I14: Klasifikátory a jejich učení - nejbližší soused, lineární klasifikátor, Adaboost, SVM, rozhodovací stromy, neuronové sítě. (RPZ)

Klasifikátory a jejich učení

"Objekt zájmu je charakterizován pozorovatelnými vlastnostmi $x \in X$ a svou příslušností ke třídě (nezsjistitelný/skrytý stav) $k \in K$. (X je prostor pozorování a K je množina skrytých stavů / tříd)"

Cílem klasifikátoru je najít strategii $q^*: X \to K$, která splňuje určité optimální vlastnosti.

Učení klasifikátorů - snižování risku.

Pokud známe rozdělení pravděpodobnosti, chceme snižovat Bayesovský risk:

$$R(q) = \sum_{\mathbf{x},k} p(\mathbf{x},k) W(q(\mathbf{x}),k)$$

Pokud neznáme pravděpodobnosti:

- 1. Předpokládáme, že p(x,k) má určitou formu/tvar (např. mix gaussiánů, apod.) s konečným (malým) počtem parametrů θ_k .
- 2. Chceme odhadnout (určit) tyto parametry z trénovací množiny T.

V tomto případě snižujeme empirický risk = R_{emp} = Chyba na trénovací množině

$$R_{emp}(q_{\theta}(\mathbf{x})) = \frac{1}{L} \sum_{i=1}^{L} W(q_{\theta}(\mathbf{x}_i), k_i) \underset{\rightarrow}{} \theta^* = \underset{\theta}{\operatorname{argmin}} R_{emp}(q_{\theta}(\mathbf{x}))$$

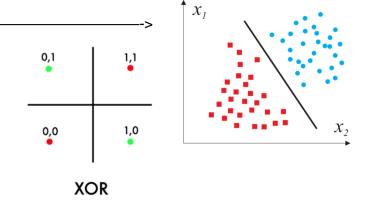
Minimalizace empirického risku může vést k overfitování.

(Problém generalizace je klíčový problém v rozpoznávání patternů. Malý R_{emp} ⊅ malý opravdový risk)

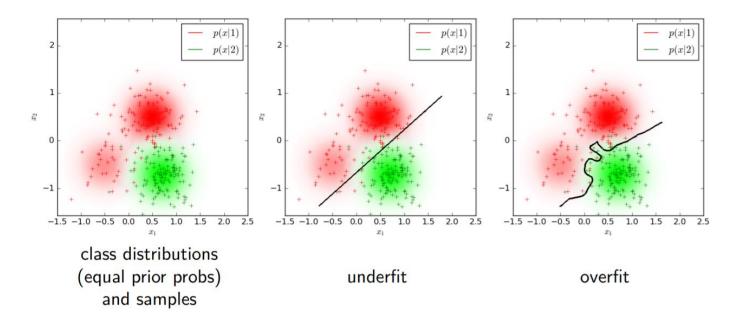
Další, kdo chce jít do hloubky: Vapnik and Chervonenkis inequality a VC dimenze

Lineárně separabilní (oddělitelná) data

- lineární klasifikátor je bezchybně oddělí: -
- XOR problém lineárně neseparabilní data nejde oddělit lineárním klasifikátorem



Přeučený klasifikátor vede k <u>overfitting</u>u (tedy když se natrénuje "příliš přesně" podle trénovacích dat):



Lineární klasifikátor

Vhodný pro lineárně oddělitelné množiny (ale vstupní data typicky obsahují chyby/šum a pak tento jednoduchý přístup není použitelný) – předpokládáme, že data jsou lineárně separovatelná (klasifikovatelná).

