

1 Úvod

Bude se pracovat v Matlabu, v moodle je skupina.

Poznámka (Úkoly počítačového vidění)

Detekovat, najít a určit věci (tváře, jestli se smějí, nádory, znaky jako znaky na SPZ, biometrika jako oko, tvář, podpis, budovy, lidi, auta, ...) na obrázku

Poznámka (Rozpoznávání objektu tradičním náhledem)

pixels -> určení feature (např. rozdělení oblastí podle tvarů, důležitých bodů, barev, umístění hledaného objektu) expertem -> učení klasifikátoru -> rozpoznávání

Definice 1.1 (Feature vektor)

Univerzální převedení obrázku do důležitých věcí.

Měl by být invariantní (měl by být stejný při rotaci a škálování), diskriminační (dobře rozdělovat objekty), kompaktní (co nejmenší)

Definice 1.2 (Rozpoznávání)

Feature vektory tvoří prostor, kde se algoritmus naučí najít hranici, která odděluje objekty, co jsou nějaké a co jsou jinaké.

Poznámka (Klasifikace může být za pomoci)

Statistiky – Bayesova teorie rozhodování

Pravidel – Rozhodovací strom

Metriky – Technika nejbližšího souseda, diskriminační analýza?, podpůrné vektorové stroje?

Biologické inspirace – Neuronové sítě

Definice 1.3 (Učení s učitelem)

Na training setu víme správné odpovědi.

Definice 1.4 (Naivní Bayesův klasifikátor)

Vychází z podmíněné pravděpodobnosti na základě věcí, co víme.

┌

Například

Rozeznávání falešného úsměvu. 91,3%.

└

Definice 1.5 (Rozhodovací stromy)

Pravidly určíme, kterou větví se vydáme. Výhodou je, že nepotřebujeme koncept vzdálenosti.

Například

Rozpoznávání, co se děje na videu (např. vražda). 70% - 100%.

Definice 1.6 (K nejbližších sousedů)

Podíváme se na nejbližší známé objekty a rozhodneme se podle nich.

Například

Čtení znaků. 99% čísla, 94% velká a 89% malá písmena.

Definice 1.7 (Lineární klasifikace)

Rozdělení prostoru nadrovinou. Zlepšením je tzv. podpůrné vektorové stroje? (support vector machines)

Například

Rozpoznávání lidí a věku. 66,9 - 80% lidí, 63,8 - 75,7% věk.

Definice 1.8 (Umělé neuronové sítě)

Sítě z neuronů, které jsou velmi jednoduše simulovány, viz moje maturitní práce. (Na GitHubu pod uživatelem JoHavel).

Například

Rozpoznávání tváře. 90%. (80% na portrétech.)

Poznámka (Hluboké učení)

pixels -> učení se včetně feature -> rozpoznávání

Například

AlexNet (top 5 error cca. 16%)

Každoročně se pořádá ILSVRC (Imagenet Large Scale Visual Recognition Challenge), kde už se dosáhlo méně než 4% chyby (152 vrstev NN)...

Poznámka (Kombinovaný přístup)

pixely \rightarrow featury nalezené NN \rightarrow trénování klasifikátoru \rightarrow rozpoznávání

Hluboké učení se ale zdá účinnější.

Definice 1.9 (Klasifikační pipeline)

Features \rightarrow Výběr featur, jejich normalizace, ... \rightarrow klasifikace (výběr klasifikátoru, trénování klasifikátoru a následná klasifikace) \rightarrow evaluace \rightarrow features (respektive výstup, pokud jsme spokojeni).

Definice 1.10 (Features)

Vlastnosti objektu, spojují nějakým způsobem podobné objekty. Musí být diskriminativní (pokud nejsou dostatečně diskriminativní, jakože často nejsou, dá se ještě hledat rozdělení s nejmenší chybou). Měly by být kompaktní (co nejmenší, protože s příliš featurami nelze ve stejném čase dostatečně naučit klasifikátor).

