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

AUTOMATICKÉ OZNAČOVÁNÍ OBRÁZKŮ

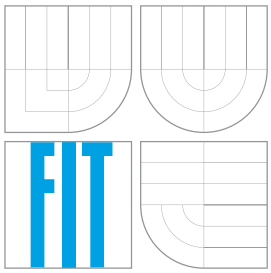
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MICHAL SÝKORA

BRNO 2012

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

AUTOMATICKÉ OZNAČOVÁNÍ OBRÁZKŮ

AUTOMATIC IMAGE LABELING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. MICHAL SÝKORA
AUTHOR

VEDOUCÍ PRÁCE — Ing. MICHAL HRADIŠ
SUPERVISOR

BRNO 2012

Abstrakt

Tato práce se zabývá automatickým zařazováním obrázků do sémantických tříd na základě jejich obsahu. Hlavně různými způsoby klasifikace založenými na SVM. Hlavním cílem této práce je zlepšit přesnost klasifikace na velkých datových sadách. Práce se zabývá jak lineárními tak i nelineárními SVM klasifikátory a také možností použití RBM pro transformaci příznaků pro lineární SVM klasifikátory. Dále jsou všechny tyto přístupy srovnány. Srovnává se nejen přesnost, ale i časová náročnost, využití zdrojů a možnosti budoucího pokračování ve výzkumu.

Abstract

This work focuses on automatic classification of images into semantic classes based on their contentc, especially in using SVM classifiers. The main objective of this work is to improve classification accuracy on large datasets. Both linear and nonlinear SVM classifiers are considered. In addition, the possibility of transforming features by Restricted Boltzmann Machines and using linear SVM is explored as well. All these approaches are compared in terms of accuracy, computational demands, resource utilization, and possibilities for future research.

Klíčová slova

zpracování obrazu, strojové učení, SVM, SVM jádra, RBM, SGE

Keywords

image processing, machine learning, SVM, SVM kernel, RBM, SGE

Citace

Michal Sýkora: Automatické označování obrázků, diplomová práce, Brno, FIT VUT v Brně, 2012

Automatické označování obrázků

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Michal Hradiše

.....
Michal Sýkora
22. května 2012

Poděkování

Rád bych poděkoval vedoucímu mé diplomové práce panu Ing. Michalu Hradiši za jeho ochotu, trpělivost a odbornou pomoc při tvorbě této práce.

Obsah

Kapitola 1

Úvod

Zpracování obrazu je v dnešní době rychle se rozvíjející odvětví a automatické zařazování obrázků do sémantických kategorií (venku / vevnitř, člověk, letadlo atd.) je jeho významnou součástí. Schopnost rozeznat objekty na obrázku je zajímavá sama o sobě, ale také otevírá další možnosti zpracování obrazových informací. Mohou vzniknout vyhledávače indexující a vyhledávající obrázky skutečně podle jejich obsahu, a ne jenom podle jejich popisu. Stejně tak může vzniknout automatické vyhledávání ve videu nebo jeho třídění podle různých kategorií [18]. Indexováním videa a hledání v něm se zabývá například Trecvid [17], což je výzkumná soutěž, jejímž cílem je podporovat výzkum ve vyhledávání informací tím, že poskytuje rozsáhlou testovací sadu, jednotné zhodnocení výsledků a jejich srovnání s ostatními. Jako další lze uvést soutěž PASCAL VOC [24], ve které je cílem co nejpřesněji určit jednotlivé objekty na obrázcích a zařadit je do příslušných kategorií.

Automatické označování obrázků je složitý proces. Tento proces je popsán v kapitole číslo 2 - Postup automatického označování obrázků. Tato práce se zabývá hlavně poslední částí toho postupu a to je klasifikace obrázků do daných kategorií. Cílem mé práce je zlepšit současné metody klasifikace založené na SVM klasifikátorech popsaných v kapitole číslo 3 - SVM. Vzhledem k tomu, že jsou tyto experimenty výpočetně náročné problémy, tak jsem je všechny prováděl na školních serverech. Školní servery pro řízení výpočtů používají software SGE, který je popsaný v kapitole číslo 4 - SGE. Experimenty s SVM klasifikátory jsou popsané v kapitole 5 - SVM v praxi. Je zde popsán celý postup od datové sady až po experimenty s SVM klasifikátory. Jako první jsem začal s lineárními SVM klasifikátory 5.2, poté jsem prozkoumal možnosti nelineárních SVM klasifikátorů 5.3. Následuje srovnání těchto přístupů 5.4. Na základě tohoto srovnání pokračuji využitím neuronových sítí RBM k transformaci obrazových příznaků a následnou klasifikací lineárními klasifikátory 6. Je zde nejen popsán princip RBM 6.1, ale i následná implementace řešení 6.2, zhodnocení výsledků a je tam také naznačena možnost dalšího pokračování práce 6.3.

Kapitola 2

Postup automatického označování obrázků

Tato kapitola popisuje postupy používané v současnosti při automatickém zařazování obrázků do sémantických kategorií na základě jejich obsahu. Celý postup se skládá z několika dílčích úloh, při kterých se využívají různé druhy algoritmů jak ze zpracování obrazu, tak i z jiných počítačových odvětví jako je např. umělá inteligence.

Jednotlivé implementace automatického označování obrázků se od sebe liší, ale základní princip je většinou stejný. Tento základní princip je následující: nejprve je nutno získat příznaky popisující vstupní obrázek. Existují různé druhy používaných příznaků, několik nejčastěji používaných příznaků je popsáno v sekci Extrakce příznaků 2.1. Pro lepší práci s příznaky je nutná jejich reprezentace v nějaké vhodné podobě, například jak je popsáno v kapitole Reprezentace příznaků 2.2, kde jsou nejprve všechny příznaky převedeny na kódová slova pomocí slovníku 2.2.1. Potom jsou tyto kódová slova reprezentovány jako Bag of words (BOW) 2.2.2, který popisuje celý vstupní obrázek. Posledním krokem je klasifikace 2.3 jednotlivých BOW do různých sémantických tříd. Tento postup je zobrazen na obrázku 2.1.

2.1 Extrakce příznaků z obrazu

V této kapitole bude vysvětleno jaké druhy obrazových příznaků se používají v automatickém označování obrázků. Dále také budou popsány detektory významných oblastí.

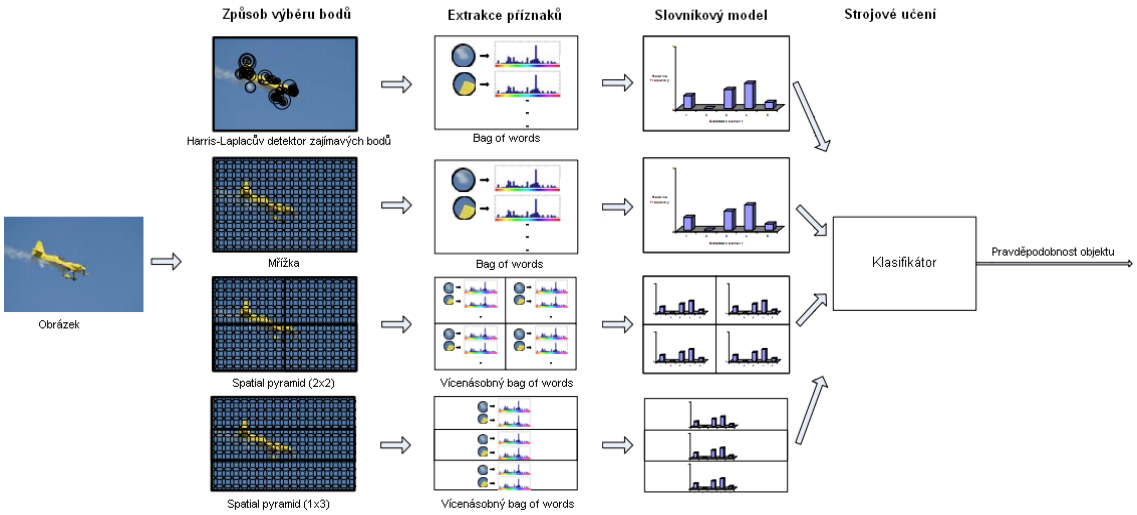
Příznaky lze rozdělit podle oblasti, kterou popisují, na lokální a globální. Globální příznaky odpovídají celému obrázku, na rozdíl od lokálních, které popisují jen malou část obrazu.

Většina metod pro automatické zařazování obrázků pracuje z různou kombinací jak lokálních, tak globálních příznaků.

2.1.1 Globální příznaky

Globální příznaky popisují celý obrázek, a proto obsahují nejen vlastnosti hledaných objektů, ale i jejich okolí. Následují používané příznaky a jejich popis.

Histogram směrů hran: Je to na měřítku nezávislý příznak. Pro jeho výpočet je potřeba nějaký detektor hran, například Cannyho detektor. Po detekci hran se určí jejich lo-



Obrázek 2.1: Postup při zařazování obrázků do sémantických tříd. Obrázek je převzat z [23] a upraven.

kální směr a z těchto směrů se vypočítá histogram. Pokud je histogram normalizovaný, tak není závislý na velikosti obrázku [25].

Barevné momenty: Reprezentují obraz jako rozdělení pravděpodobnosti barevných složek v něm obsažených. Každá složka může být reprezentována třemi hodnotami: průměr (E), standardní odchylka (σ) a nesouměrnost(s):

$$E_i = \sum_{j=1}^{N} \frac{1}{N} p_{ij}, \quad \sigma_i = \sqrt{\left(\frac{1}{N}\sum_{j=1}^{N}(p_{ij} - E_i)^2\right)}, \quad S_i = \sqrt[3]{\left(\frac{1}{N}\sum_{j=1}^{N}(p_{ij} - E_i)^3\right)}, \tag{2.1}$$

kde i-tá barevná složka a j-tý pixel obrazu je zapsán jako p_{ij} a N je počet pixelů obrázku [10].

Textura reprezentovaná Gáborovým filtrem: Informace o textuře lze interpretovat pomocí Gáborova filtru. Vytvoří se sada filtrů s různými frekvencemi a orientací. Konvolucí těchto filtrů s obrazem získáme histogram odezev. Tyto hodnoty vyjadřují energii textury v různých měřítcích a rotacích obrazu [12].

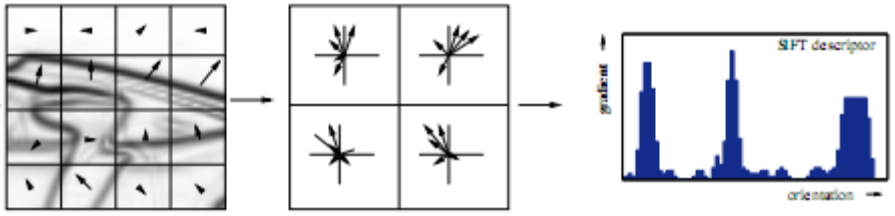
2.1.2 Lokální příznaky

Lokální příznaky odpovídají jen malé části z celkového obrázku, a i přes to jsou častěji používané než příznaky globální. Do globálních příznaků velmi často zasahuje vliv pozadí a jiných objektů v obraze. Hledaný objekt často nezabírá celou plochu obrázku, ale je umístěn v reálném světě. Na obrázku také může být více různých objektů a globální příznaky jsou vypočítány z celého obrazu. Zahrnutím více objektů a pozadí do deskriptoru obrázku zaniká chybu, a ten potom zcela neodpovídá popisovanému objektu.

V této kapitole jsou popsány nejpoužívanější lokální příznaky a detektory významných oblastí. Detektory významných oblastí slouží k určení míst obrázků, ze kterých se tyto

příznaky počítají. V praxi se používá více než jeden druh příznaků reprezentujících obraz. Získá se tak ucelenější informace o vzhledu obrazu.

SIFT: Scale Invariant Feature Transformation(SIFT) je nejčastěji používaným typem příznaků při automatickém označování obrázků [13]. Tento deskriptor reprezentuje jeden bod obrazu a jeho okolí. V okolí daného bodu je spočten gradient a následně se vypočtou histogramy směrů gradientů, kde k binu histogramu odpovídajícímu směru gradientu je přičtena jeho velikost. Histogram neobsahuje všechny možné směry, ale jen osm směrů. Směr gradientu se diskretizuje do osmi. Velikosti gradientů jsou váženy Gaussovou funkcí. Na mřížce 4x4 se potom tyto Histogramy Orientovaných Gradientů (HOG) konkatenují a normalizují. Samotný deskriptor je pak vektor těchto histogramů. Takový deskriptor je částečně nezávislý na malých posunech, rotacích nebo změně měřítka obrázku. Je částečně nezávislý na změně osvětlení a robustní vůči lokálním geometrickým deformacím.



Obrázek 2.2: Určování deskriptoru SIFT. Obrázek je převzat z [13].

OpponentSIFT: OpponentSIFT je deskriptor založený na SIFT. Vzniká konkatenací deskriptorů SIFT na všech kanálech barevného prostoru, který je definovaný třemi kanály O_1, O_2 a O_3, kde:

$$O_1 = \frac{R-G}{\sqrt{2}}, \qquad O_2 = \frac{R+G-2B}{\sqrt{6}}, \qquad O_3 = \frac{R+G+B}{\sqrt{3}}. \tag{2.2}$$

Kanály O_1 a O_2 obsahují informaci o barvě, zatímco kanál O_3 obsahuje informaci o intenzitě [14].

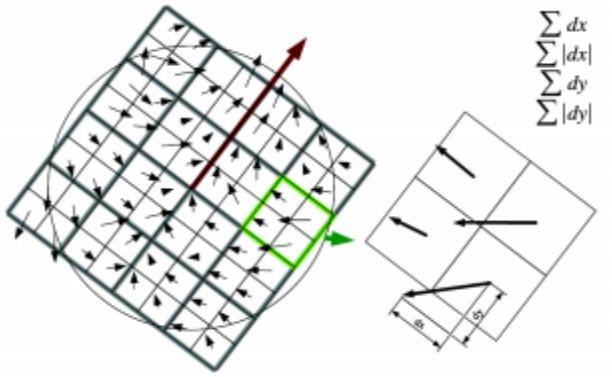
C-SIFT: Používá stejný barevný prostor jako OpponentSIFT 2.1.2. Kanály O_1 a O_2 definované v tomto prostoru obsahují stále nějakou informaci o osvětlení. Proto C-SIFT využívá C-invariant [6], který eliminuje zbývající informaci o osvětlení z těchto barevných kanálů.

rgSIFT: rgSIFT je modifikace SIFT, kde je počítaný z kanálů R a G normalizovaného modelu RGB. Protože je normalizovaný je téměř invariantní vůči změnám intenzity osvětlení.

RGB-SIFT: RGB-SIFT je SIFT, který se počítá ze všech kanálů barevného modelu RGB nezávisle na sobě. Vzhledem k normalizacím prováděným metodou SIFT je toto vyjádření ekvivalentní k OpponentSIFT [19].

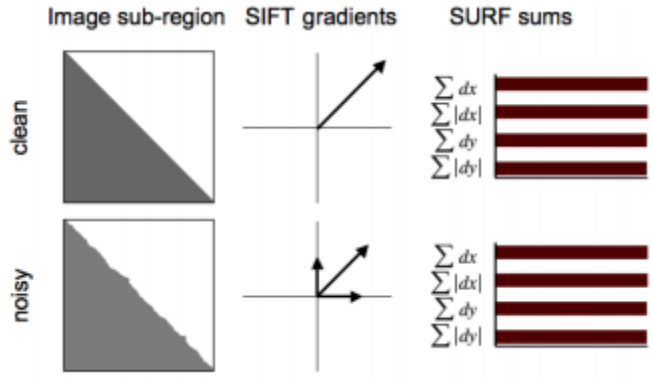
SURF: SURF je robustní deskriptor částečně inspirovaný deskriptorem SIFT. Je založený na Haarových vlnkách v x a y směru. V každém regionu 4x4 jsou spočítány 4 pří-

znaky $\sum dx, \sum \|dx\|, \sum dy, \sum \|dx\|$. Výsledný příznakový vektor je potom konkatenací těchto příznaků. Výsledný vektor tedy obsahuje 64 příznaků vnitřních regionů [2].



Obrázek 2.3: Určování deskriptoru SURF. Obrázek je převzat z [2].

SURF je několikrát rychlejší než SIFT. Je také méně citlivý na šum než SIFT jak je vidět na obrázku 2.4 [2].

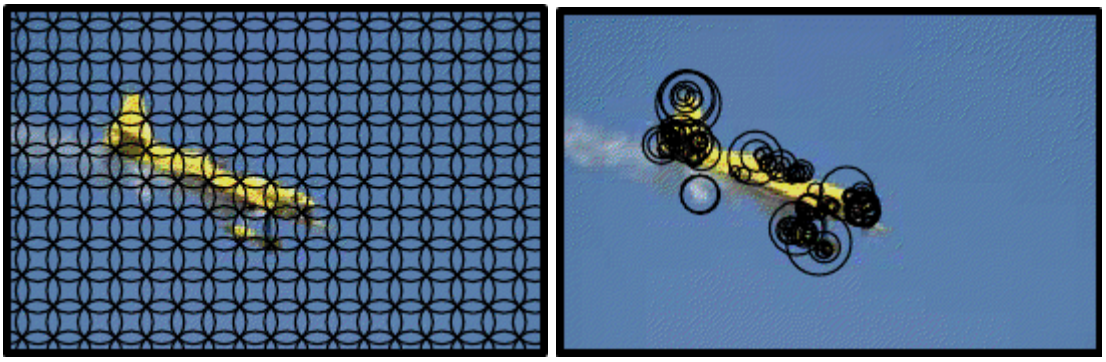


Obrázek 2.4: Srovnání příznaků SIFT a SURF při zašuměném vstupním obrázku. Obrázek je převzat z [2].

