FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

KATEDRA
KYBERNETIKY

Metody Počítačového Vidění
8. přednáška: Strojové učení

Ing. Lukáš Bureš
Ing. Petr Zimmermann
Katedra kybernetiky

Plzeň
13. listopadu 2014

Obsah

Kapitola 1

Metody Počítačového Vidění: 8. přednáška

1.1 Princip klasifikace

Ve strojovém učení a statistice je klasifikace problém identifikace, ke kterému ze souboru kategorií patří nová pozorování a to na základě trénovací množiny dat. Příkladem by mohl být antispamový filtr, který na základě jména klasifikuje do složek „je spam“ nebo „není spam“.

V terminologii strojového učení je klasifikace za instanci učení s učitelem, tj učení, kde je k dispozici trénovací sada správně označených pozorování. Podobně, klasifikaci, kde není informace od učitele se říká shlukování a zahrnuje seskupování dat do kategorií na základě jisté míry přirozené podobnosti nebo vzdálenosti.

Často mají jednotlivá pozorování sadu měřitelných vlastností. Tyto vlastnosti mohou představovat jednotlivé kategorie například typ krevní skupiny, pořadové číslo („malé“, „střední“, „velké“.

Algoritmus, který provádí klasifikaci a to zejména v konkrétním provedení je známý jako klasifikátor. Termín „klasifikátor“ někdy také odkazuje na matematické funkce, realizované klasifikačním algoritmem, který mapuje vstupní data do kategorií.

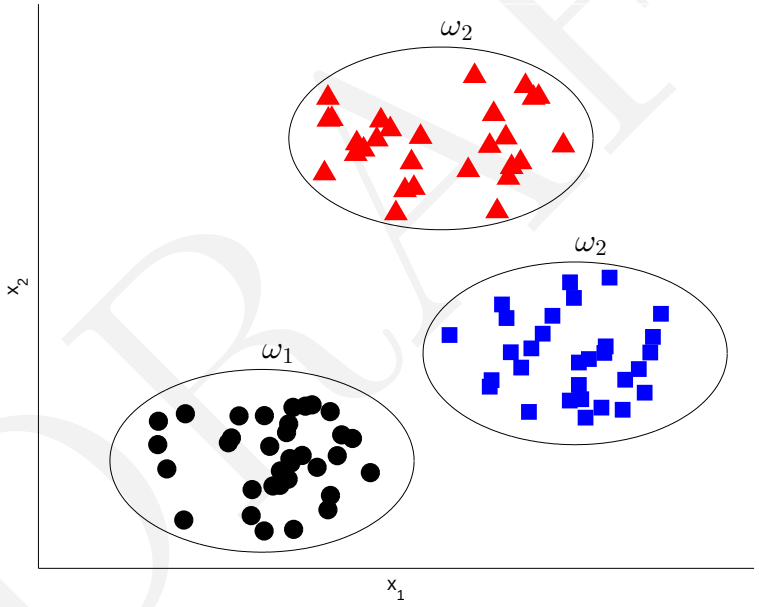
Terminologie skrze vědecké oblasti je velice pestrá. Ve statistice, kde se klasifikace často provádí binární regresí nebo obdobným postupem se pozorované proměnné nazývají „nezávislé proměnné“ a predikované kategorie jsou známé jako výsledky, které je možné považovat za závislé proměnné. Ve strojovém učení jsou pozorování známá jako instance a nezávislé proměnné jsou nazývány příznaky, které se sdružují do příznakových vektorů. Dále jsou kategorie nazývány třídami.

1.1.1 Porovnání příznakových vektorů

Jednotlivé příznakové vektory mohou být chápány jako body v mnohodimenzionálním prostoru a úkolem klasifikace (klasifikátoru) je rozdělení tohoto prostoru na podprostory představující klasifikační třídy $\omega_{1,\dots,N}$. Cílem je tedy rozdělit množinu příznakových vektorů X na podmnožiny, tak aby platila následující podmínka

$$\begin{aligned} \omega_{1,\dots,N} &\subset X, \\ \bigcup_{i=1}^{N} &= X, \\ \omega_i \cap \omega_j &= \emptyset, \text{pro} \quad i,j = 1,\dots,N, \quad i \neq j, \end{aligned}$$

Třídy (shluky) v sobě sdružují vektory, které jsou si navzájem podobné (viz Obr. 1.1). Podobnost vektorů je subjektivní pojem a výsledek klasifikace tak musí být ověřen expertem, který rozhodne, zda je výsledek klasifikace přípustný. Obecně platí, že pokud je vhodně sestaven příznakový vektor, lze vektory rozřazovat do shluků na základě tzv. míry podobnosti či odlišnosti.



Obrázek 1.1: Klasifikace do tří tříd.

Obě míry interpretují vzdálenost dvou vektorů v prostoru. Liší se pouze tím, že míra podobnosti pro dva stejné vektory dosahuje maximálních hodnot, naopak míra odlišnosti je pro takové vektory minimální.

1.1.2 Míra podobnosti

Míra podobnosti s dvou vektorů z množiny X je definována jako

$$s : X \times X \longrightarrow \mathcal{R}, \tag{1.1}$$

kde $\mathcal{R}$ je množina reálných čísel, tak že

$$\exists s_0 \in \mathcal{R} : -\infty < s(\boldsymbol{x}, \boldsymbol{y}) \geq s_0 < +\infty, \qquad \forall \boldsymbol{x}, \boldsymbol{y} \in X, \tag{1.2}$$

$$s(\boldsymbol{x}, \boldsymbol{y}) = s(\boldsymbol{y}, \boldsymbol{x}), \qquad \forall \boldsymbol{x}, \boldsymbol{y} \in X, \tag{1.3}$$

$$s(\boldsymbol{x}, \boldsymbol{x}) = s_0, \qquad \forall \boldsymbol{x} \in X. \tag{1.4}$$

V praxi nejpoužívanější míry podobnosti jsou:

- Skalární součin

$$s_{skal.souč.}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}^\top \boldsymbol{y} = \sum_{i=1}^{l} x_i y_i, \tag{1.5}$$

  kde l je dimenze vektorů $\boldsymbol{x}$ a $\boldsymbol{y}$. Skalární součin je používán v případě, že jsou porovnávané vektory normalizované, což znamená, že mají stejnou délku. Míra $s_{skal.souč.}(\boldsymbol{x}, \boldsymbol{y})$ vyjadřuje úhel svíraný vektory $\boldsymbol{x}$ a $\boldsymbol{y}$.

- Kosinová podobnost

$$s_{cos}(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x}^\top \boldsymbol{y}}{\|\boldsymbol{x}\| \|\boldsymbol{y}\|}, \tag{1.6}$$

  kde $\|\boldsymbol{x}\| = \sqrt{\sum_{i=1}^{l} x_i^2}$ a $\|\boldsymbol{y}\| = \sqrt{\sum_{i=1}^{l} y_i^2}$ je délka vektoru $\boldsymbol{x}$, resp. $\boldsymbol{y}$.

- Tanimotova míra

$$s_T(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x}^\top \boldsymbol{y}}{\|\boldsymbol{x}\|^2 + \|\boldsymbol{y}\|^2 - \boldsymbol{x}^\top \boldsymbol{y}}. \tag{1.7}$$

1.1.3 Míra odlišnosti

Míra podobnosti d dvou vektorů z množiny X je definována jako

$$d : X \times X \longrightarrow \mathcal{R}, \tag{1.8}$$

kde $\mathcal{R}$ je množina reálných čísel, tak že

$$\exists d_0 \in \mathcal{R} : -\infty \leq d_0 < d(\boldsymbol{x}, \boldsymbol{y}) < +\infty, \qquad \forall \boldsymbol{x}, \boldsymbol{y} \in X, \tag{1.9}$$

$$d(\boldsymbol{x}, \boldsymbol{y}) = d(\boldsymbol{y}, \boldsymbol{x}), \qquad \forall \boldsymbol{x}, \boldsymbol{y} \in X, \tag{1.10}$$

$$d(\boldsymbol{x}, \boldsymbol{x}) = d_0, \qquad \forall \boldsymbol{x} \in X. \tag{1.11}$$

Mezi nejčastěji používané míry odlišnosti patří:

- l_p-norma

$$d_p(\boldsymbol{x}, \boldsymbol{y}) = \left(\sum_{i=1}^{l} |x_i - y_i|^p \right)^{\frac{1}{p}}. \tag{1.12}$$

- l_1-norma (tzv. norma Manhattan)

$$d_1(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{l} |x_i - y_i|. \tag{1.13}$$

- l_∞-norma

$$d_\infty(\boldsymbol{x}, \boldsymbol{y}) = \max_{1 \le i \le l} |x_i - y_i|. \tag{1.14}$$

Výše uvedené míry mají i tzv. vážené formy. Ty získáme doplněním koeficientu $w_i \geq 0$ před jednotlivé absolutní hodnoty, koeficient pak vyjadřuje váhy příslušných prvků příznakového vektoru. Díky svému chování jsou míry odlišnosti často označovány jako vzdálenost dvou vektorů. Známá Euklidova vzdálenost je získána pouhou modifikací l_p-normy.