Definice 1.11 (Normalizace feature)

Aby nenastával problém např. s odlišnými jednotkami, nebo s různě naškálovanými featury, normalizuje se vydělením referenční hodnotou, respektive různými statistickými metodami, např. standardizací ($\tilde{x}_i = \frac{x_i - \mu}{\sigma}$), nebo 3σ škálováním ($\tilde{x}_i = \frac{x_i - \mu}{3\sigma} + 1$).

Poznámka

Rozhodovací stromy (a náhodné lesy), naivní bayesova metoda atd. nepotřebují normalizaci.

Definice 1.12 (Výběr featur)

Některé featury mohou být totožné, některé zas redundantní.

Takže vybereme nějakou podmnožinu featur a vyzkoušíme. Nebo se naopak podíváme na jednotlivé, ohodnotíme je a vybereme K nejlepších. Další možnost je přidávat je po jedné a testovat je ne samotné, ale s již vybranými. Nebo můžeme začít se všemi a odstraňovat nejhorší (zase oběma způsoby, sekvenčním i jednokrokovým K). Existuje i kombinovaný, který udělá nejdříve jedno a pak druhé. Pak existují i genetické a další algoritmy.

Podle čeho měřit: konzistence (jestli shodné hodnoty jsou ve shodné třídě, viz vzorec v prezentaci), nezávislost na ostatních featurách (opak tzv. korelace) + korelace s třídami, množství informace ($\mathcal{I} = -\log(P(A = a_i))$, $E(\mathcal{I}) = -\sum P(A = a) \cdot \log_2(P(A = a))$), co nám dá, vzdálenost mezi třídami po použití dané featury, ...

Definice 1.13 (Transformace featur)

Unsupervised (minimalizována je ztráta informací): Principal Component Analysis (PCA), Latent Semantic Indexing (LSI), Independent Component Analysis (ICA), ...

Supervised (maximalizuje se vzdálenost mezi třídami): Linear Discriminant Analysis (LDA), Canonical Correlation Analysis (CCA), Partial Least Squares (PLS), ...

Definice 1.14 (PCA)

Také Kaurhunen-Loeve (K-L) method. Hledá v rotacích a deformacích os největší varianci ($b_1^T \Sigma b_1$, kde b_1 je vektor projekce, Σ je matice kovariance). Maximum se najde Lagrangeovy multiplikátory jako místo, kde $b_1^T \Sigma b_1 = \lambda$, kde λ je vlastní číslo Σ , je největší. Pro druhý vektor dostaneme totéž.

$$\begin{aligned}\Sigma &= \frac{1}{N} X X^T (X = \text{rozdíl od průměru}) \\ \Sigma b_j &= \lambda b_j (b_j = \text{vlastní vektory}) \\ X' &= B^T X (B = [b_j])\end{aligned}$$

Definice 1.15 (SVD (singular value decomposition))

Hledáme USV^T tak, že U a V jsou vlastní vektory $A^T A$ a AA^T a V^T je poté diagonální matice vlastních čísel.

$$\begin{aligned}Y &= \frac{1}{\sqrt{N}} X^T \implies Y^T Y = \Sigma \\ Y &= USV^T \\ V &= \text{vlastní čísla matice } Y^T Y = \Sigma\end{aligned}$$

Definice 1.16 (ICA)

Báze nejsou kolmé. Je potřeba aby data byla nezávislá, tedy vycentrujeme odečtením průměru, a vyčistíme tím, že vynásobíme odmocninami vlastních čísel. Počítá se přes entropii (viz prezentace).

Poznámka

Unsupervised transformace featur může vést ke ztrátě oddělení tříd.

Definice 1.17 (LDA)

Snažíme se dostat průměr každé třídy co nejdále od průměru všeho.

Hodně vzorců, viz prezentace. Hledáme tolikarozměrnou projekční nadrovinu (v pre-

zentaci její báze označena w^T), kolik máme tříd - 1.

Poznámka

Lze použít nejdříve LDA, abychom snížili rozměry a následně použít LDA.

Poznámka

Na LDA potřebujeme hodně tréninkových dat. I při hodně dat jsou situace, kdy PCA je lepší (hlavně, když se třídy překrývají).

