VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

STROJOVÉ UČENÍ V KLASIFIKACI OBRAZU

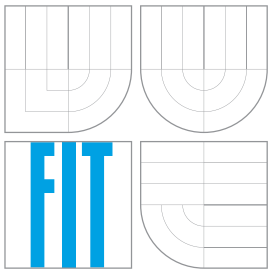
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JIŘÍ KRÁL

BRNO 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

STROJOVÉ UČENÍ V KLASIFIKACI OBRAZU

MACHINE LEARNING IN IMAGE CLASSIFICATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JIŘÍ KRÁL

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MICHAL HRADIŠ

BRNO 2011

Abstrakt

Práce se zabývá hledáním a analýzou statistických modelů a algoritmických postupů, které mají potenciál zlepšit výsledky FIT VUT v Brně na soutěžích zabývajících se klasifikací obrazu jako jsou ImageNet Large Scale Visual Recognition Challenge a TRECVID. V práci byl otestován multinomiální model, také byl použit model Phonotactic Intersession Variation Compensation (PIVCO) pro adaptaci náhodných vlivů v obrazové reprezentaci a dále pak pro redukci dimenzionality. Dále byl analyzován model KPCA, kterým se emulovala Kernel SVM klasifikace. Všechny statistické modely byly testovány na Pascal VOC 2007 datasetu.

Abstract

This project deals vith analysis and testing of algorithms and statistical models, that could potentionaly improve resuts of FIT BUT in ImageNet Large Scale Visual Recognition Challenge and TRECVID. Multinomial model was tested. Phonotactic Intersession Variation Compensation (PIVCO) model was used for reducing random effects in image representation and for dimensionality reduction. PIVCO – dimensionality reduction achieved the best mean average precision while reducing to one-twenyth of original dimension. KPCA model was tested to approximate Kernel SVM. All statistical models were tested on Pascal VOC 2007 dataset.

Klíčová slova

Strojové učení, Klasifikace obrazu, PIVCO, Multinomiální model, Kernel PCA, Redukce dimenzionality

Keywords

Machine Learning, Image classification, PIVCO, Multinomial model, Kernel PCA, Dimensionality reduction

Citace

Jiří Král: Strojové učení v klasifikaci obrazu, diplomová práce, Brno, FIT VUT v Brně, 2011

Strojové učení v klasifikaci obrazu

Prohlášení

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

.....
Jiří Král
25. května 2011

Poděkování

Chtěl bych poděkovat Ing. Michalu Hradišovi za vedení Diplomové práce. Dále bych chtěl poděkovat Ing. Ondřeji Glembekovi, za seznámení mě s modelem Phonotactic Intersession Variation Compensation.

Obsah

Kapitola 1

Úvod

Klasifikace obrazu, tak jak je chápána v této práci, se zabývá přiřazováním tagů neboli sémantických značek fotografiím nebo videosekvencím. Současné databáze obrazů a videí jsou většinou označovány ručně. Klasifikace obrazu se snaží tento proces z automatizovat a tím celý proces zrychlit. Dále se také zabývá kategorizací již existujících databází. Příkladem kategorizovaných obrazových dat může být například obrazová databáze Flikr, nebo google.images. Hlavní motivací pro klasifikaci obrazu je tedy usnadnění práce s velkými objemy obrazových dat.

V současné době je klasifikace obrazu velmi důležitou součástí informačních technologií. Rozpoznávání písma, nebo vad výrobků je již běžnou součástí našeho života, ale přesto úspěšnost metod klasifikace je často omezena některým z rušivých faktorů, kterými jsou například velikost, natočení nebo překrytí objektu jiným objektem. Dalším elementem komplikujícím proces klasifikace obrazu je velikost datových sad [16, 18, 12]. Ta pozitivně ovlivňuje úspěšnost samotné klasifikace, tím že je v datové sadě zachyceno více informací o skutečném rozložení dat, ale většinou negativně ovlivňuje časovou náročnost ($O()$) procesu učení klasifikátoru. Dále musíme také vzít v úvahu, zda klasifikujeme do sémantických tříd, kdy hledáme objekty stejného významu, a nehledáme přesný tvar, nebo klasifikujeme do vizuálních tříd, hledáme objekty stejného tvaru. V této práci se budu zabývat pouze klasifikací do sémantických tříd.

Jedním z velmi úspěšných přístupů pro klasifikaci a kategorizaci obrazových dat, je popsat obrazy pomocí "bag of visual words"(BoV) histogramů a následně BOV reprezentaci klasifikovat pomocí Support vector machine (SVM) klasifikátorů [7, 12]. Pro lineární SVM je časová náročnost trénování klasifikátoru lineární $O(N)$, kde N je počet vzorků dat [8, 3], zatímco u nelineárních SVM je kvadratická $O(N^2)$ [2]. Tento fakt je velmi pádným argumentem pro využití lineárních SVM klasifikátorů, ale nelineární SVM podávají výrazně lepší výsledky klasifikace obrazu než lineární [12].

Cílem této práce je hledat, analyzovat a testovat nové algoritmické postupy a statistické modely, které mají potenciál zlepšit výsledky SVM klasifikace na velkých datových sadách, popřípadě najít jiné úspěšnější modely, které by zcela nahradily SVM klasifikátory. Dále je cílem hledat postupy, které umožní použít nebo emulovat nelineární SVM klasifikátory v úlohách, kde byly doposud používány pouze lineární SVM kvůli jejich menší časové a paměťové složitosti. Konečným cílem práce, je dosáhnout zlepšení výsledků v evaluacích ImageNet Large Scale Visual Recognition Challenge a TRECVID na Fakultě Informačních Technologií VUT v Brně (FIT) 2.1.

První možností, jak tohoto cíle dosáhnout, je náhrazení Kernel SVM modelu jiným, nelineárním modelem, který by byl použitelný i pro velmi velké datové sady. Zde jsem využil

dva modely, které se s úspěchem využívají pro zpracování řečových signálů. Jedná se o Multinomiální model, a model Phonotactic Intersession Variation Compensation (PIVCO). Reprezentace obrazu i zvukových signálů jsou podobné, obě zaznamenávají počty výskytů lokálních primitiv. Pro řeč je to četnost fonémů, pro grafiká data je to počet výskytů obrazových primitiv nebo lokálních příznaků.

Druhou perspektivní možností pro dosažení cíle této práce, je předzpracování nebo zmenšení dimenzionality dat nelineární transformací, aby mohla být efektivněji provedena lineární respektive nelineární SVM klasifikace. V práci rozeberu dva postupy, jak této nelineární transfomace dosáhnout. Prvním je redukce dimenzinality metodou PIVCO. V tomto případě probíhá nelineární transformace dat. Trénovací i testovací datasety jsou promítnuty do nižší dimenze. Tím se sníží čas nutný pro tvorbu jádra pro Kernel SVM i samotný čas potřebný na počítání odezev jádra.

Druhou cestou k dosažení nelineárního předzpracování dat je aproximace nebo emulace nelineárního klasifikátoru pomocí Kernel PCA. Tento přístup spočívá v nelineární transformaci dat pomocí kernel-PCA a řešením lineárního SVM nad takto transformovanými daty.

Hlavním cílem této práce je tedy zlepšit výsledky pro evaluace TRECVID a ImageNet Large Scale Visual Recognition Challenge, které jsou představeny v sekcích 1.1 a 1.2, kde jsou stručně rozebrány jejich cíle a charakterizuji jejich datové sady.

Struktura práce je následující. V kapitole 2, popíši jak se postupuje v klasických metodách užitých pro klasifikaci obrazu. Dále rozeberu alternativní přístupy, které jsem použil ve své práci. Jedná se o multinomiální rozložení 2.3 , PIVCO [5, 6], redukci dimenzionality pomocí modelu PIVCO 2.4 a kernel PCA 2.5. Následující kapitola 3 popisuje implementaci jednotlivých algoritmů. Následně v kapitole 4 budu popisovat experimenty a rozeberu dosažené výsledky. A v závěru 5 provedu shrnutí výsledků.

1.1 TRECVID 2011

TREC je série konferencí sponzorovaná Národním Institutem Standardů a Technologie (NIST) za podpory vlády USA [18]. Cílem těchto konferencí je podporovat výzkum v oblasti analýzy založené na obsahu a získávání informací. Od roku 2003 se z těchto konfererencí vydělila evaluace TRECVID. Každoročně zde dochází ke srovnávání výsledků dosažených jednotlivými účastníky na společných úlohách a nejúspěšnější řešitelé prezentují svoje postupy.

Na našem ústavu se zabýváme úlohou Semantické indexace (SIN), což je proces automatického přiřazování označení (tagů) sekvencím videa, které pak mohou být využívány pro filtraci, kategorizaci prohlížení nebo vyhledávání.

Každý účastník soutěže obdrží testovací a trénovací datovou sadu a sémantické kategorie pro trénovací data, a odevzdává seznam testovacích sekvencí videa pro každou třídu, seřazený podle pravděpodobnosti toho, že je přítomna daná sémantická třída. Dříve se tato úloha nazývala High Level Feature extraction (HLF).

Poskytovaná datová sada se každoročně vyznačuje svou velikostí. Letos obsahuje 400 hodin videa pro trénování a 200 hodin videa pro testování [1]. Celkem se tedy jedná o stovky tisíc videosekvencí v délce od 10 sekund po 3.5 minuty. A velký je také počet sémantických tříd, kterých je 500. Nejedná zde pouze o typické třídy jakými by mohl být například

[1] http://www-nlpir.nist.gov/projects/tv2011/tv2011.html#data

(a) Těžná plošina

(b) Koště

(c) Kolotočové autíčko

(d) Burský ořech

Obrázek 1.1: Ukázka sémantických tříd soutěže ImageNet Challenge

člověk nebo míč, ale také třídy vyšší úrovně jako například lidé hrající fotbal, nebo třída reprezentující venkovní prostředí.

1.2 ImageNet Large Scale Visual Recognition Challenge 2011

ImageNet Challenge je vedena jako soutěž spojená s Pascal Visual Object Challenge [17]. Je to soutěž, která si klade za cíl, srovnat výsledky účastníků v oblasti odhadu obsahu fotografií neboli rozpoznávání tříd vizuálních objektů v realistických scénách.

Pro letošní ImageNet Challenge bylo stanoveno 1000 sémantických tříd, které jsou převážně zaměřeny na faunu a flóru. Ze zástupců fauny jsou zastoupeny například různé rasy koček nebo psů, a flóra je zastoupena rozličnými druhy zeleniny ovoce i jiných okrasných květin.

Rozměr datasetu je srovnatelný s datasetem TRECVID, jedná se o 1.2 milionu trénovacích a 200 tisíc testovacích fotografií získaných z databází ImageNet [2], které byly ručně označeny značkami. Na obrázku 1.1 jsou vidět ukázky sémentických tříd.

[2] http://www.image-net.org/challenges/LSVRC/2010/index#data

Kapitola 2

Teoretický základ

Existuje mnoho modelů používaných pro klasifikaci. Základním rozdělením modelů je na modely generativní, které popisují společnou distribuci pravěpodobnosti příznaků a tříd, a modely diskriminativní, které popisují podmíněnou pravděpodobnost třídy na základě daných příznaků. Mezi generativní modely řadíme rozhodovací stromy nebo gausovské modely. Diskriminativní modely jsou například neuronové sítě a Support Vector Machines (SVM). V sekci 2.1 rozeberu, jaké modely se v současnosti využívají na FIT. Dále popíši multinomiální model, který se využívá v oblasti zpracování řeči [5, 6]. V kapitole 2.3 budu popisovat model směsic multinomiálních rozložení použitý pro klasifikaci obrazu. Dalším modelem převzatým z oblasti zpracování řeči bude model PIVCO, který na FIT aktivně využívá skupina Speech@FIT pro adaptaci řečových modelu na specifický kanál, Čímž modelují různé přenosové charakteristiky každého zařízení. Já jej ale využiji pro adaptaci vizuálních kanálů, neboli pro potlačování náhodných vlivů na reprezentaci obrazu 2.4. Dále ho využiji pro redukci dimenzionality, což rozeberu v sekci 2.4. V závěru kapitoly rozeberu teorii nelineární transformace Kernel PCA 2.5.

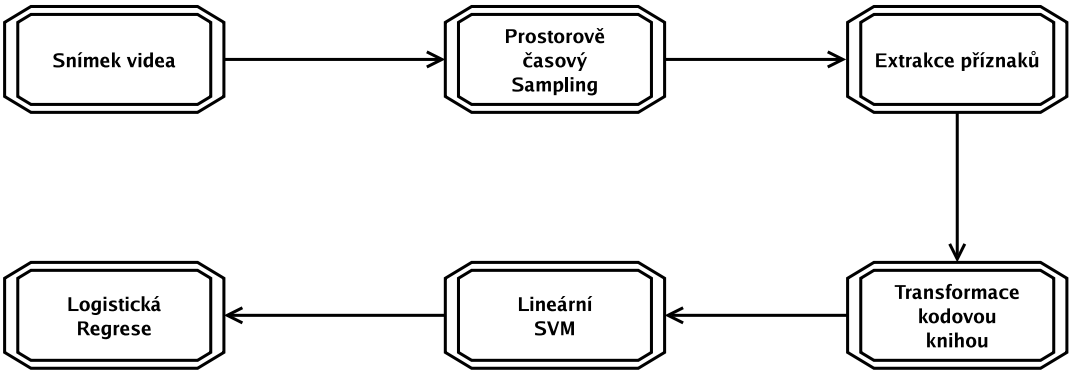
2.1 Sémantické indexování na TRECVID

Na FITu, je pro úlohu sémantického indexování použita reprezentace videosekvencí pomocí BoW reprezentace v kombinaci s Kernel SVM klasifikátorem, případně lineárním SVM klasifikátorem pro problémy, pro které již není možné udržet matici jádra v paměti. Tvorba klasifikačního modelu se dá rozdělit do dvou částí. První je konstrukce deskriptoru pro videa. Ta se dále dělí na vzorkování, extrakci příznaků a transformaci kódovou knihou. Druhou částí je trénování klasifikátorů na deskriptorech vzniklých v první části, která se dělí na trénování SVM klasifikátorů a případnou fůzi pomocí logistické regrese, která slouží ke spojení SVM klasifikátorů založených na různých typech deskriptoru. Na obrázku 2.1, je přehledně vidět celý proces.

Prostorově časový sampling

Jsou používány různé druhy samplingu (vzorkování). Dense-sampling a detekce významných oblastí pomocí Harris-Laplace bodového detektoru Pouze jeden klíčový snímek je vybrán, aby reprezentoval celou videosekvenci [7]. Navíc byly použity dva prostorově časové bodové detektory pro nalezení charakteristického obsahu přímo ve videu.

