SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE

Fakulta chemickej a potravinárskej technológie

Evidenčné číslo: FCHPT-5415-61570

Použitie Support Vector Machines na klasifikáciu dát

Bakalárska práca

2016 **Endre Hamerlik**

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE

Fakulta chemickej a potravinárskej technológie

Evidenčné číslo: FCHPT-5415-61570

Použitie Support Vector Machines na klasifikáciu dát

Bakalárska práca

Študijný program: automatizácia, informatizácia a manažment v chémii a potravinárstve
Študijný odbor: 5.2.14. automatizácia, 5.2.52. priemyselné inžinierstvo
Školiace pracovisko: Ústav informatizácie, automatizácie a matematiky
Vedúci záverečnej práce: doc. Ing. Michal Kvasnica, PhD.

Bratislava 2016 **Endre Hamerlik**

Slovenská technická univerzita v Bratislave
Ústav informatizácie, automatizácie a matematiky

Fakulta chemickej a potravinárskej technológie
Akademický rok: 2015/2016
Evidenčné číslo: FCHPT-5415-61570

ZADANIE BAKALÁRSKEJ PRÁCE

Študent:	**Endre Hamerlik**
ID študenta:	61570
Študijný program:	automatizácia, informatizácia a manažment v chémii a potravinárstve
Kombinácia študijných odborov:	5.2.14. automatizácia, 5.2.52. priemyselné inžinierstvo
Vedúci práce:	doc. Ing. Michal Kvasnica, PhD.

Názov práce: **Použitie Support Vector Machines na klasifikáciu dát**

Špecifikácia zadania:

Cieľom práce je navrhnúť a implementovať systém na klasifikáciu dát založený na Support Vector Machines (SVM). Takýto systém bude použitý na klasifikáciu rukou písaných číslic. Vstupnými dátami budú obrázky rukou písaných číslic, ktoré budú následne rozoznané pomocou vhodne zvoleného klasifikačného algoritmu. Čiastkovými cieľmi práce sú: 1. analýza možností extrakcie vlastností z obrázkov, 2. syntéza separačných funkcií pomocou metódy SVM, 3. konštrukcia klasifikačného algoritmu, 4. verifikácia a vyhodnotenia úspešnosti klasifikácie. Pri implementácii sa použije programové prostredie Matlab a Python v spolupráci s knižnicou OpenCV.

Rozsah práce: 30

Riešenie zadania práce od: 15. 02. 2016
Dátum odovzdania práce: 08. 05. 2016

L. S.

Endre Hamerlik
študent

prof. Ing. Miroslav Fikar, DrSc.
vedúci pracoviska

prof. Ing. Miroslav Fikar, DrSc.
garant študijného programu

Abstract

The aim of the present thesis is to recognize handwritten digits from raster graphics using Support Vector Machines. The first stage of the project is a step by step guide of the problem analysis with Matlab source codes. These sources (prepared using Machine learning and Image processing toolbox) were intended to help better describe and understand matter discussed in corresponding section; and also to give a benchmark for further softvare implementation. In the remaining part is implemented the algorithm in Python 3.5 developed in the first part. During the development in Python were used Scikit-learn, Numpy, Matplotlib and OpenCV for Python libraries.

Keywords:

Support Vector Machines, Histogram of oriented gradients, kernel function, handwritten digit recognition, Matlab, Python, OCR

Abstrakt

Cieľom tejto bakalárskej práce je rozpoznávanie ručne písaných číslic na digitálnych obrázkoch pomocou Support Vector Machine. V prvej časti sa rozoberala postupne problematika rozpoznávania obrázok, a vôbec grafických útvarov na rastrových obrázkach. V každej jednej podkapitole sa nachádzajú zdrojové kódy k danej problematike. Tieto zdrojové kódy pomáhajú pri exaktnej formulácii daného problému, ale zároveň sú názornou ukážkou riešenia daných problémov. Súbor týchto zdrojových kódov implementovaných v prostredí Matlabu (za pomoci toolbox-ov Machine Learning a Image processing) počas ďalšej práci slúžila aj ako prototyp algoritmu. Druhá časť je už o implementácii v programovacom jazyku Python, teda o vytvorení softvérového základu pre rozpoznávanie číslic, ktorý je širokospektrálne použiteľný ako súčasť iných, aj webových apilácii. Pri implementácii v Python 3.5 som používal knižnice z Anacondy, a to najmä Scikit-learn, Numpy a openCV pre Python.

Kľúčové slová:

Support Vector Machine, Histogram orientovaných gradientov, kernelove funkcie, rozpoznávanie číslic, Matlab, Python, OCR

Obsah

Kapitola 1

Úvod

Aj keď problém optického rozpoznávania znakov sa rieši už necelé polstoročie, neexistuje metóda ktorá by mohla byť všeobecne uznanou ako štandardný postup pri riešení tohto problému. Je to jednak kvôli komplexnosti tejto záležitosti, a aj vďaka neustále sa meniacim požiadavkám. V súčasnosti využívame funkcionalitu OCR (Optical Character Recognition) od rozpoznávania ŠPZ, čísel domov, (re)digitalizácii dokumentov, až po rozpoznávanie pohybov zaznamenaných na dotykovej obrazovke.

Obr. 1.1: Príklad využitia rozpoznávania číslic

Síce na prvý pohľad sa náš problém zdá byť záležitosťou spracovania obrázkov, okrem prispôsobení pomerov v obrázku potrebujeme k rozpoznaniu aj logiku, ktorá z vlastností danej vzorky (obrázka alebo jeho časti) zistí jeho totožnosť. V ponímaní učenia sa s učiteľom (Supervised learning) to bude otázka: Ku ktorej množine (v tomto prípade obrázkov) má najbližšie naša vzorka? Na to aby sme mohli odpovedať potrebujeme poznať už niekoľko tried (v prípade rozpoznávania číslic sú to číslice od 0 po 9) ktoré môžme charakterizovať. Teda až po charakteristike nejakej znalostnej bázy môžme nové vzorky porovnánvať s jednotlivými triedami, čo je už klasifikácia; najvšeobecnejší súbor výsledkovo orientovaných logických úkonov.

Klasifikáciou rozoznávame dobré investície od zlých, bezpečné od nebezpečnej; a práve tá jeho všeobecná exaktná podpora rozhodnutí ma zaviedla ku výbere metódy Support Vector Machines. Support Vector Machine je dnes jedným z najpoužívanejších klasifikačných metód, ale je aj priekopníkom umelej inteligencie. Jeho autori (Boser, Guyon and Vapnik 1992) ho prepracovali do dnešnej podoby po 20 ročnom tichu o publikácii ('statistical learning theory' Vapnik a Chervonenkis, 1974.) v ktorom sa prvýkrát sformuloval nápad štatistického učiaceho sa algoritmu. Dnes je súčasťou 'State of art' Machine Learning systémov gigantov Google, Facebook, IBM, ako je aj základom väčšiny pattern recognition softvérov. Samotný autor, Vapnik od roku 2014 tiež pracoval na AI projekte Facebook-u.

V rámci tejto práce som sa oboznámil s matematickým aparátom, ktorý tvorí teoretický základ pre zvolenú metódu klasifikácie, a skúmal som štruktúry vlastností digitálnych obrázkov a spôsoby ich separácie. Po obsiahlej teoretickej príprave som si vybral konkrétny dataset, súbor 60000 ručne písaných číslic (od 0 po 9) o ktorých (na základe môjho modelu) má zistit môj softvér o akú číslicu ide.

Kapitola 2

Hromadenie vstupných dát

Zdrojom klasifikačných dát, ručne písaných číslic, bude jedna zo základných datasetov pre klasifikáciu s názvom 'MNIST Database'-(Mixed National Institute of Standards and Technology (Maryland, U.S.) database), teda databáza Národného inštitútu pre štandardy a technológie, ktorá obsahuje 60.000 ručne písaných vzoriek, predupravených na štandardnú veľkosť (28 x 28 pixel) a štandardné pomery veľkosti číslic a obrázku. Po stiahnutí databázových súborov, ich potrebujeme dekódovať a následne spracovať.

Z binárneho databázového súboru prostredníctvom nasledujúceho skriptu vyextrahujeme maticu, ktorá bude obsahovať všetky obrázky; a vrátime maticu normalizovaných obrázkov.

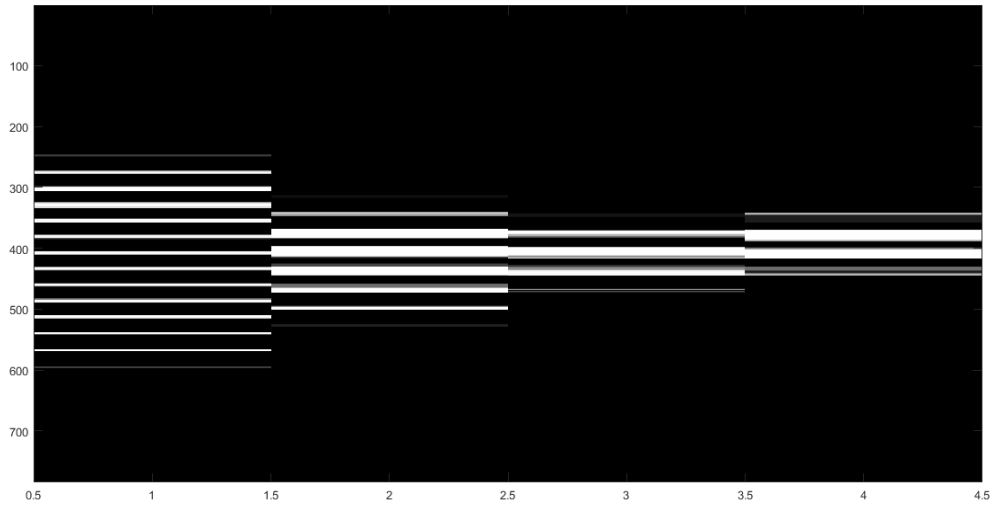
```
function images = loadImages(dbfile)

fp = fopen(dbfile,'rb');
magic = fread(fp, 1, 'int32', 0, 'ieee-be');
numOfImages = fread(fp, 1, 'int32', 0, 'ieee-be');
numOfRows = fread(fp, 1, 'int32', 0, 'ieee-be');
numOfCols = fread(fp, 1, 'int32', 0, 'ieee-be');
images = fread(fp, inf, 'unsigned char');
images = reshape(images,numOfCols,numOfRows,numOfImages);

fclose(fp);
images = reshape(images,size(images,1)*size(images,2),size(images,3));

images = double(images)/255;
```

```
end
```

Podobne si vyextrahujeme aj štítok každej číslice tzv. label, ktorý nám udáva, akú číslicu daný obrázok zobrazuje.

```
function labels = loadLabels(dbfile)

fp = fopen(dbfile,'rb');
magic = fread(fp, 1, 'int32', 0, 'ieee-be');
numOfLabels = fread(fp, 1, 'int32', 0, 'ieee-be');
labels = fread(fp, inf, 'unsigned char');
numOfCols = fread(fp, 1, 'int32', 0, 'ieee-be');

fclose(fp);

end
```

Naimportovaná štruktúra dát nevyhovuje pre všetky ďalšie úkony, ani na kontrolu správnosti parametrov.Totiž naimportovaná matica má rozmery 784 x 60.000, čo znamená, že každý jeden obrázok je reprezentovaný jediným stĺpcovým vektorom.