Lineární klasifikátor ... nadrovina – $g(\mathbf{x}) = \mathbf{w}^{\mathsf{T}}\mathbf{x} + w_0 = 0$ d ... dimenze vstupního prostoru, $\mathbf{w} = (w_1, ..., w_d)$... váhový vektor, w_0 ... práh rozdělení prostoru na dvě části: jedna strana $(g(\mathbf{x}) > 0)$ a druhá strana $(g(\mathbf{x}) < 0)$

Příklady lineární klasifikace: Perceptron, Logistická regrese, SVM, Bayesovský klasifikátor

Perceptron - cílem nalézt parametry $w_0, w_1, \dots w_d$

- Pro lineárně separabilní množiny existuje nadrovina w*T·x = 0:
 w*T·x > 0 pro každý x z ω₁ a w*T·x < 0 pro každý x z ω₂
- Zavádí se rozšířený (d+1)-dimenzionální prostor: w ≡ [w*, w₀], x ≡ [x,1]

$$\rightarrow$$
 pak $\mathbf{w}^{*T} \cdot \mathbf{x} + \mathbf{w}^{*}_{0} = \mathbf{w}^{T} \cdot \mathbf{x}'$

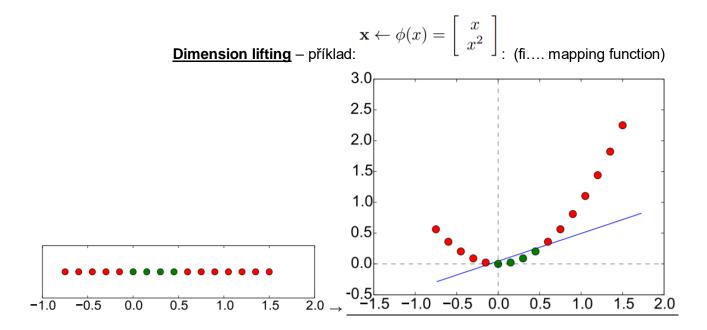
- Perceptronový algoritmus zkonverguje k řešení (nějakému) v konečném počtu kroků pokud jsou data lineárně separabilní, jinak pojede napořád
- Nalezená řešení jsou nejednoznačná → <u>existuje více</u> než jedna dělící nadrovina (nemusí být "nejlepší" - ve smyslu třeba max. šířky marginu (pásu) jako SVM)
- Algoritmus:
 - o inicializace: $\mathbf{x}' \equiv [\mathbf{x}_i 1] \rightarrow \mathbf{x}_i = \mathbf{x}_i^* \mathbf{y}_i$ ($\mathbf{y}_i \in \{-1,1\}$), $\mathbf{w}' = \operatorname{zeros}(D+1)$
 - o přidáváme do vektoru w chybně klasifikovaný bod v dané iteraci Find a wrongly classified observation \mathbf{x}_j :

$$\mathbf{w}^{(t)} \cdot \mathbf{x}_j \le 0, \qquad (j \in \{1, 2, ..., L\}.)$$

If there is no misclassified observation then terminate. Otherwise,

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \mathbf{x}_j .$$

- o Terminace pokud jsou všechny body klasifikovány správně.
- Co když <u>data nejsou lineárně separabilní</u>?: <u>dimension lifting</u>



Logistická regrese (viz. 113)

$$p(1|\mathbf{x})$$

Poměr posteriorních pravděpodobností $\overline{p(2|\mathbf{x})}$ je nazýván <u>odds ratio</u>.

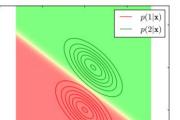
 $\frac{\ln \frac{p(1|\mathbf{x})}{p(2|\mathbf{x})}}{\ln \frac{p(1|\mathbf{x})}{p(2|\mathbf{x})}}$

Zlogaritmováním získáváme log odds:

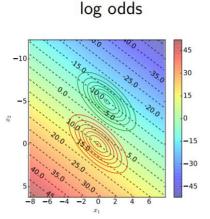
Příklad:



conditionals



posteriors



 $p(1|\mathbf{x}) = \frac{1}{1 + e^{-a(\mathbf{x})}}$

Posteriorní pravděpodobnosti (*posteriors*) lze modelovat takto:

$$p(2|\mathbf{x}) = \frac{1}{1 + e^{a(\mathbf{x})}}$$

$$q(\mathbf{x}) = 1 \quad \text{ if } \quad p(1|\mathbf{x}) > p(2|\mathbf{x}) \quad \Leftrightarrow \quad \frac{p(1|\mathbf{x})}{p(2|\mathbf{x})} > 1 \quad \Leftrightarrow \quad \ln \frac{p(1|\mathbf{x})}{p(2|\mathbf{x})} > 0$$

a obrácené znaménko pro q(x) = 2

Odhad parametrů daného modelu se provádí obvykle metodou maximální věrohodnosti.