1.1.4 Porovnání vektoru a třídy

V mnoha klasifikačních algoritmech dochází k začlenění příznakového vektoru $\boldsymbol{x}$ do třídy ω na základě porovnání $\rho(\boldsymbol{x}, \omega)$, přičemž porovnání ρ může představovat jakoukoli míru podobnosti, resp. odlišnosti. V praxi se porovnání mezi vektorem a třídou provádí dvěma způsoby. V prvním jsou při stanovení $\rho(\boldsymbol{x}, \omega)$ brány v potaz všechny vektory tvořící třídu ω. Mezi typické představitele prvního způsobu porovnání vektoru a třídy patří:

- Maximální podobnostní funkce

$$\rho_{max}(\boldsymbol{x}, \omega) = \max_{\boldsymbol{y} \in \omega} \rho(\boldsymbol{x}, \boldsymbol{y}). \tag{1.15}$$

- Minimální podobnostní funkce

$$\rho_{min}(\boldsymbol{x}, \omega) = \min_{\boldsymbol{y} \in \omega} \rho(\boldsymbol{x}, \boldsymbol{y}). \tag{1.16}$$

- Průměrná podobnostní funkce

$$\rho_{prům.}(\boldsymbol{x}, \omega) = \frac{1}{n_\omega} \sum_{\boldsymbol{y} \in \omega} \rho(\boldsymbol{x}, \boldsymbol{y}), \tag{1.17}$$

  kde n_ω značí počet vektorů tvořících třídu ω.

V druhém způsobu porovnání je třída zastoupena svým reprezentantem. Porovnání je v tomto případě stanoveno na základě vztahu $\rho(\boldsymbol{x}, \omega_{rep})$, přičemž ω_{rep} představuje vektor reprezentující třídu ω. Je zřejmé, že za ρ lze opět dosadit jakoukoli míru podobnosti či odlišnosti.

1.1.5 Reprezentant třídy

Mezi typické volby reprezentanta třídy patří:

- Průměr

$$\boldsymbol{\omega}_p = \frac{1}{n_\omega} \sum_{\boldsymbol{y} \in \omega} \boldsymbol{y}, \tag{1.18}$$

  kde n_ω je počet prvků uvnitř třídy ω. Průměr je nejčastější volbou reprezentanta třídy, jelikož je velice jednoduché ho stanovit. Nevýhodou ovšem je, že průměr nemusí být součástí třídy. Pokud usilujeme o to, aby reprezentant byl opravdu prvkem dané třídy je vhodnější zvolit střed či medián.

- Střed $\boldsymbol{\omega}_c \in \omega$ je vektor splňující:

$$\sum_{\boldsymbol{y} \in \omega} d\left(\boldsymbol{\omega}_c, bmy\right) \leq \sum_{\boldsymbol{y} \in \omega} d\left(\boldsymbol{z}, \boldsymbol{y}\right), \quad \forall \boldsymbol{z} \in \omega, \tag{1.19}$$

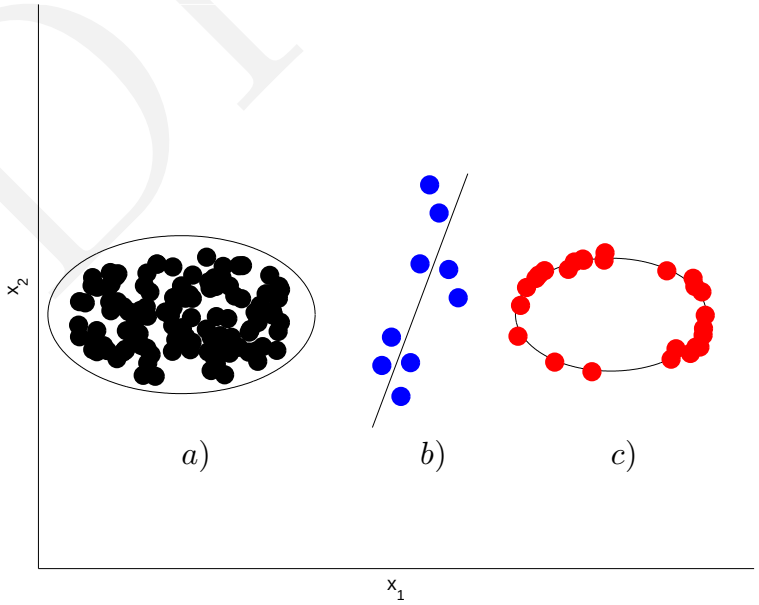
  kde d značí míru odlišnosti (pokud by byla použita míra podobnosti s je nutné v předpisu otočit znaménka nerovnosti).

- Medián $\boldsymbol{\omega}_{med} \in \omega$ je vektor splňující:

$$med\left(d\left(\boldsymbol{\omega}_{med}, \boldsymbol{y}\right) | \boldsymbol{y} \in \omega\right) \leq med\left(d\left(\boldsymbol{z}, \boldsymbol{y}\right) | \boldsymbol{y} \in \omega\right), \quad \forall \boldsymbol{z} \in \omega, \tag{1.20}$$

kde d je míra odlišnosti.

Uvedené reprezentanty třídy lze použít pouze v případě, že je třída kompaktní (viz Obr. 1.2). Pokud jsou prvky uvnitř třídy uspořádané lineárně či do kružnice (sféry) je vhodnější porovnávat vektory s těmito typy tříd na základě vzdálenosti od přímky (roviny), resp. kružnice (sféry).



Obrázek 1.2: Příklad třídy a) kompaktní, b) lineární a c) sférické..

1.1.6 Co je strojové učení?

Pojem strojové učení nemá dodnes pevně danou definici, můžeme si ukázat několik pokusů o jeho definování

- Arthur Samuel (1959). Strojové učení: *Obor, který dává počítačům schopnost učit se bez toho, aby byly přímo naprogramovány.*
- Tom Mitchell (1998): Dobře definovaný problém strojového učení: *Předpokladem je, že se počítačový program učí ze zkušeností* ***E*** *s respektováním úlohy* ***T*** *a s měřičem výkonu* ***P****, pokud se jeho výkon na úloze* ***T*** *měřený pomocí* ***P*** *zvyšuje se zkušeností* ***E****.*

Pokusme se jednotlivé části rozebrat na příkladu: Předpokládejme, že náš emailový klient sleduje, které email označíme nebo neoznačíme jako spam. Na základě tohoto rozhodnutí se náš emailový klient učí lépe rozpoznávat co je a co není spam. Nyní si můžeme položit otázku co z následujících tvrzení je úloha **T**?

1. Klasifikace emailů jako spam nebo vyžádaný email.
2. Sledování značek od uživatele, který email označil jako spam nebo vyžádaný email.
3. Počet (nebo poměr) emailů, které byly správně klasifikovány jako spam nebo vyžádaný email.
4. Nic z výše uvedeného - není to problém strojového učení.

Pokud si rozebereme jednotlivé body výše, tak bod jedna je definice úlohy **T**, dále bod dva je zkušenost **E** a třetí bod je náš měřič výkonu **P**.

Existuje několik rozdílných typů učících se algoritmů. Hlavní dva typy se nazývají

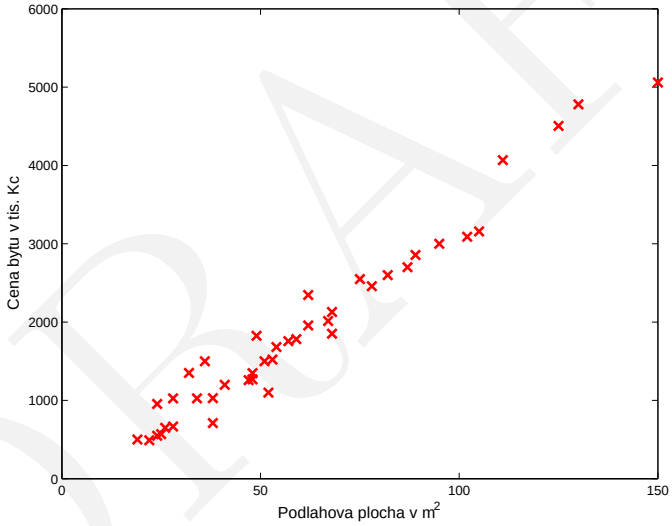
- učení s učitelem a
- učení bez učitele.

1.2 Lineární regrese

Lineární regrese je matematická metoda používaná pro proložení souboru bodů v grafu přímkou. O bodech reprezentujících měřená data se předpokládá, že jejich x-ové souřadnice jsou přesné, zatímco y-nové souřadnice mohou být zatíženy náhodnou chybou, přičemž předpokládáme, že závislost y na x lze graficky vyjádřit přímkou. Pokud měřené body proložíme přímkou, tak při odečítání z grafu bude mezi y-novou hodnotou měřeného bodu a y-novou hodnotou ležící na přímce odchylka. Podstatou lineární regrese je nalezení takové přímky, aby součet druhých mocnin těchto odchylek byl co nejmenší. Lineární regresi lze zobecnit i pro prokládání jinou funkcí než přímkou.

1.2.1 Motivace

Jako motivaci můžeme použít příklad, kdy si budete chtít zakoupit byt v Plzni, proto by bylo dobré znát, kolik zaplatíte za jakou podlahovou plochu, v ideálním případě pro všechny velikosti bytů. Nebo-li se snažíte predikovat cenu bytu v závislosti na podlahové ploše. Na Obr. 1.3 lze vidět vývoj ceny bytu v závislosti na podlahové ploše.