Harris-Laplace detektor vybírá stabilní oblasti s vysokou intenzitou změn a také poskytuje informace o měřítku lokální oblasti. Dense sampling vzorkuje obraz v pravidelné



Obrázek 2.1: Diagram klasifikátoru

prostorové mřížce.

Extrakce vizuálních příznaků

Jsou využívány různé druhy deskriptoru SIFT. V [13] je provedena analýza vlastností jednotlivých deskriptorů se zaměřením na invarianci vůči osvětlení. V rámci TRECVID byla použita pouze podmnožina deskriptorů. Konkrétně SIFT, rgb-SIFT, rg-SIFT, Oponent-SIFT, Hue-SIFT, HSV-SIFT, C-SIFT a oponent-histogram. Deskriptor pro 3D detektory byl SURF.

Transformace kódovou knihou

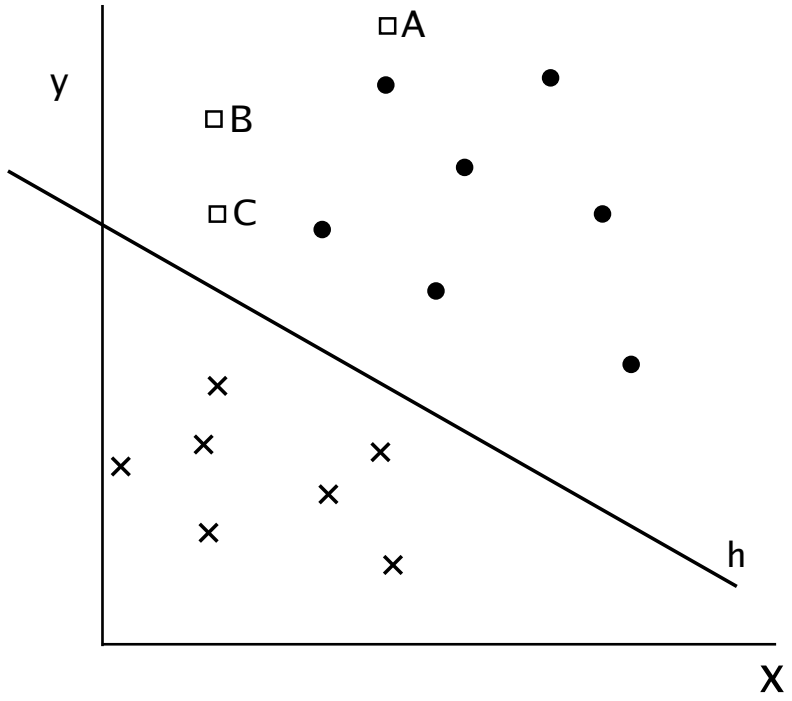
Transformace kódovou knihou převádí výsledky předchozích kroků, na prostorově nezávislou reprezentaci pomocí bag-of-words. Každé slovo (word) je prototyp rozložení vlastností, a vzorky dat jsou k těmto typům přiřazovány na základě vzdálenosti. Dohromady skupina slov tvoří kódovou knihu. Reprezentace pomocí bag-of-words zachycuje četnost výskytu jednotlivých slov [7]. Kódové knihy je možné vytvářet pomocí K-means nezávisle pro každou kombinaci vzorkování a deskriptoru. Aby se zamezilo ztrátě informace byly použity mimo hard přižazování také soft přiřazení lokálních příznaků ke kódovým slovům [4].

Klasifikace

Hlavní výzvou klasifikace je vypořádat se s obrovským množstvím dat a vysokým počtem sémantických tříd. Pro klasifikaci je v současnosti používáno lineární SVM, který popíši v sekci 2.2. Pro každou třídu je vytvořen samostatný lineární SVM klasifikátor pro každou BoW reprezentaci. Tyto klasifikátory jsou následně fůzovány pomocí logistické regrese. Pro trénování SVM je používána knihovna LIBSVM.

2.2 Support vector machine

SVM patří k nejlepším algoritmům učení s učitelem [11]. Aby mohl být SVM model popsán, musím nejdříve rozebrat teorii týkající se *margin* (hraničních) klasifikátorů. Pokud si představíme situaci na obrázku 2.2. Tak vidíme situaci kdy pozitivní (kolečka) a negativní (křízky) body jsou rozděleny klasifikátorem, např. logistickou regresí. Pravděpodobnost



Obrázek 2.2: Lineární klasifikátor s pozitivními a negativními vzorky dat

$p(y = 1|x; w, b)$, tj. pravděpodobnost, že daný vstup x, při daných parametrech w a b bude pozitivním případem, je modelována vztahem $h_{w,b} = g(w^T x + b)$[1]. Dále zde vidíme body A, B a C. Můžeme říct, že klasifikátor $h_{w,b}$ bude s větší důvěrou predikovat pozitivní případ pro bod A kdy je $w^T x + b \gg 0$. Naproti tomu pro bod C, u kterého by jen malý posun klasifikační linie mohl způsobit změnu predikce z pozitivního případu na negativní, je důvěra v predikci mnohem nižší. Proto se SVM snaží najít takovou klasifikační linii resp. hyper plochu, aby hraniční pásmo (*margin*), tj. pásmo mezi nejnedůvěryhodnějšími pozitivními a negativními predikcemi, bylo co nejširší.

Funkční a geometrická hranice – *margin*

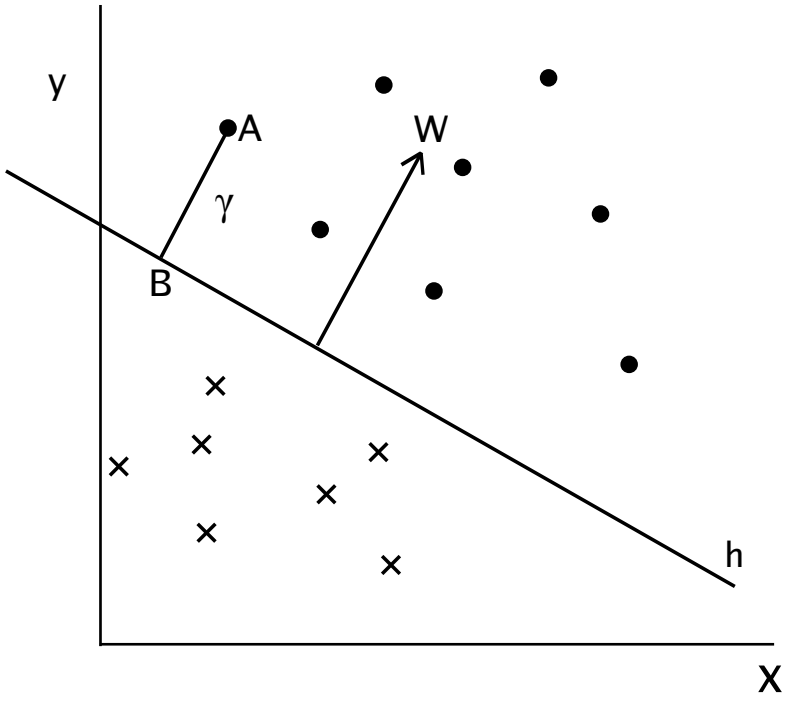
Zaveďme nyní tedy funkční a geometrické hranice – *margin*[2]. Mějme tedy trénovací vzorek $(x^{(i)},y^{(i)})$ reprezentovaný příznakovým vektorem $\mathbf{x}$ a odpovídající třídou $y \in -1, 1$. Funkční hranice je definována pomocí parametru klasifikátoru w a b následovně:

$$\hat{\gamma}^{(i)} = y^{(i)}(w^T x^{(i)} + b). \tag{2.1}$$

Pokud je $y^{(i)} = 1$, tak aby byla funkční hranice velká, tj. předpověď byla důvěryhodná a správná, je nutné, aby $w^T x^{(i)} + b$ bylo velké kladné číslo. Naproti tomu pokud je $y^{(i)} = -1$, je nutné aby $w^T x^{(i)} + b$ bylo velké záporné číslo. Takto postavená hranice říká o klasifikátoru to, že v případě, že hranice $\hat{\gamma}^{(i)}$ je kladná, je klasifikace tohoto vzorku správná.

[1] $g(z) = 1$ pro $z \geq 0$ a $g(z) = -1$ pro $z < 0$

[2] Jako překlad slova „margin“ jsem zvolil svovo hranice. Proto v této kapitole kdykoliv použiji svovo hranice, je tím myšleno právě slovo margin.



Obrázek 2.3: Grafická reprezentace geometrické hranice

Pro lineární klasifikátor a funkci g (nabývající pouze hodnot -1 a 1), je tu jedna vlastnost, která dělá z funkční hranice nedobrou míru důvěry v klasifikátor. Při této volbě funkce g platí, že i v případě náhrady w a b hodnotami $2w$ a $2b$ se výsledek predikce nezmění, protože $g(w^T x + b) = g(2w^T x + 2b)$. Protože predikce závisí pouze na znaménku a ne na velikosti. Nicméně záměna (w, b) za $(2w, 2b)$ znamená dvojnásobné zvýšení funkční hranice. Z toho vyplývá, že můžeme získat funkční hranici libovolně velkou, aniž bychom změnili cokoliv zásadního v klasifikátoru.

Při datasetu $S = (x^{(i)}, y^{(i)}); i = 1, \ldots, m$ definujeme funkční hranici pro (w, b) jako minimum z funkčních hranic jednotlivých datových bodů následovně:

$$\hat{\gamma} = \min_{i=1,\ldots,m} \hat{\gamma}^{(i)}. \tag{2.2}$$

Nyní zavedeme geometrickou hranici. Na obrázku 2.3 je vidět její grafická reprezentace. Jedná se tedy o vzdálenost mezi datovým vzorkem (bodem A) a patou kolmice B vedenou tímto bodem vůči dělící přímce h. Formální vyjádření geometrické hranice je následující:

$$\gamma^{(i)} = y^{(i)} \left(\left(\frac{w}{|w|} \right)^T x^{(i)} + \frac{b}{|w|} \right). \tag{2.3}$$

Pokud je velikost vektoru $|w|$ rovna jedné, pak se funkční a geometrická hranice rovnají. Takto definovaná geometrická hranice 2.3 je invariantní vůči změně velikosti w a b. A geometrická hranice celého datasetu $S = (x^{(i)}, y^{(i)}); i = 1, \ldots, m$ je definována takto:

$$\gamma = \min_{i=1,\ldots,m} \gamma^{(i)}. \tag{2.4}$$

Optimální *margin* klasifikátor

Hlavní myšlenkou optimálního klasifikátoru je nalezení hraniční linie respektive hyperplochy, která maximalizuje geometrickou hranici. Takovýto klasifikátor by tedy dával pouze velmi důvěryhodné predikce pro trénovací dataset. Předpokládejme nyní, že dataset je lineárně separovatelný. Formálně definovaný problém hledání optimálního klasifikátoru je pak následující:

$$\max_{\gamma,w,b} \quad \gamma \tag{2.5}$$

$$\text{aby platilo:} \quad y^{(i)}(w^T x^{(i)} + b) > \gamma, i = 1, \ldots, m \tag{2.6}$$

$$|w| = 1. \tag{2.7}$$

Maximalizujeme γ tak, aby pro každý trénovací vzorek byla geometrická hranice rovna alespoň γ. Omezující podmínka $|w| = 1$ zajišťuje to, aby se funkční a geometrická hranice rovnaly. Tudíž vyřešení tohoto optimalizačního problému by znamenalo nalezení hotnot (w, b) takových, že by všechny trénovací vzorky dosahovaly co největší geometrické hranice.

Pokud bychom byli schopni tento problém vyřešit, byla by diskuze o SVM ukončena, problémem však je omezující podmínka $|w| = 1$, která je nekonvexní a tudíž neexistují standardní nástroje pro hledání řešení. Řešení je ale možné nalézt převedením na ekvivaletní problém [11]:

$$\max_{\gamma,w,b} \quad \frac{\hat{\gamma}}{|w|} \tag{2.8}$$

$$\text{aby platilo:} \quad y^{(i)}(w^T x^{(i)} + b) > \hat{\gamma}, i = 1, \ldots, m. \tag{2.9}$$

Zde maximalizujeme $\frac{\hat{\gamma}}{|w|}$ tak, aby funkční hranice byla nejméně $\hat{\gamma}$. Zbavili jsme se sice nekonvexní podmínky $|w| = 1$, ale místo toho jsme získali nekonvexní cílovou funkci, kterou musíme optimalizovat. Vzpomeňme si tedy na fakt, že pokud koeficienty (w, b) vynásobíme libovolnou konstantou, nezmění se výsledek klasifikace. Tohoto faktu budeme nyní využívat. Zavedeme tedy škálovací konstantu takovou, že funkční hranice $\hat{\gamma}$ bude rovna 1. Takto upravenou funkční hranici $\hat{\gamma} = 1$ a konstantou vynásobené koeficienty (w, b) použijeme opět v našem optimalizačním problému, s tím, že nahradíme maximalizaci $\frac{1}{|w|}$ za minimalizaci $|w|^2$, která poskytuje stejné výsledky. Formalizace problému je pak následující:

$$\min_{\gamma,w,b} \quad \frac{1}{2}|w|^2 \tag{2.10}$$

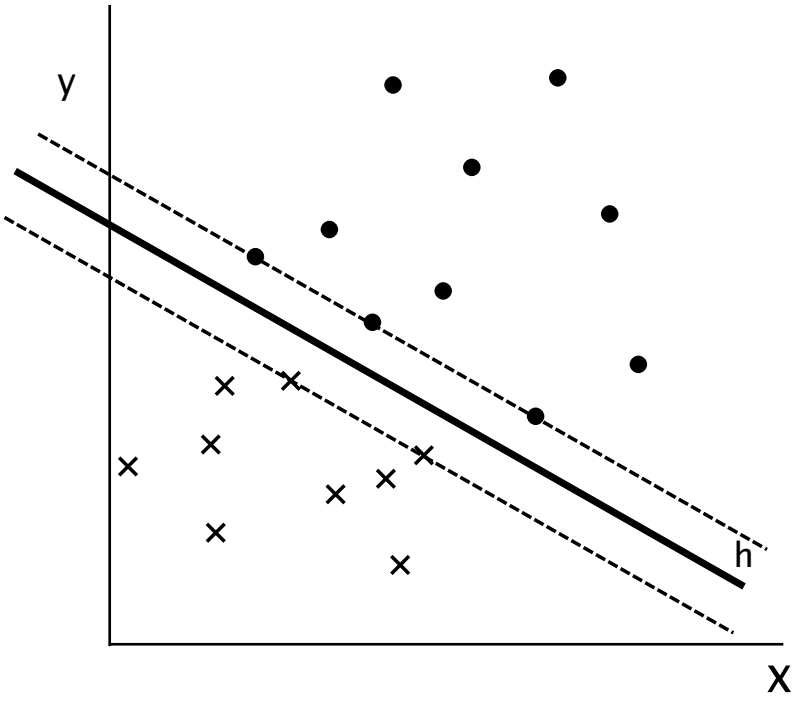
$$\text{aby platilo:} \quad y^{(i)}(w^T x^{(i)} + b) > 1, i = 1, \ldots, m. \tag{2.11}$$

Takto upravený problém je již možné řešit nástroji pro optimalizaci a výsedek tohoto problému poskytuje **optimální *margin* klasifikátor**.