Definice 1.18 (Rozhodovací hranice (decision boundaries))

Rozděluje prostor na jednotlivé třídy.

Cílem je najít rozhodovací funkci, která bude správně přiřazovat třídy, nebo najít funkci rozumně definující hranice tříd.

Definice 1.19 (Pravděpodobnostní klasifikace)

Mějme M tříd $\{\omega_i\}_{i=1}^M$. Potom pravděpodobnost, že objekt patří do třídy ω_i , za předpokladu, že vektor je \mathbf{x} , je

$$P(\omega_i|\mathbf{x}) = \frac{P(\mathbf{x}|\omega_i) \cdot P(\omega_i)}{\sum_{j=1}^M P(\mathbf{x}|\omega_j) \cdot P(\omega_j)}.$$

Objekt tedy zařadíme do nejpravděpodobnější třídy. Jelikož jmenovatel je všude stejný, tak počítáme

$$\omega_i, \text{ kde } i = \arg \max_j P(\mathbf{x}|\omega_j) \cdot P(\omega_j).$$

Hodí se zvláště, když matice kovariance je diagonální (třídy nejsou kovariantní).

Definice 1.20 (Naivní Bayesův klasifikátor)

Předpokládáme, že každá souřadnice feature vektoru je nezávislá:

$$P(x_1, x_2, \dots, x_D|\omega_i) = P(x_1|\omega_i) \cdot P(x_2|\omega_i) \cdot \dots \cdot P(x_D|\omega_i).$$

Tedy vytvoříme frekvenční tabulku tříd v závislosti na jednotlivých dimenzích a spočítáme pravděpodobnosti. Může se nám ale stát, že při mnoha dimenzích už bude násobení tolika pravděpodobností mimo přesnost, proto se používají logaritmy pravděpodobností.

Navíc pokud nemáme dostatek dat, snadno se nám může (na „nespojité dimenzi“) stát, že pravděpodobnost nějaké třídy bude nula, tedy „přidáme“ 1 ke všem výskytům ve frekvenční tabulce.

Abychom odhadli pravděpodobnost „spojité proměnné“, používáme standardní funkce

hustoty, tedy např. (\bar{x}_{ij} je průměr vzorku, σ_{ij}^2 je variance vzorku):

$$P(x_i = x | \omega_j) = \frac{1}{\sigma_{ij} \sqrt{2\pi}} e^{\frac{-(x - \bar{x}_{ij})^2}{2\sigma_{ij}^2}}.$$

Pokud máme rozložení, kde jsou třeba 2 „hrby“, můžeme rozdělit prvky do „podtříd“ a rovnici výše doplnit o sumu a váhy, které se sečtou na 1 (GMM):

$$P(x_i = x | \omega_j) = \sum_m \frac{w_m}{\sigma_{ijm} \sqrt{2\pi}} e^{\frac{-(x - \bar{x}_{ijm})^2}{2\sigma_{ijm}^2}}.$$

Na spočítání rozdělení do podtříd a vah se používá metoda EM (expectation, Maximization), kde se několikrát zopakuje rozdělení na základě aktuálních vah (v podstatě klasifikaci) a updatování průměru a variance.

2 Hodnocení klasifikátoru

Definice 2.1 (Matice zaměňování - binární klasifikace)

4 hodnoty TP, FN, FP, TN (True / False = klasifikováno správně nebo špatně, Positive / Negative = klasifikováno pozitivně nebo negativně).

$Accuracy = \frac{TP+TN}{P+N}$ má problém, když není dataset rovnoměrný (např. klasifikátor klasifikuje vždy negativně a v setu je jen 1 pozitivní, tak je accuracy 99%). Místo toho lze použít $Errorrate = 1 - Accuracy$ a jiné „rates“, např. $Recall = \frac{TP}{P}$ a $Precision = \frac{TP}{TP+FP}$ (používají se pro sledování thresholdu v grafu křivky Recall – Precision), $truepositivrate = TPR = \frac{TP}{P}$ a $FPR = \frac{FP}{N}$ (dohromady tzv. ROC křivka).