Generativní vs. diskriminativní přístup:

Nejbližší soused (1-NN)

Objekt je jednoduše přiřazován ke svému nejbližšímu sousedu.

Pro n \rightarrow inf: chyba 1-NN klasifikace </= $2e_B$

Metoda nejbližších sousedů = K-NN

Vlastnosti

- (+) jednoduchá implementace (naivní řešení složitost: D*N)
- (+) pro 1-NN: Ize najít upper-bound ($\epsilon_{NN} \leq 2\epsilon_B$) a lower-bound: ϵ_B (ϵ_B = Bayes error)
- (+) není potřeba předpoklad na distribuci, ze které data pocházejí
- (+) pro libovolný počet tříd
- (-) při naivní implementaci pomalé, vysoké nároky na paměť počítače (příp. nutná úprava dat) → lze urychlit (viz. níže): redukcí trn. množiny, pomocí k-D trees nebo Voronoi
- (-) není jasné jak zkonstruovat metriku d(potenciální problém: různý význam různých dimenzí)
- (-) vysoké nároky na paměť
- (-) může overfitovat

Algoritmus

Máme trénovací množinu $T = \{(x_i, k_i)\}_{1 \le i \le N}$. Pro testovací bod y nalezneme k nejbližších trénovacích bodů x_i a y ohodnotíme jako té třídy, která má mezi k nejbližšími body většinu.

Algorithm:

1. Given x, find K points $S = \{(x_1', k_1'), (x_2', k_2'), ..., (x_K', k_K')\}$ from the training set \mathcal{T} which are closest to x in the metric d:

$$S = \{(x'_1, k'_1), (x'_2, k'_2), ..., (x'_K, k'_K)\} \equiv \{(x_{r_1}, k_{r_1}), (x_{r_2}, k_{r_2}), ..., (x_{r_K}, k_{r_K})\}$$

$$r_i : \text{the rank of } (x_i, k_i) \in \mathcal{T} \text{ as given by the ordering } d(x, x_i)$$

$$(19)$$

2. Classify x to the class k which has majority in S:

$$k = \operatorname*{argmax}_{l \in R} \sum_{i=1}^{K} \llbracket k_i' = l \rrbracket \qquad (x_i', k_i') \in S \tag{20}$$

• **Urychlení** klasifikátoru K-NN:

- o redukcí trénovací množiny: snížení N např. z tisíců na desítky
- pomocí k-D stromů: ty urychlují vyhledávání podle vzdálenosti v geometrickém prostoru pomocí ukládání dat ve stromové struktuře. Při procházení stromu se pro každou větev určí minimální možná vzdálenost nejbližšího bodu ve větvi od posuzovaného bodu. Po prvním projití do listu pak můžeme ořezat větve, které jistě nemají bližší bod. Je to sublineární vyhledávání, ale složitost je funkcí dat, nejde tedy garantovat.
- pomocí Voronoiových diagramů: při klasifikaci stačí zjistit v jaké buňce bod leží (rychlejší než počítat vzdálenosti)

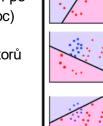
Adaboost

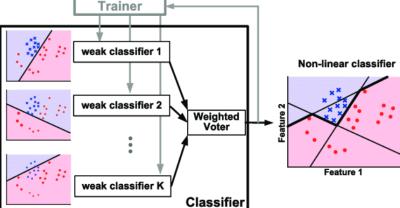
Kombinuje vícero klasifikátorů do jednoho silného klasifikátoru. Výstup dílčích slabých klasifikátorů je kombinován do váženého součtu, který představuje konečný výstup silného klasifikátoru.