Detektory významných oblastí

Lokální příznaky reprezentují jen určitou část vstupního obrázku. Je proto nutné určit, které části chci, aby reprezentovali. Výběr těchto oblastí lze rozdělit na dva způsoby: pravidelná mřížka pokrývající celý obrázek nebo zajímavé body v obraze vybrané nějakou metodou pro jejich detekci.

Mřížka: Je to pravidelná mřížka vzájemně se překrývajících kruhových oblastí, které pokrývají celý vstupní obrázek. Podle různých implementací mohou mít tyto oblasti různou velikost. Příklad takové mřížky je vidět na levém obrázku 2.5.



Obrázek 2.5: Způsoby výběru oblastí. Levý obrázek zobrazuje mřížku a na pravém je Harris-Laplacův detektor. Obrázek je převzat z [23].

Harris-Laplace: Je detektor významných oblastí v obraze, který spojuje myšlenku tradičního 2D Harrisova detektoru rohů s reprezentací prostoru Gaussovým měřítkem. Harrisův detektor je založen na auto-korelační matici, která se často používá pro detekci nebo popis lokálních struktur obrazu. Tato matice musí být adaptovaná na změny měřítka, aby byla nezávislá na rozlišení obrazu. Tato matice je definovaná jako:

$$\mu(X, \sigma_I, \sigma_D) = \sigma_D^2 g(\sigma_I) * \begin{bmatrix} L_x^2(X, \sigma_D) & L_x L_y(X, \sigma_D) \\ L_x L_y(X, \sigma_D) & L_y^2(X, \sigma_D) \end{bmatrix} \tag{2.3}$$

kde σ_I je integrační měřítko σ_D derivační měřítko a L_a je derivace vypočítaná ve směru a.

Tento detektor je průměrně robustní vůči změnám úhlu pohledu, velmi odolný proti změně měřítka, velmi stabilní při rozmazaných obrázcích a také při změnách osvětlení [15].

DOG: Difference of Gaussians je aproximací druhé derivace obrázku. Tato aproximace se provádí tak, že se obraz nejprve rozostří gaussovským filtrem. Tento výsledek se uloží. Poté se vstupní obrázek rozostří znovu gaussovským filtrem s jinou standardní odchylkou. Tyto dva obrazy se od sebe odečtou a vznikne výsledný obraz. Ve výsledném obraze se vyberou lokální extrémy, které jsou hledané významné oblasti [27].

LOG: Laplacian of Gaussian je detektor bodů, který kombinuje filtr s gaussovským jádrem a diskrétní Laplacův operátor. Diskrétní Laplacův operátor je definován jako suma druhých derivací Laplacova operátoru a vypočítána jako suma rozdílu okolních pixelů. Jádro 2D filtru:

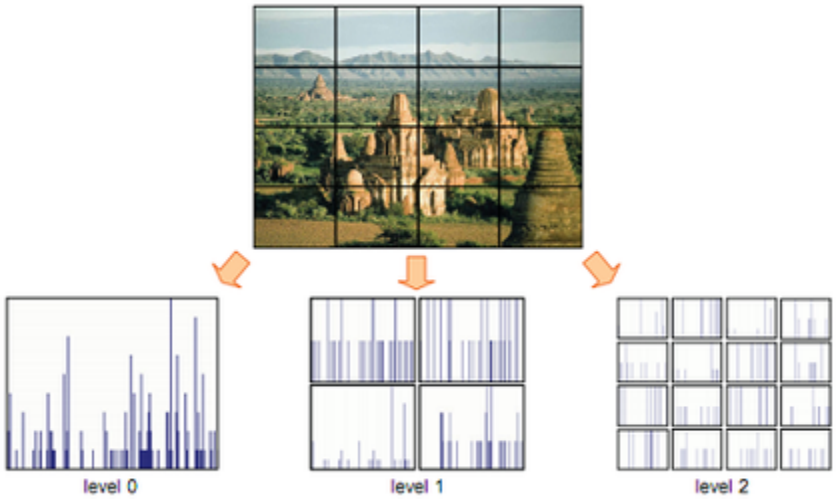
$$D_{xy}^2 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad nebo \quad D_{xy}^2 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{2.4}$$

Nejprve je na obraz použit Gaussův filtr a poté Laplacův operátor, který má velkou kladnou nebo zápornou odezvu na místa, která jsou výrazně světlejší nebo tmavší než jejich okolí. Body s velkou kladnou nebo zápornou odezvou jsou středy výsledných oblastí [26].

Hessian-Affine: Je podobný jako Harris-Laplace 2.1.2 a používá stejnou matici, se kterou se ovšem pracuje odlišně. Hlavní rozdíl je v lokální adaptaci. Tento detektor má podobné vlastnosti jako Harris-Laplacův detektor. Ale navíc oproti Harris-Laplace detektoru je odolný vůči afinním transformacím [15].

Spatial pyramid

Příznaky popsané v předešlé kapitole popisují detaily objektů a z těchto detailů vzniká sémantický popis celého obrázku. Na rozdíl od počítače je člověk schopen zařadit scénu (např. vevnitř / venku) velice rychle a bez nějakého soustředění se na detaily. Jako snaha napodobit toto chování vznikly tzv. Spatial pyramid [11], které rekurzivně rozdělují obrázek do více sektorů. Způsob rozdělování je naznačen na obrázku 2.6. Každý takto vzniklý sektor



Obrázek 2.6: Dělení prostoru obrázku do oblastí. Obrázek je převzat z [11].

nese vlastní popis příznaků, které se v něm vyskytují. Příznaky obsažené v jednotlivých úrovních mají různou váhu na výslednou klasifikaci. Čím jemnější rozdělení na oblasti, tím mají příznaky větší vliv na výsledek – analogicky k přesnosti určení polohy daného příznaku.

2.2 Reprezentace příznaků

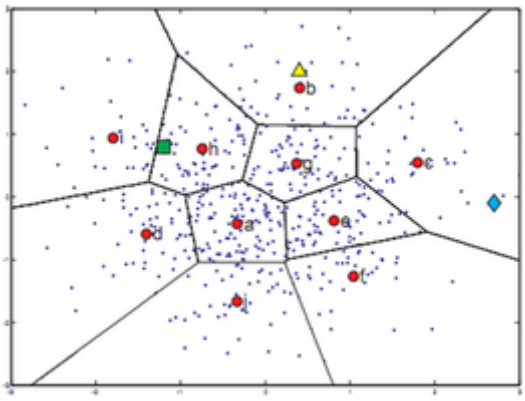
Příznaky získané z obrazu se většinou nepoužívají pro klasifikaci. Pro zjednodušení klasifikace je vhodné příznaky reprezentovat jiným způsobem. Pro reprezentaci příznaků se používá kódování pomocí slovníků, které převádí jeden příznak (typicky vektor obsahující desítky čísel) na jedno číslo. Toto číslo je odkaz do slovníku na daný typ příznaků. Pro reprezentaci celého obrazu je často používán tzv. Bag of words. Ten popisuje výskyt jednotlivých typů příznaků v obrázku.

2.2.1 Slovníky

Slovníky mají za úkol převést vektor reprezentující jeden příznak na kód, který označuje podobné příznaky. Slovníky se obvykle sestrojují dvěma různými způsoby [5]: anotačním přístupem nebo sestavením z dat. Při anotačním přístupu se slovník sestrojí tak, že se

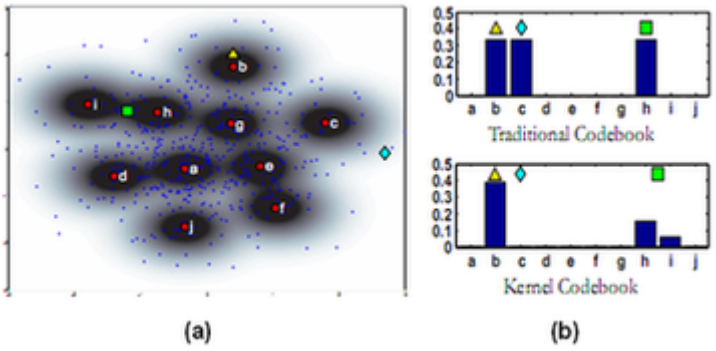
určitým oblastem v obraze přiřadí nějaký smysluplný popisek např. obloha, voda atd. Na druhou stranu při sestavovaní slovníku z dostupných dat je provedeno jejich shlukování. Pro shlukování lze použít algoritmus k-means.

Klasické slovníky přiřazují každému vstupnímu vektoru právě jedno číslo. Hodnota tohoto čísla je pořadové číslo nejbližšího shluku. Při tomto přístupu nastávají dva problémy, které ilustruje obrázek 2.7. Jsou to následující problémy: nejistota a přijatelnost [5]. Problém nejistoty nastává, pokud by vektor mohl náležet více shlukům, ale vždy je přiřazen jen jednomu. Problém přijatelnosti nastává, pokud je sice přiřazen nejbližšího shluk, ale vstupní vektor je od něj vzdálen a úplně neodpovídá příznaku, který reprezentuje daný shluk.



Obrázek 2.7: Obrázek ukazující problémy klasických slovníků. Malé tečky reprezentují obrazové příznaky, červené popsané kruhy jsou kódová slova. Trojúhelník značí datový vzorek, který je vhodný pro zakódování pomocí kódového slova. Problém nejistoty naznačuje čtvereček a problém přijatelnosti kosočtverec. Obrázek je převzat z [5].

Oba tyto problémy lze odstranit použitím jádra, které přiřadí každému vstupnímu vektoru kódová slova podle tvaru jádra a vzdálenosti od příznaku [5]. Jádro může mít tvar např. Gaussovy křivky. Slovník s Gaussovým jádrem je znázorněn na obrázku 2.8.



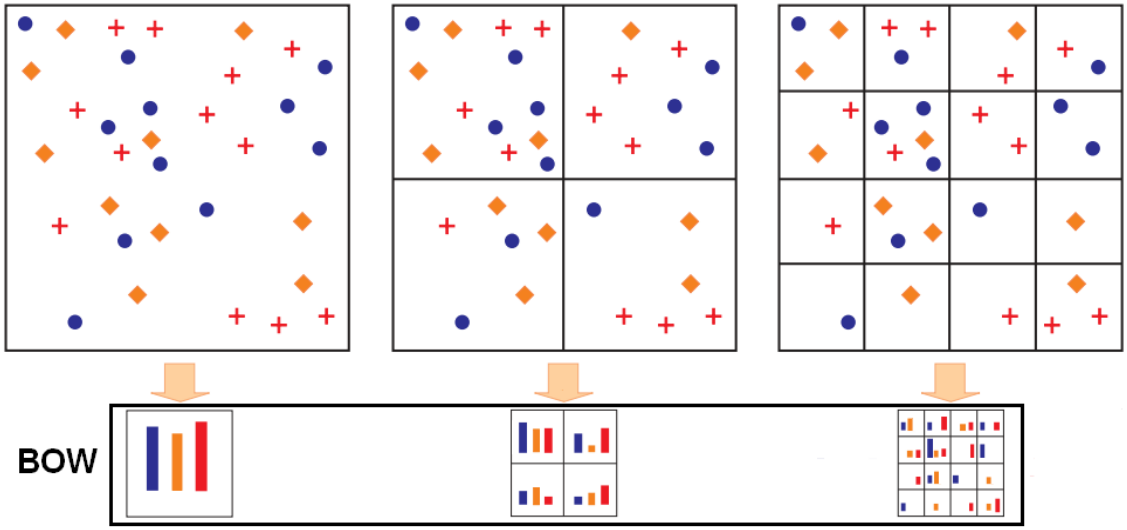
Obrázek 2.8: Obrázek (a) ukazuje slovník s Gaussovým jádrem. Na obrázku (b) je znázorněno přiřazení kódových slov příznakům zobrazeným na obrázku (a) a to jak pro klasický slovník, tak pro slovník s jádrem tvaru Gaussovy křivky. Příznaky jsou zobrazeny jako trojúhelník, čtvereček a kosočtverec. Obrázek je převzat z [5].

V praxi se pro zařazování obrázků do sémantických tříd se používají většinou slovníky s jádrem. Pokud však je použit slovník bez jádra, tak mají výsledné systémy menší spolehlivost [5].

2.2.2 Bag of words

Stejně jako jsou jednotlivé příznaky reprezentovány kódovými slovy, tak musí být nějak reprezentovaný celý vstupní obrázek. Lze použít reprezentaci pomocí Bag of words (BOW). BOW je vlastně histogram udávající počet výskytu jednotlivých kódových slov v obrázku. Tento histogram je předáván jako vektor, kde každý prvek označuje počet výskytu daného kódového slova. Takže pro slovník s velikostí 4000 kódových slov bude tento vektor mít velikost 4000.

Při použití dělení obrazu do několika oblastí, jak je popsáno v části 2.1.2, lze BOW sestavit konkatenací BOW pro jednotlivé úrovně obrázku. Je to naznačeno na obrázku 2.9, přičemž každý BOW je vynásoben koeficientem jeho váhy ve výsledku. Čím vyšší úroveň, tím větší váha příznaků dané úrovně [11].



Obrázek 2.9: Vznik BOW z obrázku při použití prostorové pyramidy.

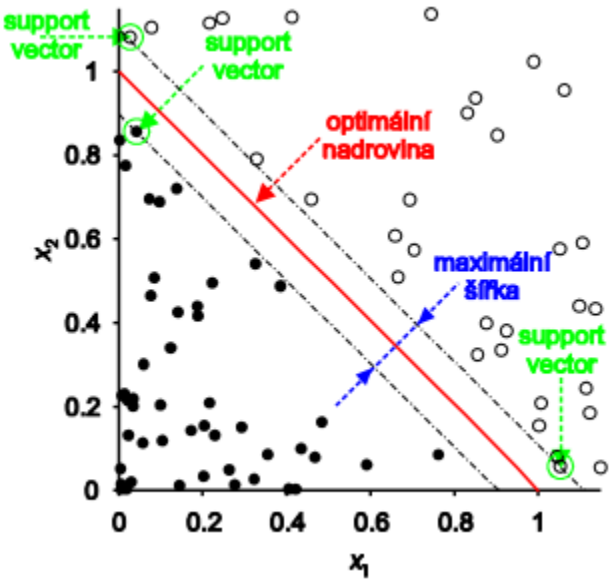
2.3 Klasifikace

Poslední část u automatického označování obrázků je jejich klasifikace do jednotlivých tříd na základě BOW, který popisuje obsah obrázku. Při klasifikaci je důležitá přesnost výsledného klasifikátoru. Studie [3] ukazuje vhodnost SVM klasifikátorů pro klasifikaci obrázkových dat. SVM klasifikátory jsou také velmi často používány v soutěži PASCAL VOC [24] a Trecvid [17]. Princip klasifikátoru SVM je popsán v kapitole 3.

Kapitola 3

SVM

Support vector machines (SVM) je sada metod učení s učitelem, které jsou používány pro klasifikaci. Základní myšlenkou SVM je rozdělit prostor příkladů nadrovinou na dva podprostory [3], kde v každém podprostoru se nachází příklady stejné třídy. Nejde jen o to najít jakoukoliv nadrovinu, ale takovou, jaká je naznačená na obrázku 3.1 jako optimální nadrovina. Pro konstrukci této nadroviny jsou důležité body, označené jako support vector, které definují optimální nadrovinu.



Obrázek 3.1: Prostor rozdělený nadrovinou na dva podprostory. Každý z těchto podprostorů obsahuje body z jedné třídy. Obrázek je převzat z [28].

3.1 Lineární SVM

Máme N trénovacích dat, kde každý vstup x_i má D atributů (tzn. je rozměru D) a každý vstup x_i je v jedné ze dvou tříd $y_i \in \{-1, 1\}$. Cílem je definovat takovou nadrovinu, která dělí vstupní příklady tak, že všechny příklady z jedné třídy jsou na jedné straně a příklady

z druhé třídy jsou na straně druhé. Nadrovina lze popsat jako: $w \cdot x + b = 0$, kde w je normála nadroviny. To odpovídá nalezení w a b takových, že:

$$y_i \left(w \cdot x_i + b \right) > 0, \ i = 1, ..., N. \tag{3.1}$$

Pokud taková nadrovina existuje, tak můžeme říct, že jsou příklady lineárně separovatelné. Potom můžeme nastavit w a b tak, že:

$$\min_{1 \leq i \leq N} y_i \left(w \cdot x_i + b \right) \geq 1, \ i = 1, ..., N.$$

Vzdálenost mezi nejbližším bodem a nadrovinou je $\frac{1}{\|w\|}$. Potom se dá rovnice 3.1 zapsat jako:

$$y_i \left(w \cdot x_i + b \right) \geq 1. \tag{3.2}$$

Pokud je vzdálenost nejbližšího bodu od nadroviny maximální, tak se taková nadrovina nazývá optimální dělící nadrovina (OSH - optimal separating hyperplane). A protože je vzdálenost nejbližšího bodu rovna $\frac{1}{\|w\|}$, odpovídá nalezení optimální dělící nadroviny minimalizování $\|w\|^2$ takového, že splňuje rovnici 3.2. Toho může být dosaženo použitím Lagrangeových multiplikátorů.

Pokud označíme $\alpha = (\alpha_1, ..., \alpha_n)$ jako N Lagrangeových multiplikátorů, kde $\alpha_i \geq 0$, omezených rovnicí 3.2. Můžeme náš optimalizační problém zapsat jako maximalizaci:

$$W(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j x_i \cdot x_j, \tag{3.3}$$

kde $\alpha_i \geq 0$ a s omezující podmínkou $\sum_{i=1}^{N} y_i \alpha_i = 0$. Toto může být dosaženo použitím standardních metod kvadratického programování.