Obrázek 1.3: Vývoj ceny bytu v závislosti na podlahové ploše.

1.2.2 Učení s učitelem

Učení s učitelem (supervised learning) dává „správnou odpověď“ pro každý příklad z množiny dat.

Problém regrese: Na základě reálného (spojitého) vstupu predikovat reálný (spojitý) výstup.

Problém klasifikace: Na základě diskrétního vstupu predikovat diskrétní výstup.

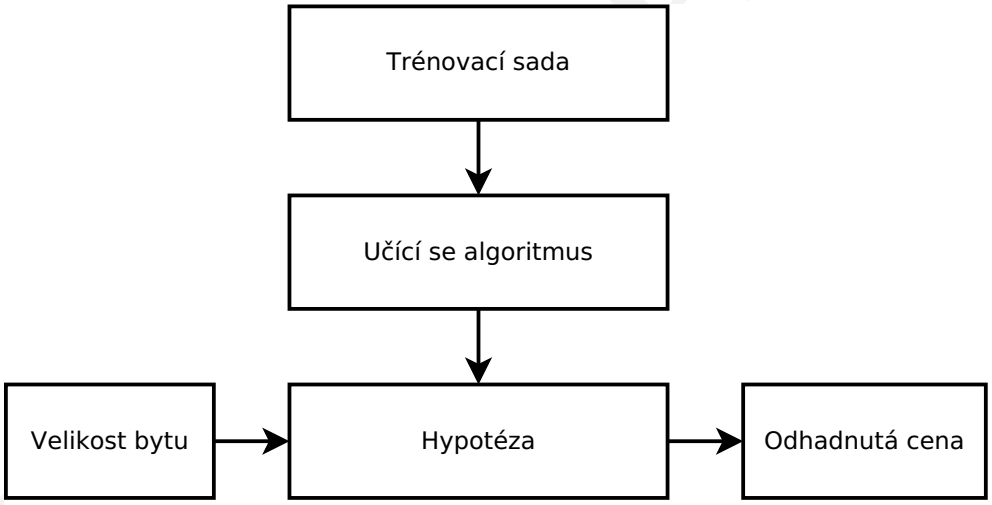
Více formálně v případě učení s učitelem potřebujeme množinu dat nazývanou trénovací množina. V našem případě predikce ceny bytu máme trénovací množinu cen bytů a k nim odpovídající podlahovou plochu. Naším úkolem je naučit se z těchto dat jak predikovat ceny bytů.

Nyní zavedeme notaci nechť m je velikost trénovací sady, x jsou „vstupní“ proměnné neboli příznaky a y jsou „výstupní“ proměnné neboli příznaky. Jeden trénovací vzorek budeme označovat jako $(\boldsymbol{x}, y)$ a odpovídá jednomu řádku v tabulce 1.1. Dále $\left(x^{(i)}, y^{(i)}\right)$ označuje i-tý trénovací vzorek.

Velikost bytu v m^2 (x)	Cena bytu v tis. Kč (y)
19	500
22	490
25	568
...	...

Tabulka 1.1: Příklady trénovacích vzorků.

Trénovací množina, v našem případě ceny bytů, je potřebná pro natrénování učícího se algoritmu, jehož výstupem je funkce (hypotézy), která se většinou označuje h. Účelem funkce (hypotézy) h je ze vstupu x (velikost bytu) odhadnout výstup y (cena bytu). Tedy h je funkce, která mapuje x na y. Schéma je naznačeno na Obr. 1.4.



Obrázek 1.4: Schéma učení s učitelem.

Když navrhujeme učící se algoritmy, další věc kterou potřebujeme rozhodnout je jak budeme reprezentovat hypotézu h. V našem motivačním příkladě si vystačíme s hypotézou v následujícím tvaru lineární funkce:

$$h_{\boldsymbol{\Theta}}(x) = \vartheta_0 + \vartheta_1 x, \tag{1.21}$$

zkráceně $h(x)$. Jedná se tedy o lineární regresi s jednou proměnnou. V případě komplexnějšího učícího se algoritmu je nutné použít složitější nelineární model.

Poznámka

ϑ je malá théta a Θ je velká théta.

1.2.3 Ztrátová funkce

Pro nejlepší napasování našeho lineárního modelu na trénovací data budeme potřebovat určit parametr ϑ_0 a ϑ_1 v naší hypotéze:

$$h_{\mathbf{\Theta}}(x) = \vartheta_0 + \vartheta_1 x, \tag{1.22}$$

Snažíme se zvolit parametry ϑ_0 a ϑ_1 tak, aby hypotéza $h_{\mathbf{\Theta}}(x)$ byla co nejblíže k y pro naše trénovací vzorky (x, y). Jinými slovy snažíme se minimalizovat kvadratickou chybu:

$$\min_{\vartheta_0,\vartheta_1} \frac{1}{2m} \sum_{i=1}^{m} \left(h_{\mathbf{\Theta}}\left(x^{(i)}\right) - y^{(i)} \right)^2, \tag{1.23}$$

kde

$$h_{\mathbf{\Theta}}\left(x^{(i)}\right) = \vartheta_0 + \vartheta_1 x^{(i)}. \tag{1.24}$$

Přesněji označíme ztrátovou funkci:

$$J(\vartheta_0, \vartheta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_{\mathbf{\Theta}}\left(x^{(i)}\right) - y^{(i)} \right)^2 \tag{1.25}$$

a minimalizujeme

$$\min_{\vartheta_0,\vartheta_1} J(\vartheta_0, \vartheta_1). \tag{1.26}$$

Tato ztrátová funkce J je také občas nazývána kvadratická chyba nebo ztrátová funkce kvadratické chyby.

Ztrátová funkce ve formě kvadratické chyby je nejčastěji používanou ztrátovou funkcí při řešení regresních problémů.

1.2.4 Gradientní algoritmus

V kapitole 1.2.3 jsme definovali ztrátovou funkci J. V této kapitole bude popsán gradientní algoritmus pro minimalizaci ztrátové funkce J. Gradientní algoritmus není omezen jen a pouze na lineární regresi nalézá uplatnění v optimalizaci a v dalších oblastech strojového učení.

Máme ztrátovou funkci $J(\vartheta_0, \vartheta_1)$ a chtěli bychom jí minimalizovat. Proto zvolíme náhodně ϑ_0 a ϑ_1, dále v každé iteraci měníme hodnoty ϑ_0 a ϑ_1, tak abychom redukovali $J(\vartheta_0, \vartheta_1)$ dokud, v nejlepším případě, nedokonvergujeme do globálního minima.

Obecně lze gradientní algoritmus použít na jakoukoliv ztrátovou funkci:

$$J(\vartheta_0, \cdots, \vartheta_n) \tag{1.27}$$

a snažíme se minimalizovat:

$$\min_{\vartheta_0,\ldots,\vartheta_n} J(\vartheta_0, \ldots, \vartheta_n) \tag{1.28}$$

Definice gradientního algoritmu: opakuj dokud není dosaženo konvergence

$$\vartheta_j = \vartheta_j - \alpha \frac{\partial}{\partial \vartheta_j} J(\vartheta_0, \vartheta_1), \tag{1.29}$$

pro $j = \{0, 1\}$, pro náš příklad (lineární regrese jedné proměnné) v sekci 1.2.1. V rovnici 1.29 proměnná α určuje, jak velké kroky bude gradientní algoritmus dělat při hledání minima a nazývá se rychlost učení. V případě velké hodnoty α bude algoritmus postupovat velkými kroky, rychle, ale není zajištěna konvergence, jelikož se může stát, že algoritmus bude kroužit kolem minima, ale nedosáhne ho, nebo bude divergovat. V případě malé hodnoty α bude algoritmus velice pomalý a nemusí dosáhnout minima pro stanovený počet iterací. Gradientní algoritmus může konvergovat do lokálního minima v případě, že je rychlost učení α konstantní. Proto se v reálných aplikacích volí nejdříve větší α, které se s rostoucí hodnotou iterací nebo času zmenšuje.