Definice 2.2 (F skóre)

Zkombinovaný recall a precision, má 1 parametr (t je threshold)

$$F_\beta(t) = \frac{(1 + \beta^2)p(t)r(t)}{r(t) + \beta^2 p(t)}.$$

Pro balancovaný model chceme maximalizovat F_1 .

Definice 2.3 (Loss funkce)

Znázorňuje cenu za špatné rozhodnutí.

Zero-one loss je standardní loss funkce (1, když klasifikoval špatně, 0 když dobře).

Definice 2.4 (Confidence Intervals)

Když spočítáme chybu, tak můžeme ještě odhadnout skutečnou chybu jako

$$error(h|X) \pm z_{1-\alpha/2} \sqrt{\frac{error(H|X)(1 - error(h|X))}{n}},$$

kde $z_{1-\alpha/2}$ je takzvaná kritická hodnota statistického rozložení, kterou najdeme v tabulkách (často se používá hodnota 1,960 odpovídající přesnosti 95%).

3 Testování

Definice 3.1

Data rozdělujeme minimálně na 3 množiny: trénovací data (tréníng), validovací data (nastavení parametrů) a testovací data (testování, MUSÍ být disjunktní s ostatními množinami).

Pokud máme malou nebo nevyrovnanou množinu dat, rozdělení nemusí být reprezentativní, tedy se často generují podmnožiny tak, aby měli reprezentativní zastoupení tříd.

Definice 3.2 (Cross validace)

Chyba se nepočítá na jedné množině testovacích dat, ale na mnoha jejích podmnožinách a dělá se průměr.

Tyto podmnožiny dat můžeme získat buď náhodným výběrem (objekt může být zvolen vícekrát) nebo při tzv. K-fold cross validaci prostě rozdělit trénovací data na K shodně velkých podmnožin (ty zas nemají reprezentativní rozdělení v každé množině). Úprava Stratified K-fold cross validation vybírá K shodně velkých podmnožin, aby třídy byly zastoupeny reprezentativně, úprava „leave one out“ volí $K = N$.

Nejčastěji se volí $K=10$, nebo pro velké sety $K=3$, nebo pro malé $K=N$.

Definice 3.3 (Bootstrap)

Vybereme náhodně (klidně opakovaně) data na trénování a pak ty, co jsme nevybrali použijeme na testování. V trénovacích datech se objeví okolo 63,2% dat.

4 Porovnávání modelů

Definice 4.1 (Porovnávání)

Modely (někdy též hypotézy) můžeme porovnávat pomocí intervalu chyby, nebo třeba tzv. McNemarův test, kde se spočítají hodnoty kontingenční matice a následně se dosadí do vzorce.

5 Další klasifikátory

Definice 5.1 (K nejbližších sousedů)

Pomocí bayesovy věty dojdeme k vzorci $p(\omega_i|\mathbf{x}) = \frac{K_i}{K}$. Učení probíhá jednoduše, prostě si zapamatujeme data. Klasifikace probíhá také jednoduše, spočítáme nejbližších K objektů a klasifikujeme jako nejvíce zastoupenou třídu.

Jediné, co můžeme nastavovat, je K (menší méně odhalí šum, ale má detailnější hranice, větší naopak přestává být senzitivní), a jakou metriku použijeme.

Důležité je standardizovat data. Druhá možnost je použít vážených K nejbližších sousedů, kde každý soused (podle vzdálenosti) bude mít jinou váhu. Nebo (a to i v případě, že jsou souřadnice „deformovány jinak“) můžeme použít metriku s parametry a používat tzv. adaptivní nejbližší sousedé, kde v každé části prostoru bude metrika vypadat jinak.

Kategorická data se pro K nejbližších sousedů buď musí převést na číselná, použít speciální metriky, nebo vektorizovat (na nuly a jedničky, dimenze pro každou kategorii).

Data se dají trochu upravovat. Kromě normalizace ještě můžeme odstranit data, která by KNS klasifikoval „velmi“ špatně (mají hodně sousedů jiné třídy).