Podmínku 2.11 můžeme rozepsat pro každý datový vzorek následovně:

$$g_i(w) = -y^{(i)}(w^T x^{(i)} + b) + 1 \leq 0. \tag{2.12}$$

Datové prvky $x^{(i)}$, které mají podmínku $g = 0$ 2.12 se nazývají **Support vektory**. Na obrázku 2.4 je vidět příklad takovýchto datových vzorků. Jsou zde celkem tři pozitivní a dva negativní support vektory.



Obrázek 2.4: Support vektory optimálního *margin* klasifikátoru

Lagrangeho dualita

Nyní se krátce oprostíme od problematiky SVM a nastíníme si problematiku duality problémů, kterou následně využijeme při převodu optimalizačního problému 2.10 na jiný ekvivalentní problém. Představme si nyní, že máme následující problém:

$$\min_{w} \quad f(w) \tag{2.13}$$

$$\text{aby platilo:} \quad g_i(w) \leq 0, i = 1, \ldots, k \tag{2.14}$$

$$h_i(w) = 0, i = 1, \ldots, l. \tag{2.15}$$

Pokud převedeme tento problém na zobecněný Lagrangian dostaneme následující:

$$\mathcal{L}(w, \alpha, \beta) = f(w) + \sum_{i=1}^{k} \alpha_i g_i(w) + \sum_{i=1}^{l} \beta_i h_i(w). \tag{2.16}$$

Zde jsou α_i a β_i jsou Lagrangeho koeficienty. Vytvoříme-li tedy funkci $\theta_{\mathcal{P}}$ definovanou následovně:

$$\theta_{\mathcal{P}}(w) = \max_{\alpha,\beta:\alpha_i \geq 0} \mathcal{L}(w, \alpha, \beta). \tag{2.17}$$

$\mathcal{P}$ zde reprezentuje primální problematiku. Takto definovaná funkce 2.17 splňuje následující podmínky. První, v případě, že nějaké w splňuje primální podmínky 2.14, je tato funkce rovna $f(w)$. Druhá podmínka zaručuje, že v případě, kdy je porušena některá z podmínek, je funkce rovna $+\infty$. Proto funkce $\theta_{\mathcal{P}}$ splňuje stejný úkol jako funkce $f(w)$, proto

pokud budeme minimalizovat funkci $\theta_{\mathcal{P}}(w)$, dostaneme stejný optimalizační problém jako 2.13. Formální zápis problému je následující:

$$\min_{w} \theta_{\mathcal{P}}(w) = \min_{w} \max_{\alpha,\beta:\alpha_i \geq 0} \mathcal{L}(w, \alpha, \beta). \tag{2.18}$$

Nyní zavedeme duální problém. Nejdříve definujeme $\theta_{\mathcal{D}}()$:

$$\theta_{\mathcal{D}}(\alpha, \beta) = \min_{w} \mathcal{L}(w, \alpha, \beta). \tag{2.19}$$

V duálním problému budeme provádět maximalizaci $\theta_{\mathcal{D}}(\alpha, \beta)$:

$$\max_{w} \theta_{\mathcal{D}}(\alpha, \beta) = \max_{\alpha,\beta:\alpha_i \geq 0} \min_{w} \mathcal{L}(w, \alpha, \beta). \tag{2.20}$$

Oba problémy, jak duální, tak primální jsou stejné, až na pořadí funkcí min a max. Vztah těchto problémů je následující:

$$d^* = \max_{\alpha,\beta:\alpha_i \geq 0} \min_{w} \mathcal{L}(w, \alpha, \beta) \leq \min_{w} \max_{\alpha,\beta:\alpha_i \geq 0} \mathcal{L}(w, \alpha, \beta) = p^* \tag{2.21}$$

kde p^* a d^* jsou řešení primálního respektive duálního problému. Pokud se budeme zabývat problematikou SVM, tak pokud jsou dodrženy Karush-Kuhn-Tuckerovi podmínky, jsou řešení primárního a sekundárního problému stejná [11, 3]. Definice podmínek je následující:

$$\frac{\partial}{\partial w_i} \mathcal{L}(w^*, \alpha^*, \beta^*) = 0, i = 1, \ldots, n \tag{2.22}$$

$$\frac{\partial}{\partial \beta_i} \mathcal{L}(w^*, \alpha^*, \beta^*) = 0, i = 1, \ldots, l \tag{2.23}$$

$$\alpha_i^* g_i(w^*) = 0, i = 1, \ldots, k \tag{2.24}$$

$$g_i(w^*) \leq 0, i = 1, \ldots, k \tag{2.25}$$

$$\alpha^* \geq 0, i = 1, \ldots, k \tag{2.26}$$

kde w^*, α^* a β^* jsou řešení jak duálního, tak primárního problému. Převod optimalizačního SVM problému 2.10 umožňuje převést veškerou práci s daty při trénování klasifikátoru na práci se skalárními součiny dat. To pak umožňuje použít při trénování jádra, místo samotných dat.

Když se tedy vrátíme k problému 2.10 bude jeho Lagrangeho funkce vypadat následovně:

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2}|w|^2 - \sum_{i=1}^{m} \alpha_i [y^{(i)}(w^T x^{(i)} + b) - 1]. \tag{2.27}$$

Ve vztahu nejsou žádné lagnangeho koeficienty β_i, protože problém nemá žádné podmínky s rovností. Pokud budeme hledat duální problém, musíme nejdříve minimalizovat $\mathcal{L}(w, b, \alpha)$ vzhledem k w a b. Abychom tak učinili, musíme nastavit parciální derivace podle w a b položit rovny 0:

$$\frac{\partial}{\partial w} \mathcal{L}(w, b, \alpha) = w - \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)} = 0 \tag{2.28}$$

$$\frac{\partial}{\partial b} \mathcal{L}(w, b, \alpha) = \sum_{i=1}^{m} \alpha_i y^{(i)} = 0 \tag{2.29}$$

Pokud vezmeme 2.28, vyjádříme w a dosadíme jej do 2.27 a zjednodušíme dostaneme:

$$\mathcal{L} = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1} y^{(i)} y^{(j)} \alpha_i \alpha_j (x^{(i)})^T x^{(j)} - b \sum_{i=1}^{m} y^{(i)}. \tag{2.30}$$

Navíc rovnice 2.29 říká, že poslední člen předchozí rovnice musí být roven 0, proto se výraz zjednoduší na:

$$\mathcal{L} = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1} y^{(i)} y^{(j)} \alpha_i \alpha_j (x^{(i)})^T x^{(j)}, \tag{2.31}$$

což už je cílová funkce našeho duálního problému. Formálně zapsaný duální problém vypadá následovně:

$$\max_{\alpha} \quad W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1} y^{(i)} y^{(j)} \alpha_i \alpha_j < x^{(i)}, x^{(j)} > \tag{2.32}$$

$$\text{aby platilo:} \quad \alpha_i \geq 0, i = 1, \ldots, m \tag{2.33}$$

$$\sum_{i=1}^{m} \alpha_i y^{(i)} = 0, \tag{2.34}$$

kde $< x^{(i)}, x^{(j)} >$ je skalární součin datových vzorků i a j. Takto upravený problém splňuje KKT podmínky, a tím pádem vyřešením tohoto problému vyřešíme i primární problém. Navíc má tato reprezentace již zmíněnou výhodu v práci s daty. Místo $< x^{(i)}, x^{(j)} >$ je možné vložit libovolné validní jádro. Stejně tak pokud budeme počítat odezvu klasifikátoru pro nová data, budeme nuceni vypočítat vztah $w^T + b$ a porovnat hodnotu výsledku s nulou. Pokud ovšem vezmeme w z rovnice 2.28 dostaneme vztah:

$$w^T x + b = (\sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)})^T x + b = \sum_{i=1}^{m} \alpha_i y^{(i)} < x^{(i)}, x > + b. \tag{2.35}$$

Opět jsme vyjádřili výpočet ve formě skalárních součinů, a tudíž je možné nahradit $< x^{(i)}, x >$ jádrovou transformací.

2.3 Multinomiální rozložení

V klasifikačních úlohách se často setkáváme s problémem [1], kdy proměnné mohou nabývat K vzájemně vyloučených hodnot. Jedná se tedy o rozšíření Bernouliho rozložení, které může nabývat pouze dvou stavů, na vícestavovou logiku. Tato situace je vyjádřena vektorem $\mathbf{x}$, ve kterém je právě jeden člen x_k roven 1 a všechny ostatní prvky rovny 0. Pro příklad, když máme proměnnou, která může nabývat $K = 6$ stavů, tak její konkrétní instance může vypadat následovně:

$$\mathbf{x} = (0, 0, 1, 0, 0, 0). \tag{2.36}$$

Za povšimnutí stojí fakt, že $\sum_{k=1}^{K} x_k = 1$. Pokud tedy řekneme, že μ_k značí pravděpodobnost, že $x_k = 1$, tak rozložení $\mathbf{x}$ je dáno následovně:

$$p(\mathbf{x}|\boldsymbol{\mu}) = \prod_{k=1}^{K} \mu_k^{x_k} \tag{2.37}$$

kde $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_K)$, a parametry μ_k jsou omezeny tak, aby splňovaly $\mu_k \geq 0$ a $\sum_k \mu_k = 1$, protože reprezentují pravděpodobnosti. Distribuční funkce 2.37 je jak již bylo řečeno, generalizací Bernouliho rozložení na více jak dva možné stavy. Nyní zvažme datovou sadu $\mathcal{D}$ o N nezávislých pozorováních $\mathbf{x_1}, \ldots, \mathbf{x_N}$. Odpovídající likelihood funkce nabývá tvaru:

$$p(\mathcal{D}|\boldsymbol{\mu}) = \prod_{n=1}^{N} \prod_{k=1}^{K} \mu_k^{x_{nk}} = \prod_{k=1}^{K} \mu_k^{(\sum_n x_{nk})} = \prod_{k=1}^{K} \mu_k^{m_k}. \tag{2.38}$$

Likelihood závisí na N datových bodech pouze skrze K hodnot:

$$m_k = \sum_n x_{nk} \tag{2.39}$$

které reprezentují počet pozorování, jež mají $x_k = 1$.

Pokud tedy budeme uvažovat společnou distribuci počtů $m_1, \ldots, m_K$, podmíněné $\boldsymbol{\mu}$ a celkovým počtem pozorování N. Tak výsledná distribuce vypadá takto:

$$Mult(m_1, m_2, \ldots, m_K|\boldsymbol{\mu}, N) = \binom{N}{m_1 m_2 \cdots_K} \prod_{k=1}^{K} \mu_k^{m_k}, \tag{2.40}$$

což je multinomiální rozložení. Normalizační koeficient je počet možností jak lze rozdělit N objektů do K skupin o velikostech $m_1, \ldots, m_K$ a vypočte se následovně:

$$\binom{N}{m_1 m_2 \cdots_K} = \frac{N!}{m_1! m_2! \cdots_K!}. \tag{2.41}$$

Multinomiální model je možné rozšířit na směsice K multinomiálních rozložení. Směsice multinomiálních rozložení mohou zajišťovat větší flexibilitu modelu v situacích, kdy je datová sada pro jeden model natolik různorodá, že tuto diverzitu není model schopen zachytit. Formulace modelu pro datovou sadu $\mathbf{x}$ následující rovnici:

$$p(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\pi}) = \sum_{k=1}^{K} \pi_k p(\mathbf{x}|\boldsymbol{\mu_k}) \tag{2.42}$$

kde $\boldsymbol{\pi} = \{\pi_1, \ldots, \pi_K\}$ jsou váhové koeficienty jednotlivých multinomiálních rozložení, jsou omezeny tak, aby splňovali $\pi_k \geq 0$ a $\sum_k \pi_k = 1$. Dále $\mathbf{x} = m_1, \ldots, m_N$ je vektor počtu pozorování, μ jsou koeficienty jednotlivých rozložení a konečně p je multinomiální rozložení.

Expectation maximization algoritmus pro směsice multinomiálních rozložení

Expectation maximization algoritmus je obecná technika pro nalezení Maximum likelihood řešení pravděpodobnostních modelů [1]. Obecný postup algoritmu je následující:

1. Inicializuj modely náhodnými hodnotami.
2. Expectation krok - Přiřaď k současné iteraci modelů prvky, které mají nejvyšší odezvu pro daný model.
3. Maximization Krok - Proveď maximum likelihood odhad parametrů modelu, se všemi prvky přiřazenými danému modelu v této iteraci.
4. Ukonči algoritmus pokud žádný z prvků nezměnil model, nebo pokud byl vyčerpán maximální počet iterací, jinak skoč na krok 2.

