacích stromů vygenerovanách při testování (cca 1500 souborů)
- soubor *testy final.xls* - pracovní soubor obsahující souhrn testů. Soubor obsahuje konkrétní výsledky a grafy.

Doc - Adresář obsahuje jeden soubor a to programovou dokumentaci vygenerovanou programem *Doxygen*.

Src - Tento adresář obsahuje soubory potřebné pro překlad a spuštění programů.

- Zdrojové soubory se nacházejí v adresáři *Source*.
- Adresáře *Testing* a *Training* obsahují MNIST databázi ručně psaných čísel.
- *Makefile* - soubor pro překlad zdrojových kódů na binární soubory

- *testGenerator.sh* - ukázka skriptu pro generování testovacích souborů
- *collector.sh* - ukázka skriptu pro sběr výslekdů testů

Other - Ostatní soubory.

- *tex* - obsahuje zdrojové soubory pro vytvoření technické zprávy + pdf *projekt.pdf* - vygenerovaná technická zpráva
- *plakat.pdf* - Obsahuje vytvořený plakát.

Dodatek B

Manual

Programy byly úspěšně překládány pod několika linuxovými operačními systémy. Pro překlad jsou potřeba knihovny jazyků C a C++. Dále je potřeba nainstalovaná nestandartní knihovna OpenCV. Pro překlad programů použijte příkaz *make*. Tento příkaz vytvoří programy *generator* a *reader*.

generator

Program slouží pro vygenerování rozhodovacích a tagovacích stromů. Vstupem programu je testovací databáze z adresáře *Training*. Výstupem programu jsou tagovací a rozhodovací stromy uložené v souborech.

Program se spouští příkazem `./generator <parametry>`. Možnosti použití a parametry jsou popsány v tabulce B.1.

Jméno parametru	**Datový typ očekávaný na vstupu**	**Základní nastavení**	**Popis parametru**
jmeno_stromu	string	Povinný parametr!!!	Kořen jména pro výstupní soubory Musí být prvním parametrem!!!
-div	double	0.9	Dělí trénovací sadu na trénovací a testovací část 0 - 0.9 je nyní trénovací částí
-tags	int	300	Počet tagů hledaných ve vzorku při trénování
-threads	int	2	Mocnina čísla 2. Udává počet vláken
-rand_tree_depth	int	20	Maximální hloubka rozhodovacího stromu
-leaf_distr	double	0.98	Minimální podíl dominantního typu číslice aby se stal z uzlu list
-min_picts	int	30	Minimální počet vzorků v uzlu, pod kterým se uzel stává listem
-pow_bal	double	2.0	Metrika - mocnina vyvažovací části
-pow_separ	double	2.0	Metrika - mocnina separační části
-kbal	double	7	Metrika - konstanta vyvažovací části
-ksepar	double	1	Metrika - konstanta separační části
-arrangments	int	100	Velikost hlavní tabulky pravidel
-samples	int	20	Velikost reprezentativní množiny vzorků
-test_arrans	int	300	Velikost vedlejší tabulky pravidel
-rbest	int	10	Počet pravidel ve vedlejší tabulce pravidel, která postoupí do hlavní tabulky pravidel

Tabulka B.1: Tabulka parametrů pro *generator*

Příklad spuštění:
`genereator jmeno_stromu -div 0.8 -threads 2 -min_picts 50 -samples 10`

Tabulka B.1 představuje výpis všech možných parametrů. Popisuje také očekávaný datový typ parametru a nastavení parametru pokud nebude zadán z příkazové řádky. U nezadaných parametrů budou použity hodnoty ze souboru /Source/settings.h. V tomto souboru je také možné před překladem nastavit parametry.

Výstupní soubory programu s tagovacím a rozhodovacím stromem se jmenují *jmeno_stromu_tagTree.txt* a *jmeno_stromu_randomizedTree.txt*. Kde „jmeno_stromu“ je kořen jména stromů který je pro oba stromy společný. Tento kořen je zadáván programu jako povinný první parametr. Tyto textové soubory jsou uloženy ve složce Trees.

reader

Program sloužící pro vyhodnocení testovací sady. Testovací sada se nachází v adresáři *Testing*. Program umožňuje otestovat i jiný vzorek než jen testovací sadu. Toto se nastavuje v konfiguračním souboru *test.txt*.

Tento soubor obsahuje na každém řádku jeden příkaz k vyhodnocení. Na začátku řádku se nachází buď číslice, která při testování slouží k porovnání správnosti rozpoznání, nebo příkaz „decide“. Příkaz „decide“ vyhodnocuje co se nachází na obrázku a výsledek vypíše na standartní výstup. Za těmito příkazy následuje pomlčka a cesta k vyhodnocovanému souboru.

Program se spouští příkazem `./reader <parametry>`.Parametry jsou popsány níže.

Příklad spuštění:
`./reader jmeno_vysledku -trees strom1 strom2 strom3`

`jmeno_vysledku` je kořenem pro soubory s výsledky. Průměrná úspěšnost čtení bude uložena do souboru *ConfusionMatrixes/jmeno_vysledku_diagonal.txt*. Matice záměn bude uložena do souboru *ConfusionMatrixes/jmeno_vysledku_confusion.txt*. Paramery jsou popsánz v tabulce B.2.

Jméno parametru	**Datový typ očekávaný na vstupu**	**Základní nastavení**	**Popis parametru**
jmeno_vysledku	string	Povinný parametr!!!	Kořen jména pro výstupní soubory Musí být prvním parametrem!!!
-div	double	0.9	Dělí trénovací sadu na trénovací a testovací část 0.9 - 1 je nyní testovací část
-distr_pow	int	0.3	Mocnina pro sčítání výsledků stromů

Tabulka B.2: Tabulka parametrů pro `reader`