FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

UIR

KLASIFIKACE RUČNĚ PSANÝCH ČÍSLIC

Martin Hamet
12. srpna 2018
Doba přípravy $\simeq$ 80h

Obsah

1 Zadání

Ve zvoleném programovacím jazyce navrhněte a implementujte program, který bude schopen klasifikovat ručně psané číslice 0 - 9. Při řešení budou splněny následující podmínky:

- vytvoření trénovacích / testovacích dat pro učení / testování systému
  - Každý student napíše alespoň jednu sadu cifer (pro tvorbu cifer použijte dodanou mřížku). Tyto budou dále oskenovány a převedeny do textového módu `.pgm` (rastr o velikosti 128 x 128 bodů nakreslených v 256 odstínech šedi). Vytvořená data budou uložena na sdílený disk, kde bude při ukládání dodržena následující konvence (osobní-číslo = adresář pro data každého studenta, v něm založit podadresáře 0 - 9, kde budou uloženy reprezentace jednotlivých cifer v uvedeném formátu).
- implementujte alespoň tři různé algoritmy (z přednášek i vlastní) pro tvorbu příznaků reprezentující číslice
- implementujte alespoň dva různé klasifikační algoritmy (vlastní implementace, klasifikace bude s učitelem, např. klasifikátor s min. vzdáleností)
- funkčnost programu bude následující:
  - spuštění s parametry:
    `trénovací-množina testovací-množina parametrizační-algoritmus klasifikační-algoritmus název-modelu`
    program natrénuje klasifikátor na dané trénovací množině, použije zadaný parametrizační a klasifikační algoritmus, zároveň vyhodnotí úspěšnost klasifikace a natrénovaný model uloží do souboru pro pozdější použití (např. s GUI).
  - spuštění s jedním parametrem `název-modelu` : program se spustí s jednoduchým GUI a uloženým klasifikačním modelem. Program umožní klasifikovat cifry napsané v GUI pomocí myši.
- ohodnoťte kvalitu klasifikátoru na vytvořených datech, použijte metriku přesnost (accuracy). Otestujte všechny konfigurace klasifikátorů (tedy celkem 6 výsledků).

2 Analýza problému a návrh řešení

Úkolem je klasifikovat ručně psané číslice. Číslice mají být klasifikovány ze souborů ve formátu `.pgm` a prostřednictvím GUI tedy z nějaké bitmapy. Problém lze rozdělit na dvě hlavní části parametrizace a klasifikace. Parametrizace zajistí jednotný popis daného obrazu, který bude možné klasifikovat pomocí klasifikátoru. Protože chceme aby bylo možné zvolit si klasifikátor a parametrizační algoritmus nezávisle na sobě musíme sjednotit vstupní parametry klasifikátorů a výstupní popis parametrizací. Pro tento účel se nabízí jednorozměrný vektor o libovolné délce (příznakový vektor).

2.1 Parametrizace

Pro snazší zpracování bude vhodné reprezentace obrazů sjednotit do bitmapy (do které se obraz v případě formátu `.pgm` převede). Vzhledem k tomu že zpracováváme obraz ve 256 odstínech šedi je zbytečné používat klasickou bitmapu (s barevnými kanály A, R, G, B). Proto bude praktičtější vytvořit vlastní bitmapu i vzhledem k jednoduché práci s ní.

Formát `.pgm` (`PGM P2`) je v podstatě textově zapsaná podoba bitmapy, kde jsou na prvních čtyřech řádcích uloženy informace o obrazu (rozměry a maximální hodnota určující obvykle bílou barvu). Dále následují hodnoty určující barvy v jednotlivých pixelech. Jediný problém který z tohoto formátu plyne je že není příliš striktní. Čtení takto uloženého obrazu musí být s nadsázkou volnější. Některé obrazové editory ukládají jednu hodnotu na řádek jiné různě řádky zalamují apod.

Při vytváření bitmapy není úplně zřejmé zda je lepší ponechávat hodnotu pixelu takovou jaká je (tj. hodnota 0 - 255), nebo diskretizovat a pomocí nějaké meze určit zda bude hodnota 0 nebo 255 (vytvořit tedy čistě černobílý obraz bez odstínů). Pomocí dobře zvolené meze pro určení zda má být pixel černý nebo bílý se dají z obrazu odstranit nechtěné ruchy okolí (okraje podél obrazu šum v případě nekvalitního skenování atp.), ale také by se mohli ztratit důležité části vlastní napsané číslice pokud by byla napsaná například světlou tužkou. Obecně by mohlo být vhodné použít kombinaci obou přístupů.