Obr. 2.1: Vektorová reprezentácia obrázkov

Preto si potrebujeme vytvoriť štruktúru, ktorá bude zrozumiteľná a prehľadná. Vybral som si štrukturované pole, kde sa budem môcť odvolať na atribúty každého obrázka, aj

pridávať ďalšie atribúty neskôr. Jednou slučkou si prejdeme všetky vektory matice vzoriek, vytvoríme si obrázok štandardnej veľkosti, a uložíme si ho vo formáte png. Potom v našom štrukturovanom poli k nemu pridáme pole so štítkom:

```
for p = 1:how_many
    h = 1;
    picture = zeros(28,28);
        for i = 1:28
            for j = 1:28
                picture (j,i) = images(h,p);
               h = h+1;
            end
         end
        imwwrite (picture,strcat('picture',num2str(p),'.png'));
        samples.(strcat('picture',num2str(p))).picture = picture;
        samples.(strcat('picture',num2str(p))).label = labels(p);
end
```

Obr. 2.2: Vygenerovaný obrázok štandardného rozmeru 28x28

Kapitola 3

Extrakcia vlastností

Samotné obrázky, resp. matice pixelov nie sú vhodné na priamu klasifikáciu. Na základe samotných informácii o mieste a miere tmavosti bodu (pixelu) nemožno dosiahnúť dostatočnú mieru určitosti triedy daného obrázku (akú číslicu obsahuje). Dva obrázky môžu mať veľmi podobné rozloženie podobných pixelov a pri tom môžu prislúchať k rôznym čísliciam. Veľkú rolu hrajú v neurčitosti:

- hrúbka čiary
- orientácia (náklon)
- počet bielych obrazových prvkov v jednotlivých častiach obrázku
- ...

Obr. 3.1: Porovnávanie dvoch obrázkov s podobným rozložením pixelov

3.1 Počty úmernosti a orientácie bielych a čiernych pixelov v jednotlivých častiach obrázka

Téza podľa ktorého tieto vlastnosti prispejú k lepšej separácii sa podložila experimentom priamo nad tréningovými dátami (z datasetu, s ktorým sa pracuje). Experiment spočíval v preskúmaní pomeru počtu čiernych a nečiernych pixelov každej triedy, a v hľadaní trendov. Na to bola potrebná funkcia, ktorá dané dáta rozklasifikuje, a vypočíta hore uvedenú konštantu:

```
function (upperCount,downCount,k) = Count_fun(input,class,images,labels)
upperCount = 0;
downCount = 0;

for i = 1:imput
    for j = 1:392
        if ((labels(i) == class) && (images(j,i)>0))
            upperCount = upperCount + 1;
        end
    end

    for k = 393:784
        if ((labels(i) == class) && (images(k,i)>0))
            downCount = downCount + 1;
       end
    end
end
 k = upperCount/downCount;
end
```

Táto funkcia nám vracia počet pixelov v hornej ako aj v dolnej polovici obrázku, aj konštantu ich úmernosti, k; na základe vstupných údajov, ktoré sú:

- input (koľko z celkových 60.000 vzoriek sa má preskúmať)
- class (trend ktorej číslice to má byť)
- images (matica stĺpcových vektorov všetkých obrázkov)
- labels (stĺpcový vektor štítkov každého obrázku samozrejme v tom istom poradí)

Použijeme predošlú funkciu na zber a následnú analýzu otáznych dát na báze 4000 obrázkov zavolaním v nasledujúcom skripte:

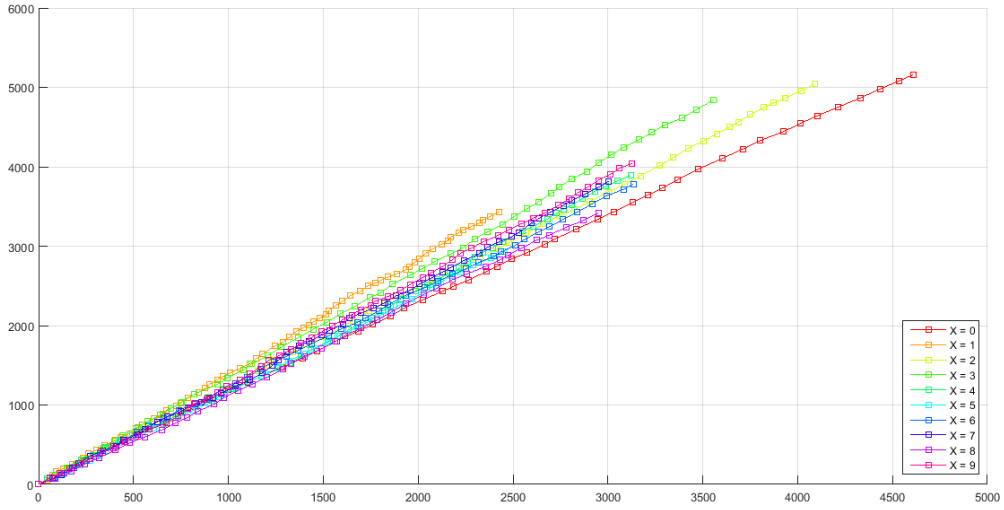
```
x = zeros (4000);
y = zeros (4000);
k = zeros (4000);

X = zeros (4000,3);
Y = zeros (4000,3);
K = zeros (4000,3);

for j = 1:40
    for i = 0:9
        [x(i+1),y(i+1),k(i+1)] = Count_fun(100*j,i,images,labels);
    end

X = [X;x];
X = [Y;y];
X = [X;k.^5];

end
```

Hore uvedený skript sa zopakoval niekoľko krát, s rôžnym počtom analyzovaných obrázkov. Pri väčšom počte ako 100 obrázkov bolo už potrebné dopredu si alokovať pamäť pre všetky premenné, lebo počet operácii ktoré bolo treba vykonať stúpala exponenciálne, a spomaľovala to opätovná alokácia, resp, rozšírenie vyhradenej pamäte pre nasledovné premenné:

- počet nečiernych pixelov v hornej polovičke obrázka
- počet nečiernych pixelov v dolnej polovičke obrázka
- ich konštanta úmernosti na piatu (kvôli lepšiemu odlíšeniu jednotlivých konštant medzi sebou)

Následná analýza spočíva vo vykreslení závislosti počtov nečiernych pixelov v hornej a dolnej polke vybraných obrázkov:

```
cmap = hsv(10);
figure;
grid on;

for i = 1:10
    hold on;

    plot(X(:,i),Y(:,i),'-s','Color',cmap(i,:));
    legendInfo{i} = ['X =' num2str(i)];
end

legend(legendInfo)
```



Obr. 3.2: Priebeh úmernosti pixelov pri zvyšovaní počtu analyzovaných obrázkov

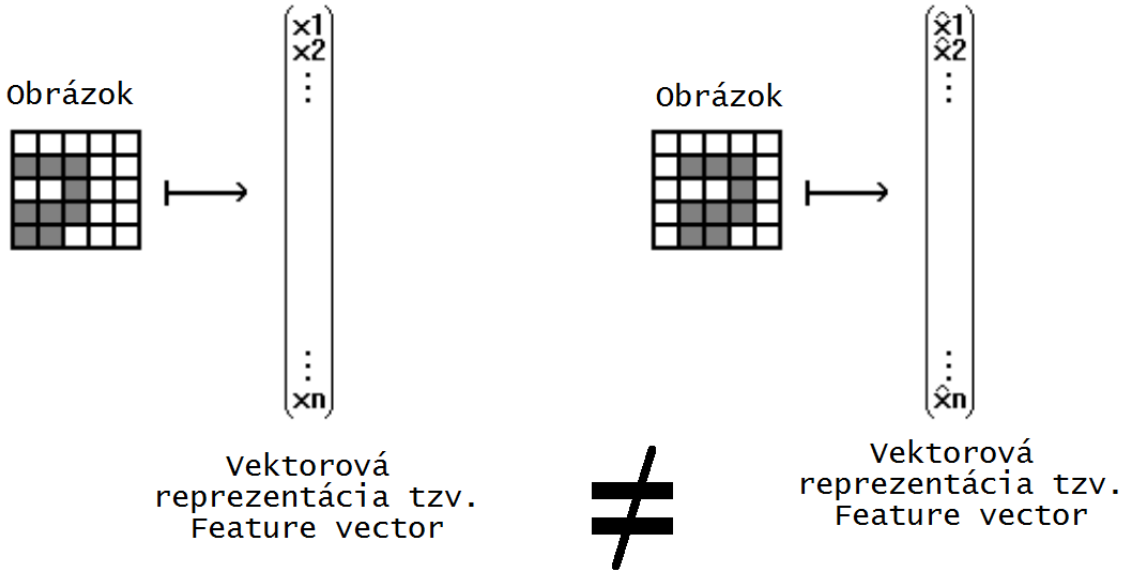
Z jednotlivých kriviek môžeme vyčítať niekoľko podstatných informácii:

- Sú dostatočne veľké rozdiely v samotných počtoch pixelov jednotlivých tried na delenie do kategórii
- väčšina číslic / skupiny číslic ukazujú podobný trend rozloženia nečiernych pixelov

Ako v predošlom odstavci bolo písané: jedine z týchto vlastností sa nedá s dostatočnou pravdepodobnosťou zistit, o akú číslicu ide, ale môžu nám pomôcť v prípade, že na základe

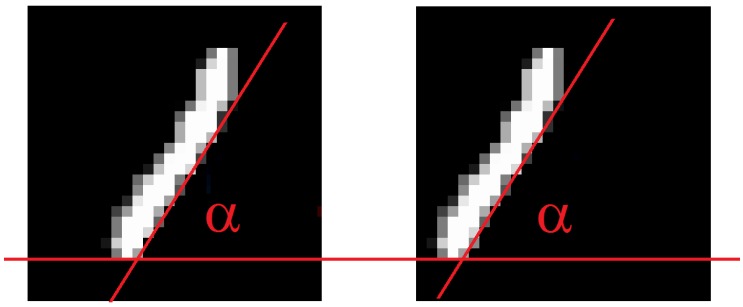
ostatných vlastností nebude možné rozhodovať.