Silný klasifikátor
$$H: H(\mathbf{x}) = \mathrm{sign}(f(\mathbf{x}))$$
, $f(\mathbf{x}) = \sum_{t=1}^T \alpha_t h_t(\mathbf{x})$ (h_t je slabý klasifikátor)

Vlastnosti:

- (+) jednoduchá implementace
- (+) dobrá generalizace (= chová se dobře k testovacím datům po natrénování, neoverfituje moc)
- (-) overfitting kvůli noise
- množina B slabých klasifikátorů $h_i(x)$



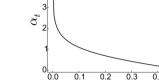


Algoritmus:

- Init: t=1, $D_{t=1}(i)=1/N$
- o vyberu h_t(x) s nejmenší chybou

$$h_t = \arg\min_{h \in \mathcal{B}} \epsilon_t; \quad \epsilon_t = \sum_{i=1}^L D_t(i) [y_i \neq h(\mathbf{x}_i)]$$

- divná závorka dá jedničku pokud výraz uvnitř platí, jinak nulu (tj. při chybě \rightarrow 1)
- pro vybraný slabý klasifikátor h_t mám chybu ε_t
 - **pokud** $\epsilon_t = 0.5 \Rightarrow \text{terminace algoritmu}$
- Váha daného slabého klasifikátoru: $\alpha_t = \frac{1}{2}\log\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$



 $D_{t+1}(i) = \frac{D_t(i)e^{-\alpha_t y_i h_t(\mathbf{x}_i)}}{Z_t} \text{, normalizační faktor}$

$$Z_t = \sum_{i=1}^{L} D_t(i) e^{-\alpha_t y_i h_t(\mathbf{x}_i)}$$
 Z_t:

- normalizační faktor lze vypočítat také takto: $Z_t = 2\sqrt{\epsilon_t(1-\epsilon_t)}$
- Nakonec (po ukončení iterování) výsledný (silný) klasifikátor: $H(\mathbf{x}) = \mathrm{sign}(f(\mathbf{x}))$

$$f(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})$$

Dependence of α_t on ϵ_t

 $\epsilon = \frac{1}{L} \sum_{i=1}^{L} [[y_i \neq H_T(\mathbf{x}_i)]] \leq \prod_{t=1}^{T} Z_t$ Upper bound theorem:

SVM

- lineární klasifikátor; nachází optimální řešení (dělící nadrovina maximalizuje šířku dělícího pásu); lze upravit i pro lin. neseparabilní data; $(w, b)^* = \operatorname{argmax}_{w,b} \min_i d(x_i)$ za podmínky $y_i(w^Tx_i + b) >$ 0, kde d(x_i) je vzdálenost x_i od nadroviny
- $T = \{x_i, y_i\} (i=1...N), x_i \in R^D, y_i \in \{-1,1\}$
- duální f: $\alpha^* = \underset{\alpha}{\operatorname{argmax}} \left\{ \sum_{i=1}^{N} \alpha_i \frac{1}{2} \sum_{i=1}^{N} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \right\}$ (podm: $\alpha \ge 0$, Sum_i y_i $\alpha_i = 0$)
 - → pomocí QP najdeme glob. optimum
- strategie pak: $q(\mathbf{x}) = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b)$, $\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i$ (pouze $\alpha > 0$, tj. SV (ostatní mají $\alpha = 0$))
- učení pomocí iteračních metod, funkce je konvexní → dospěje do globálního minima