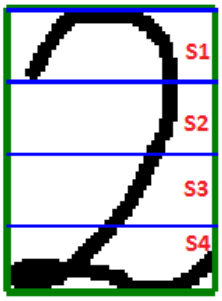
Dále pro minimalizaci nezajímavých částí obrazu (tj. bílých okrajů kolem číslice atp.) použijeme tzv. "MinMax obdelník". Tento obdélník vytvoříme tak, že si uložíme maximální/minimální index pixelu který považujeme za černý) pro sloupce i řádky.

Pomocí těchto indexů vymezíme efektivní plochu obemykající číslici v obrazu, kterou budeme dále zpracovávat (viz obr.1 zelený obdelník).

Parametrizace tedy určí parametrizační vektor podle zvolené metody (viz dále) z bitmapy, kterou vytvoříme načtením ze souboru nebo triviálním převodem z bitmapy vytvořené kreslením v GUI.

2.1.1 Řádkové průměrování (`WEIGHT`)

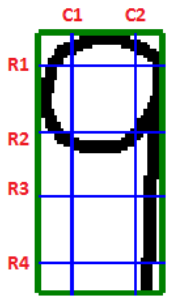
Tento způsob je velmi jednoduchý a zjistí průměrnou hodnotu intenzity v každém řádku bitmapy. V našem případě by tedy vzniklo 128 hodnot což by bylo příliš jemný popis obrazu a nemusel by dobře popisovat číslice obecně (malé rozdíly by působily velké odlišnosti). Budeme tedy dále průměrovat řádky mezi sebou tak že rozdělíme bitmapu na požadovaný počet sekcí (viz obr.1 sekce `S1 - S4`) a v každé sekci zprůměrujeme všechny příslušné řádky. Tímto způsobem dostaneme číslo pro každou sekci, které udává poměr bílé a černé barvy v dané sekci. V tomto případě je obrázek rozdělen na čtyři sekce příznakový vektor tedy bude obsahovat číslo pro každou. Tím jsme značně zkrátili příznakový vektor a jak je vidět na obr.1 tento popis by měl sloužit dobře k rozlišení mezi čísly která se ve vertikálním směru značně liší v počtu černých pixelů na sekci (např. čísla 9 a 6).



Obrázek 1: Rozdělení obrazu na sekce.

Protože se jedná o horizontální průměrování nezáleží na horizontálním posunu nebo šířce číslice. Rozdělení na velké sekce může působit problémy např. v případě číslice 4 záleží zda vodorovná čára padne do sekce `S3` nebo `S4`.

2.1.2 Vzorkování (SAMPLE)



Obrázek 2: Vzorky obrazu.

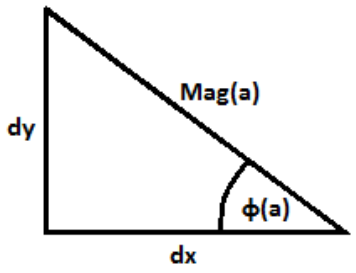
Jedná se podobný přístup jako v řádkovém průměrování s tím rozdílem že tentokrát nebudeme průměrovat řádky mezi sebou, ale vezmeme pouze reprezentativní vzorky řádek. Zvolíme vzorky řádek a sloupců a zjistíme tedy poměr bílé a černé barvy ve vybraných řádcích/sloupcích. Vzorky vybíráme rovnoměrně a pouze z efektivního obdélníku (zelený obdélník) (viz obr.2). Na obrázku jsou znázorněny vzorky řádek `R1 - R4` a vzorky sloupců `C1 a C2`. Průměry z jednotlivých řádek a sloupců zařadíme postupně do parametrizačního vektoru. Výsledný parametrizační vektor `P` by v tomto případě vypadal `P = [AR1, AR2, AR3, AR4, AC1, AC2]`, kde `A` značí aritmetrický průměr hodnot (intenzity) na příslušném řádku/sloupci.