Pro upravení obecného postupu EM algoritmu pro směsice multinomiálních rozložení budeme uvažovat datovou sadu $X = \{\mathbf{x_1}, \ldots, \mathbf{x_n}\}$, kde každý vektor $\mathbf{x_k}$ má šířku D. Potom log-likelihood funkce dat vypadá následovně:

$$\ln p(\mathbf{X}|\boldsymbol{\mu}, \boldsymbol{\pi}) = \sum_{n=1}^{N} \ln \left\{ \sum_{k=1}^{K} \pi_k p(\mathbf{x_n}|\boldsymbol{\mu_k}) \right\} \tag{2.43}$$

kde K je počet komponent a N je počet vstupních dat. Pro vytvoření EM algoritmu nejdříve musíme zavést latentní proměnnou $\mathbf{z}$ asociovanou s každou instancí $\mathbf{x}$. Vektor $\mathbf{z}$ je reprezentován K-rozměrným binárním vektorem, který má právě jednu komponentu rovnou 1 a ostatní rovny 0. Podmíněná distribuční funkce $\mathbf{x}$ za předpokladu $\mathbf{z}$ a $\boldsymbol{\mu}$ je zapsána takto:

$$p(\mathbf{x}|\mathbf{z}, \boldsymbol{\mu}) = \prod_{k=1}^{K} p(\mathbf{x}|\boldsymbol{\mu_k})^{z_k} \tag{2.44}$$

a distribuce prioru pro latentní proměnnou je tato:

$$p(\mathbf{z}|\boldsymbol{\pi}) = \prod_{k=1}^{K} \pi_k^{z_k}. \tag{2.45}$$

Pokud vynásobíme $p(\mathbf{x}|\mathbf{z}, \boldsymbol{\mu})$ s $p(\mathbf{z}|\boldsymbol{\pi})$ a marginalizujeme výsledek přes $\mathbf{z}$ dostaneme 2.42. Nyní sepíšeme log-likelihood funkci pro všechna data (včetně latentní proměnné $\mathbf{z}$):

$$\ln p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\mu}, \boldsymbol{\pi}) = \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk} \left\{ \ln \pi_k + \sum_{i=1}^{D} [x_{ni} \ln \mu_{ki} + (1 - x_{ni}) \ln(1 - \mu_{ki})] \right\} \tag{2.46}$$

kde $\mathbf{X} = \{\mathbf{x_n}\}$ a $\mathbf{Z} = \{\mathbf{z_n}\}$. Následně vezmeme očekávanou hodnotu (expectation) log-likelihood funkce dat vzhledem k distribuci posterioru latentní proměnné a získáme:

$$\mathbb{E}_Z[\ln p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\mu}, \boldsymbol{\pi})] = \sum_{n=1}^{N}\sum_{k=1}^{K}\gamma(z_{nk})\left\{\ln \pi_k + \sum_{i=1}^{D}[x_{ni}\ln \mu_{ki} + (1-x_{ni})\ln(1-\mu_{ki})]\right\} \quad (2.47)$$

kde $\gamma(z_{nk}) = \mathbb{E}[z_{nk}]$ je posteriorní pravděpodobnost, neboli responsibility (zodpovědnost), komponenty k za daný datový vzorek $\mathbf{x_n}$. V každém E-kroku jsou spočítány tyto zodpovědnosti pomocí Bayesova teorému:

$$\gamma(z_{nk}) = \mathbb{E}[z_{nk}] = \frac{\sum\limits_{z_{nk}} z_{nk}[\pi_k p(x_n|\mu_k)]^{z_{nk}}}{\sum\limits_{z_{nj}}[\pi_j p(x_n|\mu_j)]^{z_{nj}}} = \frac{\pi_k p(\mathbf{x_n}|\boldsymbol{\mu_k})}{\sum\limits_{j=1}^{K}\pi_j p(\mathbf{x_n}|\boldsymbol{\mu_j})}. \quad (2.48)$$

Pokud provedeme sumu přes všechny n v 2.47 tak vidíme, že zodpovědnosti přispívají k výpočtu pouze skrze vztah:

$$N_k = \sum_{n=1}^{N}\gamma(z_{nk}) \quad (2.49)$$

$$\overline{\mathbf{x}}_{ki} = \frac{1}{N_k}\sum_{n=1}^{N}\gamma(z_{nk})x_{ki} \quad (2.50)$$

kde N_k je efektivní počet datových vzorků asociovaných s komponentou k. Index i určuje, kterou pravděpodobnost komponenty k zrovna počítáme.

V M-kroku maximalizujeme log-likelihood funkci pro data vzhledem k parametrům $\boldsymbol{\mu_k}$ a $\boldsymbol{\pi}$. Pokud nastavíme derivaci 2.47 rovnu nule tak dostaneme:

$$\boldsymbol{\mu_k} = \overline{\mathbf{x}}_k. \quad (2.51)$$

Pro maximalizaci vzhledem k π_k zavedeme Lagrangeův multiplikátor, abychom zajistili omezení $\sum_k \pi_k = 1$. Dostaneme následující vztah:

$$\pi_k = \frac{N_k}{N} \quad (2.52)$$

což je poměr počtu efektivních datových vzorků pro komponentu k ku všem datovým vzorkům.

2.4 Phonotactic intersession variation compensation

V oblasti rozpoznávání řečičníka je v současné době Joint Factor Analysis (JFA) špičkou mezi rozpoznávacími mechanismy [9]. Důvod proč je tato metoda využívaná, je fakt, že využívá pravděpodobnostní přístup pro modelování různých typů variability parametrů modelu [6]. Myšlenkou systému PIVCO [5, 6] je adaptovat modely multinomiálního rozložení 2.3 tak, aby užívaly stejnou techniku.

V JFA se pro modelování řeči využívá směsí gausovských rozložení, a během trénování se odhadují parametry gausových rozložení. Pro Maximum likelihood odhad je úloha výpočetně náročná pro velké datové sady. Naproti tomu v multinomiálních modelech jsou

jedinnými parametry, které je nutné trénovat, pravděpodobnosti μ. Ty jsou omezeny podnínkami následovně:

$$\mu > 0 \tag{2.53}$$

$$\sum_i \mu_i = 1. \tag{2.54}$$

Abychom zabránili porušení první podmínky 2.53, zavedeme v logaritmickém počtu následující vztah:

$$q_i = m_i + \mathbf{u_i x_i} \tag{2.55}$$

kde m_i je $\log \mu_i$, u_i je i-tý řádek faktorové (kompenzační) matice U a x je vektor faktoru neboli váhový vektor. Druhá podmínka 2.54 je zajištěna následujícím vztahem:

$$\omega_i = \frac{q_i}{\sum_{j=1}^{K} q_j}. \tag{2.56}$$

Takto ošetřený prvek ω_i je pak korektní parametr pro multinomiální rozložení. Přepis rozložení $\log p(\mathbf{x}|\boldsymbol{\mu}) = \sum_{i=1}^{K} x_i \log \mu_i$ do podoby používající ω vypadá následovně:

$$\log p(\mathbf{x}|\boldsymbol{\mu}) = \sum_{i=1}^{K} x_i \omega_i \tag{2.57}$$

nebo alternativně:

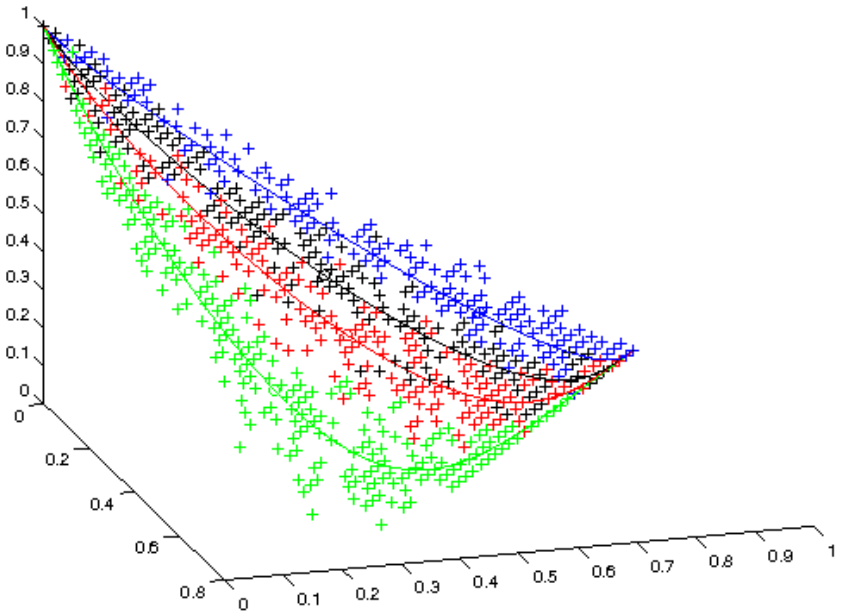
$$\log p(\mathbf{x}|\boldsymbol{\mu}) = \sum_{i=1}^{K} x_i \log \frac{e^{m_i + u_i x}}{\sum_{j=1}^{K} e^{m_j + u_j x}}. \tag{2.58}$$

Kompenzace kanálu modelem PIVCO

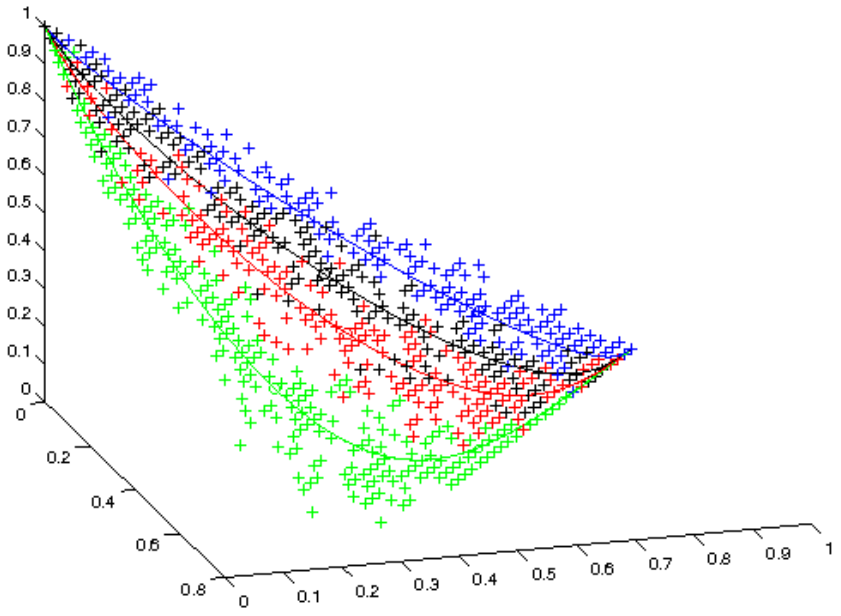
Použití systému PIVCO pro adaptaci modelu na daný kanál, je základním typem jeho použití. Demonstrační příklad adaptace modelu na kanál má následující popis. Vytvořil jsem syntetická data generovaná podle multinomiálních modelů s přidaným gausovským šumem. Celkem jsem nageneroval 2000 vzorků. V příkladu jsem použil čtyři výchozí multinomiální modely s počtem pravděpodobností (K), rovným třem. Kompenzační faktorová matice měla šířku 1 faktor.

Obrázek 2.5 je vidět situaci před trénováním. Vzorky pro každý model jsou znázorněny křížky jiné barvy. Křivka procházející datovou sadou je současný odhad kanálu pro nenatrénovanou matici U. 2.6 ukazuje situaci po natrénování matice U. Je zde vidět mírný posun křívek v pravé části směrem nahoru, a v levé části směrem dolů. Kompenzace kanálu je při takto malém počtu faktorů jen málo znatelná, nicméně i přesto, je vidět posun odhadovaného kanálu tak, aby lépe korespondoval s datovou sadou.

Tvoříme-li tedy klasifikátor pro n tříd, je třeba natrénovat n Kompenzačních matic U. Každou matici U pouze na datech patřící této třídě. Váhové vektory x se při tomto použití interpretují jako míra vlivu daného faktoru.



Obrázek 2.5: Datová sada a odhad kanálu před trénováním



Obrázek 2.6: Datová sada a odhad kanálu po trénování

Redukce dimenze modelem PIVCO

Alternativním použitím modelu PIVCO je užití jeho adaptačních schopností pro redukci dimenzionality vstupních dat. Definice modelu PIVCO v takovém případě zůstává stejná, liší se však užití tohoto modelu a interpretace jednotlivých částí. Matice U je při redukci dimenzionality trénována na **všech** datech bez ohledu na příslušnost ke třídám. Váhové vektory x se pak považují za nové vektory příznaků. Velikost těchto nových vektorů příznaků pak záleží pouze na šířce faktorové matice.

2.5 Kernel PCA

V této sekci nejdříve rozeberu co jsou to jádra, a jejich potenciální využití. Poté se v krátkosti zmíním o Analýze hlavních komponent (PCA) a v závěru rozeberu samotné Kernel PCA.

Jádra

Představme si, že máme klasifikovat data lineární regresní funkcí. Řekněme, že tato data jsou viditelně lineárně neseparovatelná. Jedna z možností je použít vstupní data $\mathbf{x}$ bez další úpravy a spokojit se s neoptimálním výsledkem. Nabízí se zde však možnost použít jako vstup lineární regresní funkce nejen $\mathbf{x}$ ale také $\mathbf{x}^2$ a $\mathbf{x}^3$. Takovýmto postupem získáme kubickou funkci a zvyšuje se šance, že data budou separovatelná. V této subsekci budu nazývat hodnoty vstupů atributy, a vektory $(\mathbf{x}, \mathbf{x}^2$ až $\mathbf{x}^3)$ příznaky. Dále budu nazývat funkci Φ mapovací funkcí. Příkladem mapovací funkce může být například toto:

$$\Phi(\mathbf{x}) = \begin{bmatrix} \mathbf{x} \\ \mathbf{x}^2 \\ \mathbf{x}^3 \end{bmatrix}. \tag{2.59}$$

Pokud bychom tedy chtěli používat příznaky, místo atributů, znamenalo by to nahradit každý výskyt $\mathbf{x}$ za $\Phi(\mathbf{x})$. Protože ale existují algoritmy, které veškerou práci s daty provádí ve formě skalárního součinu dat $\mathbf{x}^T\mathbf{z}$ je možné přepsat tyto výpočty na $\Phi(\mathbf{x})^T\Phi(\mathbf{z})$. Proto můžeme definovat skalární součin výsledků mapovacích funkcí jako Jádro – *Kernel*:

$$K(\mathbf{x}, \mathbf{z}) = \Phi(\mathbf{x})^T\Phi(\mathbf{z}). \tag{2.60}$$

Kdekoliv, kde jsme ve výpočtu měli skalární součin dvou vektorů příznaků, můžeme jednoduše nahradit skalární součin jádrem $K(\mathbf{x}, \mathbf{z})$. Zajímavým faktem je, že výpočet jádra je často velmi nenáročnou procedůrou. Můžeme nahradit v algoritmech atributy, vysokodimenzionálním vektorem příznaků, a tím pádem zlepšit rozhodovací schopnost klasifikátoru, aniž bychom museli explicitně vyčíslovat mapovací fukci Φ.

Řekněme tedy, že máme jádrovou funkci $K(x, z) = (x^T z + c)^d$. To odpovídá mapování atributů do $\binom{n+d}{d}$-rozměrného vektoru příznaků[3]. A přestože pracujeme v vysokodimenzinálním prostoru výpočet jádra je stále lineární v kontrastu k výpočtu mapovací fukce, který je $\mathcal{O}(n^d)$ [11].

Pokud tedy budeme dále přemýšlet nad jádrovými funkcemi, budeme možná chtít aby tato funkce pro data, která jsou si podobná, dávala velké hodnoty, a pro data, která jsou

[3] n zde znamená šířku vektoru $\mathbf{x}$

hodně odlišná dávala nízké hodnoty. Chceme tedy, aby jádro bylo jakousi mírou podobnosti dat. Příkladem takovéhoto jádra je Gausovské jádro:

$$K(x,z) = \exp\left(\frac{|x-z|^2}{2\sigma^2}\right), \tag{2.61}$$

kde σ^2 značí standardní odchylku. Toto jádro odpovídá převodu atributů do nekonečně rozměrného prostoru příznaků, tudíž samotná mapovací funkce odpovídající tomuto jádru není vyčíslitelná.