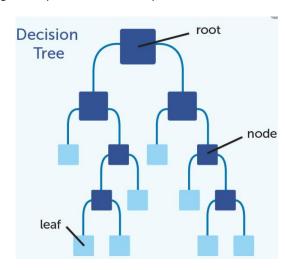
- Soft margin SVM: pro nesep. data představíme C ztráta za špatně klasifikovaný bod; $0 \leq \alpha_i \leq C; \text{ Primární úloha: (w, b)* = argmin_{w,b} 1/2 ||w||^2 + C*Sum_i \xi_i za podmínky y_i(w^Tx_i + b) >= 1 \xi_i, \xi_i >= 0; Duální úloha: <math>\alpha^* = \text{argmax}_{\alpha} \text{ Sum}_i \alpha_i 1/2 \text{ Sum}_i \text{ Sum}_j \alpha_i \alpha_j y_i y_i x_i^T x_j za podmínky 0 <= \alpha <= C, \text{ Sum}_i y_i \alpha_i = 0$
- Při použití dimension lifting $\phi(x)$: $R^D \to R^M$ nemusíme používat/znát přímo předpis mapování. Stačí znát kernel funkci $K(x, y) = \phi(x)^T \phi(y)$. Tou lze nahradit například i mapování do teoreticky nekonečně dimenzionálních prostorů apod. Učení SVM probíhá stejně, pouze duální úloha přijme kernel funkci: $\alpha^* = \operatorname{argmax}_{\alpha} \operatorname{Sum}_i \alpha_i \frac{1}{2} \operatorname{Sum}_i \operatorname{Sum}_j \alpha_i \alpha_j y_i y_i K(x_i, x_j)$ za podmínky $0 \le \alpha \le C$, $\alpha \le C$,
- Klasifikace má dva způsoby:
 - ο použijeme mapování $\phi(x)$: Získáme $w = Sum_{izSV} \alpha_i y_i \phi(x_i)$, $b = y_{sv} w^T \phi(x_{sv})$ a klasifikujeme: $q(z) = sign(w^T \phi(z) + b)$; To znamená, že musíme znát ϕ a klasifikace je zpomalena oproti lineární variantě výpočtem $\phi(z)$
 - o použijeme kernel K(x, y): b = y_{sv} Sum_{i z SV} $\alpha_i y_i K(x_i, x_{sv})$; q(z) = sign(b + Sum_{i z SV} $\alpha_i y_i K(x_i, z)$); To znamená, že si musíme pro klasifikaci pamatovat α_i , y_i , x_i pro support vektory a klasifikace je zpomalena výpočtem Sum_{i z SV} $\alpha_i y_i K(x_i, z)$
- časová složitost závislá na N (ne na D; při optimalizaci jsou počítány skalár. součiny x_i^T,x_i)
- rychlost učení závisí na inicializaci
- dobře generalizuje, používají pouze skalární součiny dat, tzn. že jsou nezávislé na vstupní dimenzi, umí pracovat i s neseparabilními daty, při klasifikaci stačí znát pouze w a b, lze zobecnit pro oddělení nelineárně separovatelných dat pomocí dimension lifting a kernel triku.
- SVM se složitě zobecňují pro oddělení více tříd a nehodí se do inkrementálního učení.
- Perceptron vs SVM
 - o Perceptron nalezne nějaké řešení, SVM "dobré" (maximalizující margin)
 - o u perceptronu pro separabilní data zaručena konvergence (Novikoff teorém)

Rozhodovací stromy (decision trees)

Výhodou je <u>interpretovatelnost</u> (*srovnejme např. s neuronovou sítí, kde takřka nevíme nic, co parametry znamenají*), zde vidíme podle kterých atributů se rozhoduje

Příklad – restaurace: (viz. tabulka níže)

- Atributy
 - o Fri = Je pátek? → T/F
 - \circ Price \rightarrow \$/\$\$/\$\$\$
 - \circ Rain? \rightarrow T/F
- atributy mohou být různé datové typy (boolean, integer; diskrétní/spojité)
- Klasifikuje do dvou tříd: True nebo False (Wait nebo neWait)