Poznámka

Pro implementaci: je nutné aktualizovat všechny Θ současně

$$temp_0 = \vartheta_0 - \alpha \frac{\partial}{\partial \vartheta_0} J(\vartheta_0, \vartheta_1), \tag{1.30}$$

$$temp_1 = \vartheta_1 - \alpha \frac{\partial}{\partial \vartheta_1} J(\vartheta_0, \vartheta_1), \tag{1.31}$$

$$\vartheta_0 = temp_0, \tag{1.32}$$

$$\vartheta_1 = temp_1. \tag{1.33}$$

Nyní budeme aplikovat gradientní algoritmus na náš problém lineární regrese

$$\frac{\partial}{\partial \vartheta_j} J(\vartheta_0, \vartheta_1) = \frac{\partial}{\partial \vartheta_j} \cdot \frac{1}{2m} \sum_{i=1}^{m} \left(h_{\mathbf{\Theta}}\left(x^{(i)}\right) - y^{(i)} \right)^2, \tag{1.34}$$

po dosazení z rovnice 1.24 dostáváme

$$\frac{\partial}{\partial \vartheta_j} J(\vartheta_0, \vartheta_1) = \frac{\partial}{\partial \vartheta_j} \cdot \frac{1}{2m} \sum_{i=1}^{m} \left(\vartheta_0 + \vartheta_1 x^{(i)} - y^{(i)} \right)^2, \tag{1.35}$$

po zderivování podle jednotlivých parametrů Θ získáme

$$\frac{\partial}{\partial \vartheta_0} \quad J(\vartheta_0, \vartheta_1) \quad = \frac{1}{m} \sum_{i=1}^{m} \left(h_{\mathbf{\Theta}} \left(x^{(i)} \right) - y^{(i)} \right), \tag{1.36}$$

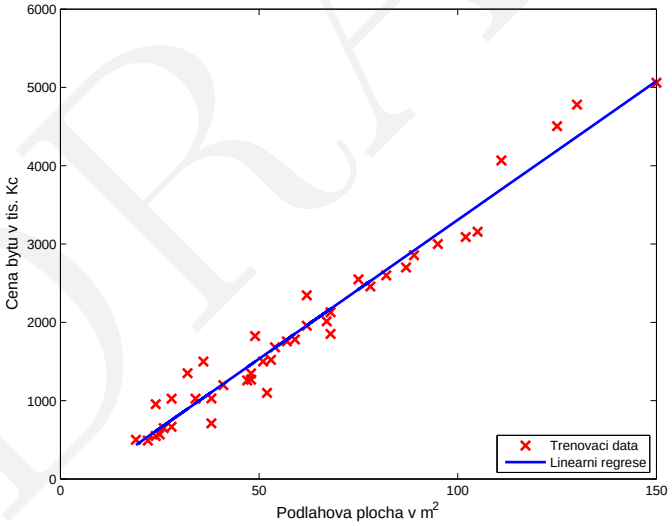
$$\frac{\partial}{\partial \vartheta_1} \quad J(\vartheta_0, \vartheta_1) \quad = \frac{1}{m} \sum_{i=1}^{m} \left(h_{\mathbf{\Theta}} \left(x^{(i)} \right) - y^{(i)} \right) x^{(i)}. \tag{1.37}$$

Výsledný konkrétní tvar rovnice 1.29 pro ϑ_0 a ϑ_1

$$\vartheta_0 \quad = \vartheta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_{\mathbf{\Theta}} \left(x^{(i)} \right) - y^{(i)} \right), \tag{1.38}$$

$$\vartheta_1 \quad = \vartheta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_{\mathbf{\Theta}} \left(x^{(i)} \right) - y^{(i)} \right) x^{(i)}. \tag{1.39}$$

Gradientnímu algoritmu popsanému výše se také někdy říká dávkový gradientní algoritmus, protože využívá všechna trénovací data, při součtu přes všech m vzorků, v každém kroku.



Obrázek 1.5: Výsledná lineární regrese našeho úvodního příkladu po výpočtu gradientním algoritmem.

1.2.5 Vícerozměrná lineární regrese

V sekci 1.2.2 byla představena lineární regrese o jedné proměnné na příkladě predikce ceny bytu (výstup y) v závislosti na velikosti podlahové plochy m^2 (vstup x).

V případě vícerozměrné lineární regrese si můžeme představit, že nemáme pouze velikost podlahové plochy bytu, ale výsledná cena závisí i dalších parametrech bytu, viz tabulka 1.2.

Velikost bytu	Počet místností	Počet poschodí	Stáří bytu	Cena bytu
x_1	x_2	x_3	x_4	y
19	5	1	45	500
22	3	2	40	490
25	2	1	30	568
...	...	...	...	...

Tabulka 1.2: Příklady vícerozměrných trénovacích vzorků.

Proměnnou n budeme označovat počet příznaků (v našem případě v tabulce 1.2 $x_{1,\dots,4}$, proto $n = 4$.) Dále budeme stále jako u jednorozměrného případu označovat vstupní příznaky (všechny = sloupcový vektor) $\boldsymbol{x}^{(i)}$ pro i-tý trénovací vzor. A hodnota jednotlivého j-tého příznaku i-tého trénovacího vzoru bude označována $\boldsymbol{x}_j^{(i)}$.

V případě jednorozměrné lineární regrese byla naše hypotéza h_Θ reprezentována

$$h_{\boldsymbol{\Theta}}(x) = \vartheta_0 + \vartheta_1 x, \tag{1.40}$$

v našem vícerozměrném případě nabývá tvaru

$$h_{\boldsymbol{\Theta}}(\boldsymbol{x}) = \vartheta_0 + \vartheta_1 x_1 + \vartheta_2 x_2 + \vartheta_3 x_3 + \vartheta_4 x_4. \tag{1.41}$$

V případě 1.40 i 1.41 je možné si všimnout, že $x_0 = 1$ čehož dále využijeme, přesněji $\boldsymbol{x}_0^{(i)} = 1$ z důvodu dodržení konvence. Proto $\boldsymbol{x} \in \mathrm{R}^{n+1}$ a to samé platí pro parametry $\boldsymbol{\Theta} \in \mathrm{R}^{n+1}$.

Poznámka

V našem příkladě odpovídají

$$\boldsymbol{x} = [x_0,\ x_1,\ x_2,\ x_3,\ x_4]^\top \tag{1.42}$$

a

$$\boldsymbol{\Theta} = [\vartheta_0,\ \vartheta_1,\ \vartheta_2,\ \vartheta_3,\ \vartheta_4]^\top. \tag{1.43}$$

Obecněji lze vícerozměrnou hypotézu zapsat jako

$$h_{\boldsymbol{\Theta}}(\boldsymbol{x}) = \vartheta_0 + \vartheta_1 x_1 + \vartheta_2 x_2 + \ldots + \vartheta_n x_n, \tag{1.44}$$

neboli jednoduše jako součin vektorů (vnitřní produkt)

$$h_{\boldsymbol{\Theta}}(\boldsymbol{x}) = \boldsymbol{\Theta}^\top \boldsymbol{x}. \tag{1.45}$$

1.2.6 Gradientní algoritmus - vícerozměrná verze

Budeme potřebovat hypotézu

$$h_{\boldsymbol{\Theta}}(\boldsymbol{x}) = \boldsymbol{\Theta}^{\top}\boldsymbol{x} = \vartheta_0 x_0 + \vartheta_1 x_1 + \vartheta_2 x_2 + \ldots + \vartheta_n x_n, \tag{1.46}$$

kde $x_0 = 1$, dále parametry $\boldsymbol{\Theta} = [\vartheta_0,\ \vartheta_1,\ \vartheta_2,\ \ldots,\ \vartheta_4]^{\top}$ a vícerozměrnou ztrátovou funkci

$$J(\boldsymbol{\Theta}) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2. \tag{1.47}$$

Gradientní algoritmus pak bude mít tvar: opakuj

$$\vartheta_j = \vartheta_j - \alpha\frac{\partial}{\partial\vartheta_j}J(\boldsymbol{\Theta}), \tag{1.48}$$

současně aktualizovat parametry pro všechny $j = 0, \ldots, n$.

Vztah z rovnice 1.48 lze zapsat jako

$$\frac{\partial}{\partial\vartheta_j}J(\boldsymbol{\Theta}) = \frac{1}{m}\sum_{i=1}^{m}\left(h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)\boldsymbol{x}_j^{(i)} \tag{1.49}$$

a po spojení rovnic 1.48 a 1.49 získáváme výsledný tvar

$$\vartheta_j = \vartheta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)\boldsymbol{x}_j^{(i)}, \tag{1.50}$$

pokud bychom výslednou rovnici 1.50 rozepsali pro první dva parametry ϑ_0 a ϑ_1 dostali bychom

$$\vartheta_0 = \vartheta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)\boldsymbol{x}_0^{(i)}, \quad \boldsymbol{x}_0^{(i)} = 1 \tag{1.51}$$

$$\vartheta_1 = \vartheta_1 - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)\boldsymbol{x}_1^{(i)} \tag{1.52}$$

$$\vdots$$

1.2.7 Features scaling

V této podkapitole si řekneme něco o velikosti příznaků. První a nejdůležitější věcí je normování velikostí příznaků na stejnou velikost. Neboli, pokud použijeme opět náš příklad nákupu bytu a definujeme si velikost bytu v dm^2 (příznak x_1) a počet místností bytu (příznak x_2) a řekněme že:

$$\begin{aligned} x_1 &\in \langle 5000, 20000 \rangle, \\ x_2 &\in \langle 1, 5 \rangle, \end{aligned}$$

tak bude pro gradientní metodu velmi obtížné a více výpočetně náročné (bude nutno spočítat více iterací) nalézt minimum. Proto je důležité, aby všechny příznaky měli stejné škálování. V našem případě by to znamenalo

$$\begin{aligned} x_1 &= \frac{\text{velikost bytu}}{20000}, \\ x_2 &= \frac{\text{počet místností}}{5}. \end{aligned}$$

Více obecně lze říci, že se většinou volí normování na interval $\langle -1, 1 \rangle$.

Mean normalization

Velice častá normalizace střední hodnotou μ, konkrétně pro i-tý příznak

$$x_i \leftarrow \frac{x_i - \mu_i}{max - min} \tag{1.53}$$

nebo

$$x_i \leftarrow \frac{x_i - \mu_i}{std} \tag{1.54}$$

(std = Standard deviation).