Aby bylo jádro validní, musíme si nejdříve definovat matici jádra, neboli *Kernel matrix*. Tu definujeme nad n datovými body (nemusí se nutně jednat o trénovací nebo testovací dataset). Jádrová matice je tedy čtvercová n-krát-n matice. Její prvky K_{ij} jsou definovány jako $K_{ij} = K(x_i, x_j)$. Aby se jednalo o validní jádro, musí tato matice splňovat Mercerův teorém:

> Nechť máme matici $K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$. Pak pokud má tato matice být validní, je nutné a dostačující, aby tato matice byla pro libovolne datové prvky $\{x_1, \ldots, x_n\}$ $(n < \infty)$ symetrická pozitivně definitní matice [11].

Pozitivní definitnost znamená, že všechna vlastní čísla této matice jsou větší než nula.

Dalšími validními jádry, se kterými budu provádět výpočty jsou addχ^2, Intersection a Bhattacharyya jádra:

$$K_{add\_chi2}(x,z) = 2\sum_{d=1}^{D}\frac{x(d)z(d)}{x(d)+z(d)} \tag{2.62}$$

$$K_{inter}(x,z) = \sum_{d=1}^{D}\min(x(d), z(d)) \tag{2.63}$$

$$K_{bha}() = \sum_{d=1}^{D}\sqrt{x(d)z(d)}, \tag{2.64}$$

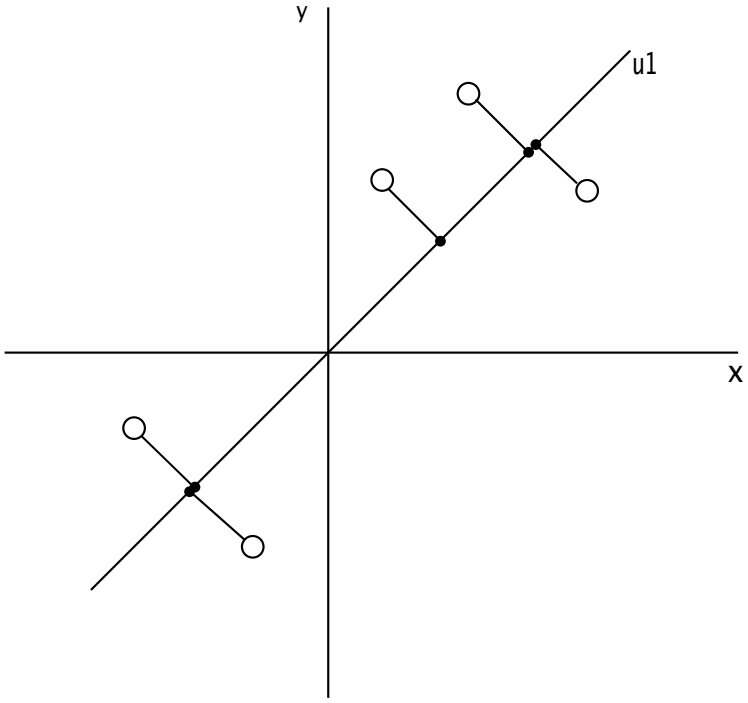
kde D je počet dimenzí vektoru příznaků a $x(d)$ je d-tá dimenze vektoru příznaků x. Všechna výše zmíněná jádra mají tu vlastnost, že jsou aditivními jádry. To znamená, že jejich výpočet lze provést odděleně pro každou dimenzi a výsledky pak sečíst. Všechna jádra mohou být převedena na do exponencální formy:

$$K_{exp} = \exp(\gamma(K-1)), \tag{2.65}$$

kde γ může být nastavena jako převrácená hodnota průměru ne-exponencializovaných hodnot jádra a -1 ve výpočtu exponentu zajišťuje to, že jsou hodnoty zhora ohraničeny 1. Jádro $K^{\exp}_{inter}$ je známo jako Laplaciánovo jádro, a $K^{\exp}_{bha}$ je známo jako *radial basis function* (RBF) jádro [12].

Dalším jádrem, které v jsem používal je neaditivní verze χ^2 jádra:

$$K^{\exp}_{chi2}(x,z) = \exp\left(-\gamma\left(\sum_{d=1}^{D}\frac{(x(d)-z(d))^2}{x(d)+z(d)}\right)\right). \tag{2.66}$$



Obrázek 2.7: Dataset promítnutý na vektor u_1. Dosažena velká variance bodů.

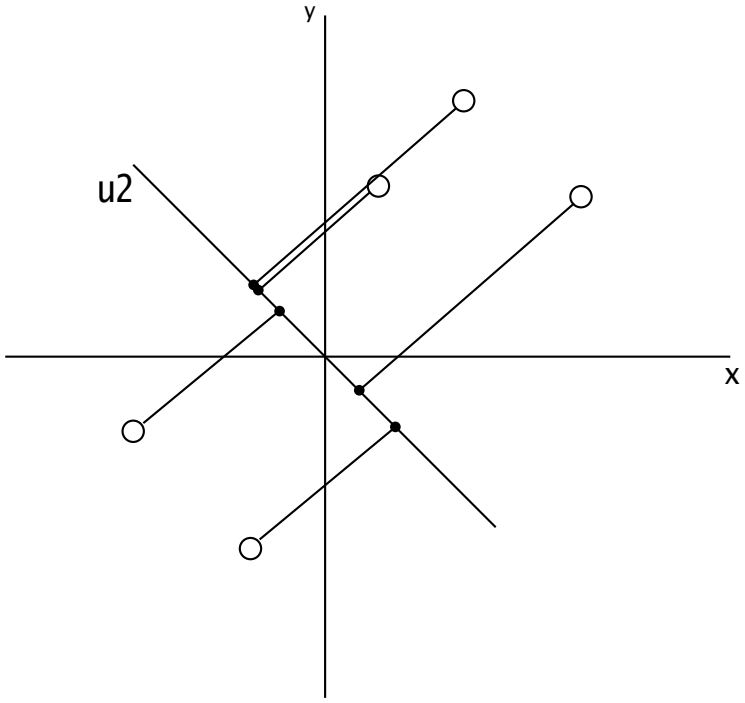
PCA

Analýza hlavních komponent se snaží v datech nalézt skryté korelace mezi prvky. Hlavní komponentou je zde vektor, na nějž se promítají datové vzorky. V Analýze hlavních komponent se snažíme najít vždy takové hlavní komponenty, abychom maximalizovali varianci promítnutých bodů. Mějme situaci na obrázku 2.7. Kroužky zde reprezentují datové vzorky. Černé tečky reprezentují data promítnutá do prostoru daného vektorem $u1$. Takto promítnuté vektory vykazují velkou varianci.

Opačná situace je na obrázku 2.8. Zde body dosahují malé variance. Úkolem PCA je hledat takové vektory u_i, které maximalizují varianci datasetu. Formálně definujeme problém takto. Mějme vektor u a bod x. Délka projekce, neboli vzdálenost od počátku, se spočte jako $x^T u$. Takže pokud chceme maximalizovat varianci, budeme maximalizovat následující vztah:

$$\max_u \frac{1}{m} \sum_{i=1}^{m} (x_i^T u)^2 = u^T \left(\frac{1}{m} \sum_{i=1}^{m} \right) u. \tag{2.67}$$

Pokud navíc stanovíme velikost velikost vektoru u na 1, dostaneme vlastní vektor kovarianční matice $\Sigma = \frac{1}{m} \sum_{i=1}^{m} x_i x_i^T$, za předpokladu, že data mají nulový průměr. Zjistili jsme tedy, že pokud chceme najít jednorozměrný podprostor, stačí vybrat jeden vlastní vektor kovarianční matice Σ a data do tohoto prostoru promítnout. Obecně tedy pokud chceme nalézt k-rozměrný podprostor, stačí vzít k vlastních vektorů Σ. Vlastní vektory u_i pak tvoří novou ortogonální bázi dat. Takto nalezené vlastní vektory se nazývají hlavní komponenty [10].



Obrázek 2.8: Dataset promítnutý na vektor u_2. Dosažena malá variance bodů.

Kernel PCA

V PCA jsme hledali vlastní čísla a vektory kovarianční matice nad daty. Nyní předpokládáme, že data jsou transformována do prostoru příznaků funkcí $\Phi(x)$ tj. $\Phi(x_i), \ldots, \Phi(x_m)$. Další podmínku, kterou klademe na je jejich centrovanost okolo 0 čili $\sum_{i=1}^{m} \Phi(x_i) = 0$. Pro provedeni PCA nad daty promítnutými do prostoru příznaků musíme spočítat kovarianční matici:

$$\bar{C} = \frac{1}{m} \sum_{i=1}^{m} \Phi(x_i)\Phi(x_i)^T. \tag{2.68}$$

Musíme pak nalézt vlastní čísla $\lambda \geq 0$ a vlastní vektory $\mathbf{V}$, které splňují podmínku:

$$\lambda \mathbf{V} = \bar{C}\mathbf{V}. \tag{2.69}$$

Při dosazení 2.68 do 2.69 zjišťujeme, že $\mathbf{V}$ leží v rozsahu $\Phi(x_i), \ldots, \Phi(x_m)$. Proto můžeme nahradit 2.69 za následující [14]:

$$\lambda(\Phi(x_i) \cdot \mathbf{V} = (\Phi(x_i) \cdot \bar{C}\mathbf{V})), \tag{2.70}$$

pro všechna $k = 1, \ldots, m$. Také víme, že existují koeficienty $\alpha_i, \ldots, \alpha_m$, takové že platí:

$$\mathbf{V} = \sum_{i=1}^{m} \alpha_i \Phi(x_i). \tag{2.71}$$

Když substituujeme 2.68 a 2.71 do 2.70, s tím, že použijeme definici jádra 2.60 dostaneme:

$$m\lambda K\boldsymbol{\alpha} = K^2\boldsymbol{\alpha}, \tag{2.72}$$

kde $\boldsymbol{\alpha}$ reprezentuje sloupcový vektor obsahující $\alpha_1, \ldots, \alpha_m$. Pro nalezení řešení 2.72 hledáme vlastní čísla následujícím vztahem [12, 14]:

$$m\lambda\boldsymbol{\alpha} = K\boldsymbol{\alpha}. \tag{2.73}$$

Výsledky 2.73 dále normalizujeme aby splňovali následující vztah:

$$1 = \lambda_i(\alpha^i \cdot \alpha^i), \tag{2.74}$$

pro všechna $i = 1, \ldots, m$. α^i zde pak reprezentuje již normalizovanou hodnotu vlatstního vektoru α_i.

PCA nad Jádrem

V průběhu testování algoritmu Kernel PCA, jsem vyzkoušel ještě tedy jiný postup pro výpočet nelineární transformace. Vycházím zde opět z jádrové transformace dat K. Nad Jádrovou matici K provedu standardní PCA redukci dimenze.

Formálně se tedy jedná o hledání vlastních vektorů a vlastních čísel nad kovarianční maticí spočítanou nad jádrem. Mějme tedy jádrovou matici K o velikosti m-krát-m. Její kovarianční matice je vyjádřena následovně:

$$\Sigma_K = \frac{1}{m}\sum_{i=1}^{m} K(i,:)K(i,:)^T, \tag{2.75}$$

kde $K(i,:)$ značí i-tý řádek matice K. Hledané vlastní čísla λ_i a vlastní vektory $\mathbf{v}$ musí splňovat následující podmínku:

$$\lambda\mathbf{v} = \Sigma_K\mathbf{v}. \tag{2.76}$$

Odezva ϕ pro nový prvek z se pak spočte následovně:

$$\phi(z) = K(z,:)\mathbf{v}, \tag{2.77}$$

kde $K(z,:) = (K(z,x_i), \ldots, K(z,x_m))$.

Kapitola 3

Implementace

V následující kapitole, budu popisovat a rozeberu implementaci jednotlivých algoritmů a modelů, popsaných v předchozí kapitole. Jako první stručně charakterizuji distribuovanou platformu SGE a skripty pro práci s Lineárním a Kernel SVM. Následně rozeberu implementaci a Expectation maximization algoritmu pro směsice Multinomiálních rozložení, poté se budu zabývat implementaci modelu PIVCO a jako poslední rozeberu implementaci Kernel PCA algoritmu.

3.1 Sun Grid Engine

SGE je distribuovanou platformou pro hromadné výpočty. Tento systém se skládá ze sítě uzlů – počítačů. Na FIT VUT v Brně to jsou hlavně Blade clustery a další servery a mimo výuku jsou do systému zapojeny i počítače v učebnách. Úlohy zpracovávané v SGE jsou dvojího druhu, krátké a dlouhé. Krátké jsou úlohy s maximálním časem zpracování tři hodiny, a dlouhé jsou všehny úlohy delší jak tři hodiny. Během testování byly všechny moje úlohy krátké, až na ty, které pracovaly s modelem PIVCO. Pro krátké i dlouhé úlohy existují oddělené fronty. Já jsem využíval fronty `all.g@@stable` a `long.q@@stable`. Přípona stable vymezuje všechny počítače, které jedou stabilně, tj. vyřazuje počítače počítače v učebnách z front.

V systému SGE je nutné aby každá úloha byla napsána jednovláknově, a všechny cesty ke zdrojům na disku, se kterými úloha pracuje byly zadány absolutní cestou. Matlab, pokud jej spustíme bez dalších parametrů, je vícevláknový, proto jsem musel každé spuštění Matlabu opatřit přepínačem `-singleCompThread`. Takto spuštěný matlabový skript je již validním skriptem pro systém SGE.

Lineární a Kernel SVM skripty

Skripty pro trénování a testování Linerárních SVM a Kernel SVM jsem převzal od Michala Hradiše. Tyto skripty byly vyvinuty pro soutěž TRECVID. Stejně tak datovou sadu a skripty s ní pracující, na které jsem testoval implementace jsem dostal od Michala Hradiše.

3.2 Směsice multinomiálních rozložení

Pro implementaci Expectation maximization algoritmu pro směsice Multinomiálních jsem zvolil platformu jazyka C++. Výchozím bodem zde byly vztahy popsané v sekci 2.3. Vy-

```
double
exp_max::multi_full(std::vector<int>  no_observations,std::vector  <double> mi)
{
  double tmp_fact = 0;
  int accumulator = 0;
  for (unsigned int i = 0; i < no_observations.size(); i++)
  {
    tmp_fact += fac(no_observations[i]);
    accumulator += no_observations[i];
  }
  tmp_fact = fac(accumulator)-tmp_fact;

  double tmp_mi = 0;
  for (unsigned int i = 0; i < mi.size(); i++)
  {
    if (mi[i] >= LOG_MIN )
    {
      tmp_mi += (no_observations[i]*mi[i]);
    }
    if (std::isnan(tmp_mi))
    {
      double asdf;
      asdf = 0;
    }
  }
  return tmp_fact+tmp_mi;
}
```

Obrázek 3.1: Implementace multnomiálního rozložení

cházel jsem ze vztahu pro výpočet $\gamma(z_{nk})$ – zodpovědnosti komponenty k za datový vzorek n 2.48, dále ještě ze vztahů 2.49 a 2.50.