Example	Attributes										Target
1	Alt	Bar	Fri	Hun	Pat	Price	Rain	Res	Type	Est	Wait
X_1	Т	F	F	Т	Some	\$\$\$	F	Т	French	0-10	Т
X_2	Т	F	F	Т	Full	\$	F	F	Thai	30–60	F
X_3	F	Т	F	F	Some	\$	F	F	Burger	0-10	Т
X_4	Т	F	Т	Т	Full	\$	F	F	Thai	10–30	Т
X_5	Т	F	Т	F	Full	\$\$\$	F	Т	French	>60	F
X_6	F	Т	F	Т	Some	\$\$	Т	Т	Italian	0-10	Т
X_7	F	Т	F	F	None	\$	Т	F	Burger	0-10	F
X_8	F	F	F	Т	Some	\$\$	Т	Т	Thai	0-10	Т
X_9	F	Т	Т	F	Full	\$	Т	F	Burger	>60	F
X_{10}	Т	Т	Т	Т	Full	\$\$\$	F	Т	Italian	10-30	F
X_{11}	F	F	F	F	None	\$	F	F	Thai	0-10	F
X_{12}	Т	Т	Т	Т	Full	\$	F	F	Burger	30–60	Т

$$IG(A) = H(P) - \sum_{i=1}^{k} \frac{|P_i|}{|P|} H(P_i)$$

Information Gain (IG) pro daný atribut (A):

Při trénování (učení) rozhodovacího stromu - trénování na trénovacích datech:

$$IG(A) = H(P) - \sum_{i=1}^{k} \frac{|P_i|}{|P|} H(P_i)$$

Spočítám IG pro každý atribut (

Ten atribut, u kterého vyjde nejvyšší, ten použiju.

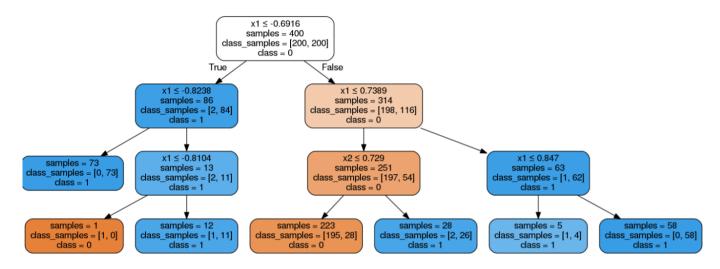
$$H(P) = -\sum_{i} \frac{|P^{(j)}|}{|P|} \log \frac{|P^{(j)}|}{|P|}$$

 $H(P) = -\sum_j \frac{|P^{(j)}|}{|P|} \log \frac{|P^{(j)}|}{|P|}$ kde $P^{(j)}$ je počet bodů které spadnou do j-té dané třídy

entropy of
$$P_i$$
:
$$H(P_i) = -\sum_j \frac{|P_i^{(j)}|}{|P_i|} \log \frac{|P_i^{(j)}|}{|P_i|} \quad \text{(entropy of } P_i\text{)}$$

To odpovídá tomu, že se snažím co nejvíc snižovat entropii. Entropie:

log₂; informační entropie [bit]: vysoká = nic nevím, nízká = vím co se stane



(+) Výhody

- Interpretovatelnost
- Rychlost rozhodování
- nepotřeba použít všechny měření jen ty relevantní
- outliers ošetřeny speciální větví
- pro klasifikační i regresní úlohy

(-) Nevýhody

- Neexistuje efektivní způsob trénování (všechny na bázi greedy algoritmu → není záruka optimality)
- Hrozí overfitting (→ proti tomu se používají randomizované stromy)

Top-down je příkladem greedy algorimtu ("hladový") → nemusí být "nejlepší"
Na začátku nastavím P=T
spočítám IG pro každý atribut, zvolím pro dělení ten který mi dá nejvyšší IG
rekurze pro větve