Myšlienka rozvinutá v tejto kapitole ešte ma zaviedla k ďalším experimentom, napríklad k porovnávaniu počtu pixelov na báze kvartálov. Bolo si však treba si uvedomiť že číslice vo vzorke nie sú symetrické, a ani nie sú v presnom strede obrázka. Práve preto sme potrebovali ďalší nápad, v podobe ďalšej vlastnosti, ktorá nebude závislá od umiestnenia číslice.



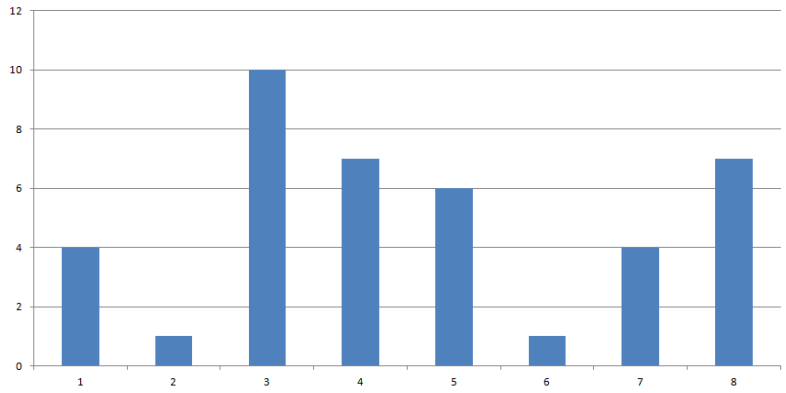
Obr. 3.3: Vplyv presunutia číslice v rámci obrázku na jeho vektor vlastností

Takouto vlastnosťou môže byť počet pixelov ktoré tvoria súvislú priamku pod nejakým uhlom. Samozrejme nemožno stanoviť jediný špecifický uhol pod ktoré dopadajúce pixely resp ich počet by 100



Obr. 3.4: Dva obrázky tej istej triedy, ktoré by vykazovali rôzne vlastnosti napr. pri kvartálnom rozdelení, ale ich podobnosť sa prejavuje v tom istom počte bielych pixelov pod uhlom alfa

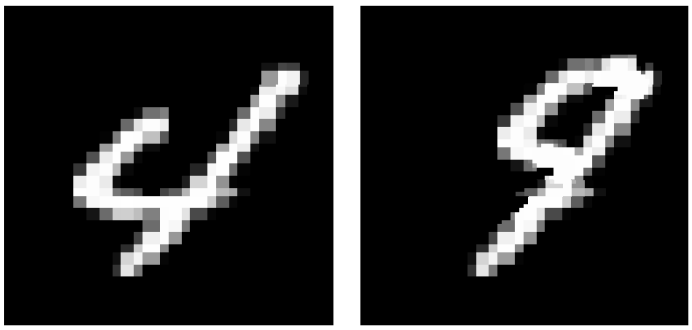
Keď si uhlové spektrum rozbijeme na niekoľko častí, napríklad s krokom 30 stupňov, relatívne malou výpočtovou náročnosťou si môžeme stanoviť ich orientáciu v danom subspektre. Výstup slučky, ktorá zaregistruje hore uvedené dáta od 0 do 180 môže byť reprezentovaný v podobe tabuľky alebo pomocou histogramu:



Obr. 3.5: Histogram orientovanych pixelov niektorej číslice

Nech je zobrazenie akékoľvek, takýto vektor charakterizujúci číslicu / obrázok je dostatočne kompaktný a má výpovednú hodnotu o objekte, ktorý je na obrázku. Túto metódu predviedli N. Dalal a B. Triggs, v roku 2005 na konferencii CVPR (Conference on Computer Vision and Pattern Recognition). Pôvodne sa vyvinula na detekciu ľudí na ulici, ale jej všeobecná použiteľnosť sa rýchlo ukázala, a je jedným zo základných metód identifikácie

obsahu obrázkov. Tejto metóde budem trošku podrobnejšie venovať ďalšiu podkapitolu.



Obr. 3.6: Dva rôzne obrázky s veľmi podobne orientovanými pixelmi, ktorých rozoznanie by mohlo byť problematické bez rozdelení obrázka na bunky.

3.2 Histogram orientovaných gradientov HoG

Extrakcia takejto vlastnosti spočíva v rozdelení obrázka na niekoľko častí (cell), v rámci ktorých sa vypočítajú -na báze niekoľkých (zvyčajne 9) pixelov- vektory gradientu. Vektorový súčet takýchto vektorov udáva výslednú orientáciu danej časti. Algoritmus výpočtu sa dá rozdeliť do 3 fáz:

- Výpočet gradientu
- Rozdelenie gradientov buniek (cells)
- Normalizácia hodnôt

3.2.1 Výpočet gradientu

Prvým krokom, predúpravou je výpočet gradientu. Dá sa k tomu dopracovať viacerými cestami, no optimálna(najefektívnejšia s minimálnou matematickou náročnosťou) vzhľadom na nespojitú funkciu f, presnejšie: diskrétne body je konvolúcia s derivačnými maskami: $\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$ a $\begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$

Gradient funkcie sa matematicky definuje ako:

$$\nabla f = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \tag{3.1}$$

kde $\frac{\partial f}{\partial x}$ je gradient v smere osi x, a $\frac{\partial f}{\partial y}$ je gradient v smere osi y.

A gradient $\frac{\partial f}{\partial x}$ sa vypočíta ako:

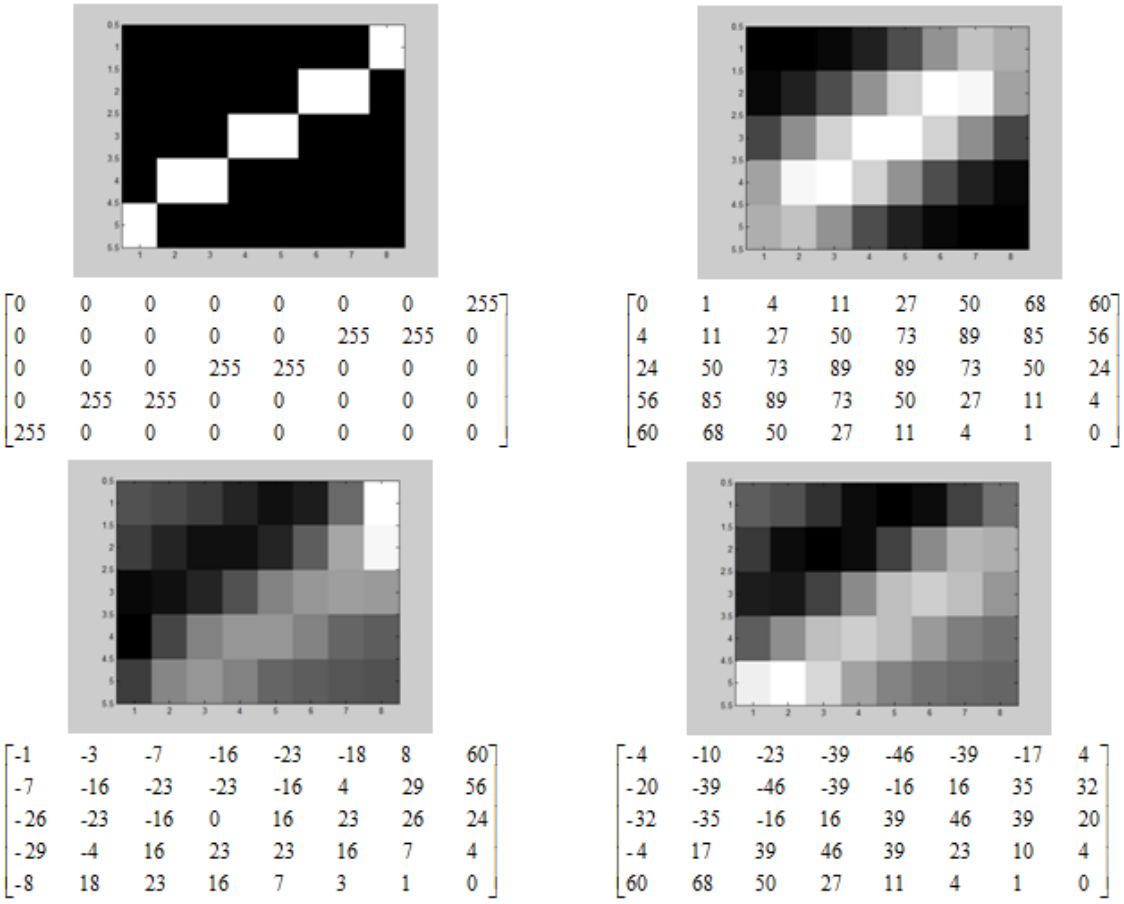
$$\frac{\partial f}{\partial x} = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} * A \tag{3.2}$$

kde * je iD konvolúcia

Teda výsledný smer orientácie pixelu na pozícii $[x, y]$, ϕ dostávame z:

$$\phi = \tan^{-1} \frac{g_y}{g_x} \tag{3.3}$$

Ukážkovým príkladom hľadania orientácie pixelu v obrázku je nižšie zobrazený obrázok naklonenej priamky:



Obr. 3.7: Príklad na 1D konvolúciu, kde matica A reprezentuje čirenobiely obrázok

Na ľavom hornom obrázku je ostrá kontúra, a na pravom je vyhladená kontúra. Na dolnon ľavom je výsledok konvolúcie v smere osi x, a na pravej v smere osi y. Z týchto výsledných matíc už priamo dostaneme orientáciu prvku každej bunky pomocou arkus tangensu pomeru príslušných buniek v gradientoch v smere x-ovej, a y-ovej osi. Napríklad keď nás zaujíma orientácia bunky $[3, 3]$ kak dostaneme:
$\arctan \frac{-16}{-16} = \arctan 1 = \frac{\pi}{4} rad$

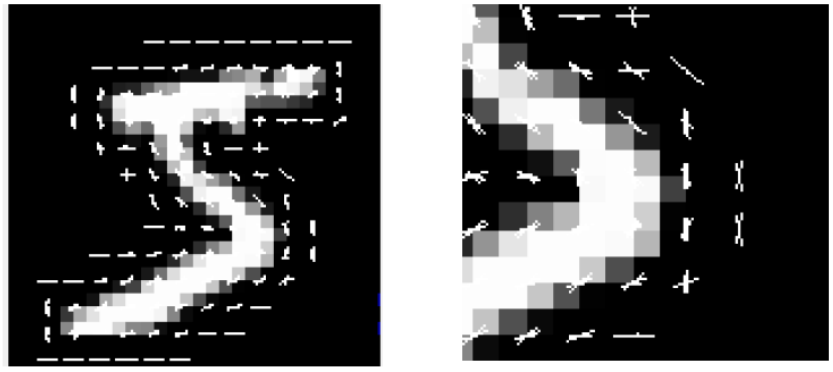
3.2.2 Rozdelenie gradientov buniek (cells)

Na základe ich orientácie pixelov si vytvoríme niekoľko kontajnerov (bin), pri čom pre každý kontajner si špecifikujeme uhlové spektrum v ktorom nachádzajúce sa pixely sa doňho premiestnia. Výsledok tohto kroku už môže byť zobrazený pomocou spomenutého

histogramu.