Předtím než jsem k tomu mohl však přistoupit, tak jsem musel naimplementovat výpočet pro multinomiální rozložení 2.40. To se skládalo z binomické části a distribuční části.

Dalším krokem bylo přímočaré naprogramování těchto vztahů, a jejich otestování na malé trénovací sadě. Důvodem pro malou testovací sadu je fakt, že pravděpodobnosti jsou čísla menší jak 1 a násobením takovýchto čísel se zvyšuje počet desetiných míst mezivýsledků. Pro velké datové sady by došlo k podtečení. Proto jsem převedl celý výpočet do logaritmického počtu. V obrazku 3.1 je vidět ukázka implementace funkce multinomiálního rozložení převedeného do logaritmického počtu.

Implementace algoritmu

Proces převodu EM algoritmu do logaritmického počtu zahrnoval převod implementace všech vztahů na logaritmy. Jako první převedl faktoriál, a multinomiální rozložení:

$$\log fac(n) = \sum_{i=1}^{n} \log(i) \tag{3.1}$$

$$\log Mult(\mathbf{x}|\boldsymbol{\mu}) = \left[\log fac(N) - \left(\sum_{k=1}^{K} \log fac(m_k)\right)\right] + \sum_{k=1}^{K} (m_k \log \mu_k) \tag{3.2}$$

kde $\mathbf{x} = (m_1, m_2, \ldots, m_k)$, a K je šířka vektoru. Dále pak výpočty zodpovědnosti komponent γ, efektivní počet datových vzorků pro komponentu N_k a výpočet $\overline{\mathbf{x}}_k$.

$$\log \gamma(z_{nk}) = \frac{\pi_k + logMult(\mathbf{x_n}|\boldsymbol{\mu_k})}{\log \sum_{j=1}^{K} \pi_j + \log Mult(\mathbf{x_n}|\boldsymbol{\mu_j})} \tag{3.3}$$

$$\log N_k = \log \sum_{n=1}^{N} \log \gamma(z_{nk}) \tag{3.4}$$

$$\log \overline{\mathbf{x}}_{ki} = \log \sum_{n=1}^{N} \log \gamma(z_{nk}) + \log x_{ki} \tag{3.5}$$

kde $\log \sum$ značí sumu logaritmů.

Při ladění jsem často narážel na problematiku práce s $-\infty$, neboli s nulou v běžném počítání, jeho násobení mezi sebou nebo násobení nulou. V ne-logaritmickém počtu je výsledek vztahu 0×0 jasný. Pokud ale tento výpočet převedeme do logaritmu dostaneme $-\infty \times -\infty$. Zde by logicky měl být výsledek $+\infty$. To by ale znamenalo špatný výsledek, proto jsem jako výsledek stanovil opět $-\infty$ aby odpovídal výpočet ne-logaritmickému počtu. Navíc při implementaci v C++ jsem narazil na fakt, že $-\infty \times -\infty = -nan$, proto jsem všechny tyto zvláštní případy musel ošetřit tak aby k nim nedocházelo, protože v případě, že by došlo byť k jen jednomu takovémuto násobení, výsledek celého výpočtu by byl roven speciální hodnotě `Not a Number`.

3.3 Model PIVCO

Pro implementaci modelu PIVCO jsem zvolil nástroj Matlab v kombinaci s distribuovaným prostředím Sun Grid Engine (SGE). Jádro implementace modelu PIVCO jsem převzal od Ondřeje Glembeka. To jsem upravil pro práci s grafickými daty. Dále jsem vytvořil řadu obslužných skriptů pro trénování a testování modelu a generování výsledků experimentů.

Algoritmus trénování matice U

Jádrem modelu je kompenzační matice U, a proto je klíčový její výpočet. Pro její trénování byla adaptována technika Dana Povey-e [6] pro multinomiální rozloženi. Trénování matice U probíhá v desíti epochách. Před první epochou dojde k inicializaci matice náhodnými čísly z rozsahu $(0, 1)$. Průběh epochy je stručně popsán takto:

1. Se současným stavem matice U proveď čtyři iterace re-estimace vektoru x pro celou datovou sadu.

2. Se současným stavem vektorů x proveď čtyři iterace re-estimace matice U.

Re-estimace vektoru x se provádí pro každý datový vzorek a je popsána v následujících krocích:

- Spočti gradient a Hessiánovu matici zpracovávaného vzorku.
- Spočti objektivní funkci - log-likelihood tohoto vzorku.
- Proveď krok metody Newton-Rhapson se současným vzorkem.
- Pokud se zlepšila objektivní fuknce, příjmi výsledek.
- Pokud se výsledek nezlepšil, cyklicky zmenšuj krok Newton-Rhapson metody dokud se likelihood nezlepší.

Re-estimace kompenzační matice U pracuje s již upravenými váhovými vektory x. Průběh re-estimace je velmi podobný předešlému, s tím, že se pracuje se všemi trénovacími daty najednou:

- Spočti první a druhou derivaci objektivní fuknce.
- Proveď Newton-Rhapson aktualizaci matice.
- Pokud se nezlepšil celkový likelihood dat, cyklicky zmenšuj krok Newton-Rhapson metody dokud se výsledek nezlepší, jinak příjmi novou estimaci matice U.

Implementace je navíc obohacena o možnost rozdělit data na shluky – *chunky*, čímž se zajišťuje to, že v jednu chvíli je objem právě zpracovávaných dat menší. Nevýhodou takovéhoto přístupu je ale prodloužení již tak časově náročného procesu trénování.

Nástroj Matlab

Jednou ze zajímavých informací o tomto nástroji je fakt, že je efektivnější pracovat se sloupcovými vektory, což je v přímém kontrastu s jazykem C++. Dalším faktem, který jsem při práci s jazykem a prostředím Matlab pociťoval, je neschopnost prostředí příjmat jakékoliv volitelné parametry z příkazové řádky. Díky tomu je každý tvůrce skriptů pro Matlab nucen při každé změně parametrů přepsat svůj zdrojový kód. Proto jsem se po poradě s Ondřejem Glembekem, rozhodl jednotlivé soubory pro různé kombinace parametrů a datových sad generovat pomocí shellových skriptů. Takto vytvořené skripty jsem pak předával k výpočtu dalším skriptům, které je pak vykonaly na platformě SGE.

3.4 Kernel PCA

Proces tvorby nelineární Kernel PCA transformace se skládá ze tří částí. První, kdy vytvářím pomocí různých jádrových transformací jádra. Druhá, kdy trénuji, nebo jinak řečeno tvořím nelineárně transformovaná data pomocí Kernel PCA a PCA nad jádry a třetí, kdy se již nad natrénovanými transformovanými daty trénuje Lineární SVM klasifikátor. V této sekci detailněji rozeberu pouze druhou část procesu, protože to je část celého postupu, který mohu ovlivnit. O zbylých částech se pouze zmíním.

Impementaci jádrových transformací jsem převzal od Michala Hradiše, který mi poskytl celý balík aplikací pro práci s Jádry, a Kernel SVM. Všechna jádra představená v sekci 2.5, jsem vytvořil pomocí skriptů bežících na platformě SGE. Jejich vypočtení trvá řádově jednotky minut.

Během tvorby jsem vyzkoušel tři implementace algoritmu Kernel PCA. První, kterou jsem naimplementoval sám a vycházel jsem z [12]. Druhá, kdy jsem vycházel z kódu Kernel PCA pana Ambarish Jash-e [1]. A třetí, kterou jsem převzal od Michala hradiše. Detailněji zde rozeberu moji implementaci, a u ostatních zmíním rozdíly.

Pro svou implementaci Kernel PCA i PCA nad jádry jsem zvolil nástroj Matlab s tím, že jsem skripty vykonával v prostředí SGE. Jako vstupní data jsem používal data vytvořená jádrovými transformacemi. Ty jsou uloženy v husté notaci. Proto jsem si nejdříve vytvořil skript `to_text.sh`, který převede hustou notaci na standardní text, který pak načtu v Matlabovském skriptu. K převodu využívám program manipulateFV, který jsem převzal od Michala Hradiše.

Jádro vytvořené nad trénovacími daty je vlastně trojúhelníková matice. V matlabu ji načtu pomocí procedury `textscan`, která všechny čísla ze vstupního souboru načte do sloupcového vektoru. Z tohoto vektoru získám jako průměr koeficient γ, který později využiji pro normalizaci jádra. Dále provedu transformaci vstupní trojúhelníkové matice na čtvercovou. Tímto mám již připravené všehny vstupy pro Kernel PCA. Vzhledem k tomu, že Matlab je velmi vysokoúrovňový nástroj, je zde již naimplementovaná procedůra pro počítání vlastních vektorů, název procedůry je `eig`.

Funkce `eig` je vícenásobně přetížena, a já jsem si vybral variantu, kdy vrací právě dva parametry:

```
[VEC,VAL] = eig(data)
```

Proměná `VEC` zde reprezentuje vlastní vektory uložené po sloupcích a `VAL` vlastní čísla. Vlastní vektory jsou seřazeny ve sloupcích podle důležitosti, čili podle velikosti vlastních čísel. Výpočet transformovaných dat tedy spočíval v maticovém násobení vstupního trénovacího a testovacího jádra s maticí `VEC` zmenšenou na cílový počet dimenzí. V matlabu je výpočet následující:

```
out_train = train_dta * VEC(:,1:output_factors);
out_test  = test_dta  * VEC(:,1:output_factors);
```

Transformovaná data jsem předal trénovacímu procesu pro lineární SVM. Ten pomocí cross-validace spočetl výsledky na jak trénovací tak testovací sadě.

Pro PCA nad jádry se pouze výpočet 3.4 nahradí následovně:

```
VEC = princomp(data).
```

Kde `princomp` je funkce počítající Analýzu hlavních komponent nad daty. Tato funkce sama provádí odečet průměrného vektoru dat, a tím data vystřeďuje okolo počátku souřadnic. Výsledná matice `VEC` obsahuje opět sloupcové vektory vlastních čísel, seřazených podle důležitosti. Zbytek výpočtu je stejný jako v případě Kernel PCA.

I druhá verze algoritmu Kernel PCA je implementována v nástroji Matlab. Zásadním rozdílem, je zde fakt, že tento program si data převádí jádrovou transformací sám. Tím

[1] http://www.mathworks.com/matlabcentral/fileexchange/27319-kernel-pca

pádem je implementace více samostatná. Pro výpočet vlastních vektorů se zde používá funkce `eigs`, která vypočte zadaný počet vlastních vektorů. Dále se s těmito vektory provádí normalizace, a další řazení podle přepočtené důležitosti. Jedním z hlavních rozdílů, je také přístup k datům. V této implementaci se užívají pouze sloupcové vektory, proto je nutné před předáním, data transponovat.

Třetí implementaci algoritmu Kernel PCA je implementována v nástroji OpenCV. Tato implementace pracuje přímo s hustou reprezentací dat a tím pádem není nutné používat předzpracovávací skritpy. Nicméně musela proběhnout transformace trénovací trojúhelníkové matice na čtvercovou.

Dalším krokem je normalizace jádra v cílovém prostoru příznaků, ta probíhá následovně:

$$K_c = K - M_n * K - K * M_n + M_n * K * M_n, \tag{3.6}$$

kde K_c značí normalizované jádro, K jádro nenormalizované a M_n normalizační matici. Normalizační matice má stejný rozměr jako jádro a její hodnoty jsou nastaveny na $\frac{1}{n}$, kde n je šířka jádra K.

Pro výpočet vlastních vektorů a vlastních čísel, je užita OpenCV funkce `cv::eigen`. Zbytek výpočtu probíhá stejně jako v předchozích implementacích. Vynásobí se Jádrová matice s maticí vlastních vektorů.

Kapitola 4

Experimenty

Tématem této kapitoly je popsat provedené experimenty, stanovit experimentální rámce, představit datovou sadu a vyhodnotit výsledky. V samotném úvodu zmíním datovou sadu, nad kterou jsem prováděl experimenty. Experimentoval jsem se dvěma klasifikačními modely, Multinomiální 2.3, který jsem podrobil analýze počtu směsí, zkoumání vlivu regularizace trénovacího algoritmu a také testu vlivu šířky vektoru příznaků. U Phonotaktického modelu 2.4 jsem analyzoval jaký vliv má kompenzace kanálu s různým počtem faktorů stanovujících kompenzaci a experimentoval jsem s trénováním bázového multinomiálního rozložení v rámci trénování faktorové matice U. Dále jsem experimentoval s modely, které předzpracovávají data pro lineární a Kernel SVM. Jako první jsem využil alternativní interpretace modelu PIVCO a použil jsem ho jako kompresní element pro redukci dimenzionality dat. Tuto PIVCO – redukci dimenzionality jsem porovnal s redukcí dimenzionality algoritmem PCA, kterou jsem využil jako referenci. Také jsem provedl experimenty s Kernel PCA 2.5, kde jsem testoval úspěšnost jednotlivých typů jader v klasifikaci a také jsem testoval vliv přesnost aproximace jádra na úspěšnost klasifikace.

Jako metriku úspěšnosti jsem použil *mean average precision* [15]. Tento termín jsem se rozhodl nepřekládat, protože jeho překlad „Střední průměrná přesnost“ není používaný. Další případnou metrikou bude průměrná přesnost.

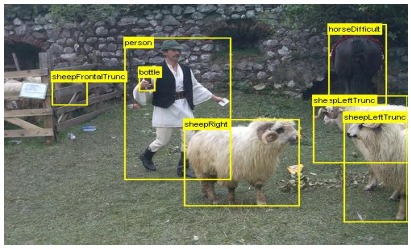
4.1 Datová sada Pascal VOC 2007

Pro experimenty jsem použil datovou sadu Pascal VOC 2007 [17]. Ta se skládá z trénovacího bloku dat, který obsahuje 5011 obrázků, a testovací sada obsahující 4952. Oba bloky dat jsou anotované. Anotace datové sady obsahuje dvacet sémantických tříd. Zde je seznam: Aeroplane, Bicycle, Bird, Boat, Bottle, Bus, Car, Cat, Chair, Cow, Dining table, Dog, Horse, Motorbike, Person, Potted plant, Sheep, Sofa, Train a TV/Monitor.

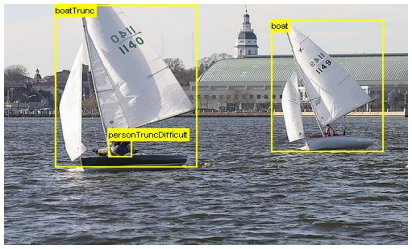
Anotace rozlišuje tři stavy přitomnosti respektive nepřítomnosti objektu třídy. První, objekt dané třídy je přítomen – označení 1. Druhý, objekt dané třídy není přítomen – označení -1, a poslední objekt dané třídy je přítomen, ale jeho vzhled není typický – označení 0. Dobrým příkladem třetího případu jsou objekty na obrázku 4.1.