Tento přístup umožňuje lepší rozlišení v horizontálním směru oproti metodě řádkového průměrování. V podstatě z výsledného vektoru můžeme určit kolikrát číslice prošla daný vzorkem pokud by jsme znali tloušťku čáry kterou je číslice napsaná. Protože se ale tato tloušťka projeví v každém vzorku jedná se o vynásobení výsledného vektoru konstantou čímž vznikne rovnoběžný vektor. Bylo by možné pokusit se o předzpracování obrazu tak aby tloušťka číslice byla pouze jeden pixel, potom by jsme skutečně dostali počet průniků se vzorkem. Pro zachování jednoduchosti zůstaneme u původního řešení.

Podobně jako u předchozí metody při zvolení velkého počtu vzorků bude metoda špatně zobecňovat číslice a při malém počtu by měli různé číslice příliš podobný výsledný parametrizační vektor.

2.1.3 Histogram orientovaných gradientů (HOG)

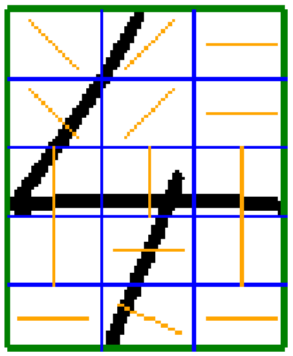
Tato metoda je založená na principu orientace gradientů v různých částech obrazu. Nejprve je nutné si uvědomit že pomocí gradientu v obrazu můžeme zjistit směr a sílu změny (intenzity barvy) v nějakém směru.



Obrázek 3: Gradient v bodě.

Aby jsme mohli určit gradient v některém bodě musíme nejprve určit hodnotu změny (derivaci) v horizontálním směru. To znamená zjistit změnu hodnoty `h` (v našem případě intenzita) předchozího a následujícího pixelu. Řekneme-li že chceme hodnotu derivace `dx` v pixelu `a` se souřadnicemi `[x, y]` použijeme vztah $dx(a) = h([x+1, y]) - h([x-1, y])$ stejně jako kdyby jsme detekovali hrany. Takto by jsme mohli pouze určit směr doleva nebo doprava. Pokud chceme určit směr ve rovině je nutné spočítat i derivaci ve vertikálním směru $dy(a) = h([x, y+1]) - h([x, y-1])$. Nyní můžeme určit gradient. Jak je znázorněno na obrázku obr.3 pomocí pravidel v pravoúhlém trojúhelníku spočítáme směr největší změny $\phi(a) = \tan^{-1}(\frac{dy(a)}{dx(a)})$ a její sílu v tomto směru $Mag(a) = \sqrt{dx(a)^2 + dy(a)^2}$.

Když se zamyslíme nad směrem největší změny v našem případě není žádoucí určovat směr ve všech směrech roviny. V případě tenké rovné čáry by se od sebe gradienty odečetli protože by měli vzájemně opačný směr, proto budeme brát směr v absolutní hodnotě. Zajímá nás tedy směr největší změny ve `180°`. Aby jsme mohli tvořit histogram těchto úhlů musíme je diskretizovat (např. po 30° dostaneme 6 částí). Každý pixel přispěje podle svého gradientu svou silou do příslušného úhlu (nejlépe váženým podílem do dvou nejbližších úhlů).



Obrázek 4: `HOG` gradienty v sekcích

Nyní už by jsme mohli vytvořit takový histogram pro celý obraz. Tím bychom ale přišli o informaci pozice jednotlivých gradientů. Aby jsme tuto informaci částečně zachovali rozdělíme efektivní oblast obrazu ještě do sekcí a pro každou sekci vytvoříme vlastní histogram (viz obr.4).

2.2 Klasifikace

Při parametrizaci jsme získali vektor pro každý obraz, který ho podle zvolené parametrizační metody popisuje. V klasifikační části potřebujeme určit do jaké klasifikační třídy zařadit vektor na základě trénovacích dat, kde známe správný výsledek. Je tedy třeba klasifikátor připravit (vytvořit model) v trénovací části tak, aby s co největší přesností určil správnou třídu bez znalosti výsledku. Klasifikátor musí být také schopen takto vytvořený model uložit a případně později načíst, proto budeme volit jednoduché datové struktury. Pro účely ukládání bude vhodnější formát `XML` místo `TXT` vzhledem k přehlednosti uložených dat.