1.2.8 Analytické řešení

V této sekci si ukážeme jak lze vypočítat minimum naší ztrátové funkce $J(\boldsymbol{\Theta})$ analyticky.

Příklad:

Předpokládejme 1D příklad ($\vartheta \in \mathcal{R}$), kdy máme ztrátovou funkci $J(\vartheta)$, která je funkcí jedné reálné proměnné

$$J(\vartheta) = a\vartheta^2 + b\vartheta + c \tag{1.55}$$

Pro minimalizaci naší kvadratické ztrátové funkce využijeme hledání extrému (položíme derivaci rovnou nule)

$$\frac{\mathrm{d}}{\mathrm{d}\vartheta} J(\vartheta) = \ldots \equiv 0 \tag{1.56}$$

a vyřešíme pro parametr ϑ.

Více obecněji pro $\boldsymbol{\Theta} \in \mathcal{R}^{n+1}$ bude naše ztrátová funkce nabývat tvaru

$$J\left(\vartheta_0,\ \vartheta_1,\ \ldots,\ \vartheta_m\right) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2 \tag{1.57}$$

dále položíme parciální derivace podle jednotlivých parametrů rovny nule (pro všechny j)

$$\frac{\partial}{\partial \vartheta_j} = \ldots \equiv 0 \tag{1.58}$$

a vyřešíme pro parametry $\vartheta_0,\ \vartheta_1,\ \ldots,\ \vartheta_m$.

Pro složení obyčejné rovnice budeme potřebovat matici $\boldsymbol{X} \in \mathcal{R}^{m\times(n+1)}$, která v řádcích obsahuje jednotlivé vektory $\left(\boldsymbol{x}^{(i)}\right)^\top$ a výstupní sloupcový vektor $\boldsymbol{y} \in \mathcal{R}^m$. Kde m je počet trénovacích vzorků (velikost trénovací množiny) a n je počet příznaků, kde $\boldsymbol{x}_0^{(i)} = 0$.

Nyní můžeme zapsat předpis pro výpočet parametrů $\boldsymbol{\Theta}$ analyticky

$$\boldsymbol{\Theta} = \left(\boldsymbol{X}^\top\boldsymbol{X}\right)^{-1}\boldsymbol{X}^\top\boldsymbol{y}. \tag{1.59}$$

Pro analytické řešení není nutné příznaky normovat, jako v případě gradientního algoritmu, na stejné měřítko, jak je popsáno v sekci 1.2.7.

1.2.9 Gradientní algoritmus vs. analytické řešení

V této sekci budou zmíněny výhody a nevýhody gradientního algoritmu versus analytického řešení. Velikost trénovací množiny m a počet příznaků n.

Gradientní algoritmus	Analytické řešení
Nutno zvolit konstantu učení α.	Není nutné volit konstantu učení α.
Potřebuje mnoho iterací.	Nepotřebuje iterovat.
Pracuje dobře, když je n velké.	Pomalé pokud je n velké.
	Nutnost vypočítat inverzi $\left(\boldsymbol{X}^\top\boldsymbol{X}\right)^{-1}$. (Složitost $\mathcal{O}\left(n^3\right)$).
Zvolit pokud $n > 10^6$.	Zvolit pokud $n < 10^4$.

1.3 Binární regrese - klasifikace

Logistická regrese je označení metody matematické statistiky zabývající se problematikou odhadu pravděpodobnosti nějakého jevu (závisle proměnné) na základě určitých známých skutečností (nezávisle proměnných), které mohou ovlivnit výskyt jevu. Událost, zda zkoumaný jev nastal, se modeluje pomocí náhodné veličiny, která nabývá hodnoty 0, pokud jev nenastal, nebo 1, pokud jev nastal.

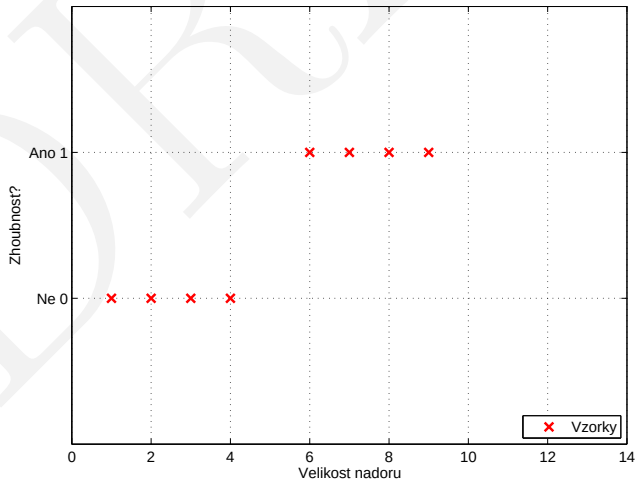
1.3.1 Klasifikace

Na úvod si uveďme několik klasifikačních příkladů:

Spam: je spam / není spam ?
Online transakce: podvodné ano / ne ?
Nádor: zhoubný / nezhoubný?

Z příkladů je zřejmé, že se jedná o rozhodnutí mezi dvěma možnostmi, tedy $y \in \{0, 1\}$, kde 0 je „negativní třída“ (nezhoubný nádor) a 1 je pozitivní třída (zhoubný nádor). Pokud máme $y \in \{0, 1\}$ jedná se o klasifikaci do dvou tříd, dále existuje klasifikace do více tříd, kde $y \in \{0, 1, 2, 3, \ldots\}$. V následujících kapitolách se budeme zabývat klasifikací do dvou tříd.

Nyní se dostáváme k problému jak vyvinout klasifikační algoritmus, což si můžeme ukázat na příkladu trénovací sady pro úlohu klasifikace nádorů. Nádor může být buď zhoubný nebo nezhoubný, tedy buďto 1 nebo 0. Dostupná data pro náš příklad jsou na obrázku 1.6.



Obrázek 1.6: Trénovací data.

Jedna z věcí, kterou bychom mohli udělat, je aplikace algoritmu lineární regrese, který známe z kapitoly 1.2. Výsledek aplikace hypotézy

$$h_{\boldsymbol{\Theta}}(\boldsymbol{x}) = \boldsymbol{\Theta}^{\top}\boldsymbol{x} \tag{1.60}$$

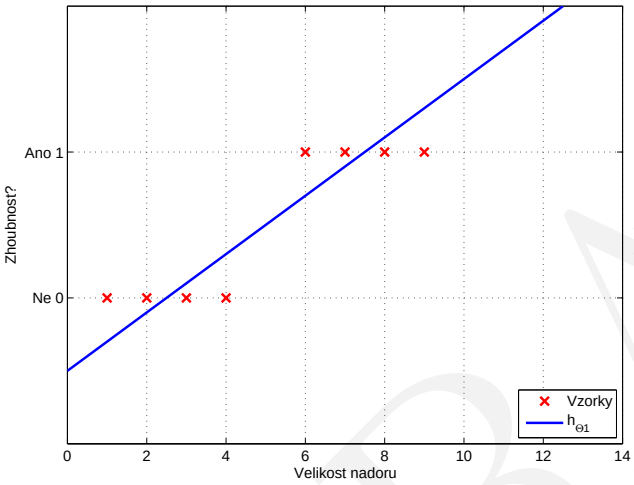
lze vidět na Obr. 1.7.

Pokud bychom chtěli udělat predikci ze získané hypotézy, tak můžeme využít klasifikátoru na základě prahování. Tedy hodnotu po jejíž překročení vzorek spadá již do druhé třídy, v případě binární regrese je to hodnota 0.5. Chceme tedy určit

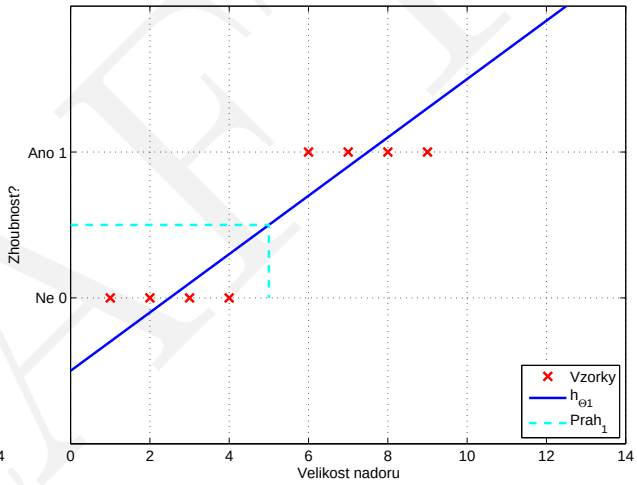
$$h_{\boldsymbol{\Theta}}(\boldsymbol{x}) \geq 0.5, \quad \text{predikuj } y = 1, \tag{1.61}$$

$$h_{\boldsymbol{\Theta}}(\boldsymbol{x}) < 0.5, \quad \text{predikuj } y = 0. \tag{1.62}$$

Tento poznatek je reprezentován na Obr. 1.8.



Obrázek 1.7: Lineární regrese.