Pokud najdeme vektor $\alpha^0 = \left(\alpha_1^0, ..., \alpha_N^0 \right)$, který je řešením maximalizačního problému, popsaného rovnicí 3.3, je OSH(w_0, b_0) daná následujícím rozvojem:

$$w_0 = \sum_{i=1}^{N} \alpha_i^0 y_i x_i. \tag{3.4}$$

Vzhledem k rozvoji 3.4 může být rozhodovací funkce nadroviny zapsána jako:

$$f(x) = sgn \left(\sum_{i=1}^{N} \alpha_i^0 y_i x_i \cdot x + b_0 \right). \tag{3.5}$$

Optimální dělící nadrovinu lze najít jen v případě, že jsou body lineárně rozdělitelné. Reálná trénovací data však bývají jen zřídka takto lineárně separovatelná [3][4].

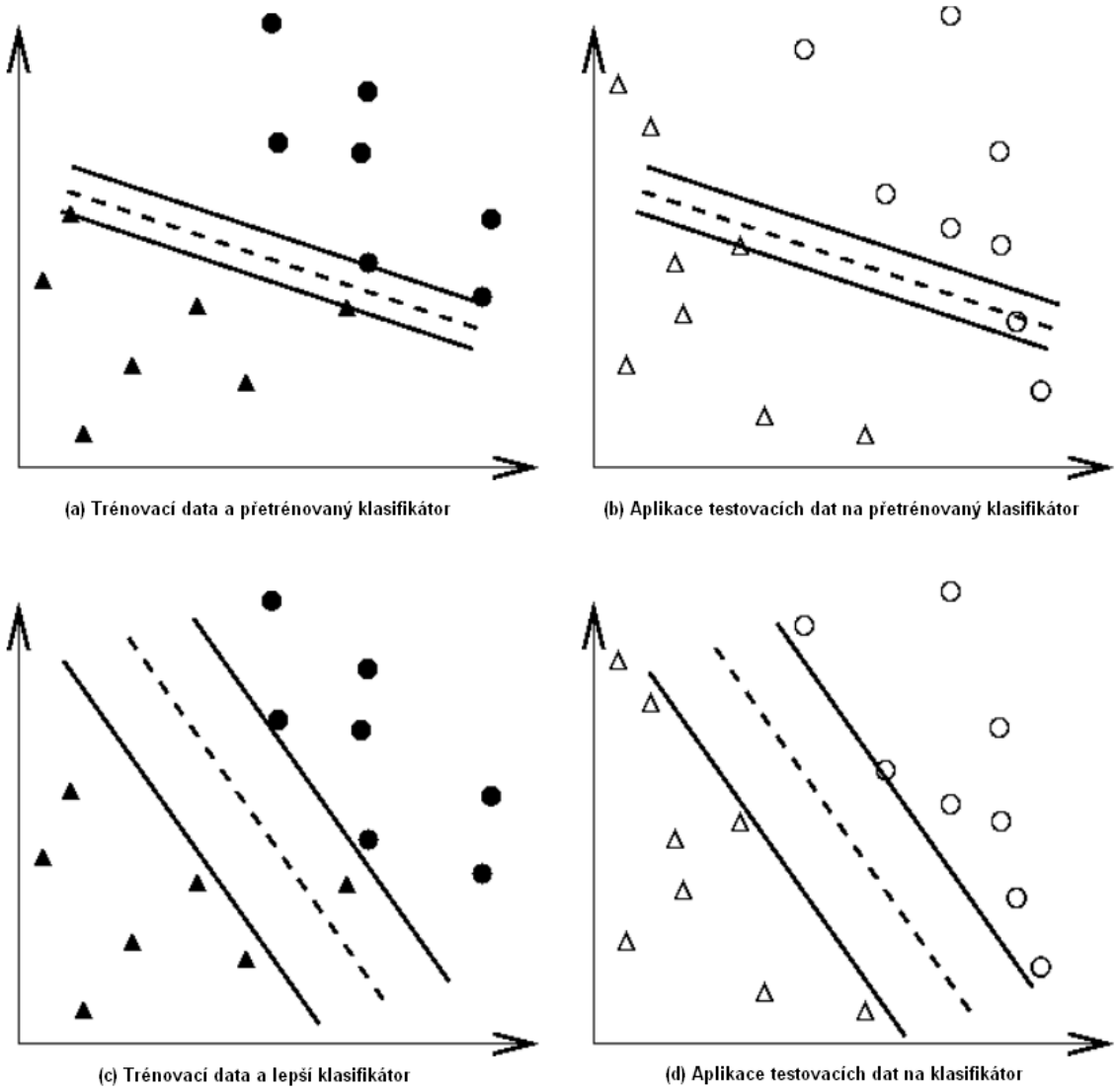
Když data nejsou lineárně separovatelná, tak lze použít volné proměnné $(\xi_1, ..., \xi_N)$, kde $\xi_i \geq 0$ takové, že:

$$y_i \left(w \cdot x_i + b \right) \geq 1 - \xi_i \ , \quad i = 1, ..., N, \tag{3.6}$$

tyto proměnné umožňují příklady, které nesplňují rovnici 3.2. Účelem proměnných ξ_i je umožnit body v trénovací sadě, které leží na špatné straně OSH. Tyto body mají svoje odpovídající $\xi_i > 1$. Tedy $\sum \xi_i$ je horní hranice počtu trénovacích chyb. Jako OSH je potom považováno řešení následujícího minimalizačního problému:

$$\frac{1}{2} w \cdot w + C \sum_{i=1}^{N} \xi_i, \tag{3.7}$$

omezeného rovnicí 3.6 a podmínkou: $\xi_i \geq 0$. Parametr C je zadáván při trénování klasifikátoru. Větší hodnota C odpovídá vyššímu postihu pro body mimo svoji třídu [3]. Pokud je ovšem hodnota C příliš velká, tak může dojít k přetrénování a nalezení neoptimální dělící nadroviny. Optimální hodnota parametru C není pro všechny data stejná, ale je nutno ji zjistit např. pomocí cross-validace. Příklad přetrénování klasifikátoru je znázorněn na obrázku 3.2.

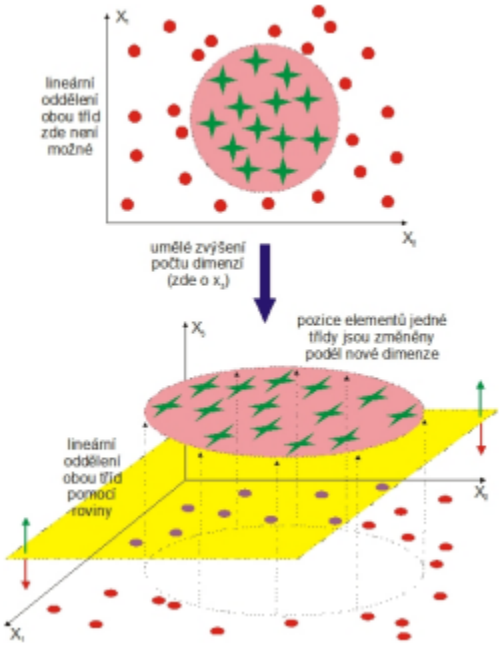


Obrázek 3.2: Příklad přetrénování klasifikátoru. Obrázek je převzat z [9].

3.2 Nelineární SVM

SVM rozdělují trénovací data lineárně. Pokud data nejsou lineárně rozdělitelné, tak na nich samozřejmě lze natrénoval lineární SVM klasifikátor, ale nebude mít velkou přesnost. I tento případ se dá vyřešit. A to tak, že se namapují trénovací data do vícerozměrného pro-

storu, kde jsou lineárně separovatelná [4]. Příklad takového namapování do vícerozměrného prostoru je na obrázku 3.3. Ve vícerozměrném prostoru se poté nalezne optimální dělící nadrovina.



Obrázek 3.3: Princip vzniku možnosti lineárního oddělení dvou tříd s nelineárními hranicemi pomocí přidané dimenze. Obrázek je převzat z [28].

Pokud nahradíme x jeho namapováním do vícerozměrného prostoru $\Phi(x)$, potom můžeme rovnici 3.3 zapsat jako:

$$W(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j \Phi(x_i) \cdot \Phi(x_j). \tag{3.8}$$

Pokud máme jádro $K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j)$, potom nám stačí pouze jádro K pro trénování klasifikátoru a mapování Φ není potřeba. A právě naopak pokud je dáno symetrické kladné jádro $K(x, y)$, pak podle Mercerova teorému [16] existuje mapování Φ takové, že $K(x, y) = \Phi(x) \cdot \Phi(y)$. Po vybrání jádra splňující podmínky Mercerova teorému, je trénovací algoritmus minimalizací:

$$W(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \tag{3.9}$$

a rozhodovací funkce má následující tvar:

$$f(x) = sgn\left(\sum_{i=1}^{N} \alpha_i y_i K(x_i, x) + b \right). \tag{3.10}$$

[3]

Takže vidíme, že k trénování klasifikátoru stačí jeho jádro. Toto jádro se dá spočítat dopředu a uložit jako matice. Trénování klasifikátoru je potom velmi rychlé.

Následují některé používané jádra. RBF (Radial Basis Kernel) jádro:

$$K(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right), \tag{3.11}$$

polynomiální jádro:

$$K(x, y) = (x \cdot y + a)^b, \tag{3.12}$$

esovité jádro:

$$K(x, y) = \tanh(ax \cdot y - b). \tag{3.13}$$

Kde a a b jsou parametry definující chování jádra a parametr σ udává šířku Gaussovi funkce [4]. Parametry jádra a, b a σ ovlivňují přesnost výsledného klasifikátoru. Optimální hodnoty těchto parametrů jsou závislé na datech a je nutné je najít např. pomocí cross-validace.

Kapitola 4

SGE

Sun Grid Engine (SGE) je software pro řízení výpočtů na výpočetním gridu. Tento software byl původně open-source, jehož vývoj zajišťovala firma Sun Microsystems. V současné době firmu Sun Microsystems vlastní firma Oracle, která přestala vydávat SGE jako open-source. Nyní jsou všechny novější verze prodávány jako komerční programy.

4.1 Výpočetní grid

Grid je kolekce výpočetních zdrojů, které vykonávají různé úlohy. Ve své nejjednodušší podobě je to z pohledu uživatele velký systém, který poskytuje jednotný přístup k výkonným distribuovaným zdrojům. Ve své složitější podobě může grid poskytovat mnoho přístupových bodů pro uživatele. Ve všech případech uživatelé používají grid jako jednotný výpočetní zdroj. Software pro řízení zdrojů, např. jako je software SGE, přijímá úlohy předložené uživateli. Software používá svoji politiku hospodaření se zdroji pro plánování běhu úloh na vhodných systémech v gridu. Uživatelé mohou tedy předložit miliony pracovních úloh najednou, aniž by věděli nebo potřebovali vědět, kde budou tyto úlohy zpracované.

Tři základní druhy gridů, které pokrývají oblast od jediného systému po superpočítačové farmy používající tisíce procesorů, jsou následující:

Cluster gridy je nejjednodušší druh. Cluster gridy jsou složeny ze skupiny počítačů, které pracují společně. Cluster grid má jeden přístupový bod pro uživatele a je určen pro jeden projekt.

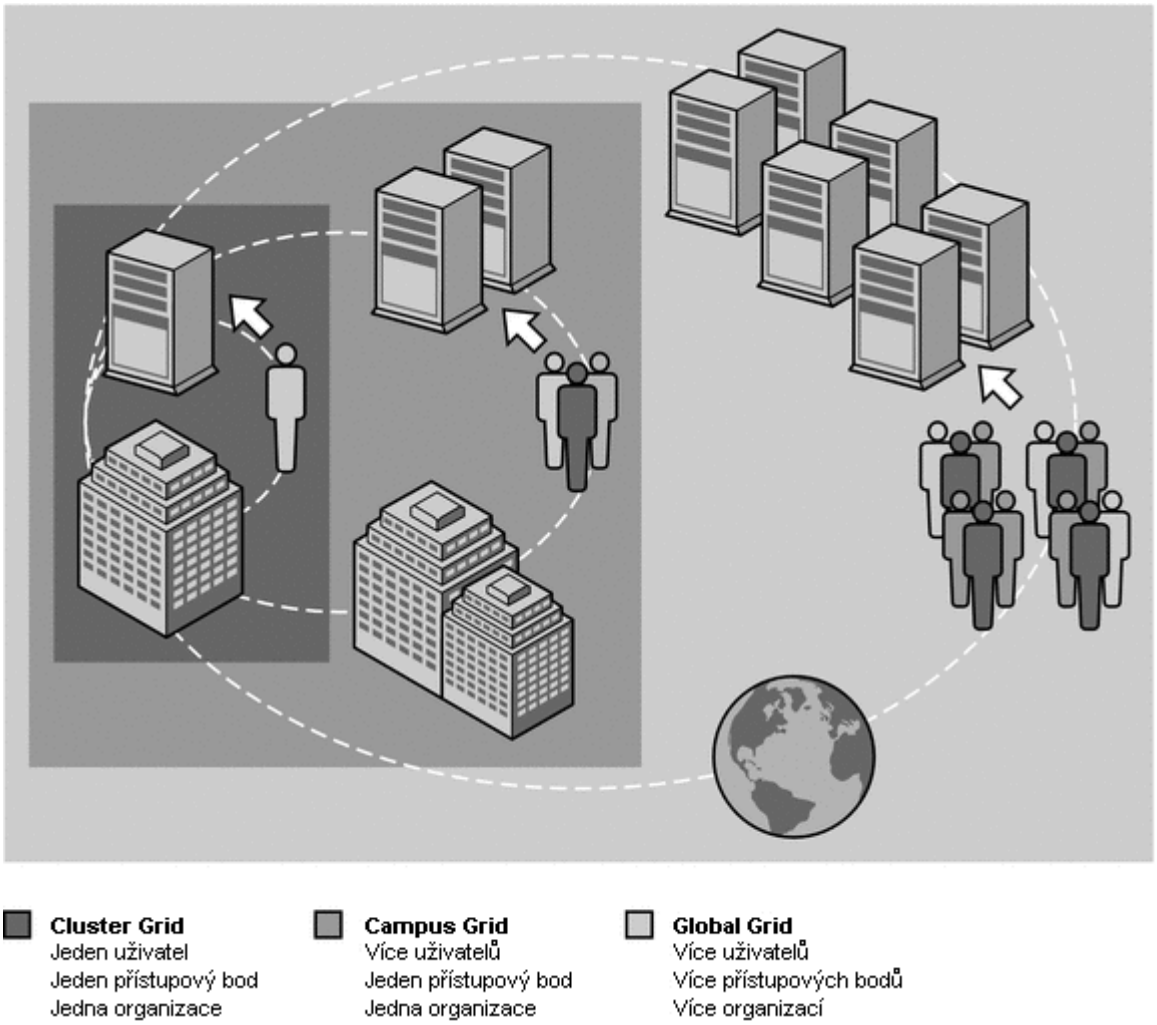
Campus gridy umožňuje souběžnou práci několika různých projektů v rámci organizace pro sdílení výpočetních zdrojů.

Global gridy je kolekce campus gridů, které překračují hranice jedné organizace, vytvářející rozsáhlé virtuální systémy. Uživatelé mají přístup k výpočetnímu výkonu, který zdaleka přesahuje zdroje, které jsou k dispozici v rámci jejich organizace.

Tyto třídy jsou naznačeny na obrázku 4.1. V cluster gridu je uživatelská úloha zpracovávána pouze jedním ze systémů v clusteru. Ale cluster grid může být součástí více komplexního campus gridu a ten muže být částí rozsáhlého global gridu. V takovém případě může být úloha zpracována jakýmkoliv výpočetním zdrojem kdekoliv na světě [22].

4.2 Základní struktura SGE

Software SGE řídí zpracování úloh na výpočetním gridu a poskytuje následující služby:



Obrázek 4.1: Obrázek ilustrující tři druhy gridů. Obrázek je převzat z [22].

- Přijímá úlohy. Úlohy jsou uživatelské požadavky na výpočetní zdroje.
- Vkládá úlohy do fronty do té doby než mohou být spuštěny.
- Přiděluje systémům úlohy z fronty k jejich provedení.
- Spravuje běžící úlohy.
- Ukládá záznamy o provedení dokončené úlohy.

[21] Tyto služby jsou poskytovány různými částmi SGE. SGE se skládá ze čtyř základních částí:

Master host řídí veškerou aktivitu clusteru. Kontroluje všechny systémy plánování a komponenty jako jsou fronty a úlohy. Dále udržuje tabulky o stavu komponent, uživatelských právech ap. Ve výchozím nastavení je master host zároveň administration host a submit host.

Execution host je systém, který má právo vykonávat úlohy. Úlohy z fronty jsou přiřazovány jednotlivým execution host systémům k jejich vykonání.

Administration host je systém, který se stará o všechnu administrativní činnost SGE.

Submit host umožňují pouze předkládat a kontrolovat úlohy.

Každý systém, který je součástí gridu, může složen z jedné a více částí [20].

Kapitola 5

SVM v praxi

Tato kapitola se zabývá trénováním SVM klasifikátorů a jejich testováním na školních serverech pomocí softwaru SGE pro řízení výpočtu.

5.1 Datové sady

Každý SVM klasifikátor pro svoje učení potřebuje trénovací datovou sadu a na otestování již natrénovaného klasifikátoru je nutná testovací datová sada. Já jsem pro všechny experimenty použil část datové sady pro Trecvid 2011 [17]. Tato sada obsahuje 20000 obrázků pro trénování a 50000 obrázků pro testování.

Pro všechny experimenty budu používat proměnou `$SGE_TOOLS` jako cestu k adresáři SGE-tools se skripty na spouštění experimentů. Všechny experimenty budou považovat za svůj kořenový adresář ten, ve kterém byly spuštěny. Pro každý experiment by to měl být jiný. V přiložených kódech jsou také příklady konfiguračních souborů. Tyto konfigurační soubory jsou komentované, takže by neměl být problém s jejich nastavením v případě opakovaní experimentů.