2.2.1 Etalonový klasifikátor (`DISTANCE`)

Tento klasifikátor nejprve načte všechna trénovací data a poté je zpracuje a vytvoří vzorový parametrizační vektoru pro každou klasifikační třídu. Při testovací fázi už pouze počítá vzdálenost vzorů (etalonů) od testovaného vektoru (ze zkoumaného obrazu). Podle etalonu s nejkratší vzdáleností od testovaného zařadí testovaný vektor do příslušné třídy. Jedná se o jednoduchý klasifikátor, kde ovšem záleží na správném vytvoření etalonů.

Nejjednodušší způsob určení etalonů je průměr v dané klasifikační třídě z vektorů vytvořených z trénovacích dat. Výhodou je že po trénovací fázi není nutné uchovávat všechna trénovací data, ale pouze výše zmíněné etalony a je tedy tento klasifikátor paměťově významně úsporný.

Pro vypočítání vzdálenosti v našem případě příznakových vektorů můžeme použít Euklidovskou vzdálenost. Je tedy důležité aby se po celou dobu funkce tohoto klasifikátoru používal stejný parametrizační algoritmus, který produkuje vektory o stejné délce.

2.2.2 K nejbližších sousedů (`KNN`)

Tento klasifikátor nevyžaduje žádné předzpracování trénovacích vektorů a pouze je zařazuje do své datové struktury podle jejich příslušnosti ke správné třídě. Testování dále probíhá na principu zjištění vzdálenosti (také Euklidovská vzdálenost) ke každému z trénovacích případů. A podle četností tříd v `K` nejbližších případech rozhodne do které třídy zkoumaný vektor zařadí.

Tento klasifikátor vyžaduje větší množství trénovacích dat, ale dokáže zpracovat různorodější třídy v našem případě různě psané číslice lépe. Tato výhoda sebou ovšem nese nevýhodu paměťové náročnosti protože je nutné,

aby klasifikátor měl přístup ke všem trénovacím vektorům po celou dobu provozu. U takového klasifikátoru by mohla hrát významnou roli délka příznakového vektoru (matice, apod.). Tento klasifikátor je také kvůli množství dat pomalejší než např. Ethalonový klasifikátor.

3 Implementace (popis řešení)

Program je sestaven tak, aby bylo snadné ho rozšířit o další klasifikátory a parametrizační algoritmy a zajištění snadného upravení již existujících (viz dále).

3.0.1 Hlavní moduly programu

`Form1` zajišťuje funkčnost GUI a poskytuje nástroje pro volání vykreslení atp.

`Program` zajišťuje hlavní funkci programu spuštění podle zadaných parametrů načítání, testování dat a spuštění GUI. Vytvoří novou instanci klasifikátoru a deskriptoru (parametrizační algoritmus). Posílá načtená data do klasifikátoru pro natrénování a následně pro testování. Obsahuje pouze jednu veřejnou metodu `evaluateBitmap()`, která je volaná prostřednictvím GUI pro vyhodnocení ručně napsané číslice v GUI reprezentované bitmapu.

`BWImage` obaluje vlastní bitmapu obrazu. Umožňuje vytvoření instance podle cesty k souboru (obrázku ve formátu `PGM P2`) nebo podle bitmapy. Při načítání jsou ukládány hodnoty o minimálních a maximálních indexech pro pozdější vytvoření MinMax obdélníku. Poskytuje možnosti převodu bitmapy do bitmapy kompatibilní s GUI pro vykreslení a získání výše zmíněných indexů.

`IDescriptor` poskytuje rozhraní pro jednotné ovládání deskriptorů.

- `getDescription()` vytvoří parametrizační vektor z předaného obrazu pomocí `BWImage`.
- `getDescriptionVectorLength()` vrátí délku parametrizačních vektorů které vytváří

`ICLassifier` poskytuje rozhraní pro jendotné ovládání klasifikátorů.

