Semestrální práce z předmětu KIV/UIR

Klasifikace ručně psaných číslic

Strávený čas: cca 25-30 hodin

student:	Jan Hinterholzinger
studijní číslo:	A19B0052P
email:	hintik@students.zcu.cz
datum:	19. 05. 2021

Obsah

1 Popis problému

Ve zvoleném programovacím jazyce navrhněte a implementujte program, který umožní klasifikovat ručně psané číslice (0-9). Pro řešení musí být použita dodaná datová sada a implementovat alespoň dva různé algoritmy pro tvorbu příznaků číslic a dva různé klasifikační algoritmy s učitele.

Program je možno spustit:

- s parametry: trénovací_množina, testovací_množina, parametrizační_algoritumus, klasifikační_algoritmus, název_modelu
  Program natrénuje klasifikátor na dané trénovací množině, použije klasifikační algoritmus a parametrizační algoritmus, spustí vyhodnocení na testovacích datech a natrénovaný model uloží do souboru.

- s jedním parametrem: název_modelu
  Program se spustí v okně s načteným modelem. Program umožní klasifikovat klasifikovat cifry napsané v rozhraní pomocí myši.

Použité klasifikátory ohodnoťte na vytvořených datech při použití metriky přesnosti. Otestujte všechny konfigurace klasifikátorů.

2 Analýza problému

Dodaná datová sada obsahuje roztříděné trénovací a testovací data. Data jsou šedotónové PNG obrázky o velikosti 28x28 pixelů. Tyto obrázky obsahují různě ručně psané cifry.

Jednotlivé obrázky z datové sady musí být reprezentovány parametrizačním algoritmem, a tím jsou vytvořeny příznaky. Na základě těchto příznaků je klasifikátor dokáže porovnávat a získat tak představu pro budoucí klasifikaci. Klasifikátor tedy na základě příznaků nějakým způsobem odvodí určitou podobnost se zadaným příznakem a dokáže na základě této podobnosti určit klasifikační třídu zadaného příznaku.

Klasifikátor má být ohodnocen metrikou `accuracy`. Jedná se o nejjednodušší metriku poměru správně klasifikovaných ku celkovému počtu klasifikovaných.

3 Návrh řešení

3.1 Grafické rozhraní

Grafické rozhraní by mělo být jednoduché a proto není zapotřebí navrhovat různé detaily a zbytečné funkce. Proto okenní aplikace obsahuje kreslící plátno, kam lze myší kreslit a napsat tak číslici, kterou chceme klasifikovat. Dalším nezbytným prvkem je tlačítko pro zahájení klasifikace a popis pro zobrazení výsledku.

3.2 Tvorba příznaků

Data v trénovacích souborech jsou šedotónová to znamená, že vždy mají všechny složky stejné barvy. Proto můžeme pro naši reprezentaci počítat pouze s barvou pouze jedné barevné složky. Při převodu je proveden také převod jednotlivých složek vektoru tak, aby maximální hodnotou ve vektoru byla hodnota 1.0.

Pro tvorbu příznaků byly zvoleny metody histogramu a přetvarování do vektoru.

3.2.1 Histogram

Metoda histogramu provede součet všech hodnot podle řádků. A následně tyto hodnoty vydělí počtem sloupců, aby bylo zachováno, že maximální možná hodnota ve výsledném vektoru je 1.0. Pro lepší představu je uvedena tabulka 1, kde v levé části je matice odpovídající obrázku s hodnotami barev a na pravé straně výsledný příznakový vektor. Při této transformaci ztrácíme spoustu informací o obrázku a mohu také předpokládat, že úspěšnost klasifikace s tímto algoritmem bude nižší.

1	.5	0		1.5		0.5
0	1	.2	→	1.2	→	0.4
1	1	1		3		1

Tabulka 1: Tvorba příznaku pomocí histogramu

3.2.2 Přetvarování do vektoru

Tato metoda obrázek v maticové (2D) formě převede na vektor tak, že jednotlivé řádky pixelů obrázku zanese za sebe do vektoru. Jak je vidět v tabulce 2, která reprezentuje obrázek ve známé maticové podobě. Následně v tabulce 3 vidíme seřazené hodnoty z maticové podoby po řádcích.