3.2.3 Normalizácia hodnôt

Funkcia **extractHOGFeatures** nám vracia vektor orientovaných gradientov, aj maticovú reprezentáciu pre ľahkú vizuálnu kontrolu (v oblasti konštantnej intenzity pixelov má gradient nulovú veľkosť, preto vo vizualizícii nie je zobrazená). Inputom pre funkciu okrem obrázku sú rozmery základnej konvolučnej bunky, ktorá nám určuje podrobnosť HOG analýzy. Veľkosť tejto bunky je kľúčovým parametrom pri škálovaní celého riešenia, totiž od tohto parametra závisí najviac rýchlosť celej predúpravy ale aj klasifikácie, vďaka vplyvu na rozmery výsledných vektorov / matíc vlastností.



Obr. 3.8: Vizualizácia orientácii buniek, ktoré obsahujú 4 pixely

Kapitola 4

Support Vector Machines

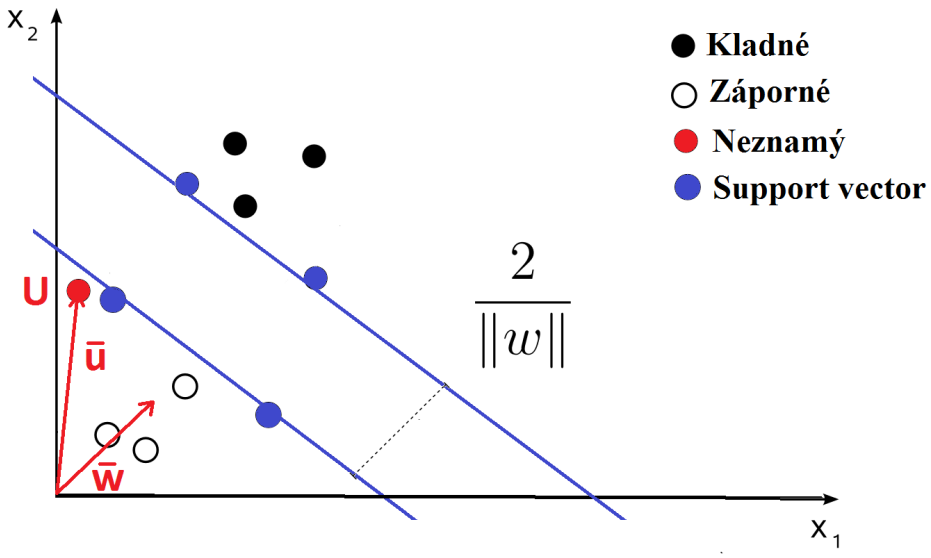
Support Vector Machines patrí do rodiny učiacich sa algoritmov, bližšie do skupiny tzv. **Supervised learning**. Supervised learning v tomto kontexte znamená že klasifikácia neznámych, neoznačovaných vzoriek nie je možná bez fázy training-u, fázy v ktorom používateľ označí všetky vzorky takzvaným štítkom (label), ktoré nám udávajú, do ktorej skupiny patrí daná vzorka (resp. aké číslo je na obrázku).

Vzorky v našom prípade môžu byť obrázky, ktoré môžme chápať ako vektory diskrétnych hodnôt, a každý jeden prvok vo vektore je reprezentovený jedným pixelom. Ale pre účinnejšiu a vôbec uskutočniteľnú separáciu sme si vybrali vlastnosť HOG, čiže vektor gradientov.

Po takomto training-u sa vytvorí model, ktorý je súborom rozhodovacích pravidiel vďaka ktorej behom niekoľkých milisekúnd dostaneme odozvu klasifikačného modelu, a nemusíme training zopakovať pri ďalších neznámych vzorkách. Ale rýchlosť odozvy závisí aj od softvérovej implementácie (o tom píšem v ďalšej kapitole) aj od zvoleného algoritmu ktorý je rozoberaný v nasledujúcich podkapitolách.

4.1 Rozhodovacie pravidlo (decision rule)

Základným problémom ktorý má riešiť SVM- je nájsť separátor v podobe parametrov priamky, ktorý separuje od seba dve množiny bodov v n-rozmernom priestore so snahou vzdialiť tento separátor od oboch množín čo najviac (maximalizácia šírky / marga). Pre jednoduchosť vizualizácie problému budeme používať príklady v dvojrozmernom priestore. Na nasledujúcom obrázku vidíme v 2D priestore dve množiny: kladné body(K) a záporné (Z), ktoré sú lineárne separovateľné, a zároveň jeden neznámy bod U, ktorý vzhľadom na jeho súradnice môžme reprezentovať vektorom $\vec{u}$



Obr. 4.1: Support Vector Machines vizualizácia

Ďalej na obrázku vidíme vektor $\vec{w}$,ktorý je normálový vektor separátora, ktorý je kolmý na separačnú priamku. Aby sme plnili účel použitia SVM, potrebujeme sformulovať rozhodovacie pravidlo (decision rule) na základe ktorého môžeme vyhodnotiť či nový bod, U patrí do jednej alebo druhej množiny.

Podľa grafu vidíme, že projekciou neznámeho vektora $\vec{u}$ do smeru $\vec{w}$ možno jednoduchým porovnávaním zistiť, na ktorej strane separátora sa nachádza bod U. Projekciu vektora $\vec{u}$ do vektora $\vec{w}$ urobíme nasledovne:

$$proj_{\vec{w}}\vec{u} = \frac{\vec{w} \cdot \vec{u}}{|w|^2}\vec{w} \tag{4.1}$$

Ale vzhľadom na definíciu skalárneho súčinu:

$$\vec{w} \cdot \vec{u} = |w| \cdot |u| \cos\phi \tag{4.2}$$

A kosínusu:

$$proj_{\vec{w}}\vec{u} = |u| \cos\phi \tag{4.3}$$

Môžme vzťah zjednodušiť vyjadrením $\cos\phi$ z rovnice č. 4.2 a dosadením do 4.4:

$$proj_{\vec{w}}\vec{u} = |u| \cdot \frac{\vec{w}}{|w|} \tag{4.4}$$

kde $\frac{\vec{w}}{|w|}$ je jednotkový vektor v smere $\vec{w}$.

Teda pre porovnávanie vektorov nám stačí skúmať ich skalárny súčin,

a vzťah sa zjednodušuje na skalárny súčin vektorov $\vec{w}$ a $\vec{u}$:

$$proj_{\vec{w}}\vec{u} = \vec{w} \cdot \vec{u} \tag{4.5}$$

Čiže dostávame vektor, ktorého veľkosť sa rovná skaláru (ktorý je výsledok skalárneho súčinu) a smer je totožný s vektorom $\vec{w}$

Tým, že separačná priamka a vektor, ktorý je kolmý na danú priamku majú jediný spoločný bod (v ktorom sa pretínajú) dá sa určiť hraničná veľkosť výsledného vektora projekcie, pri ktorom sa nachádza daný bod presne na separačnej priamke. Tá hraničná veľkosť je konštantná pre danú priamku, nazvime ju teda c

Matematický zápis tohto rozhodovacieho pravidla:

$$\vec{w} \cdot \vec{u} \geq c \implies u \in K \tag{4.6}$$

Ďalej môžme upraviť do tvaru:

$$\vec{w} \cdot \vec{u} + b \geq 0 \implies u \in K \tag{4.7}$$

Pri čom $c = -b$

Pre algoritmické využitie si zavedieme novú premennú y, ktorá bude mať hodnotu +1 pre vzorky patriace do množiny K(kladné) a -1 pre záporné (množina Z). Čo je možné vzhľadom na fakt, že v prvej fáze, vo fáze training-u poznáme pre každú jednu vzorku jeho príslušnosť do jednotlivých tried (množina , class). Rovno tieto hodnoty môžme včleniť do vektoru $\vec{y_i}$, ktorý v nasledujúcich častiach sa budú nazývať ako vektor štítkov (label).

Vďaka tomuto zjednodušeniu rozhodovacie pravidlo je uniformné pre oba prípady (kladného aj záporného bodu) v tvare:

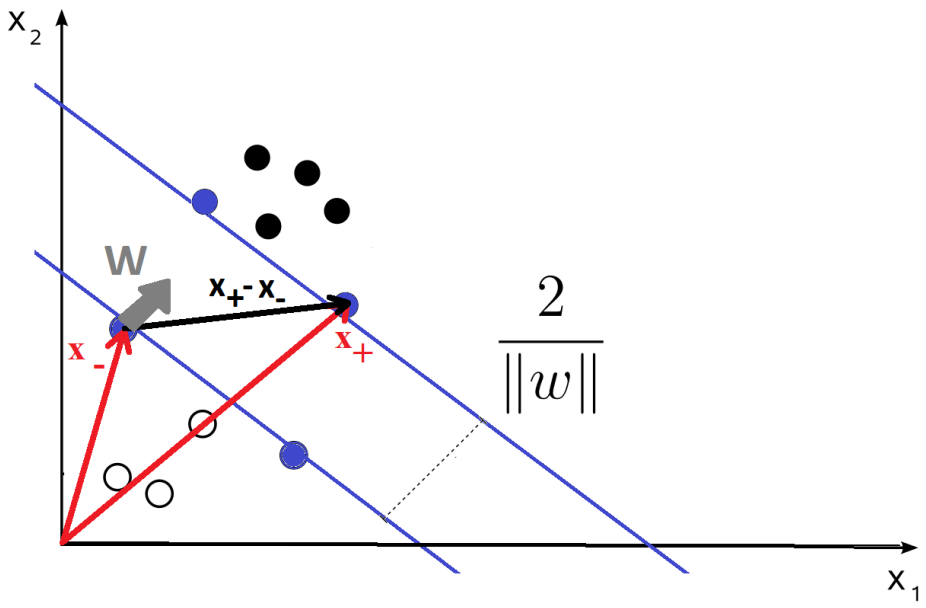
$$\vec{y_i}(\vec{w} \cdot \vec{x} + b) \geq 1 \tag{4.8}$$

Kde $\vec{x}$ je vektor vzoriek bezohľadu na množinovú príslušnosť; A jednotka na pravej strane vznikla kvôli lineárnej závislosti dvoch konečne malých čísel (numerických núl), ktorými sme nahradili ostré nuly v predošlom prípade pre bezproblémovú softvérovú implementáciu.