5.1.1 Sémantické třídy

Všechny obrázky v těchto datových sadách mohou být zařazeny do jedné nebo více sémantických kategorií. Původní datová sada pro Trecvid 2011 obsahuje 500 různých kategorií. Vzhledem k tomu, že jsem použil jen její část, tak jsem nemohl trénovat všech 500 tříd obrázků, ale jen ty, pro které je dostatek kladných i záporných vzorů pro trénování. Celkem tedy moje datová sada obsahuje 55 různých sémantických kategorií. Přehled těchto kategorií i s počty pozitivních a negativních příkladů je v tabulce 5.1.

Tabulka 5.1: Sémantické třídy v datových sadách. P je počet pozitivních vzorků, N je počet negativních vzorků a % podíl pozitivních vzorků pro danou třídu v procentech

Kategorie	**Trénovací data**			**Testovací data**		
	P	N	%	P	N	%
Actor	299	7538	3,82	680	20206	3,26
Adult	968	6189	13,53	4172	14470	22,38

Pokračování na další straně

Tabulka 5.1 – pokračování z předchozí strany

Kategorie	Trénovací data			Testovací data		
	P	N	%	P	N	%
Animal	204	9348	2,14	336	22935	1,44
Building	606	4066	12,97	1449	11845	10,90
Car	172	10770	1,57	562	27448	2,01
Carnivore	128	9431	1,34	109	23147	0,47
Celebrity_Entertainment	330	7247	4,36	1186	18801	5,93
City	144	4758	2,94	267	13943	1,88
Cityscape	151	4704	3,11	274	13816	1,94
Computer_Or_Television_Screens	659	2392	21,60	1824	5738	24,12
Crowd	106	7761	1,35	461	20959	2,15
Dark-skinned_People	110	7641	1,42	1470	18602	7,32
Daytime_Outdoor	882	3639	19,51	2541	10988	18,78
Domesticated_Animal	125	9373	1,32	128	23036	0,55
Doorway	169	3056	5,24	334	8652	3,72
Entertainment	628	2617	19,35	2279	6676	25,45
Explosion_Fire	126	2962	4,08	775	7368	9,52
Face	1718	5377	24,21	4502	14105	24,20
Female_Person	660	6721	8,94	370	24518	1,49
Female-Human-Face-Closeup	121	9396	1,27	2051	17777	10,34
Graphic	107	103	50,95	196	397	33,05
Ground_Vehicles	373	9286	3,86	1006	23774	4,06
Hand	147	3132	4,48	565	7628	6,90
Chair	101	2779	3,51	169	8052	2,06
Child	194	5465	3,43	536	14344	3,60
Indoor	901	4607	16,36	3401	12225	21,77
Instrumental_Musician	282	7462	3,64	700	20308	3,33
Landscape	361	4284	7,77	803	12642	5,97
Male_Person	1051	6060	14,78	3799	14841	20,38
Mammal	128	9371	1,35	157	23030	0,68
Mountain	125	5144	2,37	194	15063	1,27
News_Studio	162	6835	2,32	372	19251	1,90
Outdoor	2890	1756	62,20	7540	5862	56,26
Overlaid_Text	871	1820	32,37	3053	4800	38,88
Person	5093	3319	60,54	15488	7535	67,27
Plant	651	4663	12,25	1605	11616	12,14
Politics	155	2643	5,54	357	7479	4,56
Quadruped	128	9371	1,35	156	23030	0,67
Reporters	125	7607	1,62	372	20854	1,75
Road	331	4469	6,90	1063	12599	7,78
Scene_Text	265	2870	8,45	591	6965	7,82
Singing	387	7108	5,16	1504	18542	7,50
Single_Person	1072	6223	14,69	2270	17364	11,56
Sitting_Down	219	7328	2,90	653	20189	3,13

Pokračování na další straně

Tabulka 5.1 – pokračování z předchozí strany

Kategorie	Trénovací data			Testovací data		
	P	N	%	P	N	%
Sky	617	4005	13,35	1914	11447	14,33
Streets	358	4556	7,29	932	12633	6,87
Suburban	546	4116	11,71	1098	12301	8,19
Teenagers	100	8441	1,17	190	19838	0,95
Trees	897	1977	31,21	2541	4935	33,99
Two_People	150	8486	1,74	551	19990	2,68
Vegetation	1146	2271	33,54	2843	5884	32,58
Vehicle	604	7778	7,21	1620	20115	7,45
Walking	160	9425	1,67	429	24861	1,70
Walking_Running	286	7937	3,48	1051	21153	4,73
Waterscape_Waterfront	189	4715	3,85	602	13480	4,27

5.1.2 Použité BOW

Když mám datové sady na trénování a testování, je nutné mít obrazové příznaky pro trénování klasifikátorů. Příznaky z jednotlivých obrázků jsem nevytvářel. Použil jsem již připravené Bag of Words (BOW). Pro všechny experimenty jsem použil BOW uvedené v tabulce 5.2. V této tabulce je i uvedena zkratka pro každý BOW, kterou budu používat dále u výsledků experimentů. Tvorbou jednotlivých BOW se nebudu zabývat, protože programy pro jejich tvorbu jsou již hotové a používané na naší fakultě. Stejně tak nebude ve srovnání zahrnuta časová a paměťová náročnost tvorby BOW, protože z hlediska klasifikace příznaků nemá význam a vzhledem k tomu, že jsou použité stejné BOW pro všechny přístupy, tak by byla časová a paměťová náročnost u všech přístupů větší o stejnou konstantu.

Jméno BOW	Popis	Zkratka
HARLAP_SIFT_NoA.KM–L12.UNC–K32–o10–L01	Harris–Laplace, SIFT	HS
DENSE8_SIFT_NoA.KM–L12.UNC–K32–o10–L01	Mřížka průměr 8px, SIFT	D8S
DENSE16_SIFT_NoA.KM–L12.UNC–K32–o10–L01	Mřížka průměr 16px, SIFT	D16S
HARLAP_CSIFT_NoA.KM–L12.UNC–K32–o10–L01	Harris–Laplace, CSIFT	HCS
DENSE8_CSIFT_NoA.KM–L12.UNC–K32–o10–L01	Mřížka průměr 8px, CSIFT	D8CS
DENSE16_CSIFT_NoA.KM–L12.UNC–K32–o10–L01	Mřížka průměr 16px, CSIFT	D16CS

Tabulka 5.2: Použité BOW v experimentech, jejich základní popis, který obsahuje použitý detektor oblastí a příznak, a také zkratku, která bude použita dále ve výsledcích experimentů. Pro tvorbu všech BOW byl použit slovník s jádrem.

5.2 Lineární SVM

Jako první vyzkoušíme přesnost lineárních SVM klasifikátorů na daných datových sadách a s danými BOW. Potřebné programy na trénování a testování lineárních SVM klasifikátorů jsem převzal od Michala Hradiše, stejně jako pomocné skripty na SGE. Já jsem jen napsal skripty pro spuštění na školním SGE a skripty pro vyhodnocení výsledků.

5.2.1 Trénování lineárních SVM

Ke spuštění experimentu se používá skript `runLinear.sh`. Typické použití je pak následující:

```
$SGE_TOOLS/runLinear.sh -c CONF -b BB
```

Kde `BB` je soubor obsahující seznam použitých BOW a `CONF` je konfigurační soubor obsahující veškeré nastavení pro experiment. Od jmen data setů na trénování a testování, přes parametry pro samotné trénování SVM klasifikátorů až po nastavení parametrů pro SGE jako maximální potřebná paměť a konečné uzly, na kterých můžou být úlohy spuštěny. Po přečtení této konfigurace skript provádí postupně následují činnosti:

- Vytvoření adresářové struktury.
- Vytvoření konfiguračních souborů pro trénování a testování lineárních SVM klasifikátorů.
- Spuštění úloh pro trénování a testování lineárních SVM klasifikátorů na školním SGE.

Samotný běh úloh na SGE je rychlý a bez jakýchkoliv problémů. Při mých pokusech jsem nenarazil na žádné omezení ze strany SGE. Na SGE jsou spuštěny dvě úlohy. Jedna na trénování SVM klasifikátorů a druhá na jejich testování. Každá z těchto úloh je spuštěna pro každý klasifikátor jednou, to je celkem 330 krát. Vzhledem k tomu, že nastavený limit na školním SGE je 100000, tak s tímto omezením není žádný problém. Další omezení školního SGE je maximální doba běhu skriptu a to 4 hodiny. V tomto případě není ani s tím žádný problém, protože jak trénování tak testování SVM klasifikátorů je hotové v řádu minut.

5.2.2 Výsledky lineárních SVM

Po dokončení úloh na SGE je dobré vědět výsledek experimentu. Každý test vytvoří výstupní soubor s výsledky experimentu. Vzhledem k tomu, že je to celkem nepraktické takhle kontrolovat výsledky, tak jsem napsal skript, který tyto výstupní soubory projde a vytvoří jeden soubor s přehlednou statistikou, která obsahuje přesnost a chybovost jednotlivých klasifikátorů. Tyto výsledky jsou potom zobrazeny v tabulce 5.3. Skript se spouští následujícím příkazem:

```
$SGE_TOOLS/makeLinearStats.sh -c CONF -b BB
```

Kde `CONF` je stejný konfigurační soubor jako při trénování klasifikátorů. Je nutný, protože obsahuje sémantické kategorie, pro které byly trénovány SVM klasifikátory.

5.2.3 Fúze lineárních SVM

Nyní mám celkem 6 různých SVM klasifikátorů pro jednu sémantickou třídu. Když vezmu jejich průměr, tak mám celkovou přesnost 20,16%, což není úplně moc. Tuto přesnost můžu zlepšit, pokud nebudu brát průměr, ale udělám fúzi těchto klasifikátorů. Fúzi dělám opět lineárním SVM klasifikátorem a to tak, že vezmu odezvy všech klasifikátorů pro jednotlivé testovací obrázky. Z těchto odezev udělám nový příznakový vektor a na něm lze natrénovat lineární SVM klasifikátor stejně jako na obrazových příznacích. Fúzi spouštím skriptem:

```
$SGE_TOOLS/runFusion.sh -c CONF_FUSION -b BB
```

BOW	AvgP	EER
HS	0,1964	0,3866
D8S	0,1907	0,3867
D16S	0,2046	0,3782
HCS	0,2016	0,3819
D8CS	0,2050	0,3769
D16CS	0,2115	0,3721
průměr	0,2016	0,3804

Tabulka 5.3: Výsledky experimentů lineárních SVM klasifikátorů na zvolené datové sadě. V prvním sloupci je použitý BOW, ve druhém průměrná přesnost a v posledním je průměrná chybovost pro daný BOW. Na posledním řádku je potom průměr těchto hodnot.

Soubor `CONF_FUSION` je podobný konfigurační soubor jako pro trénování. Může však obsahovat jiné nastavení jednotlivých parametrů pro SVM klasifikátory. Druhý soubor obsahuje seznam použitých BOW pro fúzi.

Po spuštění se skript postará o celou fúzi. Jeho základní funkčnost lze popsat jako následující posloupnost úkolů:

- Vytvoření adresářové struktury pro fúzi.
- Spuštění úloh pro vytvoření nového příznakového vektoru na školním SGE.
- Spuštění úloh pro trénování a testování lineárních SVM klasifikátorů na školním SGE.

Po skončení výpočetních úloh na SGE lze udělat výslednou statistiku skriptem:

```
$SGE_TOOLS/makeFusionStats.sh -c CONF_FUSION
```

Jak ukazuje tabulka 5.4, tak pomocí fúze lze zlepšit konečný výsledek lineárních SVM klasifikátorů. Zlepšení není nijak výrazné, ale vzhledem k tomu, že je výpočetně nenáročné, tak je to lepší než používat jen průměr natrénovaných klasifikátorů. Takže nakonec jsem dosáhl přesnost 21,73% s lineárními klasifikátory na zvolené datové sadě.

Jiné a lepší zlepšení přesnosti lze dosáhnout použitím více různých BOW. Tím však nejspíše končí schopnosti lineárních SVM klasifikátorů pro klasifikaci obrázků do sémantických tříd podle jejich obsahu. A proto jsem se podíval na nelineární SVM klasifikátory a jejich možnosti.

	AvgP	EER
průměr	0,2016	0,3804
fúze	0,2173	0,3627

Tabulka 5.4: Výsledky fúze lineárních SVM klasifikátorů na zvolené datové sadě. Ve druhém sloupci je průměrná přesnost a v posledním je průměrná chybovost. Na prvním řádku je uveden průměr přesností klasifikátorů pro srovnání s fúzí. Lepší výsledky jsou zelenou barvou.

5.3 Nelineární SVM

Po nepříliš velkém úspěchu s lineárními SVM klasifikátory jsem se rozhodl použít nelineární SVM klasifikátory. Princip nelineárního SVM je popsán v kapitole 3.2. Jde o to převést nelineární problém na problém lineární. Pro trénování nelineárních SVM klasifikátorů je nutné tzn. jádro, což je právě ta transformace příznaků do vícerozměrného prostoru. Pro můj experiment jsem zvolil RBF jádro, které lze popsat následující rovnicí:

$$K(x,y) = \exp\left(-\frac{\chi^2}{2\sigma^2}\right) , \tag{5.1}$$

kde χ^2 je míra podobnosti vzorků, která je definováno jako:

$$\chi^2(x,y) = \sum_{i=1}^{D} \frac{(x_i - y_i)^2}{x_i + y_i} . \tag{5.2}$$

Protože je nastavení parametru σ závislé na datech, je nutné jeho určení ze vstupních dat. K tomu účelu je ve vlastním jádru spočítané jen χ^2 a vhodnou velikost parametru σ hledám pomocí cross-validace. Program pro počítání těchto jader jsem převzal od Michala Hradiše.

Pokud zavedu proměnnou γ definovanou jako:

$$\gamma = \frac{1}{2\sigma^2} , \tag{5.3}$$

mohu zapsat rovnici 5.1 jako:

$$K(x,y) = \exp\left(-\gamma\chi^2\right) . \tag{5.4}$$

5.3.1 Optimalizace parametrů nelineárních SVM

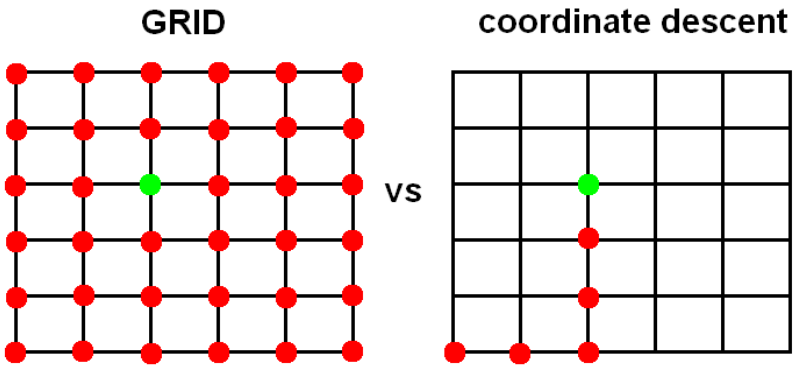
Po zavedení parametru γ je problém trénování nelineárních SVM klasifikátorů hledání správné hodnoty parametru γ a parametru C zavedeném v rovnici 3.7. Pro hledání hodnot těchto parametrů využívám cross-validace. Velikost parametrů generuji dvěma způsoby z předem daného intervalu hodnot. Jednak generováním všech možných kombinací hodnot (GRID) a pak také postupem coordinate descent, kdy optimalizuji jeden parametr. Až najdu nejlepší hodnotu, tak optimalizuji druhý. Oba postupy jsou zobrazeny na obrázku 5.1. Později mé experimenty ukázaly, že oba postupy dávají srovnatelné výsledky. Tyto výsledky jsou v tabulce 5.5, a proto používám už jen coordinate descent, jehož průměrná doba potřebná pro trénování je asi 4 krát menší než s využitím metody GRID, jak ukazuje graf na obrázku 5.2.

5.3.2 Trénování nelineárních SVM

Celý experiment lze spustit jedním skriptem:

```
$SGE_TOOLS/runNonLinear.sh -c CONF -b BB
```

Skript má stejné parametry jako skript pro trénování lineárních SVM klasifikátorů, ale obsah konfiguračního souboru `CONF` je samozřejmě trochu jiný. Obsahuje hodnoty parametrů pro nelineární SVM klasifikátory. Po jejich načtení se dá skript popsat následující posloupností činností:



Obrázek 5.1: Obrázek ilustrující generování hodnot parametrů γ a parametru C.

BOW	coordinate descent		GRID	
	AvgP	EER	AvgP	EER
HS	0,2630	0,3260	0,2647	0,327578
D8S	0,2600	0,3249	0,2600	0,3248
D16S	0,2711	0,3219	0,2710	0,3214
HCS	0,2692	0,3174	0,2707	0,3179
D8CS	0,2675	0,3251	0,2672	0,3256
D16CS	0,2768	0,3183	0,2715	0,3201
průměr	0,2679	0,3222	0,2675	0,3229

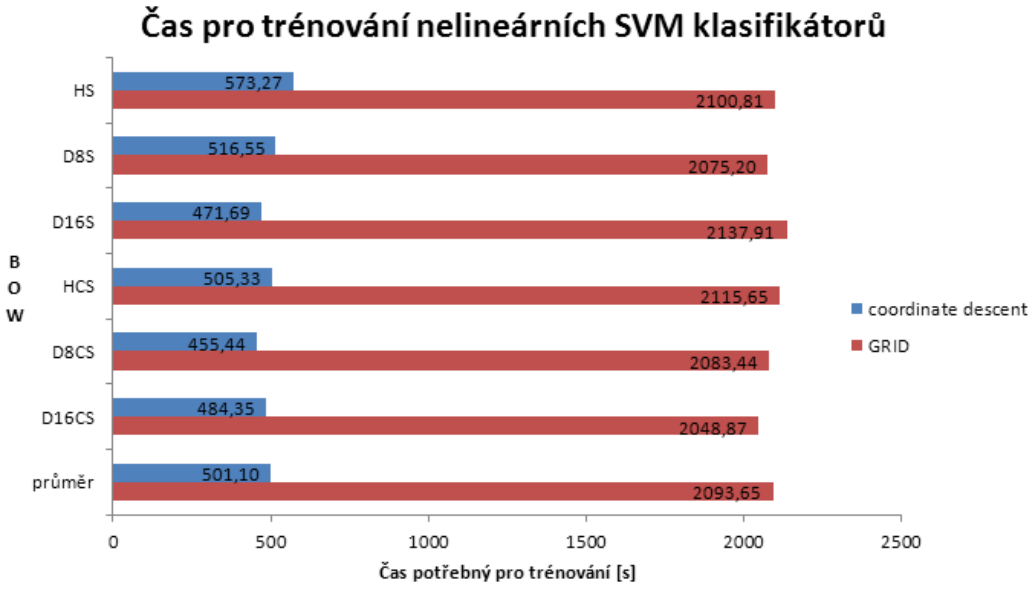
Tabulka 5.5: Porovnání výsledků experimentů nelineárních SVM klasifikátorů na zvolené datové sadě. AvgP je průměrná přesnost a EER je průměrná chybovost pro daný BOW.