5.1 Rozhodovací stromy

Definice 5.2 (Rozhodovací stromy)

Výhoda = nemusíme zavádět vzdálenosti (= můžeme klasifikovat i nominální data = data, u kterých nemá vzdálenost smysl).

V každém nelistovém vrcholu (uzel) je tzv. test, větve reprezentují výsledky testu, v listech jsou klasifikační třídy.

Bude nás zajímat, jaké všechny testy můžeme vykonat, jaký test je v každém uzlu použit a jestli je nutné ještě nějaký používat. Snažíme se snížit nečistot (množství tříd) podmnožin, na které dělíme v daném uzlu.

Každý strom lze převést na binární, tedy budeme používat pouze binární rozhodovací stromy, tedy odpověď testu bude YES / NO. Většinou (v podstatě vždy) se ptáme jen na jeden příznak, protože vybírat a následně testovat podmnožinu je těžké.

Definice 5.3 (Měření „nečistosti“)

Chceme, aby byla minimální, když obsahuje právě jednu třídu, naopak pokud každou třídu obsahuje ve stejném poměru a měla by být symetrická vůči výměně tříd.

Optimalizaci stromu následně budeme dělat maximalizováním úbytku nečistosti (vážený

průměr (přes velikosti) podmnožin ve větvích):

$$\Delta I(S, X_i) = I(S) - \sum_j \frac{|S_{ij}|}{|S|} I(S_{ij})$$

Existuje několik takových funkcí, například Gini-index (jak moc často bude náhodný objekt špatně klasifikován, když klasifikujeme náhodně podle pravděpodobnost tříd):

$$p_i = \frac{|\{X|X \in \omega_i\}|}{|x_t|}, GI = \sum_{i=1}^m p_i(1 - p_i)$$

Další možností je používat entropii (tzv. IG = information gained).

Definice 5.4 (Kdy prohlásit daný vrchol za list)

Pokud nečistota je menší než nějaký threshold, pokud je velikost množiny dostatečně malá, pokud je vrchol čistý (moc velké stromy, tzv. přeučení \implies nová data budou klasifikována špatně).

Listu následně bude přiřazena třída, která: má maximální pravděpodobnost v dané množině (nejčastější kritérium), ...

Například

Ukázáno na rozhodování, zda se bude něco hrát na základě počasí (teplota, vlhkost, vítr, ...) s IG.

Pozor

IG preferuje hodně hodnot, tedy buď musíme penalizovat rozdělení do více podmnožin (např. vydělením počtem podmnožin, pak zase ale favorizuje nebalancované rozdělení), nebo povolit pouze binární dělení (zkontrolujeme všechny možné dvojice, resp. všechny thresholdy = prahy, v případě číselných hodnot).

Definice 5.5 (Ořezávání = pruning)

Lze ořezávat za běhu (pre-pruning = když trénuji), nebo po přeučení (post-pruning).

Metoda ořezávání C4.5: veeeeeeelký vzorec v prezentaci (spočítá zlepšení rozdělení = zmenšení vážené (velikostí podmnožin) chyby, když uděláme nejlepší test v daném uzlu, pokud se příliš nezmění, nedejbože zhorší, tak místo testu bude uzel list).

Poznámka (Plusy a mínusy)

+: Jednoduché, málo přípravy dat, indikuje, co je nejlepší pro rozdělení.

-: NP-úplný problém, pracuje špatně pro mnoho tříd, dlouhé učení, nebezpečí přeučení.

Definice 5.6 (Náhodný les (Breiman 2001))

Každý strom trénujeme na stejných datech, ale testy vybíráme z náhodné podmnožiny / příznak vyrobíme jako náhodnou kombinaci původních příznaků. Klasifikace pak probíhá, že uděláme klasifikaci podle každého stromu a vezmeme nejčastější.

5.2 Lineární klasifikátor

Definice 5.7 (Lineární klasifikátor)

Hledáme nadrovinu, která prostor rozdělí na 2 třídy (podmínkou je, že musí existovat / viz dále).