Ďalej môžme konštatovať, že body, ktoré ju spĺňajú toto ohraničenie s rovnosťou, ležia na separačných hraniciach:

$$\vec{y_i}(\vec{w} \cdot \vec{x} + b) = 1 \tag{4.9}$$

Takýmto bodom hovoríme Support Vectors, pretože tieto body (vektory) znesú (ohraničia) hľadané nadroviny (hyperplane), ktorých únia tvorí margo okolo separačnej priamky. Následne ich budeme označovať $\vec{x_-}$ ak sú zo zápornej množiny Z a $\vec{x_+}$ ak patria do kladnej množiny K.



Obr. 4.2: Výpočet šírky cesty z pozície Support vectorov

Tieto vektory (na obrázku $\vec{x}_-$ a $\vec{x}_+$), podporné vektory sú nám nápomocné pri definovaní šírky separačného medzipriamkového priestoru ako:

$$W = (\vec{x}_+ - \vec{x}_-) \cdot \frac{\vec{w}}{\|w\|} \tag{4.10}$$

Kde $\vec{x}_-$ a $\vec{x}_+$ z rovnice 4.9 budú:

$$\vec{x}_+ = 1 - b \tag{4.11}$$

$$\vec{x}_- = 1 + b \tag{4.12}$$

vzhľadom na to, že $\vec{y}_i$ prislúcha i-tému prvku, a má hodnotu -1 pre záporné vzorky a naopak +1 pre kladné vzorky.

Kombináciou týchto rovníc dostávame hľadanú skalárnu hodnotu šírky v tvare:

$$W = \frac{2}{\|w\|} \tag{4.13}$$

A v tomto momente už máme k dispozícii matematickú formuláciu nášho účelu, ktorá je maximalizácia šírky marga, W, aj príslušné ohraničenia.

4.2 Skonštruovanie účelovej funkcie

Matematický zápis nášho účelu je maximalizácie šírky marga:

$$\max \quad \frac{2}{\|w\|} \tag{4.14}$$

Tým, že pôvodny odvodený vzťah pre výpočet šírky marga nie konvexný, a ani po derivácii by nebolo možné s ním ďalej pracovať uskutočníme niekoľko zmien:

- Optimalizačný problém budeme riešiť ako minimalizáciu takže minimalizujeme reciproku maximalizačnej účelovej funkcie
- Vďaka tomu, že W nie je definovaná v rozsahu záporných hodnôt, môžme nahradiť $\|w\|$ s $\|w\|^2$ lebo majú to isté minimum na $\mathbb{R}$

A potom dostaneme aj s ohraničením:

$$\min \quad \frac{1}{2}\|w\|^2 \tag{4.15}$$

$$\text{s.t.} \quad \vec{y_i}(\vec{w} \cdot \vec{x} + b) \geq 1 \tag{4.16}$$

Takáto formulácia vyhovuje formálnym kritériam pre optimalizačné softvéry, ale tento algoritmus nevyhovuje pre naše potreby:

- Separovať dáta, ktoré v tomto priestore nie sú lineárne separovateľné
- Separovať viac množím ako 2

Takže v ďalšej kapitole budem rozoberať možnosti ktoré ponúka riešenie metódou Lagrange-ových násobičov.

4.3 Metóda Lagrange-ových násobičov

4.3.1 Teória metódy Lagrange-ových násobičov

Metóda Lagrange-ových násobičov je metóda, ktorá slúži na vyriešenie optimalizačných úloh aj s nelineárnymi ohraničeniami.

Táto metóda nám ponúka algoritmus, pre nájdenie takej náhradnej, alebo modifikovanej účelovej funkcie, ktorá má optimum v tom istom bode ako pôvodná účelová funkcia s ohraničeniami, ale dá sa vyriešiť ako neohraničený optimalizačný problém. Nová účelová funkcia bude funkciou viac premenných ako pôvodná, presne o počet ohraničení v pôvodnej účelovej funkcii.

Takýmto modifikovaným účelovým funkciam, príslušným k originálnemu hovoríme Lagrangeián. Lagrange-ián je súčtom pôvodnej účelovej funkcie a lineárnej kombinácie štandardnej formy všetkých ohraničení. Pri čom násobiče, váhy každého ohraničenia sa nazývajú Lagrange-ove multiplikátory.

Hodnoty takýchto Lagrange-ových multiplikátorov majú aj hlbší význam:

- Ich nulová hodnota znamená, že dané ohraničenie v optime je splnené rovnosťou. Inak povedané dané ohraničenie je aktívne.
- Majú výpovednú hodnotu o tom, ako vplýva zmena k nemu prislúchajúceho ohraničenia na číselnú hodnotu účelovej funkcie v optime.

Aplikácia metódy Lagrange-ových násobičov na riešenie SVM

Vďaka tejto metóde môžme riešiť náš optimalizačný problém s ohraničeniami, ako bez ohraničení za cenu že nám pribudnú optimalizačné premenné v podobe Lagrange-ových multiplikátorov α
Nech L je náš Lagrangeián, ktorý zapisujeme:

$$L = \frac{1}{2}\|w\|^2 - \sum_i \alpha_i \left[y_i(\vec{w} \cdot \vec{x_i} + b) - 1 \right] \tag{4.17}$$

A hľadáme jeho extrémy pomocou jeho gradientu:

$$\nabla L = \begin{bmatrix} \frac{\partial L}{\partial \vec{w}} \\ \frac{\partial L}{\partial b} \\ \frac{\partial L}{\partial \alpha} \end{bmatrix} \tag{4.18}$$

$$\frac{\partial L}{\partial \vec{w}} = \vec{w} - \sum_i \alpha_i y_i x_i \tag{4.19}$$

$$\frac{\partial L}{\partial b} = -\sum_i \alpha_i y_i \tag{4.20}$$

$$\frac{\partial L}{\partial \alpha} = -\sum_i y_i(\vec{w} \cdot \vec{x_i} + b) - 1 \tag{4.21}$$

Následne položíme jednotlivé parciálne derivácie rovné nule a takto získame 'kandidátov', potenciálne extrémy Lagrange-iánu 4.16:

$$\vec{w} = \sum_i \alpha_i y_i x_i \tag{4.22}$$

$$\sum_i \alpha_i y_i = 0 \tag{4.23}$$

$$-\sum_i y_i(\vec{w} \cdot \vec{x_i} + b) = 1 \tag{4.24}$$

Potom si spätne dosadíme do Lagrangian-u výsledky parciálnych derivácii a upravujeme: Vďaka tomu, že b je konštanta, po vyňatí pred sumáciu si všimneme že daná sumácia extrémov je rovná nule z rovnice 4.22.

$$L = \frac{1}{2}\left(\sum_i \alpha_i y_i x_i\right) \cdot \left(\sum_j \alpha_j y_j x_j\right) - \left(\sum_i \alpha_i y_i x_i\right) \cdot \left(\sum_j \alpha_j y_j x_j\right) - \sum_i \alpha_i y_i b + \sum_i \alpha_i \tag{4.25}$$

Po ďalšej úprave dostávame tzv. Lagrange-iálny duál z ktorého si môžme odvodiť niekoľko viet:

$$\sum_i \alpha_i - \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j \vec{x_i} \cdot \vec{x_j} \tag{4.26}$$

- Výsledný Lagrange-ián, čiže nepriamo aj optimálne hodnoty parametrov separátora **závisia od skalárneho súčinu dvojíc vzoriek**
- Závislosť 4.21 nám hovorí, že **rozhodujúci vektor dostávame z lineárnej kombinácie vzoriek a Lagrange-ových multiplikátorov**

- Keď si dosadíme rozhodovací vektor $\vec{w}$ zo vzťahu č.4.21 do rozhodovacieho pravidla 4.9, zistíme, že rozhodovanie o novom bode $\vec{u}$ tiež **závisí iba od skalárneho súčinu vzorky a neznámeho vektora** $\vec{u}$

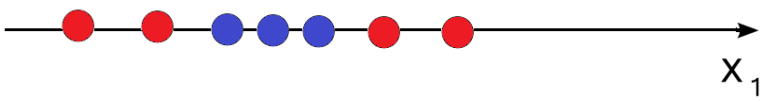
$$\sum_i \alpha_i y_i x_i \cdot \vec{u} + b \geq 0 \implies u \in K \tag{4.27}$$

- Vďaka týmto poznatkom môžme v takom tvare so zníženou matematickou náročnosťou riešiť tú istú klasifikačnú úlohu

- Účelová funkcia v takom tvare nám umožňuje extrakciu lineárne neseparovateľných množín do priestoru kde sú lineárne separovateľné pomocou tzv. 'Kernel trick'-u, ktorému je venovaná ďalšia podkapitola

4.4 Použitie Kernel funkcii

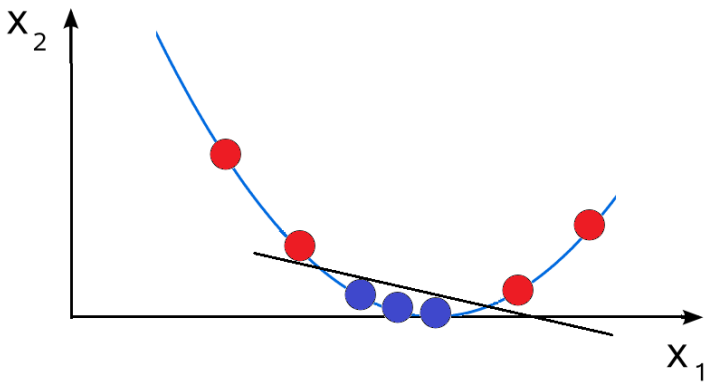
kernel funkcie sú funkcie, ktoré transformujú do nich dosadené body (ktoré nie sú lineárne separovateľné) na body (väčšinou so zväčšením počtu dimenzii) ktoré sú v tom novom priestore lineárne separovateľné.
Najjednoduchším príkladom na takú extrakciu môžu byť body na obrázku č. 4.4



Obr. 4.3: Príklad na 1D, lineárne neseparovateľné dátové množiny

Máme dve množiny, červené a modré, ktoré nie sú lineárne separovateľné v 1D priestore. Keďže tieto množiny nie sú lineárne separovateľné našou úlohou je zmapovať dáta vo viacrozmerovom priestore. A to transformáciou funkcie $f(x) = x$ na $\phi(x) = x^2$

Po takejto transformácii naše dáta v nami vytvorenom viacrozmernom (2D) priestore budú vyzerať takto:



Obr. 4.4: Funkčné hodnoty $f(x) = \phi(x) = x^2$, teda transformované dáta

Z tejto reprezentácie je nám jasné že nami hľadaný separátor existuje, ale len pre upravené, rozšírené dáta. Z priebehu je vidieť, že daný separátor bude spĺňať funkciu aj pri klasifikácii nových vzoriek presne tak ako sme to očakávali pri 1D zobrazeí.
Ale v našom prípade kernel funkciou nie je $\phi(x) = x^2$, ale k, ktorá spĺňa: $k(\vec{x_i}, \vec{x_j}) = \phi(\vec{x_i}) \cdot \phi(\vec{x_j})$