- Vytvoření potřebné adresářové struktury.
- Spuštění úloh pro vytvoření jader.
- Spuštění úloh pro trénování a testování nelineárních SVM klasifikátorů na školním SGE.

Výpočet jader je celkem náročný. Jen pro představu jádro na testování natrénovaného klasifikátoru je matice o velikosti 20000, krát 50000 což je 1000000000 čísel. Tento výpočet lze naštěstí paralelizovat, čehož také využívám. Každý výpočet jádra je rozdělen na předem daný počet částí, které se poté spojí do jednoho jádra. Pro trénování je zapotřebí jedno jádro na každé použité BOW. Stejně tak pro testování. To znamená, že v tomto experimentu bylo zapotřebí 12 různých jader. Toto jádro se potom používá při trénování, kdy je celé uložené v paměti RAM. Z toho vyplývá, že je tento přístup celkem náročný na paměť a pro datové sady o statisících obrázcích prakticky nepoužitelné, ale pro můj experiment je to celkem vhodné řešení.

5.3.3 Výsledky nelineárních SVM

Stejně jako u lineárních SVM klasifikátorů se generují výstupní soubory s výsledky trénování a testování. Pro zpracování těchto výsledků slouží skript:



Obrázek 5.2: Průměrná doba trénování nelineárních SVM klasifikátorů v závislosti na použitém BOW.

```
$SGE_TOOLS/makeNonLinearStats.sh -c CONF
```

Výstup toho skriptu je prakticky shodný s výstupem u lineárních SVM klasifikátorů. Přesnost tohoto řešení je v tabulce 5.6. Z výsledků vidíme, že přesnost je lepší než u lineárních SVM klasifikátorů a opět můžeme použít fúzi těchto klasifikátoru pro dodatečné zpřesnění klasifikace. Výsledek fúze je ve stejné tabulce. Jak je patrné, tak přesnost klasifikace po fúzi je 29,89%, což je lepší výsledek než jsme schopni dosáhnout s lineárními SVM klasifikátory. Tato přesnost by opět šla zlepšit použitím více různých BOW.

BOW	**AvgP**	**EER**
HS	0,2630	0,3260
D8S	0,2600	0,3249
D16S	0,2711	0,3219
HCS	0,2692	0,3174
D8CS	0,2675	0,3251
D16CS	0,2768	0,3183
průměr	0,2679	0,3222
fúze	0,2989	0,2978

Tabulka 5.6: Výsledky experimentů nelineárních SVM klasifikátorů na zvolené datové sadě. V prvním sloupci je použitý BOW, ve druhém průměrná přesnost a v posledním je průměrná chybovost pro daný BOW. Na předposledním řádku je potom průměr těchto hodnot a na posledním výsledek fúze. Lepší výsledky jsou zelenou barvou.

5.4 Srovnání přístupů

Nyní se podíváme zpět a zhodnotíme si oba použité přístupy. Porovnáme nejen výslednou přesnost, ale i čas potřebný pro trénování a použitou paměť v průběhu trénování.

5.4.1 Porovnání přesnosti a chybovosti

Z hlediska klasifikace je nejdůležitější přesnost výsledného klasifikátoru a co nejmenší chybovost. Z tabulky 5.7 je vidět, že nelineární SVM klasifikátory mají v tomto případě lepší přesnost. Navíc pokud se zaměříme na úspěšnost fúze výsledných klasifikátorů, tak je jasně zřejmé, že je úspěšnější u nelineárních SVM klasifikátorů. U lineárních SVM klasifikátorů je zlepšení z průměru 0,2016 na výsledek fúze 0,2173 o hodnotu 0,0157. A to odpovídá zpřesnění výsledku o 7,78% oproti průměru. Zatímco u nelineárních SVM klasifikátorů je průměrná přesnost klasifikace 0,2679. Po provedení fúze je však přesnost 0,2989, což je zlepšení o 0,031 a to odpovídá zpřesnění výsledku fúzí o 11,57% oproti průměru. Po porovnání výsledných přesností je tedy v tomto případě jasná volba nelineárních SVM klasifikátorů, které mají nejen lepší výslednou přesnost a menší chybovost, ale také jsou vhodnější pro fúzi, protože díky fúzi dosahují lepšího výsledku oproti průměru.

BOW	**lineární SVM**		**nelineární SVM**	
	AvgP	**EER**	**AvgP**	**EER**
HS	0,1964	0,3866	0,2630	0,3260
D8S	0,1907	0,3867	0,2600	0,3249
D16S	0,2046	0,3782	0,2711	0,3219
HCS	0,2016	0,3819	0,2692	0,3174
D8CS	0,2050	0,3769	0,2675	0,3251
D16CS	0,2115	0,3721	0,2768	0,3183
průměr	0,2016	0,3804	0,2679	0,3222
fúze	0,2173	0,3627	0,2989	0,2978

Tabulka 5.7: Srovnání výsledků experimentů lineárních a nelineárních SVM klasifikátorů na zvolené datové sadě. V prvním sloupci je použitý BOW, AvgP značí průměrnou přesnost a EER je průměrná chybovost pro daný BOW. Na předposledním řádku je potom průměr těchto hodnot a na posledním výsledek fúze. Lepší výsledky jsou zelenou barvou.

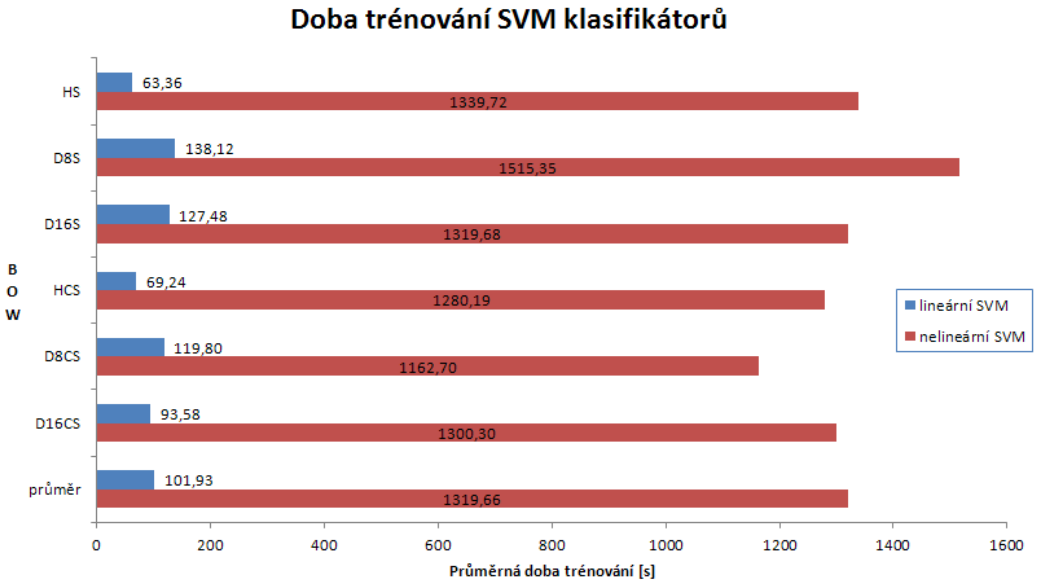
5.4.2 Porovnání doby trénování

Jako další statistika pro srovnání se nabízí časová náročnost trénování lineárních a nelineárních SVM klasifikátorů. Délka trénování lineárních SVM klasifikátorů je jasně daná, protože se trénují přímo na hotových BOW. Ale při použití nelineárních SVM klasifikátorů je jejich trénování složeno z celkem tří fází. První fáze je příprava částí jader, dále potom jejich sloučení do jednoho jádra a nakonec následné trénování. Jak je vidět v grafu na obrázku 5.3, tak průměrná doba potřebná pro natrénování lineárních SVM klasifikátorů je 101,93 vteřin, zatímco průměrná doba potřebná pro natrénování nelineárních SVM klasifikátorů je 1319,66 vteřin. Takže trénování jednoho lineárního SVM klasifikátoru je skoro 13 krát rychlejší než trénování klasifikátoru nelineárního. Z větší části za tento nepoměr může výpočet jádra, který je časově náročný. Ale i když srovnáme jen čisté trénování klasifikátorů,

tak u nelineárních SVM dostáváme průměrnou dobu potřebnou pro trénování 501,1 vteřin, což je pořád skoro 5 krát více než je potřebná doba pro nelineární SVM.

V grafu na obrázku 5.4 je vidět rozložení doby trénování nelineárních SVM klasifikátorů na čas potřebný na výpočet jádra a čas trénování vlastního klasifikátoru. Je zde patrné, že výpočet jádra trvá skoro dvakrát tak dlouho než samotné trénování. Pro výpočet jádra jsem je rozložil na 10 částí, které se počítají současně. Rozložením na více částí by tento proces nepochybně zrychlilo. Toto zrychlení je však nakonec limitováno samotným SGE, protože příliš krátké úlohy nejsou pro SGE vhodné. V krátkých úlohách se na době jejich běhu výrazně podílí plánování úloh pro běh na jednotlivých uzlech SGE.

Z tohoto srovnání vyplývá jasný vítěz a to jsou lineární SVM klasifikátory, které potřebují mnohonásobně menší čas pro svoje natrénování. Tento výsledek však šel předvídat, protože u nelineárních SVM se optimalizuje nejen parametr C jako u lineárních, ale i parametr γ a tudíž je nutné vyzkoušet více možných kombinací. Zvolený postup optimalizace coordinant descent tento proces výrazně zrychluje, jak jsem ukázal dříve, ale ani tak se nemůže rychlost trénování nelineárních SVM klasifikátorů srovnávat s rychlostí trénování klasifikátorů lineárních.



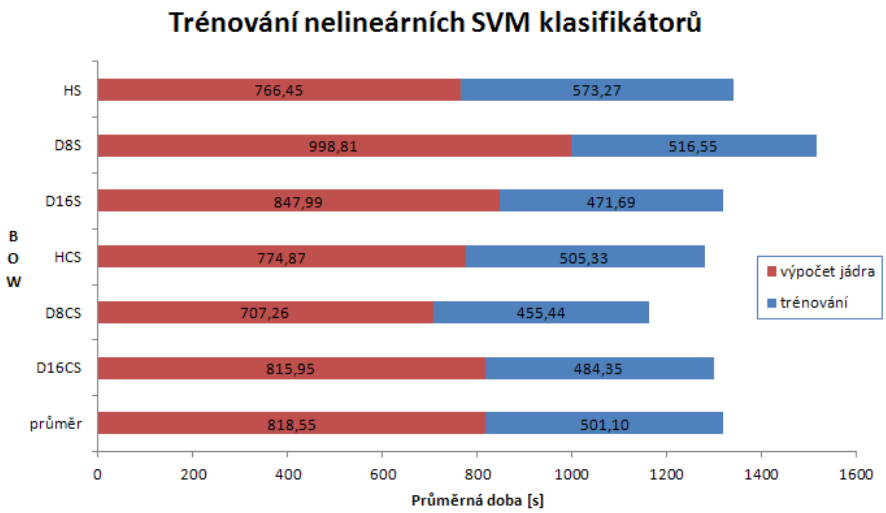
Obrázek 5.3: Průměrná doba trénování SVM klasifikátorů v závislosti na použitém BOW.

5.4.3 Porovnání paměťové náročnosti

Program pro trénování SVM zabírá určité místo v paměti RAM. Toto potřebné místo si můžeme rozdělit na dvě základní části:

- Paměť potřebná pro režii trénování.
- Paměť pro uložení trénovacích dat.

Když se podíváme blíže na tyto dvě části, tak paměť potřebná pro režii je téměř pořád stejná a velikost trénovací sady ji moc neovlivňuje. Také v mém případě, když mám v



Obrázek 5.4: Rozložení průměrné doby potřebné pro trénování nelineárních SVM klasifikátorů v závislosti na použitém BOW.

trénovací sadě 20000 různých obrázků, je její velikost zanedbatelná o proti velikosti paměti potřebné pro uložení vlastních trénovacích dat. Z toho důvodu budu porovnávat jen paměť pro vlastní uložení trénovacích dat.

Při trénování lineárních SVM klasifikátorů jsou trénovací data přímo jednotlivé BOW. Když si vyjádřím závislost potřebné paměti P_L na velikosti datové sady N tak dostanu následující vztah:

$$P_L = |BOW| \cdot p_z \cdot N \ , \tag{5.5}$$

kde $|BOW|$ je velikost jednoho BOW a p_z vyjadřuje paměť potřebnou pro uložení jednoho prvku BOW. V mém případě je dané:

$$|BOW| = 4096 \ a \ p_z = 4 \ . \tag{5.6}$$

Potom mohu psát, že paměť potřebná pro trénování lineárních SVM je:

$$P_L = 16384 \cdot N \ . \tag{5.7}$$

Nelineární SVM klasifikátory pro svoje trénování potřebují matici sestavenou z trénovacích dat. Vzhledem k způsobu jak taková matice vzniká, musí být souměrná podle diagonály. Proto můžu tuto matici rozdělit podle diagonály na dvě části a uložit jen jednu část společně s diagonálou. Potom vyjádření závislosti potřebné paměti na velikosti trénovací datové sady je následující:

$$P_L = \left(\frac{N^2}{2} + N \right) \cdot p_z \ . \tag{5.8}$$

Teď, když mám vyjádřenou paměťovou náročnost obou řešení můžu vypočítat velikost trénovací datové sady N, pro kterou jsou si obě řešení stejně paměťově náročná a určit, které je v mém případě náročnější na paměť. A to tak, že položím:

$$|BOW| \cdot p_z \cdot N = \left(\frac{N^2}{2} + N \right) \cdot p_z \ . \tag{5.9}$$

Když tuto rovnici upravím tak dostávám:

$$\frac{N^2}{2} + N\left(1 - |BOW|\right) = 0\,. \tag{5.10}$$

Po dosazení a vyřešení rovnice 5.10, dostaneme $N_1 = 0, N_2 = 8190$. Vzhledem k tomu, že používám trénovací data set o velikosti 20000 vzorků, tak je trénování nelineárních SVM klasifikátorů paměťově náročnější než trénování klasifikátorů lineárních. Navíc lineární SVM mají i lineární závislost potřebné paměti na velikosti trénovací datové sady, zatímco nelineární SVM mají tuto závislost kvadratickou. Z tohoto srovnání vychází opět lépe lineární SVM klasifikátory.

5.5 Možnost dalšího pokračování

Z výše uvedeného srovnání je patrné, že trénování nelineárních SVM klasifikátorů trvá déle a zabírá více paměti. Navzdory tomu jsou vhodnější pro tento případ klasifikace, protože mají větší přesnost a menší chybovost než klasifikátory lineární.

Obecně však nastávají případy, že je datová sada mnohem větší než používám já. Potom mají vypočtené trénovací matice velikost desítky GB. Operace s takto velkými maticemi mají celkem dva hlavní problémy:

- Výpočetní uzel musí mít dostatek volné RAM paměti pro trénování klasifikátoru.
- Musí být zajištěná dostatečná přenosová kapacita od místa uložení matice do výpočetního uzlu pro trénování.

Oba tyto problémy lze řešit pořízením dostatečného hardwaru. Toto řešení je však celkem hodně finančně náročné. A proto já bych se rád zaměřil na nalezení softwarového řešení tohoto problému. Nabízí se hned dvě cesty, kterými se mohu vydat:

Optimalizovat stávající řešení: Mohl bych optimalizovat využití matice trénovacím programem. Např. bych mohl vybrat náhodně jen podmnožinu vzorků pro trénování. V tom případě ale může být výsledný model méně kvalitní. Nebo bych mohl více využívat pevný disk místo paměti RAM. Také bych mohl optimalizovat přenos matic od místa uložení k trénovacím uzlům. To by šlo realizovat odstraněním neutrálních vzorků před přenosem pro každou sémantickou kategorii (pokud se tedy nepoužívají jako kladné nebo záporné vzorky).

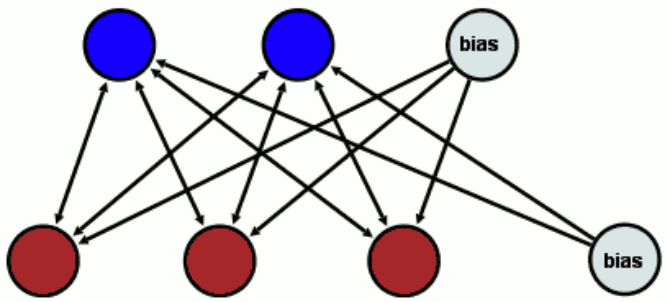
Vytvořit nové řešení na podobném principu: Mohl bych se inspirovat prvotním účelem nelineárních SVM klasifikátorů, který je převádět nelineární problém na problém lineární. Pokud tento převod provedu jiným způsoben než použitím jádra, tak budu moci využívat vysokou přesnost nelineárních SVM klasifikátorů a nízké paměťové a časové nároky klasifikátorů lineárních.