Naproti tomu přítomnosti objektů na obrázku 4.2 jsou jednoznačné a proto spadá do první kategorie přítomnosti.

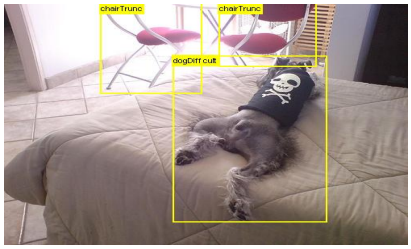
Při experimentování jsem Pascal VOC 2007 dataset interpretoval tak, že pro trénování jsem jako relevantní vzorky pro danou třídu vzal ty s označením 1 a 0 a pro testování úspěšnosti klasifikátoru jsem pak bral ty testovací vzorky označené pouze hodnotou 1.



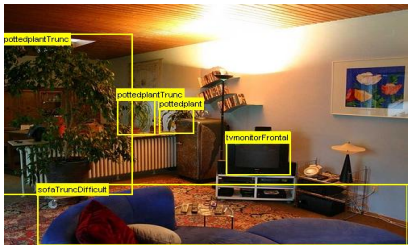
(a) Kůň



(b) Člověk



(c) Pes

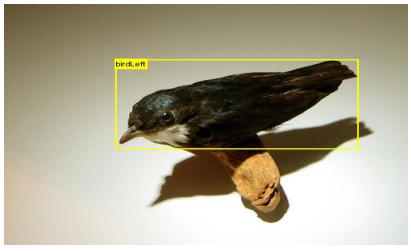


(d) Sedačka

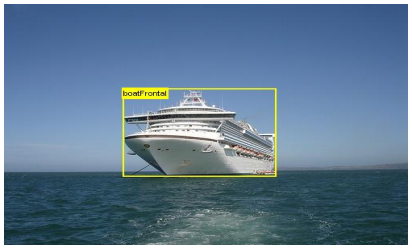
Obrázek 4.1: Příklad netypické přítomnosti objektů Kůň, Člověk, Pes a Sedačka

(a) Kůň

(b) Letadlo



(c) Pták



(d) Loď

Obrázek 4.2: Příklad typické přítomnosti objektů Kůň, Letadlo, Pták a Loď

Počet komponent	1	3	5	10
Mean avg. prec. (%)	17.0005	16.8071	15.9323	13.11

Tabulka 4.1: Tabulka závislosti mean average precision na počtu komponent

Šířka vekt. příznaků	64	128	256	512	1024	2048
Mean avg. prec. (%)	17.0005	18.6113	19.6542	21.0319	22.0585	22.9377

Tabulka 4.2: Tabulka závislosti mean average precision na počtu priznaků

4.2 Experimenty s multinomiálním modelem

Mým prvním experimentem s modelem bylo hledání nejlepšího počtu komponent pro směsici multinomiálních rozložení. Experimentální rámec jsem si zvolil následovně:

- Počet komponent směsi byl stanoven na $1, 3, 5, 10, 15$ a 20.
- Šířka vektoru příznaků byla 64, získaných dense samplingem, s rgb-SIFT deskriptorem a hard assignmentem příznaků. Bližší popis vlastností deskriptoru v [13].
- Expectation maximization algoritmus provede čtyři iterace.

Výsledky pro takto široký vektor příznaků nejsou uspokojivé, ale pro účely zjištění nejlepšího počtu komponent směsi jsou dostačující. V tabulce 4.1 je vidět efekt počtu komponent na výsledek.

Z tabulky je patrné, že úspěšnost klasifikace klesá s počtem komponent. Z tohoto důvodu jsem prováděl další experimentování na směsích s pouze jednou komponentou, čili na jednoduchých multinomiálních modelech.

V druhém experimentu jsem testoval závislost úspěšnosti klasifikace na šířce vstupního vektoru. Zvolil jsem následující podmínky:

- Počet komponent ve směsi 1.
- Šířka vstupního vektoru příznaků byla $64, 128, 256, 512, 1024$ a 2048.
- dense sampling, rgb-SIFT deskriptor a Hard assignment příznaky.

V tabulce 4.2 jsou shrnuté výsledky testu. Z tabulky je zřejmé, že s rostoucím počtem příznaků roste i mean average precision dosahovaná všech dvaceti třídách. Z tohoto jsem usoudil, že pro experimenty jejichž výsledky jsou uvedené v kapitole 4.3, jsem použil vektory o délce 2048 příznaků.

Pro úplnost jsou v tabulce uvedeny výsledky pro jednotlivé třídy.

Trénování klasifikátoru pro všech dvacet tříd trvá řádově deset až dvacet minut. Zpracování testů pak trvá řádově jednotky minut. Testování probíhalo na stroji s procesorem intel Core 1 duo s frekvencí $2GHz$ pro každé jádro a $2GB$ paměti.

Díky experimentům jsem tedy zjistil, že nahrazení lineárního SVM klasifikátoru Multinomiálním modelem, není perspektivní, protože nejlepší výsledek cirka 23 % je stále výrazně horší než pro lineární SVM, které na stejně zpracovaných datech dosahuje mean average precision okolo 37 %. Proto jsem se rozhodl dále nezkoumat tento model a přešel jsem na jiné perspektivnější modely.

Třída	**Průměrná přesnost (%)**	
	Multi. Model	Kernel SVM
aeroplane	53.6607	69.7371
bicycle	22.0639	49.9041
bird	18.0991	39.0913
boat	37.0816	60.5609
bottle	10.047	18.6106
bus	14.1501	43.5071
car	37.5181	68.1272
cat	18.3332	42.5791
chair	26.0981	45.76
cow	10.317	24.9255
diningtable	14.2371	33.6351
dog	22.2394	36.1221
horse	16.3019	70.2428
motorbike	17.5057	48.5224
person	59.8032	81.5375
pottedplant	10.1079	18.0399
sheep	13.5351	33.2556
sofa	13.3322	34.008
train	26.933	67.4208
tvmonitor	17.3903	38.2085
průměr	22.9377	46.1898

Tabulka 4.3: Přesnosti klasifikace pro jednotlivé třídy dosažené multinomiálním modelem a Kernel SVM

Původní průměrná přesnost v procentech		59.8032
Kompenzace maticí o šířce	10	48.3749
	20	55.5195
	50	53.1058
	100	54.8515
	300	48.9605

Tabulka 4.4: Tabulka závislosti průměrné přesnosti třídy osoba na počtu faktorů matice U

4.3 PIVCO – kompenzace kanálu

Pro experimenty jsem používal Pascal VOC data s tím, že bylo vyextrahováno 2048 příznaků, zíkaných metodou dense sampling s rgb-SIFT deskriptorem a hard assignmentem. Pracoval jsem pouze s třídou osoba, protože trénování matice faktorů při šířce vektoru příznaků 2048 je velmi časově a paměťově náročná úloha. Pro příklad trénování matice s třemi sty faktory zabere řádově 6 hodin a 15 GB paměti. Druhý důvod je závislost modelu PIVCO na výchozím multinomiálním modelu, proto je vhodné použít třídu s největší průměrnou přesností 59.8 %.

Prvním experimentem bylo zkoumání závislosti přesnosti klasifikátoru na počtu faktorů kompenzační matice U. Z trénovací sady jsem musel vyfiltrovat všechny datové vzorky, které nepatří do třídy osoba, tj. jsem v upravené datové sadě ponechal pouze prvky s označením 1 a 0 pro třídu osoba. Výsledkem byla trénovací sada s 2008 vzorky dat. Pro model pozadí jsem datovou sadu neupravoval, měla tedy 5011 vzorků.

Experimentální rámec byl tedy následující:

- Dataset s 2048 hard assignment příznaky.
- Dense sampling a rgb-SIFT reprezentace.
- Kompenzační matice U o šířce $10, 20, 50, 100$ a 300.

V tabulce 4.4 jsou vidět výsledky testu. Je patrné, že kompenzace kanálu výsledky modelu zhoršuje. Tento fakt neodpovídá původním předpokladům, ale je způsoben pravděpodobně tím, že PIVCO systém není vhodný pro adaptaci náhodných vlivů v obrazové reprezentaci. Nejlepšího výsledku dosáhla matice s 20 faktory, ale i ta měla přesnost o 4 % horší než původní model bez kompenzace.

Experimenty prokázaly, že model PIVCO není vhodný pro redukci náhodných vlivů v obrazové reprezentaci. Díky tomuto zjištění, jsem upustil od dalších experimentů s redukcí náhodných vlivů kompenzační maticí a začal jsem se věnovat využití modelu PIVCO jako prvek pro redukci dimenzionality.

Experimenty jsem prováděl na platformě SGE. Spouštěcí požadavky pro menší dimenzi kompenzační matice jsou $2\,GB$ paměti, a trénování trvalo maximálně dvě hodiny. Naproti tomu pro matici U se 100 a 300 faktory bylo trénování otázkou čtyř až osmi hodin pro model pozadí. Paměťové nároky jsou cirka $10\,GB$. Paměťová náročnost je způsobena zaprvé velkým rozsahem dat a zadruhé faktem, že Matlab předává všechny parametry pro funkce a procedůry hodnotou, a tak jsou v jednom okamžiku jsou v paměti data nahrána více než jednou.

	Mean average precision	
Původní dataset	0.461898	
Cílová dimenze	**PIVCO**	**PCA**
10	0.272256	0.216747
20	0.335075	0.268401
50	0.382145	0.308091
100	0.412	0.344118
300	0.406311	0.369327

Tabulka 4.5: Mean average precisions pro původní data, PCA a PIVCO

4.4 PIVCO – Redukce dimenzionality

Experimenty s redukcí dimenzionality pomocí PCA a modelu PIVCO, jsem prováděl na stejných datech jako předchozí experimenty 4.3 a 4.2. Užil jsem tedy jako výchozí dimenzi 2048 hard assignment příznaků. Vzhledem k tomu, že se jedná jen o alternativní využití modelu PIVCO z předchozích experimentů, trénování matice trvá řádově stejnou dobu. Data pro trénování jsem použil všechna, protože pro redukci dimenzionality je informace o příslušnosti ke třídě v mém případě zbytečná.

Jako referenci jsem použil redukci dimenzionality pomocí PCA se stejnými počty cílových dimenzí. Dále jsem výsledky redukce dimenzinality porovnal s neredukovanými daty. Experimentální rámec jsem stanovil takto:

- Dimenze zdrojových dat byla 2048 příznaků.
- Cílové dimenze pro kompresi dat byly $10, 20, 30, 50, 100$ a 300.
- Neredukovaná data zpracována Kernel SVM s χ^2 jádrem.
- Redukovaná data byla zpracována Kernel SVM se standardním RBF jádrem.

V tabulce 4.5 je v horní části vidět *mean average precision* Kernel SVM nad všemi Pascal VOC třídami, a pod ní už jsou výsledky klasifikátorů na redukovaných datech.

Nejlepšího výsledku dosáhl klasifikátor s daty cílovou dimenzí 100 příznaků. Nicméně i přesto nedosáhl stejného výsledku jako neredukovaná data. Dále je z tabulky patrné, že mezi výsledky klasifikace nad PCA-redukovanými daty a cílovou dimenzí je pozitivní korelace. To nasvědčuje tomu, že ve vyšších cílových dimenzích by PCA bylo pravděpodobně úspěšnější než PIVCO redukce, nicméně vzledem k časové náročnosti pro natrénování tak velké kompenzační matice U jsem od dalšího zvyšování cílové dimenze upustil. Zkoušel jsem ještě provést experiment s cílovou dimenzí 500 příznaků, ale po patnácti hodinách trénování, jsem tento experiment ukončil.

Jak jsem již řekl, nejlepšího výsledku se dosáhlo při kompresním poměru jedna-ku-dvaceti. Tento výsledek znamená, že při užití této cílové dimenze se dvacetkrát zrychlí výpočet jádra pro Kernel SVM Cenou za toto zrychlejí je však 5 % pokles mean average precision. Nicméně zrychlení výpočtu umožňuje použít větší datovou sadu, pro nelineární Kernel SVM klasifikaci, což je jedním z cílů mé práce.

K dalšímu závěru, ke kterému jsem došel, je fakt, že PIVCO – redukce dimenzionality je ve všech cílových dimenzích lepší, než redukce dimenzionality algoritmem PCA. Zvláště

pak výrazně lepších výsledků dosáhl model PIVCO ve velmi malých cílových dimenzích, kde bylo PIVCO o 4 až 7 % lepší než PCA.

Potenciální užití PIVCO – redukce dimenzionality je *on demand* počítání jádra, pro Kernel SVM, nebo právě použití větších datasetů pro trénování nelineárních klasifikátorů jako je Kernel SVM.

4.5 Kernel PCA

Testování implementací algoritmu Kernel PCA jsem prováděl na datové sade Pascal VOC 2007, ale extrahoval jsem z ní jiný druh příznaků. Jedná se o Soft assignment příznaky.

Prvním experimentem bylo hledání funkční implementace algoritmu. První dvě implementace se ukázaly být nekompletní. S největší pravděpodobností vynechávali klíčovou normalizaci v cílovém prostoru příznaků popsanou rovnicí 3.6. Dalším problémem druhé implementace 3.4 bylo nenormalizování vstupních dat, to způsobilo špatné spočtení jádrové matice a tím pádem byl celý výpočet znehodnocen. Třetí implementace 3.4 byla úspěšná, a proto nadále budu používat pouze ji. Neúspěšnost prvních dvou implementací bylo také budícím elementem k experimentu s PCA nad jádry, které se ukázalo relativně úspěšné.

Druhým experimentem je tedy srovnání jednotlivých jader. Data konvertovaná jádrovými transformacemi jsem předal algoritmu Kernel PCA, který je převedl do cílového prostoru příznaků. Data v novém prostoru příznaků jsem pak předal lineárnímu SVM klasifikátoru. Takto získané výsledky klasifikace jsem srovnával s lineární SVM klafikací nad neupraveným datasetem a také s výsledky Kernel SVM klasifikace. Experimentální rámec je tedy následující:

- Dimenze zdrojových dat je 4096 se soft assignment příznaky.
- Koeficient α ovlivňující šířku RBF funkce jádra je nastavován na $\pm 0.1, \pm 1.0$ a ± 2.0. Znaménko koeficientu záleží na typu jádra.
- Testovaná jádra jsou $K^{\exp}_{chi2}$ 2.66, $K^{\exp}_{add\_chi2}$ 2.62, $K^{\exp}_{bha}$ 2.64 a $K^{\exp}_{inter}$ 2.63.
- Kernel PCA využívá vlastní vektory v plné velikosti.