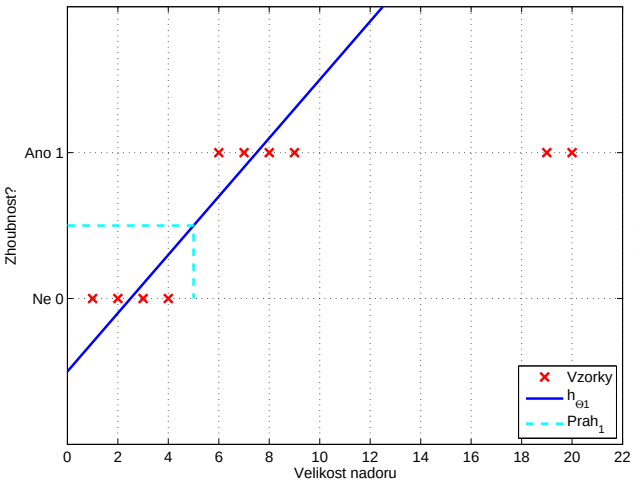


Obrázek 1.8: Lineární regrese - určení rozhodovacího prahu.

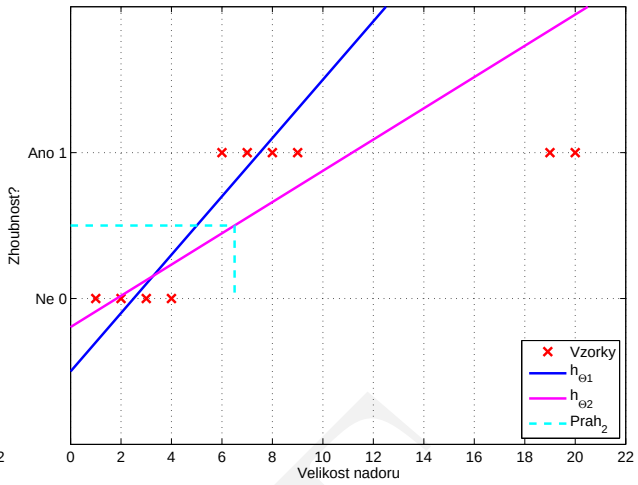
V tomto příkladě jsme si ukázali, že lineární regrese dělá něco co by se dalo považovat za řešení klasifikačního problému. Na následujícím rozšíření našeho příkladu bude ukázáno, že tomu tak není.

Rozšíříme náš příklad o další vzorky viz Obr. 1.9, kde nám přibyly dva vzorky více vpravo. Po aplikování lineární regrese, na rozšířená data, získáváme výsledek, který lze vidět na Obr. 1.10. Aktuální hypotéza $h_{\boldsymbol{\Theta}2}$ již nesplňuje podmínky definované vztahy 1.61 a 1.62, jelikož první vzorek, který má být vyhodnocen jako zhoubný by v tomto případě byl vyhodnocen jako nezhoubný, což může zásadně ovlivnit něčí život.

V tomto příkladě, lépe řečeno protipříkladě jsme si ukázali, proč je nevhodné využívat lineární regresi jako klasifikační algoritmus. Ve většině případů nefunguje správně. Pro klasifikaci na základě lineární regrese můžeme získat výsledek hypotézy $h_{\boldsymbol{\Theta}} > 1$ nebo $h_{\boldsymbol{\Theta}} < 0$, přičemž výsledek binární klasifikace je pouze $y = 0$ nebo $y = 0$. Proto bude nutné v dalších kapitolách přijít se sofistikovanějším postupem binární regrese (klasifikace), která splňuje podmínku $0 \leq h_{\boldsymbol{\Theta}} \leq 1$.



Obrázek 1.9: Rozšíření trénovací množiny dat o dva vzorky vpravo.



Obrázek 1.10: Lineární regrese - určení rozhodovacího prahu a zobrazení jeho selhání při predikci.

1.3.2 Reprezentace hypotézy

V této kapitole si ukážeme, jak správně reprezentovat hypotézu, tedy jakou funkci je vhodné použít v případě problému klasifikace.

V sekci 1.3.1 jsme zjistili, že bychom si pro binární regresní model přáli, aby platilo

$$0 \leq h_{\boldsymbol{\Theta}}(\boldsymbol{x}) \leq 1, \tag{1.63}$$

tedy aby výstupní hodnota byla mezi 0 a 1 (včetně). Pro reprezentaci hypotézy lineární regrese platí

$$h_{\boldsymbol{\Theta}}(\boldsymbol{x}) = \boldsymbol{\Theta}^{\top}\boldsymbol{x}, \tag{1.64}$$

tuto hypotézu upravíme na tvar

$$h_{\boldsymbol{\Theta}}(\boldsymbol{x}) = g\left(\boldsymbol{\Theta}^{\top}\boldsymbol{x}\right), \tag{1.65}$$

kde funkce g je definována následovně

$$g(\boldsymbol{z}) = \frac{1}{1+e^{-\boldsymbol{z}}}, \tag{1.66}$$

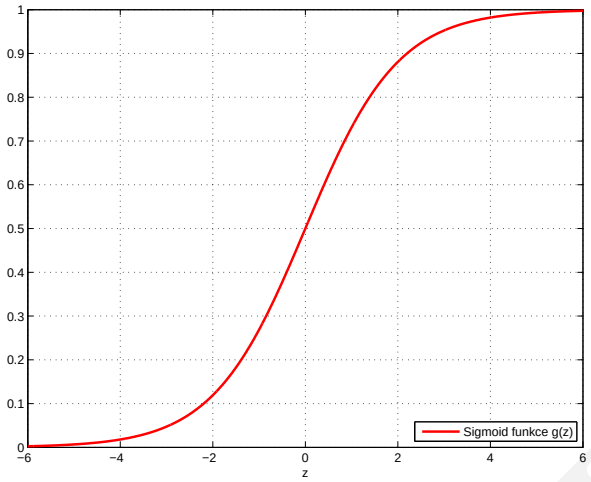
tato funkce se nazývá sigmoid funkce (Sigmoid function) nebo logistická/binární funkce (Logistic function). Jméno logistická funkce je důvod, proč vznik termín logistická/binární regrese. Pokud spojíme rovnice 1.65 a 1.66 získáme tvar

$$h_{\boldsymbol{\Theta}}(\boldsymbol{x}) = \frac{1}{1+e^{-\boldsymbol{\Theta}^{\top}\boldsymbol{x}}} \tag{1.67}$$

a průběh grafu funkce $g(\boldsymbol{z})$ lze vidět na Obr. 1.11.

Poznamenejme, že

$$\begin{aligned} \lim_{z\to+\infty} \quad g(z) \quad &= 1, \\ \lim_{z\to-\infty} \quad g(z) \quad &= 0. \end{aligned}$$



Obrázek 1.11: Sigmoidní funkce $g(z)$.

Nyní je potřeba, stejně jako dříve, přizpůsobit parametry $\boldsymbol{\Theta}$ našim datům, tedy pomocí trénovací množiny zvolit hodnoty parametrů $\boldsymbol{\Theta}$ a využít tyto parametry pro výpočet predikce.

Dále si povíme něco o interpretaci výstupu hypotézy. Hypotéza $h_{\boldsymbol{\Theta}}(\boldsymbol{x})$ = odhadnutá pravděpodobnost, že výstup $y = 1$ na základě vstupních dat $\boldsymbol{x}$.

Příklad

Pokud $\boldsymbol{x} = [x_0,\ x_1]^\top = [1, \text{velikost nádoru}]^\top$ a $h_{\boldsymbol{\Theta}}(\boldsymbol{x}) = 0.7$, což říká pacientovi, že šance, že je nádor zhoubný je 70%. Nyní naše tvrzení zapíšeme formálněji

$$h_{\boldsymbol{\Theta}}(\boldsymbol{x}) = P(y = 1|\boldsymbol{x}; \boldsymbol{\Theta}), \tag{1.68}$$

neboli, pravděpodobnost, že $y = 1$, „na základě dat $\boldsymbol{x}$“, parametrizovaná parametry $\boldsymbol{\Theta}$. V případě klasifikace víme, že výstup y může nabývat pouze hodnot 0 nebo 1, proto platí

$$P(y = 0|\boldsymbol{x}; \boldsymbol{\Theta}) + P(y = 1|\boldsymbol{x}; \boldsymbol{\Theta}) = 1, \tag{1.69}$$

neboli

$$P(y = 0|\boldsymbol{x}; \boldsymbol{\Theta}) = 1 - P(y = 1|\boldsymbol{x}; \boldsymbol{\Theta}). \tag{1.70}$$

Poznámka

Pravděpodobnost může nabývat hodnot $0 - 1$, neboli $0 - 100\%$.

1.3.3 Rozhodovací hranice

V této kapitole si povíme něco o rozhodovací hranici, abychom získali větší ponětí o výpočtu hypotézy binární regrese (klasifikace).

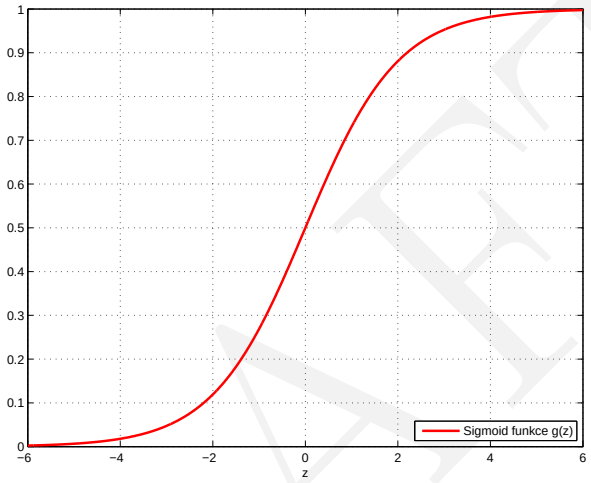
V kapitole 1.3.2 jsme si ukázali tvar hypotézy (sigmoid funkce)

$$h_{\boldsymbol{\Theta}}(\boldsymbol{x}) = g\left(\boldsymbol{\Theta}^{\top}\boldsymbol{x}\right), \tag{1.71}$$

kde

$$g(\boldsymbol{z}) = \frac{1}{1+e^{-\boldsymbol{z}}} \tag{1.72}$$

a graf lze vidět na Obr. 1.12.