1	2	3
4	5	6
7	8	9

Tabulka 2: Obrázek před převodem

1	2	3	4	5	6	7	8	9

Tabulka 3: Obrázek po převodu

Je zřejmé, že tedy tato metoda nijak neupravuje výchozí obrázek, resp. obrázek lze z této podoby opět složit. Výsledný vektor tedy bude mít velikost násobku šířky a výšky obrázku. Následně lze také říct, že tento způsob bude výpočetně náročnější, protože je zde více prvků ve vektoru, ale určitě bude přesnější než u histogramu. Následně tuto metodu budu dále v dokumentu nazývat jako „data“ nebo „datová reprezentace“;

3.3 Klasifikátory

Jsou implementovány klasifikátory minimální vzdálenosti (MD), k nejbližších sousedů (KNN) a navíc experimentální klasifikátor.

3.3.1 MD klasifikátor

Klasifikátor nejmenší vzdálenosti spočívá v nalezení třídního bodu, který reprezentuje celou třídu. Pozice tohoto reprezentanta může být vypočtena například zprůměrováním

pozic všech bodů v dané klasifikační třídě. Následně se při klasifikaci hledá, který z těchto reprezentantů je nejblíže a výsledkem je poté právě třída nejbližšího reprezentanta. Tato klasifikace neaproximuje tvar pozic bodů ideálně, proto předpokládám, že úspěšnost této metody bude nízká.

3.3.2 KNN klasifikátor

Klasifikátor k nejbližších sousedů spočítá v nalezení k nejbližších bodů. Následně je prohlášeno, že klasifikovaný příznakový vektor spadá do klasifikační třídy, která je nejčetnější v k nejbližších bodech. Jelikož se nejbližší sousedi hledají ze všech bodů, předpokládám že tato klasifikace bude nejpomalejší, ale přesnější.

3.3.3 Experimentální klasifikátor

Pokusil jsem se implementovat i vlastní klasifikátor, jehož algoritmus nyní popíši. Algoritmus nalezne centrální bod (reprezentanta) stejně jako u klasifikátoru minimální vzdálenosti. Následně je vytvořeno dalších dvakrát velikost příznaku příznaků tak, že je vždy ke každé složce příznakového vektoru reprezentanta přičtena a odečtena jednička. Následně je nad těmito body spuštěna shlukovací metoda K-Means a přebytečné příznaky jsou odstraněny. Tímto tedy vznikne více reprezentantů, což lépe aproximuje tvar dané třídy a zároveň je redukovaný počet příznaků ve kterých se hledá oproti KNN klasifikaci. Díky tomu si myslím, že klasifikace bude rychlejší než u KNN a bude přesnější než u klasifikace MD.

4 Popis řešení

Aplikace je realizována v programovacím jazyce Java a je rozdělena do tříd podle účelu funkcí.

4.1 Grafické rozhraní

Grafické řešení je řešeno knihovnami `javax.swing` a `java.awt` a zajišťují tak chod okenní aplikace. Grafické rozhraní obsahuje kreslící plátno, tlačítko pro zahájení klasifikace a výstupní textový popisek. Implementace je provedena ve třídě `Window`

Kreslící plátno je zobrazený obrázek o velikosti 28x28 pixelu roztažený na potřebnou velikost. Při přejíždění myší se stisknutým levým tlačítkem myši se pixely na plátně (obrázku) obarvují na odstíny šedé. O kreslící plátno se stará třída `Drawing`.

4.2 Příznaky

Samostatné příznakové vektory jsou v programu řešeny pomocí pole typu float. Byl zvolen datový typ float, protože hodnoty jsou omezeny na hodnoty mezi 0 a 1 a často tedy mají dlouhý desetinný rozvoj.

Aby bylo možno přiřadit k příznakovým vektorům i nějaké další informace, byla vytvořena třída `SymptomPoint` návrhového vzoru přepravka. Kromě příznakového vektoru může obsahovat i klasifikační třídu, ke které je bod přiřazen a případně nějaké další informace v proměnné `meta` například vzdálenost k nějakému vektoru.