Čiže kvôli ešte menšej výpočtovej náročnosti nehľadáme funkciu, ktorá nám transformuje vzorky (samples) do priestoru s väčším počtom dimenzii (Hilbert space), ale funkciu, ktorá nám vracia rovno skalárny súčin dvoch vektorov reprodukovaných (transformovaných) bodov; **Teda kernel funkcia nám vrátí skalár**

Potom aj rozhodovací vektor (classification vector) budeme mať v tvare:

$$\vec{w} = \sum_i \alpha_i y_i \phi(\vec{x_i}) \tag{4.28}$$

Kde α_i dostaneme z riešenia modifikovanej účelovej funkcie:

$$\sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \phi(\vec{x_i}) \cdot \phi(\vec{x_j}) \tag{4.29}$$

Resp.

$$\sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j k(\vec{x_i}, \vec{x_j}) \tag{4.30}$$

Odkiaľ po nájdení Support vectoru spätne získame:

$$b = \vec{w} \cdot \phi(\vec{x_i}) - y_i \tag{4.31}$$

Najpoužívanejšie Kernel funkcie sú :

- Polynomiálne :

  $k(\vec{x_i}, \vec{x_j}) = (\vec{x_i} \cdot \vec{x_j} + 1)^p$

- Radiálne :

  $k(\vec{x_i}, \vec{x_j}) = e^{-\gamma \|\vec{x_i} - \vec{x_j}\|^2}$

  Kde vo väčšine prípadov (aj v našom) $\gamma = \frac{1}{2}\sigma^2$

- Sigmoidne :

  $k(\vec{x_i}, \vec{x_j}) = \tanh(\kappa \vec{x_i} \cdot x_j - \delta)$

V ďalšej práci budem používať RBF (Radial Base Function) kernel s hore uvedenou gammou.

4.4.1 Testovanie SVM s HoG

K prvotnému výsledku som sa dostal pomocou funkcii Matlab Machine learning toolboxu, ktoré mi dovolili s niekoľkými príkazmi vytvoriť model mojho klasifikátora na základe dlho analyzovaných vlastností. Pri prvej klasifikácii bol použitý lineárny a potom RBF Kernel, s multiclass SVM separátorom. Po všetkých, v tomto dokumente uvedených extrakciach a úpravach, model sa vytvoril príkazmi:

```
classifier = fitcsvm(hogMatrix(1:how_many/2,:), labels(1:how_many/2));
testFeatures = hogMatrix((how_many/2)+1:how_many);
testLabels = labels((how_many/2)+1:how_many,:);
predictedLabels = predict(classifier, testFeatures);
confMatrix = confusionmat(testLabels, predictedLabels)
```

kde classifier je objekt modelu, hogMatrix je Matica všetkých vlastností, kde každý stĺpec predstavuje jeden obrázok, a labels je vektor štítok (v tom istom poradí v akom sú obrázky v

matici images a hogMatrix). Pre jednoduchosť prvej klasifikácie som rozdelil súbor všetkých vzoriek na dve časti, prvá časť bola trénovacia a druhá, testovacia. Funkciou predict sa vytvorí model číslic, v podobe klasifikačného vektoru (s kľúčovými parametrami separátorov) pomocou ktorej každú ďalšiu testovaciu vzorku môžeme klasifikovať.

Kapitola 5

Implementácia v programovacom jazyku Python

5.1 O jazyku Python

Python je jeden z mála bežne používaných interpretovaných programovacích jazykov. Vďaka tejto jeho vlastnosti dokážeme spúšťať skripty veľmi rýchlo, a obísť možné chyby pri kompilácii (bugy). Používa sa vo veľkých softvérových projektov, napr.: BitTorrent, Zope, Mnet, ale je základom aj nedávno otvoreného projektu Googlu pre strojové učenie, Tensorflow. Pri implementácii používam verziu 3.5.

Vybral som si ho kvôli jeho pozícii v špičke preferovaných jazykov pre strojové učenie a Artificial intelligence. Niekoľko ďalších dôvodov na použitie práve tohto jazyka:

- Je vyvíjaný ako Open source projekt
- Je rozširiteľný o moduly v C alebo C++
- Je dynamicky typový jazyk, teda premenné nemajú typ len hodnoty
- Dátový typ záznam
- for slučka ktorý iteruje cez prvky zoznamu / vektora
- špičkové externé knižnice na strojové učenie a spracovanie obrázkov

- ľahká implementácia v akejkoľvek podobe (web, desktop application, aj s GUI)

5.2 Import a spracovanie dát

Na začiatku každého jedného skriptu si zavoláme, naimportujeme externé knižnice, resp. potrebné časti externých knižníc. Naše dáta (MNIST database) priamo v sebe obsahuje knižnica sklearn, čiže nám stačí načítať ich ako celok do matice features a do vektora labels. Matica features podobne ako v Matlabovskom prototype obsahuje obrázky v podobe vektorov hĺbky farby, a vektor labels obsahuje štítky obrázkov v tom istom poradí ako sú umiestnené v matici:

```
from sklearn import datasets
from sklearn.externals import joblib
from sklearn.svm import LinearSVC
from skimage.feature import hog
import numpy as np

dataset = datasets.fetch_mldata("MNIST Original")
features = np.array(dataset.data, 'int16')
labels = np.array(dataset.target, 'int')
```

5.3 Extrakcia vlastností

Podobne ako v Matlabe, ku každej jednej vzorke priradíme štítok, a jeho HoG.

```
HOGfeatures = []
for feature in features:
    vector = hog(feature.reshape((28, 28)), orientations=9,\
        pixels_per_cell=(14, 14), cells_per_block=(1, 1), visualise=False)
```

```
    HOGfeatures.append(vector)
featureMatrix = np.array(HOGfeatures, dtype='float64')
```

Zoznam **listHOGfeatures** postupne napĺňame vektormi **vector** ktoré sú príslušné vektory vlastností k poradiu v matici features, ktoré nám generuje funkcia **hog()** z preformátovaného vektoru (na originálnu veľkosť 28 x 28 pixelov). Potrebujeme pri tom zadefinovať niektoré vstupné parametre pre funkciu **hog()**:

- **orientations** - počet uhlovýk subspektier, v rámci ktorých nás bude zaujímať počet prislúchajúcich pixelov. V našom prípade je to 9, takže subspektrá budú 40 stupňové.
- **pixels_per_cell** - rozmery konvolučnej masky. Čím je menší podiel rozmeru obrázka a konvolučnej masky, tým rýchlejšie pracuje algoritmus, a tým sú výsledky menej podrobné.
- **cells_per_block** - počet buniek konvolučnej masky. Čím je tento počet väčší tým je výsledok podrobnejší, a počet matematických operácii vyšší.

Z týchto atribútov vyplýva, že výsledný vektor vlastností pre jednu vzorku bude mať rozmery 36x1 z 4x9. Pretože konvolučná maska rozdelí náš obrázok na štyri časti, v rámci ktorých budeme mať 9 kontajnerov (bin) v ktorých budeme mať rôzne počty pixelov.

5.4 Training

Ďalším krokom je vytvorenie SVM objektu. Training uskutočníme procedúrou **classifier.fit** ktorého parametre sú už len vlastnosti a štítky. Vytvorený model si uložíme do súboru vzhľadom na to, že proces trainingu je aj časovo aj matematicky náročný. V asledujúcej práce takýmto spôsobom budeme načítavať uložený objekt (model) rukopisov číslic.

```
classifier = LinearSVC()
classifier.fit(featureMatrix, labels)
joblib.dump(classifier, "BC_SVMmodel1.pkl")
```

5.5 Klasifikácia novej vzorky

Nasledujúci klasifikačný skript je už spustiteľný bez akýchkoľvek training-ov, vďaka súboru, ktorý obsahuje náš model **BC_SVMmodel1.pkl**:

```
import cv2
import numpy as np
from sklearn.externals import joblib
from skimage.feature import hog

model = joblib.load("BC_SVMmodel1.pkl")

picture = cv2.imread("picture4.png")

GrayPicture = cv2.cvtColor(picture, cv2.COLOR_RGB2GRAY)
GrayPicture = cv2.GaussianBlur(GrayPicture, (5, 5), 0)

# len ak nie je pozadie cierne
#ret, im_th = cv2.threshold(im_gray, 90, 255, cv2.THRESH_BINARY_INV)

HOGfeatureVector = hog(GrayPicture, orientations=9,\
 pixels_per_cell=(14, 14), cells_per_block=(1, 1), visualise=False)
prediction = model.predict(np.array([HOGfeatureVector], 'float64'))
print('Prediction: %s' % (prediction))
```

Ako skúšku správnosti sme si načítali jeden obrázok, ktorý sme ešte v Matlabe vygenerovali. Po načítaní tohto obrázka (vzorky) a súboru modelu si prekonvertujeme vzorku na čiernobielu. Normalizujeme hĺbky farieb a vyextrahujeme si vlastnosti HOG tými istými parametrami ako pri training-u. A nakoniec funkciou predict obdržíme výsledok rozhodovacieho pravidla v podobe názvu triedy (class), do ktorého patrí vzorka podľa predikcie.

Obr. 5.1: Niekoľko testovateľných obrázkov (vzoriek)

5.6 Aplikácia na rozpoznávanie v rámci zložitých rastrových obrázkov

5.6.1 Bitové mapy

Zaujímavým využitím nášho efektívneho klasifikátora môže byť nasledujúci problém, ktorý je bližší každodenným praktickým potrebám. Majme obrázok klasickej (z bežných fotoaparátov získateľnej) veľkosti, ktorý zobrazuje viacero číslic v rôznych veľkostiach.