- `addTrainCase() přidá do klasifikátoru nový vektor se správnou třídou`. Pomocí této metody se trénuje klasifikátor.
- `evaluate()` vytvoří vektor (descriptorem klasifikátoru) z předaného obrazu `BWImage` a klasifikuje ho. Metoda vrací číslo třídy do které byl tento vektor klasifikován.
- `saveModel()` uloží natrénovaný model do souboru ve formátu `XML` pro pozdější použití.
- `loadModel()` načte natrénovaný model ze souboru a připraví tak klasifikátor na vyhodnocování.

`WeightDescriptor` reprezentuje parametrizační algoritmus řádkového průměrování. Pomocí konstanty `DESCRIPTION_LENGTH` lze nastavit délka výsledného vektoru a tím i počet sekcí na které se obraz rozdělí. Descriptor nejprve vytváří vektor o délce 128 (původní výška obrazu) a dále ho normalizuje na danou hodnotu metodou `getNormalizedDescription()`.

`SampleDescriptor` reprezentuje parametrizační algoritmus vzorkování. Pomocí konstant `EXAMINED_ROWS` a `EXAMINED_COLUMNS` lze nastavit počet vzorků na řádek a sloupec. Délka výsledného vektoru je udaná součtem těchto konstant. Deskriptor nejprve zjistí indexy řádků a sloupců v efektivním obdélníku podle počtu požadovaných vzorků metodou `getSamples()`. Dále projde tyto řádky a sloupce v předaném obrazu `BWImage` a vytvoří výsledný vektor.

`HOGDescriptor` reprezentuje parametrizační algoritmus histogramu orientovaných gradientů. Pomocí konstant `BUCKET_ROWS` a `BUCKET_COLUMNS` lze nastavit počet sekcí na které se obraz rozdělí a tím počet histogramů na celý obraz což ovlivňuje délku výsledného vektoru. Konstantou `HISTOGRAM_ANGLES` lze ovlivnit jemnost diskretizace směru gradientu. Descriptor prochází obraz `BWImage` a pro každý pixel zjistí jeho gradient a sekci do které patří metodou `getIndexOfBucket()` a přispěje váženou hodnotou gradientu do příslušné sekce a jejího histogramu. Výsledný vektor je potom vytvořen serializací histogramů ze všech sekcí.

`DistanceClassifier` reprezentuje etalonový klasifikátor. Přidávané trénovací vektory ukládá do seznamů ve slovníkové struktuře tak, že klíčem je správná klasifikační třída. Při prvním zavolání metody `evaluate()` se projdou všechny trénovací vektory metodou `compileReadings()` a vytvoří se příslušné etalony do seznamu `finalPatterns`. Při ukládání modelu zanikají informace o trénovacích vektorech a ukládá se pouze seznam etalonů.

`NearestNeighborClassifier` reprezentuje klasifikátor K nejbližších sousedů. Přidávané trénovací vektory ukládá do seznamů ve slovníkové struktuře, tak že klíčem k seznamu je správná klasifikační třída. Stejně jako předchozí klasifikátor ovšem při ukládání se žádná informace neztrácí a veškeré trénovací případy se zachovávají.

4 Uživatelská příručka

4.0.1 Formát uložených dat

Formát obrazů ve formátu `PGM P2` viz 1 Zadání.

Trénovací a testovací data musejí být uložena tak, že složka obsahuje podsložky 0 - 9 a v každé podsložce jsou samotné obrazy. Každý obraz musí být umístěn v takové složce která odpovídá jeho správné klasifikační třídě, aby bylo jasné co za třídu klasifikátor právě trénuje. Stejně u testovacích dat kde je toto umístění nutné pro určení přesnosti klasifikace. Na vlastních názvech souborů nezáleží je pouze nutná koncovka `.pgm`.

4.1 Spuštění programu

Program se spouští z příkazové řádky. Jsou dvě možnosti spuštění. Program je napsaný v jazyce `C#` bude tedy nutné pros spuštění mít stažené knihovny `Mcrosoft .NET Framework`.