Když se vydám první cestou, tak s největší pravděpodobností částečně odstraním problémy s použitím nelineárních SVM klasifikátorů, ale pro velmi velké datové sady (stovky tisíc trénovací vzorků) bude toto řešení nedostatečné. Proto jsem se rozhodl jít druhou cestou a hledat vhodné řešení pro převádění nelineárních problémů na problémy lineární. Po dohodě s Michal Hradišem jsem zkoumal možnost natrénovat *RBM* na datové sadě a poté použít odezvy ze skryté vrstvy jako příznaky pro lineární SVM klasifikátory.

Kapitola 6

RBM

Restricted Boltzmann Machine neboli RBM je stochastická neuronová síť, která má dvě vrstvy. Jedna vrstva je viditelná a ta druhá je skrytá. Neurony v každé vrstvě nejsou propojeny, ale jsou spojeny s každým neuronem ve druhé vrstvě. Všechna spojení jsou oboustranná a symetrická přesně jak je znázorněno na obrázku 6.1.



Obrázek 6.1: Tvar neuronové sítě u Restricted Boltzmann Machine. Obrázek je převzat z [1].

6.1 Teoretický úvod k RBM

Vezměme si trénovací sadu binárních vektorů, které budeme předpokládat, že jsou binární obrázky pro účely vysvětlení. Tato trénovací sada může být modelována použitím dvou-vrstvé neuronové sítě, ve které stochastické binární pixely jsou spojeny se stochastickými binárními detektory pomocí symetricky vážených spojení. Jednotlivé pixely jsou viditelné neurony a jednotlivé detektory jsou skryté neurony. Společná konfigurace viditelných a skrytých jednotek (v, h) má energii danou:

$$E(v,h) = -\sum_{i \in viditeln} a_i v_i - \sum_{j \in skryt} b_j h_j - \sum_{i,j} v_i h_j w_{ij} \,, \tag{6.1}$$

kde v_i, h_j jsou binární stavy viditelného neuronu i a skrytého neuronu j, a_i, b_j jsou jejich biasy a w_{ij} je váha spojení mezi nimi. Síť přiřadí pravděpodobnost všem možným párům viditelným a skrytým vektorům prostřednictvím této energetické funkce:

$$p(v,h) = \frac{1}{Z} e^{-E(v,h)} \,, \tag{6.2}$$

kde "rozdělovací funkce" Z je dána součtem přes všechny možné dvojice viditelných a skrytých vektorů:

$$Z = \sum_{v,h} e^{-E(v,h)} \ . \tag{6.3}$$

Pravděpodobnost, že síť aktivuje viditelný vektor v je dána součtem přes všechny možné skryté vektory:

$$p\left(v\right) = \frac{1}{Z} \sum_{h} e^{-E(v,h)} \ . \tag{6.4}$$

Pravděpodobnost, že se síť aktivuje trénovacím obrázkem, lze zvýšit úpravou váhy a biasů a snížit tak energii tohoto obrázku a zvýšit energii ostatních obrázků, zejména těch, které mají nízkou energii a tak mohou mít velký příspěvek k rozdělovací funkci. Derivace průběhu pravděpodobnosti trénovacího vektoru s ohledem na váhy je překvapivě jednoduchá.

$$\frac{\partial \log p\left(v\right)}{\partial w_{ij}} = \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model} \tag{6.5}$$

Úhlové závorky jsou použity k označení pravděpodobnosti v rámci distribuce určené popisem, který následuje. To vede k velmi jednoduchému pravidlu učení pro provádění stochastických rychlých růstů průběhů pravděpodobnosti trénovacích dat:

$$\Delta w_{ij} = \epsilon \left(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model} \right) \ , \tag{6.6}$$

kde ϵ je rychlost učení.

Protože neexistuje spojení mezi skrytými neurony u RBM, je velmi jednoduché získat vzorek $\langle v_i h_j \rangle_{data}$ bez biasu. Když náhodně vyberu trénovací obrázek v, tak každý binární stav h_j každého skrytého neuronu j je nastaven na hodnotu 1 s pravděpodobností:

$$p\left(h_j = 1 \mid v\right) = \sigma \left(b_j + \sum_i v_i w_{ij} \right) \ , \tag{6.7}$$

kde $\sigma\left(x\right)$ je logistická esovitá funkce $1/\left(1 + \exp -x\right)$. $v_i h_j$ je potom vzorek bez biasu.

A protože neexistuje ani spojení mezi viditelnými neurony, je také velmi jednoduché získat stav viditelného neuronu bez biasu s danými skrytými neurony:

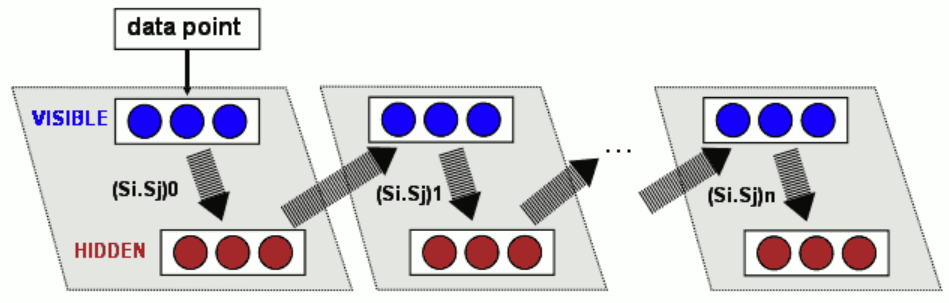
$$p\left(v_i = 1 \mid h\right) = \sigma \left(a_i + \sum_j h_j w_{ij} \right) \ . \tag{6.8}$$

Na druhou stranu získání vzorku $\langle v_i h_j \rangle_{model}$ bez biasu je mnohem složitější. Lze toho dosáhnout nastavením náhodného stavu neuronů viditelné vrstvy a potom provádění střídavého Gibbsova vzorkování po velmi dlouhou dobu. Jedna iterace Gibbsova vzorkování se skládá z aktualizace celé skryté vrstvy paralelním použitím rovnice 6.7 následovanou aktualizací viditelné vrstvy paralelním použitím rovnice 6.8.

Mnohem rychlejší postup je nastavení stavů neuronů viditelné vrstvy podle trénovacího vektoru. Potom vypočtení stavů prvků skryté vrstvy pomocí paralelního užití rovnice 6.7. Jakmile jsou nastaveny stavy neuronů skryté vrstvy, tak se provede rekonstrukce nastavením každé v_i na hodnotu 1 s pravděpodobností danou rovnicí 6.8. Změna vah spojení je potom dána jako:

$$\Delta w_{ij} = \epsilon \left(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{rekon} \right) \ , \tag{6.9}$$

Zjednodušená verze toho samého učícího pravidla, která používá stav jednotlivých neuronů místo dvojic je použita pro nastavení jednotlivých biasů. Tento druh učení RBM se nazývá Contrastive Divergence (CD). Opakování tohoto postupu lze celkem rychle natrénovat RBM síť [8]. Tento postup je naznačen na obrázku 6.2.



Obrázek 6.2: Trénování RBM použití metody Contrastive Divergence. Obrázek je převzat z [1].

6.2 RBM v praxi

Po lehkém teoretické úvodu do toho jak jsou RBM definovány se podíváme, jak funguje jejich trénování a jaké mají výsledky na zvolené datové sadě.

6.2.1 Implementace

Program pro vlastní trénování RBM sítí jsem převzal od Michala Hradiše. Program je napsán v Matlabu, což s sebou přináší řadu výhod jako jednodušší programování operací s maticemi a jejich efektivní a optimalizované provádění, ale také to má řadu nevýhod. Jako první nevýhoda je určitě nemožnost předání parametrů z příkazové řádky přímo do programu. Musím to obcházet vytvářením různých variant programu s různými nastaveními parametrů. Další nevýhoda je nemožnost pracovat se zavedeným formátem souborů. Všechny předchozí programy byli psány v jazyce C++ a používají jednotný formát pro ukládání příznaků. Tento formát samozřejmě Matlab nezná a tak je třeba převod do binární podoby a potom zase zpět. A právě tím se zavádí to trénovacího řetězce dvě další položky, které celé trénování prodlužují.

Další omezení plyne přímo ze školního SGE. Školní SGE obsahuje dvě fronty pro plánování úloh. Jedna fronta slouží pro krátké úlohy a druhá pro dlouhé. Krátké úlohy mohou na SGE běžet jen 4 hodiny než dojde k jejich ukončení bez ohledu na čas, který opravdu měly k dispozici procesor. Dlouhé úlohy mají čas na dokončení 14 dní, je však omezen počet úloh na jednoho uživatele. Trénování RBM sítí při velké skryté vrstvě může trvat celkem dlouho, ale dlouhé úlohy nepřichází v úvahu kvůli omezení na počet úloh a malému počtu slotů pro ně vyhrazených. Proto bylo nutné výpočet rozdělit do více krátkých částí. Tohle naštěstí není velký problém. Mohu si uložit stav sítě do souboru a potom ho zase načíst. Další omezení SGE je maximálně 100000 úloh v jedné sadě úloh. Pro trénování RBM to samozřejmě stačí, ale pro trénování SVM na příznacích z RBM to může být málo, protože pro každou sémantickou kategorii je nutný jeden SVM klasifikátor. Z toho důvodu jsem vytvořil skript `SGE-partRun.sh`.

Většina mých skriptů pro SGE funguje tak, že si načtou jeden řádek z konfiguračního souboru jako svoje nastavení a provedou s tím nastavením nějakou úlohu. Skript `SGE-partRun.sh` potom rozděluje konfigurační soubor a postupně spouští zadaný skrip. Jeho činnost se dá popsat následující posloupností operací:

1. Zkontroluji jestli je co počítat.
2. Když není co počítat spusti příkaz po skončení výpočtu a skonči.
3. Vytvoř dočasný soubor s nastavením pro další část výpočtu.
4. Spusť výpočet s dočasným souborem.
5. Spusť sám sebe v úloze, která čeká na dokončení výpočtu, s příznakem další části výpočtu.

Takto se dá rozdělit jakákoliv úloha pro SGE, která načítá svoje nastavení jako jeden řádek v souboru. Navíc jsou úlohy spuštěny postupně a neblokují zbytečně fronty SGE, jak je naznačeno na obrázku 6.3. Toto může být také výhodné pro rozdělení více náročných úloh, pro které není na školních serverech výpočetní kapacita, na menší výpočetní celky, aby se nestalo, že se nestihne výpočet v časovém limitu stanoveném školním SGE. Tento jednoduchý skript vlastně potom slouží jako "obal"výpočetních úloh na SGE. Takto obalené podúlohy lze spouštět bez další starostí s rozložením výpočtu na části nebo zatížením SGE. Stačí správně zvolit parametry skriptu. Potřebné parametry jsou:

NUMBER: číslo spouštěné části - začíná číslem 1

FILE_LIST: soubor s konfigurací

MAX_TASK: maximální počet úloh v jedné sadě

SCRIPT: řetězec obsahující příkaz pro výpočet

NEXT: řetězec obsahující příkaz, který se vykoná po skončení výpočtu

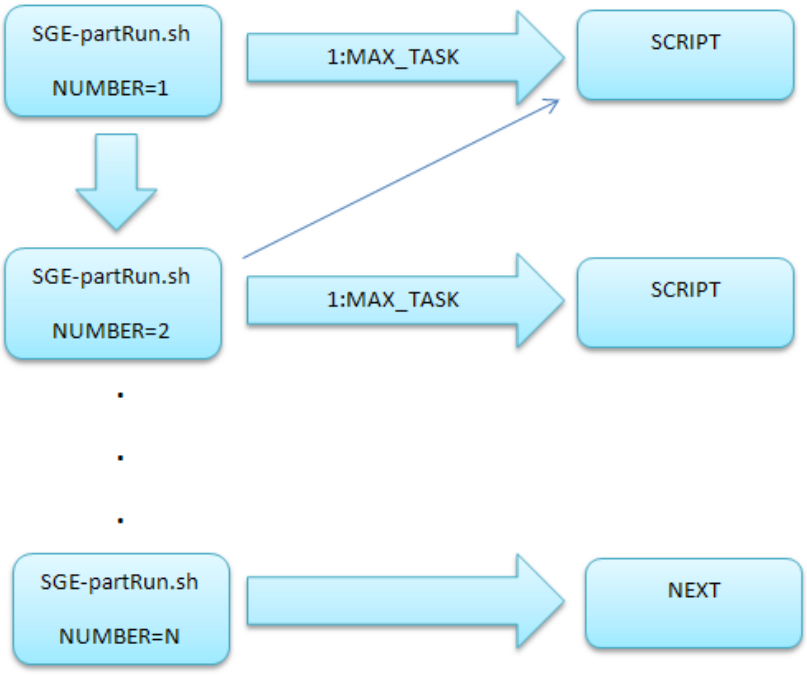
PART_RUN_NAME: volitelný parametr - je použitý jako část jména spouštěných výpočtů

Samotné trénování RBM a SVM klasifikátorů se spouští následovně:

```
$SGE_TOOLS/runRBM.sh -c CONF -b BB
```

Soubor `BB` je stejný jako v případě trénování SVM klasifikátorů a obsahuje použité BOW. `CONF` obsahuje nastavení experimentu, hodnoty parametrů pro RBM síť, nastavení parametrů pro trénování SVM a nastavení SGE. Po spuštění je proveda následující posloupnost činností:

- Vytvoření adresářové struktury.
- Vytvoření konfiguračního souboru pro převod souborů s BOW do binární podoby.
- Vytvoření programů v Matlabu s nastavenými parametry pro trénování RBM.
- Vytvoření programů v Matlabu pro získání odezvy skryté vrstvy pro všechny vstupy.



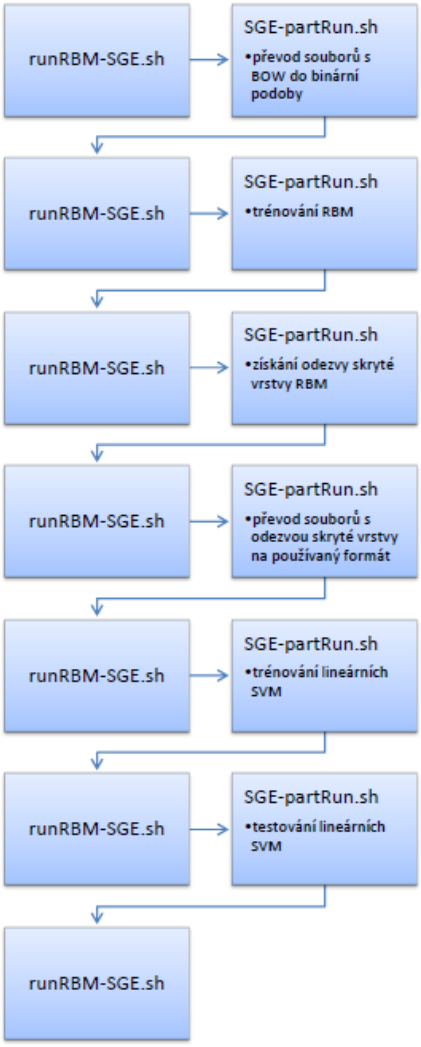
Obrázek 6.3: Princip fungování skriptu `SGE-partRun.sh` na SGE. Obdélníky označují spuštěné úlohy. Široké šipky reprezentují spuštění úlohy a případně obsahují kolik úloh je spuštěno v sadě. Úzké šipky potom ukazují závislost jednotlivých úloh. Úloha, od které vede šipka, čeká na dokončení úlohy, na níž šipka ukazuje. Použité popisky mají návaznost na parametry skriptu.

- Vytvoření konfiguračních souborů pro trénování RBM a výpočet skryté vrstvy.
- Vytvoření konfiguračního souboru pro převod odezvy skryté vrstvy z binární podoby do formátu používaného SVM klasifikátory.
- Vytvoření konfiguračního souboru pro trénování SVM na vytvořených příznacích.
- Vytvoření konfiguračního souboru pro testování natrénovaných SVM.
- Spuštění skriptu `runRBM-SGE.sh` na školním SGE.

Jak je vidět, tak skript `runRBM.sh` pouze vytvoří potřebné konfigurační soubory potřebné pro celý řetězec trénování. Celé trénování potom řídí skript `runRBM-SGE.sh`, běžící na SGE. Postupně potom vkládá další úlohy do SGE, v závislosti na průběhu trénování. Všechny takto vložené úlohy jsou obaleny skriptem `SGE-partRun.sh`. Celý průběh trénování je znázorněn na obrázku 6.4. Tento jednoduchý princip je umožněn hlavně díky tomu, že jsem použil skript `SGE-partRun.sh`, který po skončení výpočtu spustí další úlohu. Na konci toho řetězce lze potom jednoduše přidat další úlohy jako vytvoření statistik, informování o konci výpočtu mailem atd.

6.2.2 Nastavení parametrů

Při trénování RBM hraje roli řada parametrů, které na ně mají menší či větší vliv. Ovlivňují celé trénování sítě, od jeho rychlosti až po kvalitu výsledného modelu. Nastavení těchto



Obrázek 6.4: Princip fungování skriptu `runRBM-SGE.sh` na SGE. Obdélníky označují spuštěné úlohy. Šipky mezi nimi reprezentují spuštění úlohy.

parametrů je součástí programu v Matlabu, který je generován skriptem `runRBM.sh`. Jen některé parametry mají velký vliv na výsledný model a má cenu se jimi zabývat. Ostatní parametry jsem zvolil podle doporučení z dokumentu [8].