Decision trees vs. SVM

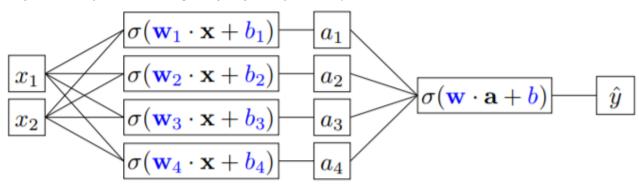
- <u>DT</u>: žádné záruky na generalizaci/optimalitu řešení X <u>SVM</u>: dobře generalizuje, jelikož minimalizuje VC dimenzi maximalizací margin mezi třídami dat, díky konvexnosti rozhodovací funkce a QP optimalizaci nalezne globální optimum
- <u>DT</u>: pro jakékoliv prostory, i nenumerické, s různými hodnotami/měřítky v různých dimenzích, apod. X <u>SVD</u>: jen pro data z R^D
- trénovací množina může být libovolně velká, při klasifikaci se nevyužívá (klasifikace je sublineární vůči dimenzi dat) X trénovací množina může být libovolně velká, při klasifikaci se nevyužívá, pouze se z její části (tzv. support vektorů) vypočítá lineární klasifikátor q(x) = sign(w^Tx)
- při učení je potřeba uchovávat veškerá data, při klasifikaci pak již DT samotný (posloupnost otázek/rozhodnutí) X při učení stačí uchovávat skalární součiny všech xi, pro klasifikaci stačí uchovávat jediný vektor w z R^D
- učení je závislé na složitosti stromu, respektive nic jako učení neprobíhá, strom je pouze sestaven, při stavbě stromu lze dodržovat určitá pravidla pro jeho obecně lepší klasifikaci (TDIDT) X učení je závislé na velikosti trénovací množiny a rychlosti konvergence QP optimalizace, která kriticky závisí na počáteční inicializaci
- odchýlené hodnoty lze jednoduše odchytávat jako speciální větve stromu X vliv odchýlených hodnot lze regulovat velikostí ztráty za špatně klasifikovaný bod C, ta určuje směr "úzký margin bez špatně klasifikovaných bodů (vyšší C)" vs "širší margin se špatně klasifikovanými body (nižší C)"

Neuronové sítě

Doporučuju: 3Blue1Brown - YT

Vrstvy

Příklad s jednou skrytou vrstvou: (jinak jich je obvykle více)



input layer

hidden layer

output layer

, což je to samé co (in)
$$\xrightarrow{\mathbf{x} \in \mathbb{R}^2} \boxed{\sigma(\mathbf{W}\mathbf{x} + \mathbf{b})} \xrightarrow{\mathbf{a} \in \mathbb{R}^4} \boxed{\sigma(\mathbf{w} \cdot \mathbf{a} + b)} \xrightarrow{\hat{y} \in \mathbb{R}} (\text{out})$$

Motivace

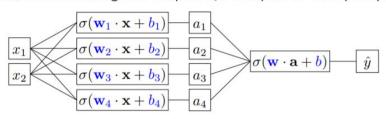
Recall the Perceptron: given the perceptron parameters $\mathbf{w} \in \mathbb{R}^n$ and $b \in \mathbb{R}$, the classification \hat{y} to two classes $\{-1,1\}$ for a vector $\mathbf{x} \in \mathbb{R}^n$ is performed as

$$\hat{y} = \operatorname{sign}(\mathbf{w} \cdot \mathbf{x} + b), \tag{1}$$

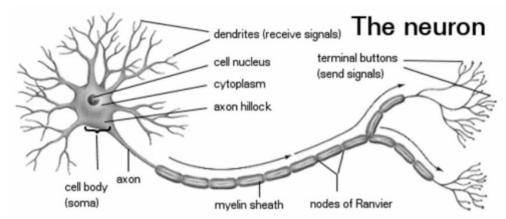
which is an affine (often called linear) function $l(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$, followed by a non-linear function $\sigma : \mathbb{R} \to \mathbb{R}$, $\sigma(z) = \text{sign}(z)$. The perceptron is also often depicted as follows:

$$x_1$$
 x_2
 $\sigma(\mathbf{w} \cdot \mathbf{x} + b)$
 \hat{y}

to make the linear combination of elements of the input vector explicit. Perceptron is a linear classifier; these are well understood, have low VC dimension, etc. and thus it is a natural next step to combine them to a more complex classifiers: By letting more of them sharing the input, as well as using their outputs a_i 's as inputs to other perceptrons:



'Biological' motivation - a real neuron is known to combine inputs to an output which is then passed to other neurons.