Obr. 5.2: Vzorová bitová mapa pre rozpoznanie

V tomto prípade našou úlohou bude:

- Rozpoznať časti obrázka ktoré s najväčšou pravdepodobnosťou zobrazujú nejakú číslicu
- Rozpoznané plochy preformátovať na rozmer, v akom training prebiehal.
- Predikovať triedu vzorky, na základe uloženého modelu

- Výsledné predikcie zobraziť na obrázku spolu s vyznačením skúmaných plôch

```
import cv2
import numpy as np
from sklearn.externals import joblib
from skimage.feature import hog

model = joblib.load("BC_SVMmodel1.pkl")

picture = cv2.imread("try.png")

GrayPicture = cv2.cvtColor(picture, cv2.COLOR_RGB2GRAY)
GrayPicture = cv2.GaussianBlur(GrayPicture, (5, 5), 0)

ret, im_th = cv2.threshold(GrayPicture, 97, 255, cv2.THRESH_BINARY_INV)

_, contours,hier = cv2.findContours(im_th, \
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

V tejto prvej časti kódu po naimportovaní sme si načítali náš klasifikačný model, a rastrový obrázok, ktorý je zdrojom vzoriek. Tento obrázok sa ďalej prekonvertoval na čierno-bielú a normalizoval sa. Pomocou funkcie **findContours()** nájdeme kontúry, a následne s funkciou **boundingRect()** získame súradnice blokov (obdĺžnikového tvaru), v ktorých sú koncentrované kontúry. Práve tieto bloky obsahujú s najväčšou pravdepodobnosťou číslice. Tieto štyri súradnice sú: [ľavý dolný roh, pravý dolný roh,výška ľavej strany, výška pravej strany].

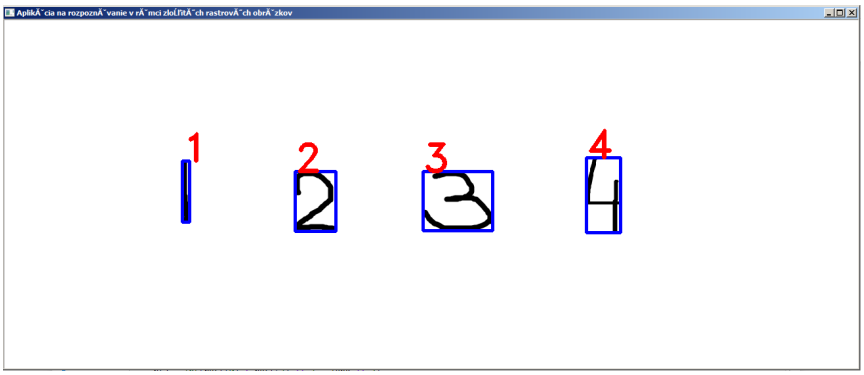
```
rectangles = [cv2.boundingRect(contour) for contour in contours]

for rect in rectangles:  # Vykreslovanie obdlznikov
    cv2.rectangle(picture, (rect[0], rect[1]), (rect[0] \
```

```
        + rect[2], rect[1] + rect[3]), (255, 0, 0), 3)
    # Rozsirenie vyberu
    leng = int(rect[3] * 1.6)
    pt1 = int(rect[1] + rect[3] // 2 - leng // 2)
    pt2 = int(rect[0] + rect[2] // 2 - leng // 2)
    stvorec = im_th[pt1:pt1+leng, pt2:pt2+leng]
    # Preformátovanie na 28x28
    stvorec = cv2.resize(stvorec, (28, 28), interpolation=cv2.INTER_AREA)
    stvorec = cv2.dilate(stvorec, (3, 3))
    # Extrakcia vlastnotí
    hog4sample = hog(stvorec, orientations=9, pixels_per_cell=(14, 14), \
        cells_per_block=(1, 1), visualise=False)
    prediction = model.predict(np.array([hog4sample], 'float64'))
    # Vykreslenie predikcie do obrázka
    cv2.putText(picture, str(int(prediction[0])), \
        (rect[0], rect[1]),cv2.FONT_HERSHEY_DUPLEX, 2, (0, 0, 255), 3)
cv2.imshow("Aplikácia na rozpoznávanie v rámci zložitých \
 rastrových obrázkov", picture)
cv2.waitKey()
```

Ale tým, že všetky úkony potrebujeme vykonať pre všetky identifikované zhluky gradientov, môžme uskutočniť všetky úpravy v jednej for slučke kde iterujeme cez všetky zhluky gradientov:

- Dokresliť obdĺžniky ohraničujúce gradientové zhluky do obrázka funkciou rectangle():
- Rozšíriť obdĺžniky pre všetky zhluky na štvorcový tvar
- Preformátovať vzorku na rozmer: 28x28
- Pre každú takto vytvorenú vzorku vyextrahovať vlastnosti HOG
- Urobiť predikciu na základe nášho modelu

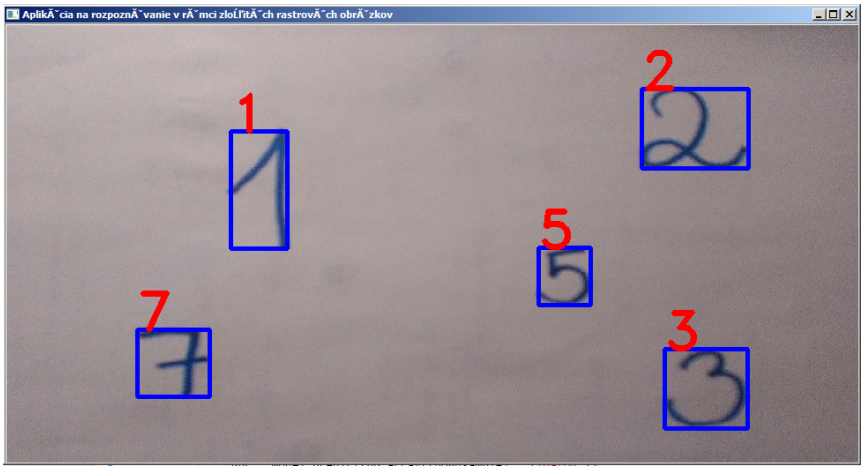
- Vypísať predikciu do obrázku
- Vykresliť obrázok aj s označeniami a predikciami



Obr. 5.3: Odpoveď programu v podobe okna, v ktorom sa nacháza pôvodný obrázok s vykreslenými predikciami

5.6.2 Fotografie

Pridávaním filtrov, alebo aplikovaním ďalších predúprav vzoriek možno funkcionalitu našej aplikácie rozšíriť aj na rozpoznávanie vzoriek zo šumivého prostredia. Napríklad ladením Threshold-ových parametrov môžme získať požadovaný výsledok aj z takejto fotky:



Obr. 5.4: Rozpoznané číslice z fotografie

Vidíme že napriek minimálnej analyzovanej ploche, aj nepodrobnej analýze klasifikátor robí svoju prácu, ale na ďalšie rozšírenie funkcionality už nemá vplyv. To sú už veci predúpravy

zdrojových obrázkov. Pre dosiahnutie súčasného technologického maxima tohto algoritmu by sa mali ešte použiť tzv. **haar kaskády** pre podporu relevantných hitov pri hľadaní potenciálnych číslic aj vo viac zašumených obrázkoch; čo už ale nie je v súlade s predmetom tejto práce, takže radšej niekoľko poznámok k dosiahnutým výsledkom:

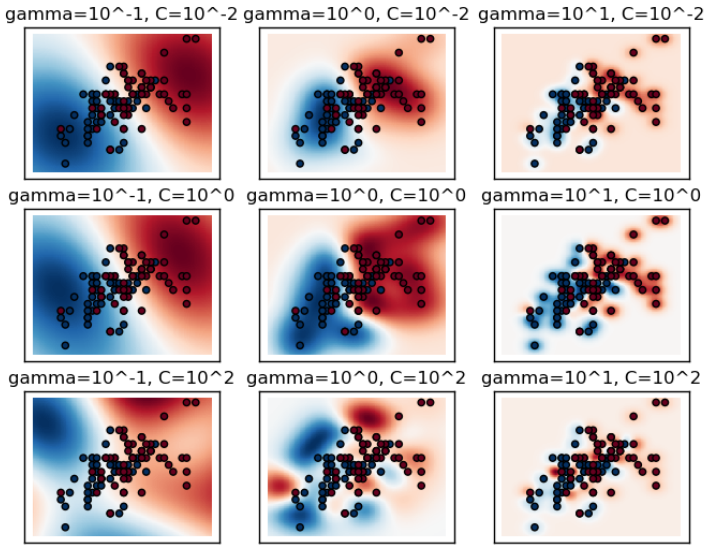
Kapitola 6

Vyhodnotenie výsledkov

Za výsledky práce považujem tak funkcionalitu pripraveného softvéru, ako samotnú úspešnosť vytvoreného klasifikačného algoritmu. V tejto kapitole ale budem merať merateľné, percentuálnu úspešnosť môjho klasifikačného algoritmu, pri rôznej parametrizácii modelu a extrakčného procesu. Totiž pri implementácii sa vyskytlo niekoľko parametrov, (vstupujúce do extrakcie vlastností alebo klasifikácie) ktorých hodnotu sme nastavili ľubovoľne. Ale tým, že tieto parametre majú priamy dopad na výsledky, vznikne požiadavka vyšetriť dopad týchto parametrov na klasifikovaný systém. Tieto parametre sú:

- Počet uhlových subspektier, v rámci ktorých zbierame orientované pixely
- Rozmery konvolučnej masky
- Váha trainingových dát (γ)
- Konštanta hladkosti separátora (c)

Kvôli súdružnosti dvoch parametrov klasifikácie (c,γ) ich nebudem skúmať zvlášť. Všeobecné optimum konštanty c sa nachádza blízko jednotky, preto túto konštantu používam v každom jednom prípade.



Obr. 6.1: Znázornenie vzájomného vzťahu parametrov c a γ

Následné testovanie parametrov urobíme na báze 30000 obrázkov, z krorých 20000 bude trainingových a 10000 testovacích, pre zníženie časovej náročnosti tohto testu. Kvôli zníženému počtu trainingových vzoriek očakávam nižší podiel správne klasifikovaných vzoriek, ale pre účel porovnávania na rovnakej báze to nie je invazívne.

Náš dataset je zoradený, takže si potrebujeme náhodne vybrať v hore uvedených počtoch trainingové a testovacie vzorky. Potom sme si vyextrahovali HOG vlastnosti, pri čom parametre postupne budeme modifikovať:

```
from sklearn import datasets
from sklearn.externals import joblib
from sklearn.svm import LinearSVC
from skimage.feature import hog
import numpy as np

dataset = datasets.fetch_mldata("MNIST Original")
```

```
features = np.array(dataset.data, 'int16')
labels = np.array(dataset.target, 'int')
```

```
import random

labeltrain = []
zoznam = []
imgtrain = []
for i in range(0,20000):
    zoznam.append(random.randint(0, 60000))
for i in zoznam:
    imgtrain.append(features[i])
    labeltrain.append(labels[i])

imgTrain = np.array(imgtrain)
labelTrain = np.array(labeltrain)

HOGfeatures = []
for img in imgtrain:
    vector = hog(img.reshape((28, 28)), orientations=8,\
                 pixels_per_cell=(7, 7), cells_per_block=(2, 2), visualise=False)
    HOGfeatures.append(vector)
featureTrain = np.array(HOGfeatures)

labeltest = []
zoznamt = []
imgtest = []
for i in range(0,10000):
    zoznamt.append(random.randint(0, 60000))
```

```
for i in zoznamt:
    imgtest.append(features[i])
    labeltest.append(labels[i])

imgtest = np.array(imgtest)
labeltest = np.array(labeltest)

HOGfeatures = []
for img in imgtest:
    vector = hog(img.reshape((28, 28)), orientations=8,\
                 pixels_per_cell=(7, 7), cells_per_block=(2, 2), visualise=False)
    HOGfeatures.append(vector)
featureTest = np.array(HOGfeatures)
```

Po vygenerovaní trainingových a testovacích matíc, a k nim príslušných štítok uskutočnili sme training. Klasifikovali sme s radiálnou kernel funkciou, a $\gamma smepostupnemenili$.