Jednotlivé příznakové vektory jsou vypočítávány instancemi třídy `Symptom`. Tato třída je abstraktní a pouze definuje jaké metody musí jejich potomek mít. Je tak vyžádáno, aby potomci měli metody pro trénování, načtení, uložení a klasifikaci. Tato třída také obsahuje metody pro vypočtení euklidovské vzdálenosti mezi dvěma vektory a vygenerování nového náhodného příznakového vektoru. Tato třída má poté dva potomky. Třídy `ImageDataSymptom` a `HistogramSymptom`, kde každá z nich implementuje svůj způsob tvoření příznakových vektorů dle návrhu řešení.

4.3 Klasifikátory

4.3.1 MD klasifikátor

Klasifikátor je implementován tak, že nejprve spustí načítání trénovacích dat podle aktuálně zpracovávaného čísla. Z dat získá příznaky a ty sčítá do jednoho vektoru. Nakonec je tento vektor vydělen počtem sčítaných vektorů (provede se aritmetický průměr) a tento vektor uloží mezi reprezentanty.

Nakonec tedy klasifikátor má 10 reprezentantů, mezi kterými při klasifikaci hledá nejbližší prvek. Klasifikační třída nejbližšího reprezentanta je teda předána jako výsledek klasifikace.

4.3.2 KNN klasifikátor

Algoritmus je implementován tak, že se spustí načítání trénovacích dat, ty převede na příznakové vektory a následně uloženy do kolekce. Při klasifikaci příznakového vektoru se vždy tato kolekce seřadí tak, aby na vrchu byly nejbližší prvky. Algoritmus poté zjistí počty jednotlivých klasifikačních tříd u pěti (k je zvoleno 5) prvních prvků a první nejpočetnější klasifikační třída je navrácena jako výsledek klasifikace.

4.3.3 Experimentální klasifikátor

Experimentální klasifikátor je implementován následovně. Jsou načteny všechny trénovací data podle jednotlivých tříd a jsou převedeny na příznakové vektory. Klasifikátor následně nalezne obdobně jako u MD klasifikace příslušného reprezentanta. Následně je od vektoru reprezentanta vytvořeno několik dalších vektorů tak, že ke každé složce je vždy přičtena a odečtena hodnota 1. Reprezentanta poté můžeme „vyhodit“ a spustíme shlukovací algoritmus K-Means nad vytvořenými vektory. Po ustálení nebo po pěti (aby generace modelu netrvala příliš dlouho) krocích se odstraní vektory bez přiřazených bodů a zbytek bodů je uložen jako reprezentanti.

4.4 Ukládání a načítání

Struktura souboru začíná hlavičkou, kde je uvedeno pro jaký klasifikační a parametrizační je model určen. To je zejména důležité při načítání souboru, protože při načítání souboru uživatel nezadává pro jaké algoritmy je model určen. Po hlavičce následuje výpis vektorů, kdy na prvním místě je třídní příslušnost vektoru a následně pokračují jednotlivé hodnoty složek vektoru.

Ukládání a načítání probíhá skrz třídu `Main`, která načte/uloží hlavičku a následně volá příslušné metodu přímo v klasifikátorech.

4.5 Výsledky

Výsledky ohodnocení jednotlivých klasifikátorů za použití jednotlivých parametrizačních algoritmů jsou vloženy do tabulky 4.

	MD		KNN		EXPERIMENTAL	
	Histogram	Data	Histogram	Data	Histogram	Data
0	58.878%	90.102%	93.469%	99.490%	75.306%	98.673%
1	92.775%	96.564%	98.326%	99.736%	98.062%	99.648%
2	58.236%	77.422%	81.395%	97.093%	73.547%	96.899%
3	52.277%	81.782%	68.713%	96.931%	49.703%	96.139%
4	73.829%	81.670%	92.872%	96.334%	86.965%	96.334%
5	16.368%	64.574%	44.170%	96.749%	40.919%	96.413%
6	82.046%	86.430%	93.006%	98.852%	86.848%	98.121%
7	75.389%	83.463%	91.926%	96.595%	85.992%	96.109%
8	44.045%	76.386%	75.975%	95.072%	69.097%	95.688%
9	73.241%	81.665%	87.512%	95.738%	81.962%	95.144%
0-9	63.59%	82.36%	83.31%	97.29%	75.47%	96.95%

Tabulka 4: Úspěšnost v procentech pro jednotlivá čísla a jednotlivé kombinace algoritmů