- `NumberClasification.exe arg1 arg2 arg3 arg4 arg5`
  - `arg1` název složky s trénovacími daty
  - `arg2` název složky s testovacími daty
  - `arg3` zkratka vybraného parametrizačního algoritmu (viz 4.1.1)
  - `arg4` zkratka vybraného klasifikátoru (viz 4.1.2)
  - `arg5` název souboru do kterého se natrénovaný model uloží
- `NumberClasification.exe arg1`
  - `arg1` název souboru s modelem který se má načíst

Příklad spuštění by mohl například vypadat takto:
`NumberClasification.exe TrainingData TestingData HOG DISTANCE ulozenyModul`

Po spuštění programu prvním způsobem se načtou trénovací data proběhne natrénování klasifikátoru podle zvolených parametrů. Dále se načtou testovací data proběhne testování a je do konzole vypsán výpis o úspěšnosti. Natrénovaný model se poté uloží do aktuální složky se jménem zadaným v parametrech.

Pro spuštění programu druhým způsobem je třeba mít již uložený natrénovaný model. Program načte model připraví se klasifikační a parametrizační algoritmy které byli použity při ukládání modelu a zobrazí jednoduché uživatelské rozhraní (viz 4.1.3).

Při spuštění programu s chybným počtem parametrů se zobrazí nápověda.

4.1.1 Zkratky parametrizačních algoritmů

- `WEIGHT` řádkové průměrování
- `SAMPLE` vzorkování
- `HOG` histogram orientovaných gradientů

4.1.2 Zkratky klasifikačních algoritmů

- `DISTANCE` etalonový klasifikátor
- `KNN` klasifikátor K nejbližších sousedů

4.1.3 Grafické uživatelské rozhraní GUI

Po spuštění GUI je okamžitě možné sním pracovat. Do bílého panelu uprostřed je možné levým tlačítkem kreslit (resp. psát) pravé tlačítko myši umožňuje gumování. Plátno je interaktivní ve smyslu změny velikosti okna. Při nepoměrné velikosti okna je plátno doplněno o bílé okraje kam není možné psát z důvodu zachování poměru stran. Plátno má stejný rastr jako obrázky uložené v souborech.

Po levé straně panelu se nabízí možnosti pro vymazání kreslícího plátna a pro vyhodnocení. Tlačítko vyhodnocení vyhodnotí obraz na kreslícím plátně vykreslí doplňující informace k obrázku podle zvoleného parametrizačního algoritmu a zobrazí výsledek a podrobnosti o vyhodnocení v levém panelu.

Seznam čísel udává vzdálenosti jednotlivých tříd (případně jednotlivých trénovacích vektoru u `KNN`) k vyhodnocovanému obrazu na plátně. Pokud by byl obraz příliš malý nebo prázdné plátno nebude obraz vyhodnocen stejně jako u souborových obrazů.

5 Závěr

Program je připraven pro další rozšiřování o klasifikátory a parametrizační algoritmy. Zvolené algoritmy byly vybrány tak, aby byly snadno implementovatelné a dali se dále rozšiřovat. Výsledky byly překvapivě dobré vzhledem k jedoduchosti algoritmů. Nejlépe klasifikovala kombinace histogramu orientovaných gradientů a ethalonový klasifikátor i přesto že byla zvolena horší varianta `HOG`, kde se sekce navzájem nepřekrývají. Mezi klasifikátory nebyl příliš velký rozdíl, ale to je způsobeno především zkoušením na menší trénovací množině, kde `KNN` klasifikátor nemá dostatečný podklad trénovacích dat. Přesto i při 5 vzorcích od každé číslice klasifikoval s nadprůměrnou úspěšností. Pokusil jsem se o implementaci i jiných parametrizačních algoritmů např. LBP (Local Binnary Pattern) bohužel se nepodařilo rozpoznávání dovést alespoň do 50%. Bylo patrné že LBP by mohl mýt dobré výsledky pro rozpooznávání textur. Neúspěch byl pravděpodobně způsoben chybnou implementací, nebo nepochopením problematiky pro modifikování na rozpoznávání číslic, proto byl nahrazeny jiným algoritmem a LBP byl vyřazen z konečného projektu.

Testovaná kombinace	Počet vzorků	Správně	Přesnost
$WEIGHT + DISTANCE$	541	319	58.96%
$WEIGHT + KNN$	541	301	55.64%
$SAMPLE + DISTANCE$	541	358	66.17%
$SAMPLE + KNN$	541	335	61.9%
$HOG + DISTANCE$	541	453	83.7%
$HOG + KNN$	541	436	80.59%