Rovina je dána normálovým vektorem $\mathbf{w}^T \mathbf{x} + b = 0$. Rozhodování je pak dáno tím, zda je $\mathbf{w}^T \mathbf{x} + b$ kladné, nebo záporné. Často to uděláme tak, že přidáme dimenzi a rovina bude procházet počátkem. Dalším trikem je jednu třídu vynásobit -1 a následně máme jen jednu podmínku.

Hledání lineárního klasifikátoru se dělá gradientovou metodou. Tedy iterativně: vybereme náhodný vektor a potom vždy spočítáme gradient účelové funkce, následně od vektoru odečteme learning rate krát gradient a opakujeme počítání gradientu a odčítání. Z Taylorova rozvoje navíc můžeme najít přibližně nejlepší learning rate.

Definice 5.8 (Účelové funkce)

Nejjednodušší je spočítat, kolik dat jsme klasifikovali špatně krát jak moc špatně $O_P = \sum_{z \in Z} -\mathbf{u}^T \mathbf{z}$ (Z je množina špatně klasifikovaných). Není spojitá, ale dá se použít. Takovéto funkce se říká Piece-wise...

Aby byla spojitá, dá se použít tzv. kvadratická funkce $O_Q = \sum_{z \in Z} (\mathbf{u}^T \mathbf{z})^2$. Ta však často konverguje k tomu, že budou data přesně na hranici.

Definice 5.9 (Sekvenční učení)

Pokud víme, že daná rovina existuje, tak vezmeme vektor a následně od něj budeme sekvenčně odečítat špatně klasifikované objekty.

TODO

Definice 5.10 (Metoda pomocných vektorů)

Místo abychom hledali rozdělení prostoru nadrovinou, hledáme rozdělení „tlustou“ nadrovinou (budeme se snažit najít rozdělující rovinu tak, aby objekty byly od této roviny co nejdále).

V metodě hladkých okrajů navíc budeme ignorovat špatně klasifikované objekty blízko

(blízko nastavíme na validačních datech) nadroviny.

Definice 5.11 (Nelineární SVM)

Převeďme data nějakým způsobem (např. na kužel při kruhových třídách) do vícerozměrného prostoru. Transformaci nemusíme hledat, hledáme jen jádro (skalární součin transformovaných dat).

Klasická jádra: Polynomické $K(x, y) = (a + x^T y)^p$, gaussovská $K(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right)$.

6 Kombinované modely (ensemble methods)

Definice 6.1

Dá se kombinovat homogenně (stejný klasifikátor s různými parametry / daty / začátkem) nebo homogenně (různé klasifikátory).

Dá se kombinovat průměrem, nebo hlasováním.

Prezentace, tohle jsem nějak nezvládl zapisovat. Ve zkratce, buď se můžou klasifikátory na ensemble trénovat úplně náhodně, nebo postupně s přidáním dat, na kterých se předchozí sekly, nebo se natrénují náhodně a následně se trénuje meta klasifikátor na jejich klasifikování (bude z nich „vybírat“).

7 Neuronové sítě

Poznámka (Historie)

Už v roce 1943 McCulloch a Pitts vytvořili logický neuron, který měl práh, skokovou aktivizační funkci a skupinu inhibitorů (stačil jediný inhibitor a výstup byl 0, tzv. absolutní inhibice) a skupinu excitorů, všude je signál pouze 0 / 1.

Lze z nich složit libovolnou logickou funkci. Absolutní inhibice je navzájem záměnná s relativní inhibicí.

Nelze tyto sítě učit, pouze modelovat, proto vznikl v roce 1949 Hebbianovo učení (síla spojení by měla být podle toho, jak spolu dané neurony souvisí).

1957 Rosenblatt vytvořil tzv. perceptron (= lineární klasifikátor, jak jsme se ho učili), který fungoval až do roku 1969 (Golden Age), kdy Minsky ukázal, že jedním perceptronem nelze řešit neseparabilní data (XOR-problem). Pro více vrstvou síť však neexistoval učící algoritmus (Dark Age = „AI Winter“).

Ten byl vytvářen od 60. let, ale teprve od 80. byl zpopularizován. Tzv. backpropagation. Od roku 1995 existují Support Vector Machines, které umí totéž, ale jednodušeji. 2006

vznikla teorie Deep Neural Network, které se používají dodnes jako jeden z nejlepších modelů.