Obrázek 1.12: Sigmoidní funkce $g(z)$.

Podrobněji vysvětlíme, kdy naše hypotéza bude predikovat na výstupu 0 nebo 1. Konkrétně naše hypotéza bude na výstupu predikovat $y = 1$ v případě

$$h_{\boldsymbol{\Theta}}(\boldsymbol{x}) = g\left(\boldsymbol{\Theta}^{\top}\boldsymbol{x}\right) = P(y = 1|\boldsymbol{x}; \boldsymbol{\Theta}). \tag{1.73}$$

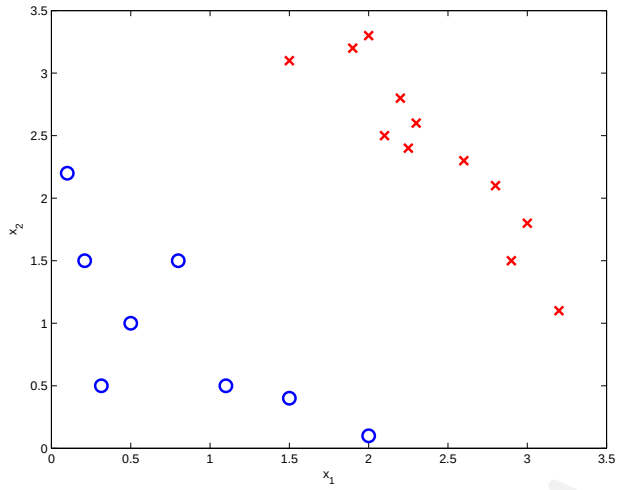
Předpokládejme, že klasifikátor predikuje výstup $y = 1$ pokud $h_{\boldsymbol{\Theta}}(\boldsymbol{x}) \geq 0.5$, což platí pro $\boldsymbol{\Theta}^{\top}\boldsymbol{x} \geq 0$ a klasifikátor predikuje výstup $y = 0$ pokud $h_{\boldsymbol{\Theta}}(\boldsymbol{x}) < 0.5$, což platí pro $\boldsymbol{\Theta}^{\top}\boldsymbol{x} < 0$.

Příklad - lineárně separabilní třídy

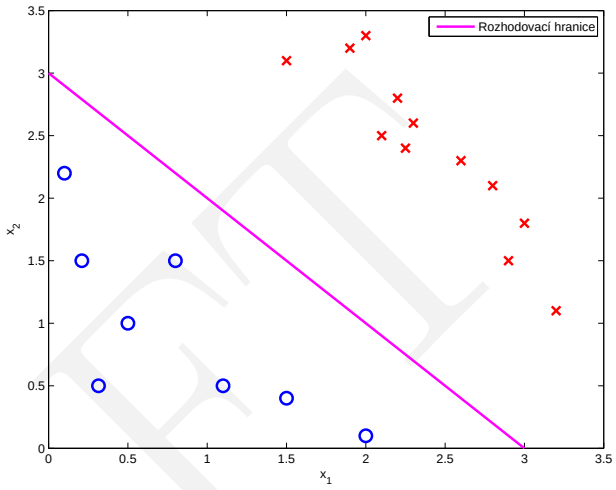
Předpokládejme, že máme trénovací sadu, která je znázorněna na Obr. 1.13 a naše hypotéza má tvar

$$h_{\boldsymbol{\Theta}}(\boldsymbol{x}) = g(\vartheta_0 + \vartheta_1 x_1 + \vartheta_2 x_2). \tag{1.74}$$

Nyní se nebudeme zabývat tím, jak přizpůsobit parametry tomuto modelu. Předpokládejme, že $\boldsymbol{\Theta} = [\vartheta_0,\ \vartheta_1,\ \vartheta 2]^\top = [-3,\ 1,\ 1]^\top$.



Obrázek 1.13: Vizualizace dvou tříd.



Obrázek 1.14: Vizualizace rozhodovací hranice.

Na příkladu si ukážeme, kdy hypotéza bude predikovat, jako výstup 0 a kdy 1. Hypotéza bude predikovat výstupní hodnotu $y = 1$ pokud

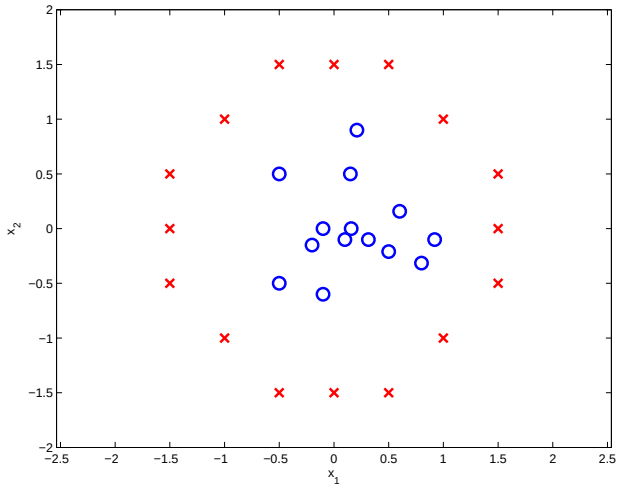
$$-3 + x_1 + x_2 \geq 0 \tag{1.75}$$

($\boldsymbol{\Theta}^\top \boldsymbol{x} \geq 0$), jinými slovy pro všechna x_1 a x_2, které splňují rovnici 1.75 bude naše hypotéza predikovat výstup $y = 1$. Řešení rovnice 1.75 je naznačeno na Obr. 1.14 a odpovídají mu červené křížky nad rozhodovací hranicí.

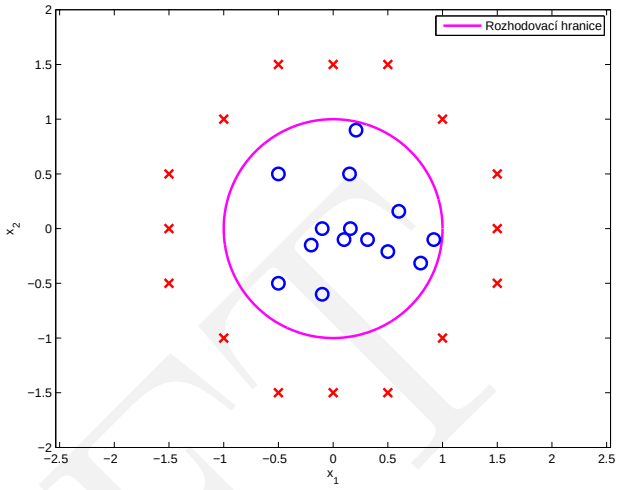
V případě, kdy $x_1 + x_2 = 3$, tak to odpovídá přesně $h_{\boldsymbol{\Theta}}(\boldsymbol{x}) = 0.5$. Jinak řečeno řešení odpovídá přesně rozhodovací hranici mezi třídami.

Příklad - lineárně neseparabilní třídy

Předpokládejme složitější příklad, kde nelze jednoduše použít binární regresi (lineární rozhodovací hranici) a musíme využít polynomiální regrese, tedy musíme využít vyšší stupně polynomů jako příznaky viz Obr. 1.15.



Obrázek 1.15: Vizualizace lineárně neseparabilních tříd.



Obrázek 1.16: Vizualizace lineárně neseparabilních tříd společně s rozhodovací hranicí.

Naše hypotéza má tvar $h_{\boldsymbol{\Theta}}(\boldsymbol{x}) = g(\vartheta_0 + \vartheta_1 x_1 + \vartheta_2 x_2 + \vartheta_3 x_1^2 + \vartheta_4 x_2^2)$, kde jsme oproti předcházejícímu příkladu (rovnice 1.74) přidali dva příznaky x_1^2 a x_2^2 a máme 5 parametrů $\vartheta_{0,\ldots,4}$. Nebudeme zabývat tím, jak vypočítat parametry $\boldsymbol{\Theta}$ (bude popsáno v následujících sekcích), řekněme, že $\boldsymbol{\Theta} = [\vartheta_0,\ \vartheta_1,\ \vartheta_2,\ \vartheta_3,\ \vartheta_4]^\top = [-1,\ 0,\ 0,\ 1,\ 1]^\top$. Což znamená, že naše hypotéza bude predikovat výstup $y = 1$ pokud $-1 + x_1^2 + x_2^2 \geq 0$ (pro směr ven od rozhodovací hranice), tato rovnice odpovídá rovnici kružnice $x_1^2 + x_2^2 = 1$ o poloměru $r = 1$. Rozhodovací hranice je vidět na Obr. 1.16.