Koeficient α použitý v experimentálním rámci zde vyjadřuje násobitel normalizačního koeficientu γ v jádrové transformaci $K_{\exp} = \exp(-\alpha\gamma K)$. Nastavováním násobitele α simuluji odpovídající optimalizaci hyperparametrů při trénování Kernel PCA. Znaménko koeficientu je záporné pro jádro $K^{\exp}_{chi2}$ jinak je kladné.

V experimentu jsem srovnával výsledky lineární SVM klasifikace dat transformovaných Kernel PCA, a vzhledem k tomu, že hlavní cíl Kernel PCA transformace je emulace nelineárního Kernel SVM klasifikátoru, srovnávám úspěšnost klasifikace i výsledky Kernel SVM.

V první tabulce 4.6 je výchozí stav. V horní části je uveden výsledek pro lineární SVM klasifikátor a jeho *mean average precision*, ve spodní části tabulky jsou pak vidět výsledky Kernel SVM klasifikace pro jednotlivá jádra.

Následující tabulka 4.7 ukazuje výsledky lineární SVM klasifikace pro algoritmus Kernel PCA. Nejlepšího výsledku dosáhlo jádro $K^{\exp}_{chi2}$ a to 44.11 % *mean average precision*. Ač došlo ke zhoršení o 3 % oproti Kernel SVM, došlo k radikálnímu zlepšení oproti lineárnímu SVM a to o 7 %. Pokud vezmeme nejlepší výsledky pro každé jádro, tak pokaždé došlo ke zlepšení

Lineární SVM			36.92 %
$K^{\exp}_{chi2}$	$K^{\exp}_{add\_chi2}$	$K^{\exp}_{bha}$	$K^{\exp}_{inter}$
47.212 %	47.0943 %	45.2403 %	44.7786 %

Tabulka 4.6: Mean average precision pro lineární a Kernel SVM

α	$K^{\exp}_{chi2}$	$K^{\exp}_{add\_chi2}$	$K^{\exp}_{inter}$	$K^{\exp}_{bha}$
± 0.1	43.185 %	43.02 %	41.665 %	40.845 %
± 1.0	44.11 %	39.89 %	39.495 %	41.05 %
± 2.0	43.96 %	31.91 %	31.385 %	36.63 %

Tabulka 4.7: Mean average precision lineární SVM klasifikace pro algoritmus Kernel PCA

vůči lineární klasifikaci. Jako nejvíce nadějná jádra se tedy jeví $K^{\exp}_{chi2}$ a $K^{\exp}_{add\_chi2}$, které by při důkladnějším prohledání prostoru násobitelů, mohli dosáhnout ještě na lepší výsledky.

Dalším faktem patrným z tabulky je to, že nejhorším násobitelem je $\alpha = \pm 2.0$. Tento násobitel přinesl téměř vždy zhoršení i vůči lineární klasifikaci. Velmi nadějný je naproti tomu prostor násobitelů v rozmezí 0 až 1, kde klasifikátory dosahovaly vysoké přesnosti.

Dosáhl jsem tedy emulace nelineárního SVM pomocí Kernel PCA s tím, že se výsledky zhoršily jen o 3 %. Tato transformace dat má tedy velký potenciál zlepšit výsledky na soutěžích TRECVID a ImageNet Large Scale Visual Recognition Challenge, ve kterých není možné použít Kernel SVM kvůli velikosti datových sad. Vzhledem k tomu, že právě toto je cílem mé práce, mohu Kernel PCA transformaci považovat za úspěšnou. Využití Kernel PCA transformace je tedy zřejmé, a to emulace nelineárního SVM pro velké datové sady, a zmenšování cílového prostoru příznaků.

PCA nad jádry

Experimentoval jsem se stejnými daty jako v případě Kernel PCA. Experimentální rámec byl stanoven takto:

- Dimenze zdrojových dat je 4096 soft assignment příznaků.
- Koeficient α ovlivňující šířku jádrové matice nastavován na ± 1.0. Znaménko koeficientu záleží na typu jádra.
- Testovaná jádra jsou $K^{\exp}_{chi2}, K^{\exp}_{add\_chi2}, K^{\exp}_{bha}$ a $K^{\exp}_{inter}$.
- PCA nad Jádry využívá nezkrácené hlavní komponenty a komponenty zkrácené na 2048 příznaků.

V tabulce 4.8 jsou shrnuty výsledky. Tato technika dosahuje velmi dobrých výsledků. Nejlépe dopadla při plném počtu hlavních komponent a to 45.33 % *mean average precision* pro jádro $K^{\exp}_{inter}$. Úspěšnost byla dokonce vyšší než nejlepší výsledek pro Kernel PCA. Pokud ale srovnáme výsledky Kernel PCA a PCA nad jádry, je úspěšnost přibližně srovnatelná.

Tato technika je tedy srovnatelně využitelná jako Kernel PCA, ale výpočetní nároky jsou zde o něco vyšší, protože je zde nutné před počítáním vlastních vektorů kovariační matice, neboli hlavních koponent, tuto kovarianční matici nejdříve spočíst.

Délka hlavních komponent	$K^{\exp}_{chi2}$	$K^{\exp}_{add\_chi2}$	$K^{\exp}_{inter}$	$K^{\exp}_{bha}$
2048	44.635 %	42.445 %	42.445 %	39.49 %
Plná délka	44.525 %	42.555 %	45.33 %	39.39 %

Tabulka 4.8: Mean average precision pro PCA nad Jádry

Kapitola 5

Závěr

Cílem mé práce bylo hledání a testování modelů a algoritmických postupů, které mají potenciál zlepšit výsledky FIT VUT v Brně na soutěžích TRECVID a IMAGENET Large Scale Visual Recognition Challenge. Hlavní výzvou pro řešitele je u obou soutěží rozsah datové sady a počet sémantických tříd, který je velký. Abych mohl tuto problematiku řešit, musel jsem nastudovat strukturu obou soutěží. Zabýval jsem se hlavně metodami klasifikace, které jsou schopné zpracovat velké množství trénovacích dat. Jako potenciálně přínosné metody jsem vybral nahrazení SVM klasifikátoru, Multinomiálním modelem, nebo modelem PIVCO, nebo předzpracování dat pro SVM nelineární transformací, tak aby se emulovala rozhodovací schopnost nelineárního Kernel SVM. Testoval jsem redukci dimenzionality modelem PIVCO a Kernel PCA transformaci. Jako experimentální techniku jsem vyzkoušel PCA nad jádry. Právě techniky využívající jádrových transformací se ukázaly být nejúspěšnější.

Vytvořil jsem nástroj pro trénování směsic multinomiálních rozložení algoritmem expectation maximization, a ten jsem podrobil experimentům nad datovou sadou Pascal VOC 2007. Zjistil jsem, že nejlépe dopadla klasifikace multinomiálním modelem, pokud je ve směsi právě jedna komponenta, tj. je použit standardní Multinomiální model. Dále jsem hledal optimální šířku vektoru příznaků a zjistil jsem, že nejlépe klasifikace dopadne při použití příznakových vektorů o délce 2048 příznaků na datový vzorek. *Mean average precision* je 22.9 %. Výsledky multinomiálních modelů celkově nejsou uspokojivé a nebudou dále používány pro soutěžě TRECVID a ImageNet Challenge.

Použil jsem nástroj PIVCO, pro odstraňování náhodných vlivů v reprezentaci obrazu. Testoval jsem úspěšnost kanálové adaptace v závislosti na velikosti faktorové matice U. Zjistil jsem, že kanálová kompenzace výsledky zhoršuje. Pro model třídy `person` se zhoršila klasifikace o 4 % z 59.8032 % na 55.5195 %. To je pravděpodobně způsobeno tím, že systém PIVCO, není schopen náhodné vlivy eliminovat. Proto jsem se rozhodl dále nezkoumat využití modelu PIVCO pro kompenzaci náhodných vlivů v reprezentaci obrazu, ale použil jsem ho pro redukci dimenzionality.

Po neúspěchu s náhradou SVM klasifikátoru jiným modelem, se ukázalo právě předzpracování datové sady, jako jedinnou cestou jak zlepšit výsledky ve výše zmíněných soutěžích. Z tohoto důvodu jsem se začal věnovat redukci dimenzionality modelem PIVCO. Provedl jsem experiment s redukcí datové sady z prostoru 2048 příznaků do prostoru s desíti až třemi sty příznaky. Pro srovnání jsem provedl ještě redukci dimenzionality pomocí PCA. Nejlépe dopadl model PIVCO pro cílovou dimenzi 100 příznaků. Kernel SVM klasifikator s takto předzpracovanými daty dosáhl `mean average precision` 41.2 %. Došlo tedy ke zhoršení oproti neredukované datové sadě, a to o 5 %, ale za to jsem dosáhl dvacetinásobného

zrychlení výpočtu jádrové matice pro Kernel SVM. Dále jsem zjistil, že PIVCO – Redukce dimenzionality byl ve všech cílových dimenzích lepší než PCA. Tento model je použitelný v praxi. Využití této redukce je například pro zrychlení trénování Kernel SVM v případě, že je jádro počítané *on demand*.

Posledním metoda, kterou jsem testoval bylo Kernel PCA. Experimentoval jsem s různými jádry, a zjistil jsem, že nejlepších výsledků dosahuje χ^2 jádro $K^{\exp}_{chi2}$. Lineární SVM klasifikátor dosáhl na datech předzpracovaných Kernel PCA s tímto jádrem *mean average precision* 44.11 %. Tímto výsledkem se téměř vyrovnal nelineárnímu Kernel SVM klasifikátoru, který dosáhl *mean average precision* 47.212 %. Využití Kernel PCA je emulace nelineárního Kernel SVM. Kombinace Kernel PCA a lineárního SVM umožňuje tedy možné použít emulovaný nelineární klasifikátor nad velkými datasety soutěží ImageNet Challenge a TRECVID.

Posledním model, který jsem podrobil experimentům bylo PCA nad jádry. Tuto experimentální techniku jsem vytvořil, jako reakci na neúspěchy nekterých implementací Kernel PCA. Předzpracování dat pomocí PCA nad Jádry je velmi podobné jako předzpracování modelem Kernel PCA. Nejlepšího výsledku dosáhlo jádro *histogram intersection* $K^{\exp}_{inter}$. Lineární SVM s daty z tohoto jádra dosáhlo *mean average precision* 45.33 %. Ostatní výsledky jsou srovnatelné s Kernel PCA. Výpočetní náročnost této techniky je vyšší než u Kernel PCA, což částečně snižuje její použitelnost, nicméně i přes tento handicap, je možné tuto techniku použít pro emulaci nelineárního klasifikátoru pro větší datové sady.

Budoucím vývojem této práce by mohla být například optimalizace metody Kernel PCA, tak aby dosahovala srovnatelných výsledků jako Kernel SVM. Dále bych mohl hledat modely závislostí tříd mezi sebou. Jak v TRECVID tak v ImageNet Challenge jsou mezi skupinami tříd položky, které nějakým způsobem podmiňují přítomnost, nebo nepřítomnost jiné třídy. Současný přístup tyto závislosti mezi třídami ignoruje a považuje všechny třídy za nezávislé. Hledání modelů by tedy spočívalo v naleznutí podmíněných pravděpodobností $p(trida_1|trida_2)$.

Literatura

[1] Bishop, C. M.: *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006, ISBN 0387310738.

[2] Cao, D.; Boley, D.: On Approximate Solutions to Support Vector Machines. In *SDM*, 2006.

[3] Chang, C.-C.; Lin, C.-J.: LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, ročník 2, 2011: s. 27:1–27:27, software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[4] van Gemert, J. C.; Veenman, C. J.; Smeulders, A. W.; aj.: Visual Word Ambiguity. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, ročník 32, 2010: s. 1271–1283, ISSN 0162-8828, doi:http://doi.ieeecomputersociety.org/10.1109/TPAMI.2009.132.

[5] Glembek, O.; Matějka, P.; Burget, L.; aj.: Advances in Phonotactic Language Recognition. In *Proc. Interspeech 2008*, 9, International Speech Communication Association, 2008, ISSN 1990-9772, str. 4.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=8727

[6] Glembek, O.; Plchot, O.: PIVCO Phonotactic Intersession Variation Compensation. Faculty of Information Technology, Brno University of Technology - Lecture notes, 2009.

[7] Hradiš, M.; Beran, V.; Řezníček, I.; aj.: Brno University of Technology at TRECVid 2010. In *TRECVID 2010: Participant Notebook Papers and Slides*, National Institute of Standards and Technology, 2010, str. 11.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=9444

[8] Joachims, T.: Training linear SVMs in linear time. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, New York, NY, USA: ACM, 2006, ISBN 1-59593-339-5, s. 217–226, doi:10.1145/1150402.1150429.
URL http://dx.doi.org/10.1145/1150402.1150429

[9] Kenny, P.; Boulianne, G.; Ouellet, P.; aj.: Joint Factor Analysis Versus Eigenchannels in Speaker Recognition. *IEEE Transactions on Audio, Speech & Language Processing*, ročník 15, 2007: s. 1435–1447, doi:10.1109/TASL.2006.881693.

[10] Ng, A.: Principal Component Analysis.
http://www.stanford.edu/class/cs229/notes/cs229-notes10.pdf, 2009, lecture notes.

[11] Ng, A.: Support Vector Machines.
http://www.stanford.edu/class/cs229/notes/cs229-notes3.pdf, 2009, lecture notes.

[12] Perronnin, F.; Senchez, J.; Xerox, Y. L.: Large-scale image categorization with explicit data embedding. In *CVPR*, Červen 2010, s. 2297–2304, doi:10.1109/CVPR.2010.5539914.
URL http://dx.doi.org/10.1109/CVPR.2010.5539914

[13] van de Sande, K. E. A.; Gevers, T.; Snoek, C. G. M.: Evaluating Color Descriptors for Object and Scene Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence volume 32*, 2010: s. 1582–1596.

[14] Scholkopf, B.; Smola, A.; Müller, K.-R.: Kernel principal component analysis. In *ADVANCES IN KERNEL METHODS - SUPPORT VECTOR LEARNING*, MIT Press, 1999, s. 327–352.

[15] WWW stránky: Information Retrieval.
http://en.wikipedia.org/wiki/Information_retrieval.

[16] WWW stránky: Large Scale Visual Recognition Challenge 2010.
http://www.image-net.org/challenges/LSVRC/2010/.

[17] WWW stránky: Pascal Visual Object Classes.
http://pascallin.ecs.soton.ac.uk/challenges/VOC/.

[18] WWW stránky: TREC Video Retrieval Evaluation.
http://trecvid.nist.gov/.

Příloha A

Obsah CD

- Text
- Program
  - Multinomiální model
  - Model Pivco
  - KPCA
- Video