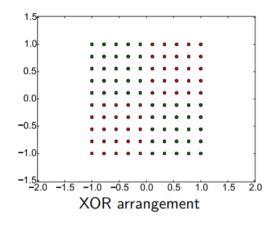


Historická poznámka

Perceptron (Rosenblatt, 1956) with its simple learning algorithm generated a lot of excitement

Minsky and Papert (1969) showed that even a simple XOR cannot be learnt by a perceptron

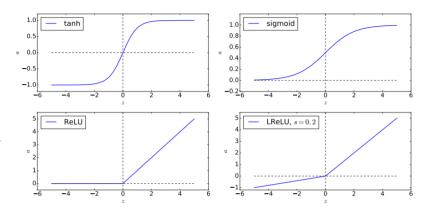
But chaining perceptrons to a network (Multi-Layer Perceptron, MLP) can learn XOR



U perceptronu byl na konci threshold, .. Aktivační funkce mají takovéto tvary:

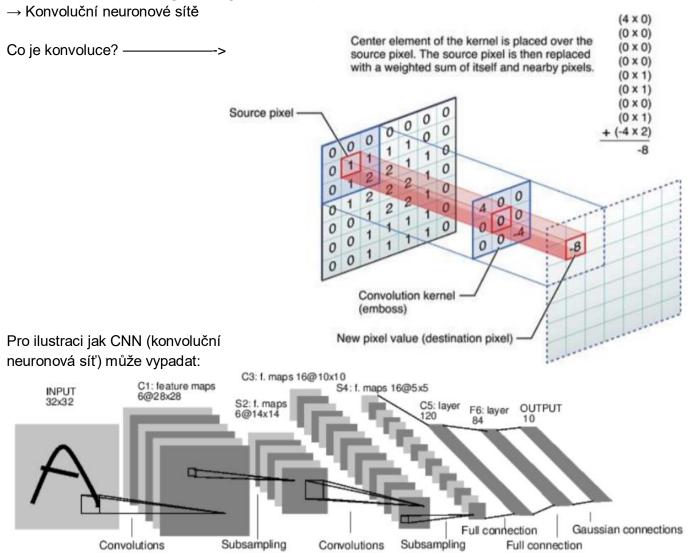
- sigmoida (logistická nebo tanh)
- ReLU (Rectified Linear Unit)
- Leaky ReLU

<u>Nepoužívá se sign</u> (jako u perceptronu) - ta není diferencovatelná v 0



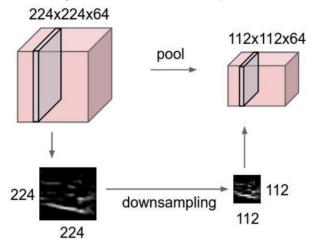
Gradient a Back propagation: 3Blue1Brown (LINK nahoře)

Neuronové sítě pro Image Recognition (Rozpoznáv

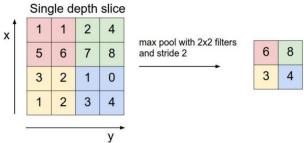


Pooling

Pooling layers reduce the dimensions of data by combining the outputs of neuron clusters at one layer into a single neuron in the next layer.



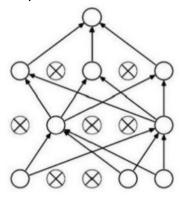
• Max pooling:



The maximum pixel value of the batch is selected

- Min pooling: The minimum pixel value of the batch is selected.
- Average pooling: The average value of all the pixels in the batch is selected.

Dropout



Data augmentation - vytvoří se tak více dat pro trénování

- Náhodná rotace
- Afinní transformace
- Náhodné škálování
- Color augmentation
- Noise input
- Thin plate deformation
- ...cokoliv si umíš přestavit

Poznámky

Předpoklad: Features jsou navzájem nezávislé -> pak mluvíme o <u>naive Bayes approach</u>. Neuronové sítě nelze zapsat jako jednu lineární funkci (kvůli aktivační fci, ..)

→ je popsána tunou parametrů