Základním poznatkem vidíme, že klasifikace minimální vzdálenost dopadla nejhůře a nejlépe obstál klasifikátor k nejbližších sousedů. Také je patrné, že datová reprezentace měla výrazně vyšší úspěšnost oproti histogramu tak, jak bylo předpokládáno v návrhu řešení. Vidíme také, že klasifikátory nejvíce chybují při klasifikaci cifry „5“, která zejména (ale ne výlučně) v případě použití reprezentace histogramu zaznamenává výrazné poklesy v úspěšnosti dokonce až na 16.368%. Za to nejlépe si vede klasifikace cifry „1“, která v případě KNN dosahuje až na 99.736% úspěšnost.

5 Uživatelská dokumentace

5.1 Předpoklady

Pro spuštění programu je potřeba, aby na klientském počítači byla nainstalované prostředí pro spuštění Java aplikací. Minimální verze Java prostředí je verze 58.0 (Java 14)!

5.2 Vytvoření a ohodnocení klasifikátoru

Vytvoření a ohodnocení klasifikátoru můžeme dosáhnout zadáním všech pěti parametrů při spuštění programu. Program se tedy spustí příkazem
`java -jar UIR-SP.jar trénovací_data testovací_data parametrizační_algoritmus klasifikační_algoritmus název_modelu`,
kde jednotlivé parametry znamenají:

- **trénovací_data** cesta ke složce s trénovacími daty
- **testovací_data** cesta ke složce s testovacími daty

- **parametrizační_algoritmus** identifikátor algoritmu, který se má použít pro tvorbu příznaků (možné hodnoty: `DATA`, `HISTOGRAM`)
- **klasifikační_algoritmus** identifikátor algoritmu, který se má použít pro klasifikaci (možné hodnoty: `KNN`, `MD`, `EXPERIMENTAL`)
- **název_modelu** cesta pro uložení výsledného modelu

Po takovém spuštění programu se začne klasifikátor trénovat na trénovacích datech, poté uloží model do souboru a následně spustí testování. Průběh událostí je vypisován do příkazové řádky včetně výsledků ohodnocení.

5.3 Použití klasifikátoru v GUI

Pro spuštění aplikace s existujícím modelem stačí zadat příkaz do příkazové
`java -jar UIR-SP.jar název_modelu`,
kde `název_modelu` je cesta k souboru modelu.

Po načtení dat ze souboru aplikace spustí jednoduché GUI, kde lze „psát“ na kreslící plátno a nechat poté nakreslenou cifru klasifikovat. Výsledek klasifikace se zobrazí pod tlačítkem „Klasifikovat“.

5.4 Použití textového rozhraní

Při spuštění aplikace bez parametrů, spustí se textové ovládání přes které lze aplikaci ovládat. Tato funkce je přívětivější pro uživatele, kteří neví jak a jaké parametry mají být zadány.

Pro spuštění tohoto režimu stačí spustit aplikaci následujícím příkazem z příkazové řádky:
`java -jar UIR-SP.jar`

6 Závěr

Výsledná aplikace při zvolení určitých algoritmů klasifikuje vskutku obstojně a výsledky potvrzují předpokládané ohodnocení klasifikátoru zmíněné v návrhu řešení. Klasifikace je zejména při použití klasifikátoru KNN pomalejší, ale to lze vysvětlit řazením obsáhlé kolekce prvků.

Při ručním kreslení cifer myší v GUI se mi často stává, že aplikace špatně klasifikuje nakreslenou cifru, dávám to za chybu jednak mé schopnosti napodobit cifry z trénovacích dat a pravděpodobně i chybějícímu předzpracování dat ať už trénovacích, nebo těch nakreslených.

Pokud bych měl zmínit věc, kterou bych případně ještě dodal je v GUI, kdy by bylo dobré ukazovat popis, zda právě probíhá nebo neprobíhá klasifikace. To by eliminovalo zmatení, kdy klasifikujeme dvě stejná čísla za sebou a nevíme, zda se již výsledek změnil nebo při použití klasifikátoru, který vyžaduje delší čas na klasifikaci.

Jinak zadání se mi líbilo a myslím, že vedlo k určitému poznání a určitě se vyjímá svou zajímavostí.