Mě zajímá hlavně závislost velikosti skryté vrstvy na přesnost výsledného klasifikátoru. Na kvalitu výsledné sítě při různé velikosti skryté vrstvy má vliv parametr `weightcost`, který ovlivňuje výpočet vah. Než se tedy pustím do optimalizací RBM s různou velikostí skryté vrstvy, musím určit další parametry. Rychlými experimenty s jedním BOW a skrytou vrstvou velikosti 50 jsem dospěl k následujícím hodnotám:

- Rychlost učení vah: 0,1.
- Rychlost učení biasů viditelné vrstvy: 0,1.
- Rychlost učení biasů skryté vrstvy: 0,1.

Teď, když mám hodnoty parametrů pro experimenty, potřebuji ještě vstupní data. RBM je binární neuronová síť, která má hodnoty svých neuronů 0 nebo 1. Při trénování RBM sítě metodou Contrastive Divergence se nastaví stavy neuronů viditelné vrstvy podle trénovacího vektoru. BOW, které mám jako příznaky trénovacích obrázků, je histogram hodnot. Takže obsahuje kladná čísla, proto BOW nelze použít přímo pro trénování RBM sítě a je nutná jeho normalizace. Normalizovat se samozřejmě musí i testovací BOW.

Provedl jsem několik experimentů s normalizací BOW, abych určil, která je nejvhodnější. Všechny experimenty byli prováděny na jednom BOW a s velikostí skryté vrstvy 50. Toto nastavení jsem zvolil kvůli rychlosti trénování BOW. Velmi často BOW obsahuje nuly nebo velmi malé čísla, proto jsem zkusil i netradiční přístupy normalizace a odstranění tohoto nedostatku. Vyzkoušel jsem následující postupy:

- Normalizace L2.
- Podělení všech prvků maximálním prvkem (H).
- Podělení všech prvků maximálním prvkem. Potom přičtení konstanty 0,05 a podělením 1,05, kvůli odstranění nul (HPL).
- Podělení všech prvků maximálním prvkem. Potom odmocnění všech prvků, kvůli zvětšení čísel (HSQ).
- Podělení všech prvků maximálním prvkem. Potom odmocnění všech prvků a následně přičtení konstanty 0,05 a podělením 1,05 (HSQPL).

Z výsledků v tabulce 6.1 je vidět, že nejlepší výsledky dosahuje normalizace pomocí podělení každého vektoru jeho největší hodnotou, a proto jsem ji také použil dále při mém experimentu.

Normalizace	**AvgP**	**EER**
L2	0,1927	0,3763
H	0,1969	0,3763
HPL	0,1764	0,3895
HSQ	0,1729	0,3974
HSQPL	0,1672	0,4031

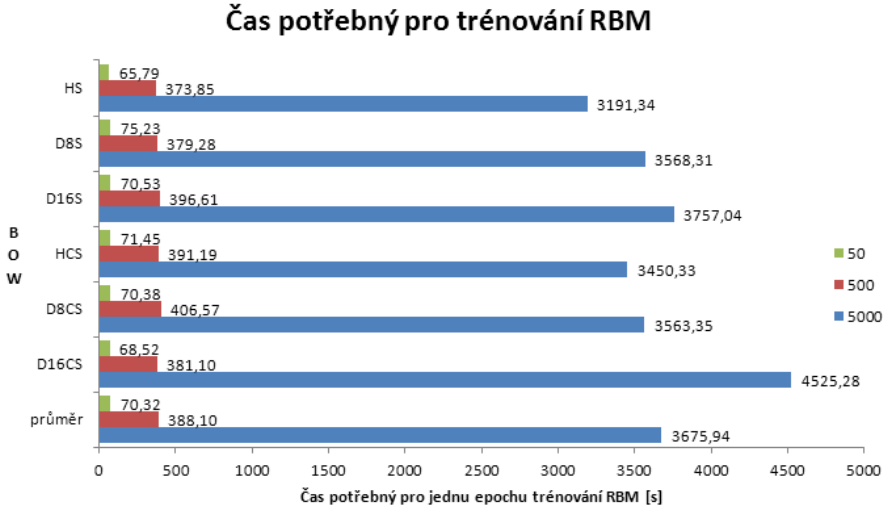
Tabulka 6.1: Výsledky experimentů s normalizací trénovacích BOW.

6.2.3 Výsledky

Když mám nastavené nejvhodnější parametry a připravené trénovací a testovací BOW, můžeme se pustit do experimentu. Spuštění experimentu jsem popsal v sekci 6.2.1. Tady se podíváme na výsledky mého experimentu. Zaměřím se nejen na jednotlivé výsledky podle velikosti skryté vrstvy, ale i na možnosti fúze takto vytvořených klasifikátorů.

Začneme tedy trénováním RBM s různou velikostí skryté vrstvy. Experimenty jsem provedl pro 50, 500 a 5000 neuronů ve skryté vrstvě. Trénování RBM sítě je rozloženo na epochy, kdy v každé epoše je provedeno pět iterací algoritmu Contrastive Divergence. Každá epocha potom potřebuje určitý čas pro svůj výpočet. Jak ukazuje graf na obrázku 6.5, tak počet neuronů ve skryté vrstvě značně ovlivňuje tuto dobu výpočtu. Trénování RBM s 50

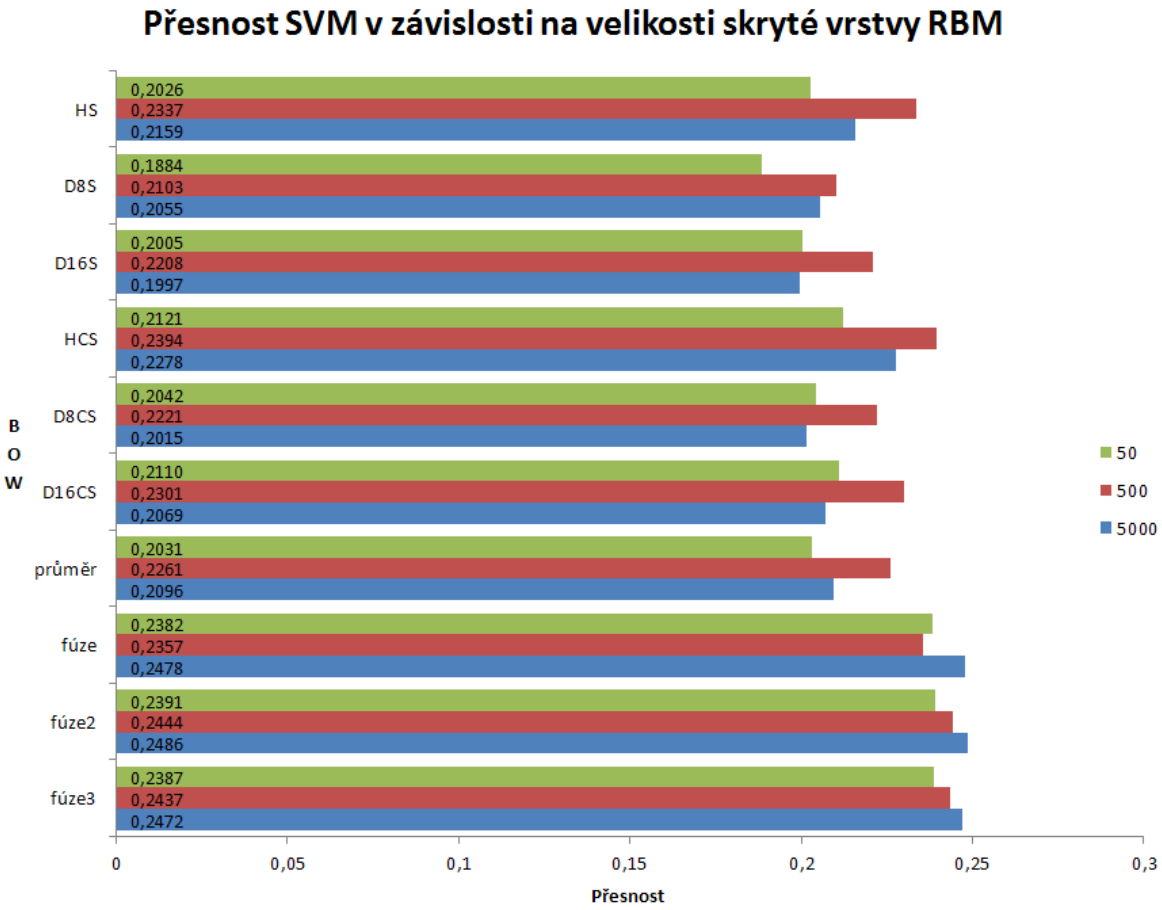
neurony ve skryté vrstvě je asi 5,5 krát rychlejší než RBM s 500 neurony ve skryté vrstvě a asi 52 krát rychlejší než RBM s 5000 neurony ve skryté vrstvě.



Obrázek 6.5: Časová náročnost epochy trénování v závislosti na velikosti skryté vrstvy RBM sítě.

Počet neuronů ve skryté vrstně má také vliv na přesnost natrénovaného SVM klasifikátoru. Výsledná přesnost je naznačena v grafu na obrázku 6.6. Pro každou velikost skryté vrstvy jsem trénoval šest různých RBM sítí. Je to kvůli správné hodnotě parametru `weightcost` u každé sítě. Je to analogické s optimalizací parametru C u lineárních SVM klasifikátorů. Na každé z těchto sítí jsem po natrénování vzal odezvu skryté vrstvy na vstupní data a na této odezvě potom natrénoval jeden SVM klasifikátor. V grafu je vždy zobrazen jen klasifikátor s nejlepším výsledkem pro danou velikost skryté vrstvy. Výsledek je celkem zajímavý. Pokud se podíváme jen na jednotlivé BOW tak jsou nejpřesnější SVM klasifikátory natrénované na odezvách skryté vrstvy RBM sítě s 500 neurony ve skryté vrstvě. Jak je však vidět z grafu tak po fúzi tyto klasifikátory mají výslednou přesnost 23,57%. A to je nejhorší výsledek po fúzi z celého experimentu. Tento výsledek se dá zlepšit fúzí, kdy vezmu místo nejlepšího klasifikátoru pro daný BOW dva nejlepší. Tím dosáhnou SVM přesnosti 24,44%, což je ovšem pořád méně oproti SVM trénovaných na RBM s velikostí skryté vrstvy 5000 neuronů s přesností 24,86%. Pokud vezmu v úvahu časovou náročnost trénování jednotlivých RBM a skutečnost, že tento rozdíl v přesnosti není tak velký, tak jsou SVM klasifikátory, trénované na odezvách skryté vrstvy RBM sítě s 500 neurony ve skryté vrstvě, rozumný kompromis.

Když jsem otestoval vliv počtu neuronů skryté vrstvy na výslednou přesnost, tak mě začala zajímat vhodnost takto trénovaných SVM na fúzi. Konkrétně jsem chtěl zjistit zda více natrénovaných RBM může výrazně zlepšit výslednou přesnost podobným způsobem jak se dá zlepšit použitím více různých BOW. Výsledky mých experimentů zobrazuje graf na obrázku 6.7. Jako první jsem vyzkoušel fúzi úplně všech natrénovaných SVM. Dosáhl jsem výsledku 24,33%, což není špatné vzhledem k tomu, že ta fúze je z většiny složena z RBM sítí, které nemají optimální hodnotu parametru `weightcost`. Dále jsem zkoušel různé kombinace nejlepších SVM pro jednotlivé BOW. Výsledky jsou srovnatelné s fúzí všech klasifikátorů. Nakonec jsem se zaměřil i na rychlost. Nejrychleji se trénuje RBM s 50 neurony



Obrázek 6.6: Přesnost natrénovaného SVM klasifikátoru v závislosti na velikosti skryté vrstvy RBM sítě. Fúze2 označuje fúzi provedenou z dvou nejlepších SVM pro daný BOW a fúze3 analogicky označuje fúzi ze tří nejlepší SVM pro daný BOW.

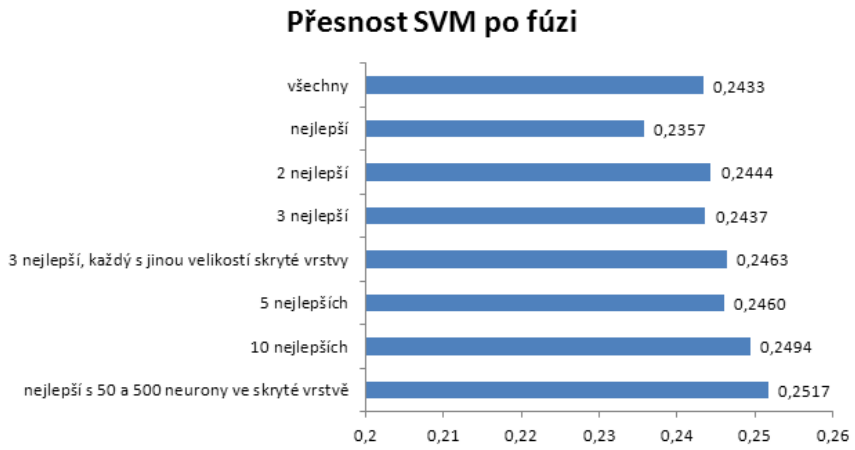
ve skryté vrstvě a RBM s 500 neurony ve skryté vrstvě je také celkem rychle natrénované. Proto jsem se rozhodl udělat fúzi nejlepších SVM pro každou BOW natrénované na RBM s 50 a 500 neurony ve skryté vrstvě. Výsledek mě mile překvapil. Výsledná přesnost této fúze je 25,17%. To je nejlepší přesnost ze všech provedených experimentů. Jinak, ale jsou možnosti fúze velice omezené a přidáním více RBM se stejnou velikostí skryté vrstvy nemá moc velký pozitivní vliv na výslednou přesnost.

6.3 Srovnání s klasickými SVM

Po provedení experimentů s RBM mohu tento postup srovnat s předchozími experimenty na lineárních a nelineárních SVM klasifikátorech.

6.3.1 Přesnost

Když se podíváme na přesnost a chybovost SVM klasifikátorů natrénovaných na odezvě skryté vrstvy RBM sítě, tak dávají lepší výsledek než lineární SVM klasifikátory. Přesnosti jakou dosahují nelineární SVM klasifikátory se mě nepodařilo v mých experimentech



Obrázek 6.7: Přesnost SVM klasifikátorů po fúzi.

dosáhnout. Srovnání přesnosti a chybovosti jednotlivých řešení obsahuje tabulka 6.2. Pro srovnání jsem vzal vždy nejlepší hodnoty. Z tabulky je zřejmé, že použití RBM pro transformaci příznaků, je kompromisní řešení mezi lineárními a nelineárními klasifikátory, alespoň podle přesnosti a chybovosti výsledné klasifikace.

BOW	**lineární SVM**		**RBM**		**nelineární SVM**	
	AvgP	**EER**	**AvgP**	**EER**	**AvgP**	**EER**
HS	0,1964	0,3866	0,2337	0,3467	0,2630	0,3260
D8S	0,1907	0,3867	0,2103	0,3625	0,2600	0,3249
D16S	0,2046	0,3782	0,2208	0,3644	0,2711	0,3219
HCS	0,2016	0,3819	0,2393	0,3466	0,2692	0,3174
D8CS	0,2050	0,3769	0,2220	0,3587	0,2675	0,3251
D16CS	0,2115	0,3721	0,2301	0,3559	0,2768	0,3183
průměr	0,2016	0,3804	0,2261	0,3558	0,2679	0,3222
fúze	0,2173	0,3627	0,2517	0,3387	0,2989	0,2978

Tabulka 6.2: Srovnání výsledků experimentů RBM pro transformaci příznaků s lineárními a nelineárními SVM klasifikátory na zvolené datové sadě. V prvním sloupci je použitý BOW, AvgP značí průměrnou přesnost a EER je průměrná chybovost pro daný BOW.

6.3.2 Časová náročnost

Časová náročnost trénování SVM s použitím RBM pro transformaci příznaků lze rozdělit na základní kroky, které je nutné provést:

- Převod fv souborů na binární formát.
- Trénování RBM sítě.
- Získání odezvy skryté vrstvy RBM sítě pro trénovací data.

- Převod binární podoby odezvy skryté vrstvy RBM sítě pro trénovací data na fv soubory.
- Trénování lineárních SVM klasifikátorů na těchto datech.

Samotné trénování RBM sítě je naprogramováno, tak aby se síť trénovala po určitou předem danou dobu. Já jsem v mém experimentu provedl deset iterací trénování po 1 hodině. Dohromady se tedy každá RBM síť trénovala minimálně 10 hodin. Takovou dobu jsem zvolil kvůli RBM sítím s 5000 neurony ve skryté vrstvě, kde jedna epocha trénování se průměrně počítá více jak hodinu. Potom můžu ostatní doby zanedbat, protože potřebují pro svoje dokončení čas v řádu minut. Pokud bych bral jako dobu potřebnou pro trénování 10 hodin, tak je tento přístup 27 krát pomalejší než trénování nelineárních SVM klasifikátorů a 353 krát pomalejší než trénování lineárních SVM klasifikátorů.

Vzhledem k tomu, že přesnost jsem porovnával jen u RBM s 50 a 500 neurony ve skryté vrstvě a doba trénování byla nastavená kvůli RBM s 5000 neurony ve skryté vrstvě, mohu tento odhad upravit. Podle průměrných hodnot vah u jednotlivých RBM je zřejmé, že průměrně trvá 50 epoch k natrénování RBM na zvolených testovacích datech. Lze to poznat podle ustálené hodnoty průměru vah. Pokud budu uvažovat spuštění experimentu bez RBM s 5000 neurony ve skryté vrstvě. Tak mohu nastavit menší dobu trénování. Tuto dobu nastavím, tak aby RBM s 500 neurony ve skryté vrstvě stihlo provést 50 epoch učení. Potom stačí nastavit čas trénování na 19405 vteřin. I to je, ale skoro 15 krát pomalejší než trénování nelineárních SVM klasifikátorů a 190 krát pomalejší než trénování lineárních SVM klasifikátorů.