Definice 7.1 (Umělé neuronové sítě)

My se budeme zabývat pouze tzv. feed forward topologií, která má vrstvy a hrany jdou pouze z vrstvy do další v pořadí.

Poznámka (Kdy přestat učit?)

Nastavit maximální počet, nějaký cíl, nebo když dosáhneme minima chybové funkce, to však bude mít za následek overfitting, kterému ale můžeme předejít tím, že použijeme validační data a budeme sledovat chybovou funkci na nich (neznámých datech) místo testovacích.

Poznámka

Počet vrstev: 0 = lineární klasifikátor, 1 = z libovolného konečného prostoru do libovolného konečného, 2 = libovolná hranice v prostoru, ...

Počet neuronů ve skryté vrstvě by měl být mezi počtem ve vstupní a počtem ve výstupní (nejvýše dvojnásobek výstupní), třeba $2/3$ vstupní + výstupní.

Definice 7.2 (Konvoluční NN)

Na každý „čtverec“ dat použijeme nějaký filtr, který z něho vypočítá 1 hodnotu. (Respektive z každé vrstvy filtru jednu hodnotu, tj např. $32 \times 32 \times 3 \xrightarrow{5 \times 5 \times 3} 30 \times 30 \times 3$.)

Hodnoty mimo data můžeme doplnit, nebo používat filtr jen tam, kam se „vejde“. Používá se i tzv. stride = kde bude následující filtr.

Na výsledky se ještě většinou použije ReLU ($f(x) = \max(0, x)$), nebo její modifikace Leaky ReLU, ELU, SReLU (aby se neztrácely záporné hodnoty);

Konvoluční vrstvy se označují feature extraktor a dají se použít nejen s touto neuronovou sítí, ale můžeme použít (natrénovat na něm) tento extraktor s jiným klasifikátorem / nacvičit na nich jinou NN.

8 Učení bez učitele

Definice 8.1 (Clustering)

Klasifikujeme na základě blízkosti ve feature space.

Můžeme buď hierarchicky (do menších a menších množin), nebo nehierarchicky (jen na jednu skupinu tříd).

Přístup sdola / shora = spojujeme prvky do množin / rozdělujeme množinu prvků na

menší.

Hard / soft = jeden objekt patří vždy do jedné třídy / jeden prvek může patřit do více tříd.

Definice 8.2 (Dendrogram)

Graf („binární strom“) odpovídající hierarchii.

Definice 8.3 (Splitting Process of DIANA)

Iterujeme: vybereme nejodlišnější objekt od ostatních, oddělíme ho a následně k němu přidáme ty objekty, které jsou v naší metrice blíže jemu než ostatním.

Definice 8.4 (Principal Directions Divise Partition)

Zvolím osy dat jako při změně souřadnic a podle jedné z těchto os (té s „menším“ (ve smyslu projekce) průnikem s daty) to rozdělíme.

Existují vylepšení, kde tu osu posunut tak, aby lépe oddělovala (xx|yyzz místo xxy|yzz).

Definice 8.5 (Median cut)

Rozdělují v nejvíce variantní osu souřadnic, objekty rozdělím podle mediánu projekce na této osu.

Definice 8.6 (Agglomerative clustering)

Spojování aktuálně nejbližších clusterů.

Vzdálenost může být single-link (nejbližších prvků), complete-link (nejvzdálenějších), centroid-link (těžišť), average-link (průměrná vzdálenost).

Definice 8.7 (Ward's method)

Spojování aktuálně nejmenší ceny za spojení (třeba podle počtu členů krát vzdálenost) clusterů. Například:

$$MergeCost(A, B) = \frac{N_A \cdot N_B}{N_A + N_B} \cdot d(A, B)$$

Definice 8.8 (Iterative shrinking)

Iterujeme: vybereme shluk „nejlevnější“ (třeba jako v předchozím algoritmu) na rozdrobení (= rozdělíme tento shluk mezi okolní) a rozdrobíme.