```
from sklearn import datasets, svm, metrics

n_samples = len(featureTrain)
data = featureTrain.reshape(n_samples, -1)

classifier = svm.SVC(gamma=2)

classifier.fit(data, labeltrain)
```

Po trainingu sme urobili predikciu testovacích vzoriek, ktoré sme porovnali s ich štítkom. Výstupom z testovania za pomoci reportovacieho tool-u scikit-learn modulu, budú priemerná presnosť klasifikácie všetkých číslic pri daných nastaveniach a porovnávacia matica (Confusion matrix).

```
n_samples = len(featureTest)
datatest = featureTest.reshape(n_samples, -1)

expected = labeltest
predicted = classifier.predict(datatest)

print("Classification report for classifier %s:\n%s\n"
      % (classifier, metrics.classification_report(expected, predicted)))
print("Confusion matrix:\n%s" % metrics.confusion_matrix(expected, predicted))
```

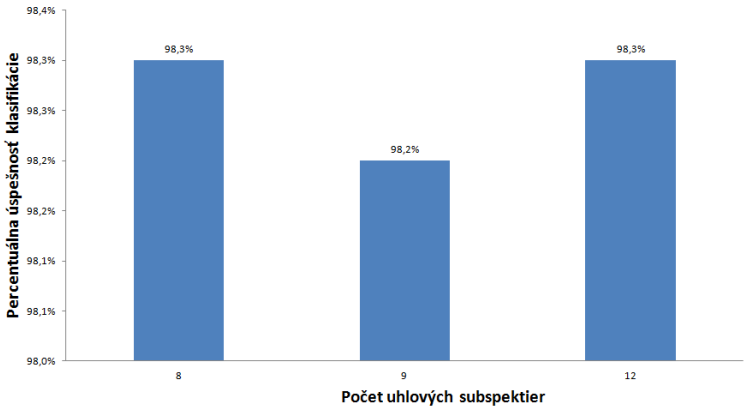
Hore uvedený skript sme zopakovali so stúpajúcimi parametrami:

- Počet uhlových subspektier [8,9,12]
- Rozmery konvolučnej masky [[7,7],[14,14]]
- γ v rozmedzí [0.001:100]

Výstupné reporty sú v prílohe, výňatok dôležitých informácii nižšie, zostupne:

- Počet bin-ov

  Daný parameter ovplyvňoval veľkosť uhla v rámci ktorých sa zhromaždovali pixely. Testoval sa v rámci zmyslúplnývh intervalov (od 30až po 45stupňov).

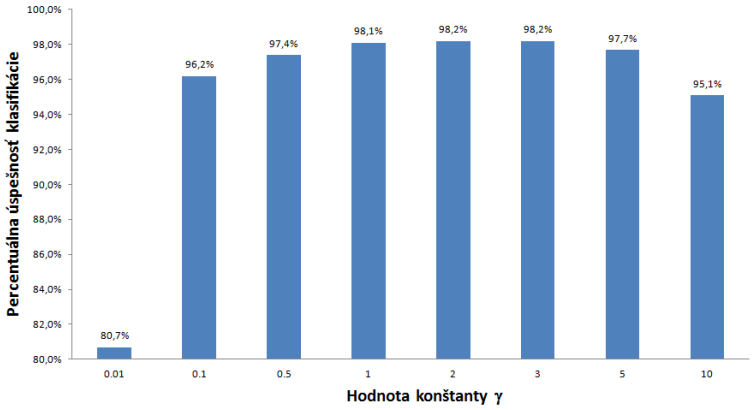


Obr. 6.2: Závislosť presnosti klasifikácie od počtu uhlových subspektier

Z grafickej závislosti je vidno, že tento parameter nemá veľký dopad na zvyšovanie percentuálnej úspešnosti, ale použijú sa informácie: ďalej budem pracovať s počtom subspektier rovných 8.

- γ s dvomi možnými rozlíšeniami konvolučnej masky

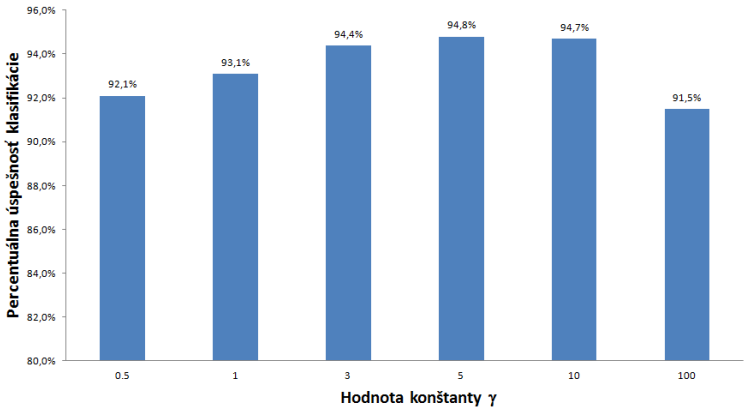
  Na obrázku 6 vidíme vplyv parametra γ na klasifikáciu vzorky s extrakciou HoG vlastností na báze kontajnera 7x7:



Obr. 6.3: Závislosť presnosti klasifikácie od parametra γ pri rozlíšení konvolučnej masky [7x7]

Maximálny počet správne klasifikovaných vzoriek sme dosiahli so skúmaným parametrom v intervale (2,3), čo predstavuje 98,2%-nú presnosť.

V prípade dvojnásobne väčšej konvolučnej masky [14x14], menej podrobnej analýzy sme dosiahli o niečo slabšie výsledky:



Obr. 6.4: Závislosť presnosti klasifikácie od parametra γ pri rozlíšení konvolučnej masky [14x14]

V tomto prípade najlepšie výsledky (94,8%) boli dosiahnuté s $\gamma = 5$.

Na záver sa urobilo training všetkých obrázkov (ktoré sú k dispozícii), a otestoval sa takto vytvorený klasifikačný model. Výsledky predčili očakávania, totiž v prípade 5000 náhodne vybraných testovacích vzoriek sme dosialhi numericky 100%-ný pomer výskytu a rozpoznaniu číslic. Podrobný rozpis klasifikácii sa nachádza nižšie v confusion matici:

Predikcie \ Napísaná číslica	0	1	2	3	4	5	6	7	8	9
0	527	0	0	1	0	0	0	0	0	0
1	0	539	0	0	0	0	0	2	0	0
2	1	0	539	2	0	0	0	0	0	0
3	0	0	1	469	0	1	0	1	0	0
4	0	0	0	0	487	0	0	0	0	0
5	0	0	0	0	0	419	0	0	1	0
6	0	0	0	0	0	0	470	0	0	0
7	0	0	0	0	0	0	0	527	0	0
8	0	0	0	0	0	0	0	0	496	1
9	0	0	0	0	0	0	0	4	2	503

V tabuľke každý jeden prvok zobrazuje počet klasifikácii tej číslice, v ktorej stĺpci sa nachádza do triedy, s číslom jeho riadku. Podľa toho na hlavnej diagonále sú počty správnych klasifikácii, a všetko ostatné (mimo hlavnej diagonály) je chybne klasifikované. Keď vypčítame pomer 24 nesprávnych klasifikácii ku 5000 klasifikácii vychádza nám neuveriteľných 0,48% čo je špičkový výsledok aj v porovnaní s inými prácami s podobným zámerom[1].

[1] http://yann.lecun.com/exdb/mnist/

Kapitola 7

Záver

V rámci bakalárskeho projektu sme sa zameriavali na prípravu základného softvéru na rozpoznávanie a klasifikáciu rukou písaných číslic.

V prvej časti zo spracovaných 60000 obrázkov sme sa snažili vyextrahovať čo najviac relevantných a jedinečných informácii. Od obyčajných počtov čiernych a nečiernych pixelov cez vyšetrenie orientácie sme sa dostali k populárnemu deskriptoru: Histogramu orientovaných gradientov. Po analýze možností tejto metódy sa aj otestoval s rôznymi parametrami, vtedy ešte ľubovoľnými.

Po tom, ako som sa oboznámil so štruktúrou dát, ktoré reprezentujú vyextrahované vlastnosti v euklidovskom priestore som zistil, že dáta nie sú lineárne separovateľné. Aj keď existujú extenzie lineárnej separácie pre taký prípad, využil som túto možnosť spoznať najvyspelejšiu formuláciu Support Vector Machine, SVM s kernel funkciami.

Použitie kernel funkcii ale vyžadovalo iný postup pri riešení optimalizačnej úlohy - použitie metódy Lagrange-ových násobičov. Vďaka tejto metóde sme si odvodili niekoľko zjednodušení pre výpočet optimálnych hodnôt separátora, a vybral som si radiálnu kernelovu funkciu -RBF kernel.

Druhá časť bakalárskeho projektu sa začína implementáciou základných klasifikačných čŕt

v python-e, za pomoci knižníc podporujúce vývoj podobných aplikácii. Pokračovaním je základný modul aplikácie na rozpoznanie číslic aj zo zložitejšieho obrázka. Takouto vzorkou môže byť fotografia z telefónu s nie veľmi výraznými gradientmi pozadia či okolia číslic. Pretože naša aplikácia nie je rozšírená o filtrovanie kaskád, rozpoznáva len zhluky gradientov, a tie posiela na rozpoznávanie.

V poslednej časti sa preskúmali vplyvy parametrov klasifikácie a extrakcie pre optimálne využitie možností klasifikačnej metódy. Tie parametre sa následne otestovali na báze celej databázy, a poskytli perfektné výsledky vyjadrené v percentuálnej úspešnosti.

Môžem konštatovať, že cieľ práce bol splnený, klasifikačný softvér ručne písaných číslic so svojou presnosťou vyhovuje našim požiadavkám.

Poďakovanie

Týmto by som rád poďakoval vedúcemu mojej bakalárskej práce doc. Ing. Michalovi Kvasnicovi, PhD. za jeho rady pri vypracovávaní bakalárskej práce. Ďalej chcem poďakovať Tomášovi Tkáčovi za jeho pripomienky k štýlu tejto práce.

Literatúra

[1] Hunter, J. D., *Matplotlib: A 2D graphics environment*, IEEE COMPUTER SOC, Computing In Science & Engineering, 2007.

Eric Jones and Travis Oliphant and Pearu Peterson and others, *SciPy: Open source scientific tools for Python*, [Online; accessed 2016-05-08], "http://www.scipy.org/", 2001–

[2] Vapnik, Vladimir N., *Statistical Learning Theory*, Wiley-Interscience, 1998.

[3] Dalal, Navneet and Triggs, Bil *Histograms of oriented gradients for human detection*, Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on, IEEE, 2005.

[4] LeCun, Yann and Cortes, Corinna and Burges, Christopher JC, *The MNIST database of handwritten digits*, 1998.