Když vezmeme v úvahu jen časovou náročnost trénování, tak nelze o transformaci příznaků pomocí RBM sítí ani uvažovat. Časová náročnost tohoto řešení je opravdu značná.

6.3.3 Paměťová náročnost

Pokud se budu držet předchozího srovnání paměťové náročnosti, kdy jsem porovnával jen paměť potřebnou pro uložení trénovacích dat, tak je trénování RBM sítě stejně paměťově náročné jako trénování lineárního SVM klasifikátoru. Je to dané tím, že používá stejné vektory BOW, jenom jsou normalizové pro použití s RBM sítěmi. Mohl bych do srovnání zavést velikost skryté vrstvy, protože potřebuji udržoval v paměti váhy pro mezi všemi viditelnými a skrytými neurony. Tato matice může být velice rozměrná při velikostech skryté vrstvy v řádu desítek tisíc neuronů. Jak jsem však ukázal v experimentech, tak stačí skrytá vrstva o velikosti 50 a 500 neuronů pro dobrou přesnost. Proto můžu tuto paměť zanedbat. Výsledná paměťová náročnost učení RBM sítí je srovnatelná s paměťovou náročností trénování lineárních SVM klasifikátorů.

6.4 Možnosti dalšího pokračování

Když se podívám na experimenty s RBM, tak i přes časovou náročnost jejich učení, je to celkem použitelné řešení. Dosáhl jsem lepší přesnosti než mají lineární SVM klasifikátory, při srovnatelné náročnosti na paměť. Už jen kvůli tomu má cenu se zamýšlet, jak by se dalo pokračovat v této práci a zlepšit výslednou přesnost minimálně na úroveň nelineárních SVM klasifikátorů. Také by nebylo špatné urychlit trénování RBM sítí. Nikdy asi nebude trénování tak rychlé jako u nelineárních SVM klasifikátorů, ale třeba jen dvojnásobné zrychlení by, vzhledem k současné době potřebné pro učení RBM sítě, byl veliký posun.

Existuje několik postupů, kterými by se tohoto cíle mohlo dosáhnout:

Optimalizovat trénovací data: Trénovací data se normalizují před použitím v RBM. Já jsem vyzkoušel několik možností jejich normalizace, ale je možné, že existuje nějaká vhodnější metoda pro tento případ. Jejím použitím by se mohla zvýšit výsledná přesnost klasifikace.

Změna algoritmu trénování: Pokud by algoritmus učení RBM sítě měl jiný způsob ukončení než uplynulá doba od počátku učení, tak by mohlo být trénování dokončeno dříve. Tohle by šlo například pomocí sledováním hodnot rekonstrukční chyby nebo průměrné hodnoty vah.

Použít více vrstev: RBM sítě jsou ze své definice neuronové sítě, které mají dvě vrstvy. Ovšem po natrénování RBM sítě lze použít skrytá vrstva jako vrstva viditelná. K takovéto vrstvě lze natrénovat další skrytá vrstva atd. Pokud bych takovou síť natrénoval a jako příznaky pro SVM klasikátor použil hodnoty až druhé nebo třetí vrstvy, tak by se mohla zlepšit výsledná přesnost. Více informací lze najít zde [7].

Toto byli jen příklady vylepšení, které by se mohli realizovat pro zlepšení použití RBM sítí pro transformaci příznaků. Po dalším zkoumáním problému určitě půjde nalézt další možnosti jak tento postup zlepšit.

Kapitola 7

Závěr

Prostudoval jsem používané metody pro automatické zařazování obrázků do sémantických kategorií na základě jejich obsahu. Na naší fakultě jsou již hotové a používané detektory významných oblastí v obraze, deskriptory těchto oblastí a stejně tak slovníky, které přiřazují kódová slova na základě jádra. Proto jsem se také rozhodl vylepšit poslední krok tohoto postupu. Tímto krokem je klasifikace, kde jsem se rozhodl prozkoumat možnosti SVM klasifikátorů a pak také možnosti RBM sítí na transformaci obrazových příznaků.

Po zvolení datové sady, popsané v kapitole 5.1, jsem začal s lineárními SVM klasifikátory, s kterými jsem dosáhl výsledné přesnosti klasifikace 21,73%. Dále jsem otestoval možnosti nelineárních SVM klasifikátorů, kdy jsem dosáhl přesnosti 29,89%. Po porovnání těchto přístupů a zjištění jejich výhod a nevýhod, jsem se snažil najít lepší řešení problému automatického zařazování obrázků do sémantických tříd podle jejich obsahu. Z tohoto důvodu jsem experimentoval s RBM sítěmi jako s možností pro transformaci nelinárního problému na problém lineární. Po dokončení experimentů jsem dosáhl na výslednou přesnost 25,17%. Tato přesnost je sice menší než u nelineárních SVM klasifikátorů, ale alespoň jsem zlepšil paměťovou náročnost klasifikace. Řešení s RBM sítěmi je tedy kompromis mezi lineárními a nelineárními SVM klasifikátory. V případech, kdy nelze použít nelineární SVM klasifikátory může být toto řešení využíváno pro zlepšení přesnosti.

Po provedení navržených zlepšení, uvedených v kapitole 6.4, by toto řešení mohlo konkurovat nelineárním SVM klasifikátorům v přesnosti a nahradit je v této oblasti. Pokud se to nepodaří, je pořád možné využívat RBM sítě pro tvorbu dalších smysluplných příznaků na trénování SVM klasifikátorů a zpřesnit výslednou přesnost fúzí z více různých klasifikátorů.

A proto si myslím, že jsem splnil hlavní cíl této práce. Povedlo se mně zlepšit možnosti klasifikace obrázků do sémantických tříd. Hlavně pak spojit výhody lineárních a nelineárních SVM klasifikátorů a odstranit nedostatky tohoto postupu za cenu určitých kompromisů v přesnosti.

Literatura

[1] Restricted Boltzmann Machine. [cit. 2012-05-01].
URL http://imonad.com/rbm/restricted-boltzmann-machine/

[2] Bay, H.; Ess, A.; Tuytelaars, T.; aj.: Speeded-Up Robust Features (SURF). *Computer Vision and Image Understanding*, ročník 110, č. 3, 2008: s. 346 – 359, ISSN 1077-3142, doi:DOI:10.1016/j.cviu.2007.09.014, similarity Matching in Computer Vision and Multimedia.
URL http://www.sciencedirect.com/science/article/B6WCX-4RC2S4T-2/2/c2c03b6165996e30312e5b7c7b681155

[3] Chapelle, O.; Haffner, P.; Vapnik, V.: Support vector machines for histogram-based image classification. *Neural Networks, IEEE Transactions on*, ročník 10, č. 5, sep 1999: s. 1055 –1064, ISSN 1045-9227, doi:10.1109/72.788646.

[4] Fletcher, T.: Support Vector Machines Explained. 2009-3-1 [cit. 2011-01-09].
URL http://www.tristanfletcher.co.uk/SVM%20Explained.pdf

[5] van Gemert, J.; Geusebroek, J.-M.; Veenman, C.; aj.: Kernel Codebooks for Scene Categorization. In *Computer Vision ECCV 2008*, *Lecture Notes in Computer Science*, ročník 5304, editace D. Forsyth; P. Torr; A. Zisserman, Springer Berlin / Heidelberg, 2008, s. 696–709, 10.1007/978-3-540-88690-7_52.
URL http://dx.doi.org/10.1007/978-3-540-88690-7_52

[6] van Gemert, J. C.; Burghouts, G. J.; Seinstra, F. J.; aj.: Color invariant object recognition using entropic graphs. *International Journal of Imaging Systems and Technology*, ročník 16, č. 5, 2006: s. 146–153, ISSN 1098-1098, doi:10.1002/ima.20082.
URL http://dx.doi.org/10.1002/ima.20082

[7] Hinton, G.: To Recognize Shapes, First Learn to Generate Images. 2006 [cit. 2011-05-04] Version 1.
URL http://www.cs.toronto.edu/~hinton/absps/guideTR.pdf

[8] Hinton, G.: A Practical Guide to Training Restricted Boltzmann Machines. 2010 [cit. 2011-05-01] Version 1.
URL http://www.cs.toronto.edu/~hinton/absps/guideTR.pdf

[9] Hsu, C.-W.; Chang, C.-C.; Lin, C.-J.: A Practical Guide to Support Vector Classification. 2000.
URL http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf

[10] Keen, N.: Color Moments. 2005-2-10 [cit. 2011-01-02].
URL http:
//pascallin.ecs.soton.ac.uk/challenges/VOC/voc2008/workshop/tahir.pdf

[11] Lazebnik, S.; Schmid, C.; Ponce, J.: Spatial Pyramid Matching. [cit. 2011-01-02].
URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.173.16&rep=rep1&type=pdf

[12] Manjunath, B.; Ma, W.: Texture features for browsing and retrieval of image data. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, ročník 18, č. 8, aug 1996: s. 837 –842, ISSN 0162-8828, doi:10.1109/34.531803.

[13] Marszałek, M.; Schmid, C.; Harzallah, H.; aj.: Learning Representations for Visual Object Class Recognition. 2007-10-15 [cit. 2011-01-02].
URL http://eprints.pascal-network.org/archive/00003693/01/MarszalekSchmid-VOC07-LearningRepresentations-slides.pdf

[14] Meslouhi, O.; Allali, H.; Gadi, T.; aj.: Colposcopic image registration using opponentSIFT descriptor. In *I/V Communications and Mobile Network (ISVC), 2010 5th International Symposium on*, 30 2010-oct. 2 2010, s. 1 –4, doi:10.1109/ISVC.2010.5656239.

[15] Mikolajczyk, K.; Schmid, C.: Scale & Affine Invariant Interest Point Detectors. *International Journal of Computer Vision*, ročník 60, 2004: s. 63–86, ISSN 0920-5691, 10.1023/B:VISI.0000027790.02288.f2.
URL http://dx.doi.org/10.1023/B:VISI.0000027790.02288.f2

[16] Minh, H.; Niyogi, P.; Yao, Y.: Mercer's Theorem, Feature Maps, and Smoothing. In *Learning Theory*, *Lecture Notes in Computer Science*, ročník 4005, editace G. Lugosi; H. Simon, Springer Berlin / Heidelberg, 2006, s. 154–168, 10.1007/11776420_14.
URL http://dx.doi.org/10.1007/11776420_14

[17] National Institute of Standards and Technology: TREC Video Retrieval Evaluation. [cit. 2011-01-02].
URL http://trecvid.nist.gov/

[18] Over, P.; Awad, G.; Fiscus, J.; aj.: TRECVID 2009 - Goals, Tasks, Data, Evaluation Mechanisms and Metrics. 2010-4-8 [cit. 2011-01-02].
URL
http://www-nlpir.nist.gov/projects/tvpubs/tv9.papers/tv9overview.pdf

[19] Snoek, C.; van de Sande, K.; de Rooij, O.; aj.: The MediaMill TRECVID 2009 Semantic Video Search Engine. 2009 [cit. 2011-01-02].
URL http://www-nlpir.nist.gov/projects/tvpubs/tv9.papers/mediamill.pdf

[20] Sun Microsystems: Grid Engine System Components. [cit. 2011-01-09].
URL
http://wikis.sun.com/display/GridEngine/Grid+Engine+System+Components

[21] Sun Microsystems: How the System Operates. [cit. 2011-01-09].
URL http://wikis.sun.com/display/GridEngine/How+the+System+Operates

[22] Sun Microsystems: What Is Grid Computing? [cit. 2011-01-09].
URL http://wikis.sun.com/display/GridEngine/What+Is+Grid+Computing

[23] Uijlings, J.; Yan, F.; Gevers, T.; aj.: UvA & Surrey PASCAL VOC 2008. 2008 [cit. 2011-01-02].
URL http://pascallin.ecs.soton.ac.uk/challenges/VOC/voc2008/workshop/tahir.pdf

[24] University of Southampton: The PASCAL Visual Object Classes Homepage. [cit. 2011-01-02].
URL http://pascallin.ecs.soton.ac.uk/challenges/VOC/

[25] Wang, F.; Merialdo, B.: Eurecom at TRECVID 2009. 2009 [cit. 2011-01-02].
URL http://www-nlpir.nist.gov/projects/tvpubs/tv9.papers/eurecom.pdf

[26] Wikipedia, t. f. e.: Blob detection. [cit. 2010-12-31].
URL
http://en.wikipedia.org/wiki/Blob_detection#The_Laplacian_of_Gaussian

[27] Wikipedia, t. f. e.: Difference of Gaussians. [cit. 2010-12-31].
URL http://en.wikipedia.org/wiki/Difference_of_Gaussians

[28] Žižka, J.: Support vector machines (SVM). 2005-10-21 [cit. 2011-01-09].
URL http://is.muni.cz/el/1433/podzim2006/PA034/09_SVM.pdf

Příloha A

Obsah CD

Přiložené CD obsahuje:

- Zdrojové kódy této práce pro LaTeX.
- Zdrojové kódy pro používané programy.
- Skripty pro spouštění experimentů a vytváření statistik.
- Konfigurační soubory pro jednotlivé experimenty.

Příloha B

Manuál

Před spuštěním experimentů je nejprve nutné sestavit používané programy. Lze to udělat příkazem `make` v adresáři `/src`. Poté je nutné nakopírovat všechny soubory z adresáře `/src/bin` do adresáře `/SGE-tools/bin`. Nyní se mohou spustit jednotlivé experimenty.

B.1 Lineární SVM

Experiment s lineárními SVM má uložené konfigurační soubory v adresáři `/tv11.SVM.20k.Linear` a všechny následující příkazy je nutné spouštět z tohoto adresáře. Trénování:

```
/SGE-tools/runLinear.sh -c CONF -b BB
```

Vytvoření statistik jednotlivých SVM:

```
/SGE-tools/makeLinearStats.sh -c CONF -b BB
/SGE-tools/makeLinearTimeStats.sh -c CONF -b BB
```

Fúze:

```
/SGE-tools/runFusion.sh -c CONF_FUSION -b BB
```

Vytvoření statistik fúze:

```
/SGE-tools/makeFusionStats.sh -c CONF_FUSION
```

B.2 Nelineární SVM

Konfigurační soubory pro nelineární SVM jsou v adresáři `/tv11.SVM.20k.NonLinear` a opět je nutné všechny následující skripty spouštět z této složky. Trénování:

```
/SGE-tools/runNonLinear.sh -c CONF -b BB
```

Vytvoření statistik:

```
/SGE-tools/makeNonLinearStats.sh -c CONF -b BB
/SGE-tools/makeNonLinearTimeStats.sh -c CONF -b BB
```

Fúze:

```
/SGE-tools/runFusion.sh -c CONF_FUSION -b BB
```

Vytvoření statistik fúze:

```
/SGE-tools/makeFusionStats.sh -c CONF_FUSION
```

B.3 RBM

Všechny potřebné soubory pro RBM jsou v adresáři `/tv11.SVM.20k.RBM`, všechny skripty je nutné spouštět z adresáře s konfiguračními soubory. Trénování:

```
/SGE-tools/runRBM.sh -c CONF -b BB
```

Vytvoření statistik:

```
/SGE-tools/makeRBMStats.sh -c CONF -b BB
```

Fúze:

```
/SGE-tools/runRBMFusion.sh -c CONF_FUSION_X
```

Vytvoření statistik fúze:

```
/SGE-tools/makeRBMFusionStats.sh -c CONF_FUSION_X
```

Kde `CONF_FUSION_X` může být jeden z konfiguračních souborů. Jednotlivé varianty se liší pouze v tom jaké SVM budou použity pro fúzi. Možnosti:

CONF_FUSION_ALL: Všechny natrénované SVM.

CONF_FUSION_BEST: Z každého BOW jen ten nejlepší SVM.

CONF_FUSION_BEST2: Z každého BOW dva nejlepší SVM.

CONF_FUSION_BEST3: Z každého BOW tři nejlepší SVM.

CONF_FUSION_BEST_ALL: Z každého BOW tři nejlepší SVM, každý s jinou velikostí skryté vrstvy.

CONF_FUSION_BEST5: Z každého BOW pět nejlepších SVM.

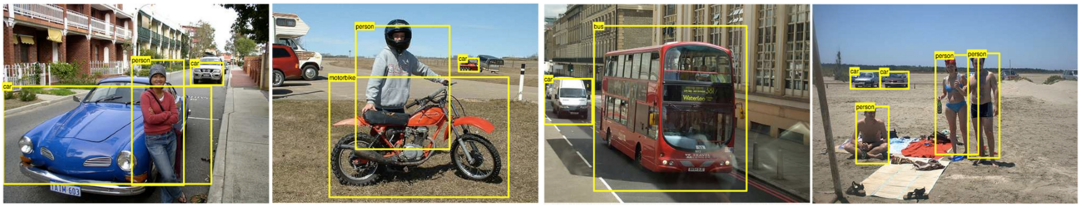
CONF_FUSION_BEST10: Z každého BOW deset nejlepších SVM.

CONF_FUSION_BEST_ALL2: Z každého BOW nejlepší SVM s 50 a 500 neurony ve skryté vrstvě.

Příloha C

Plakát

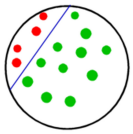
Automatické označování obrázků



JAK?

Tradiční přístupy

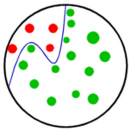
Lineární SVM



\+ Paměťová náročnost

\− Přesnost

Nelineární SVM



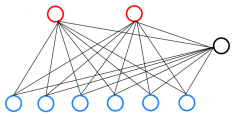
\+ Přesnost

\− Paměťová náročnost

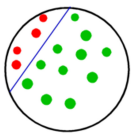
Moje řešení

RBM + SVM

Transformace příznaků pomocí RBM sítě



Klasifikace pomocí lineárních SVM



\+ Přesnost

\+ Paměťová náročnost

Autor: Bc. Michal Sýkora
Vedoucí: Ing. Michal Hradiš