Poznámka

Rozhodovací hranice je vlastnost hypotézy, konkrétně parametrů $\boldsymbol{\Theta}$, není to vlastnost trénovací sady. Trénovací sada je pouze využita k nalezení parametrů hypotézy $\boldsymbol{\Theta}$.

1.3.4 Ztrátová funkce

V této sekci si povíme jak vypočítat parametry $\boldsymbol{\Theta}$ pro binární regresi / klasifikaci.

Definujme problém učení s učitelem pro binární regresní model. Mějme trénovací množinu

$$\left\{\left(\boldsymbol{x}^{(1)}, y^{(1)}\right), \left(\boldsymbol{x}^{(2)}, y^{(2)}\right), \ldots, \left(\boldsymbol{x}^{(m)}, y^{(m)}\right)\right\} \tag{1.76}$$

o velikosti m vzorků, kde

$$\boldsymbol{x}^{(i)} = \left[x_0,\ x_1,\ \ldots,\ x_n\right]^\top, \tag{1.77}$$

$\boldsymbol{x}^{(i)} \in \mathrm{R}^{n+1}$ a $x_0 = 1$, $y \in \{0, 1\}$. Naše hypotéza má tvar

$$h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}\right) = \frac{1}{1 + e^{-\boldsymbol{\Theta}^\top \boldsymbol{x}}}. \tag{1.78}$$

Nyní si položme zásadní otázku, jak zvolit parametry $\boldsymbol{\Theta}$ v závislosti na naší trénovací množině?

Připomeňme si ztrátovou funkci $J\left(\boldsymbol{\Theta}\right)$ pro lineární regresi (viz sekce 1.2.3), kterou zapíšeme v následujícím tvaru

$$J\left(\boldsymbol{\Theta}\right) = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}\left(h_\Theta\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2, \tag{1.79}$$

dále můžeme ztrátovou funkci přepsat do tvaru

$$J\left(\boldsymbol{\Theta}\right) = \frac{1}{m}\sum_{i=1}^{m} Cena\left(h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}^{(i)}\right), y^{(i)}\right), \tag{1.80}$$

kde

$$Cena\left(h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}^{(i)}\right), y^{(i)}\right) = \frac{1}{2}\left(h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2. \tag{1.81}$$

Nyní lze z rovnice 1.80 lépe vidět, že ztrátová funkce je $\frac{1}{m}$ krát součet Cen přes trénovací sadu. Rovnici 1.81 lze dále zjednodušit vynecháním horních indexů

$$Cena\left(h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}\right), y\right) = \frac{1}{2}\left(h_\Theta\left(\boldsymbol{x}\right) - y\right)^2 \tag{1.82}$$

a říká nám jakou $Cenu$ zaplatíme pokud výstupní hodnota hypotézy bude $h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}\right)$ a výstupní informace od učitele bude hodnota y.

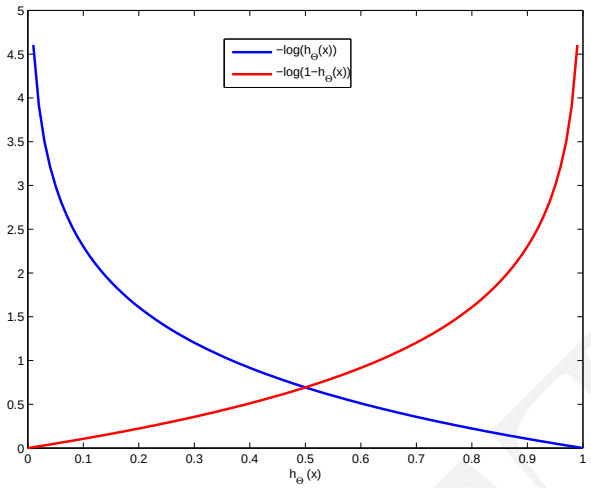
Tento předpis 1.81 pro výpočet $Ceny$ platí pro lineární regresi. Avšak v této sekci se zajímáme o binární regresi. Pokud bychom chtěli minimalizovat tuto ztrátovou funkci v rovnici 1.80, tak tato funkce může být nekonvexní vzhledem k parametrům $\boldsymbol{\Theta}$ (není zaručeno, že dosáhneme globálního minima, jako v případě konvexní ztrátové funkce).

Uveďme si ztrátovou funkci, která se využívá při binární regresi

$$Cena\left(h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}\right), y\right) = \begin{cases} -\log\left(h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}\right)\right), & \text{pro } y = 1 \\ -\log\left(1 - h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}\right)\right), & \text{pro } y = 0 \end{cases} \tag{1.83}$$

jinými slovy lze říci, že se jedná jakou ztrátu (kolik nás stojí) utrpíme pokud $y = 0$ nebo $y = 1$.

Nyní si ztrátovou funkci pro binární regresi, z rovnice 1.83, rozebereme podrobněji. Na Obr. 1.17 lze vidět průběh ztrátové funkce.



Obrázek 1.17: Průběh ztrátové funkce.

Pokud bude $y = 1$ a naše hypotéza $h_{\boldsymbol{\Theta}}(\boldsymbol{x}) = 1$, tak utrpíme ztrátu rovnu $Cena = 0$. Ale pokud se bude hodnota hypotézy zmenšovat $h_{\boldsymbol{\Theta}}(\boldsymbol{x}) \to 0$, tak se utrpěná ztráta bude zvětšovat $Cena \to \infty$. V druhém případě pokud $h_{\boldsymbol{\Theta}}(\boldsymbol{x}) = 0$ (predikce $P(y = 1|\boldsymbol{x}; \boldsymbol{\Theta}) = 0$), ale $y = 1$, tak utrpěná ztráta bude velmi velká.

Pro $y = 0$ platí obdobně druhá část rovnice 1.83.

1.3.5 Zjednodušená ztrátová funkce a gradientní metoda

Naše ztrátová funkce $J(\boldsymbol{\Theta})$ ve tvaru

$$J(\boldsymbol{\Theta}) = \frac{1}{m} \sum_{i=1}^{m} Cena\left(h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}^{(i)}\right), y^{(i)}\right), \tag{1.84}$$

kde

$$Cena\left(h_{\boldsymbol{\Theta}}(\boldsymbol{x}), y\right) = \begin{cases} -\log\left(h_{\boldsymbol{\Theta}}(\boldsymbol{x})\right), & \text{pro } y = 1 \\ -\log\left(1 - h_{\boldsymbol{\Theta}}(\boldsymbol{x})\right), & \text{pro } y = 0 \end{cases} \tag{1.85}$$

poznamenejme že $y = 0$ nebo $y = 1$ (pokaždé, žádná jiná možnost není).

Naší *Cenu* z rovnice 1.85 můžeme zapsat v jedné rovnici, která má tvar

$$Cena\left(h_{\boldsymbol{\Theta}}(\boldsymbol{x}), y\right) = -y \log\left(h_{\boldsymbol{\Theta}}(\boldsymbol{x})\right) - (1 - y) \log\left(1 - h_{\boldsymbol{\Theta}}(\boldsymbol{x})\right), \tag{1.86}$$

pokud $y = 1$, tak rovnice 1.86 přechází na první část rovnice 1.85 a pokud $y = 0$, tak rovnice 1.86 přechází na druhou část rovnice 1.85.

Po spojení a úpravě rovnic 1.86 a 1.84 získáme výsledný tvar

$$J(\boldsymbol{\Theta}) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log\left(h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \log\left(1 - h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}^{(i)}\right)\right)\right]. \tag{1.87}$$

Cílem je nalézt takové parametry $\boldsymbol{\Theta}$, které minimalizují ztrátovou funkci J

$$\min_{\boldsymbol{\Theta}} J\left(\boldsymbol{\Theta}\right), \tag{1.88}$$

a následně pro vypočítáme predikci naší hypotézy (na základě nového vstupu x)

$$h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}\right) = \frac{1}{1 + e^{-\boldsymbol{\Theta}^{\top}\boldsymbol{x}}}. \tag{1.89}$$

Nyní můžeme konečně zapsat celý gradientní algoritmus, opakuj

$$\vartheta_j = \vartheta_j - \alpha \frac{\partial}{\partial \vartheta_j} J\left(\boldsymbol{\Theta}\right), \tag{1.90}$$

po každé iteraci aktualizuj všechny parametry ϑ_j najednou dokud algoritmus nedokonverguje.

Poznámka

V rovnici 1.90 bude využit tvar

$$\frac{\partial}{\partial \vartheta_j} J\left(\boldsymbol{\Theta}\right) = \frac{1}{m} \sum_{i=1}^{m} \left(h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right) \boldsymbol{x}_j^{(i)}. \tag{1.91}$$

a proto je výhodnější psát rovnici 1.90 ve tvaru

$$\vartheta_j = \vartheta_j - \alpha \sum_{i=1}^{m} \left(h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right) \boldsymbol{x}_j^{(i)}. \tag{1.92}$$

Algoritmus vypadá totožně jako v případě lineární regrese (viz 1.50) s rozdílnou hypotézou $h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}\right)$, v případě lineární regrese

$$h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}\right) = \boldsymbol{\Theta}^{\top}\boldsymbol{x} \tag{1.93}$$

a v případě binární regrese se jedná o

$$h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}\right) = \frac{1}{1 + e^{-\boldsymbol{\Theta}^{\top}\boldsymbol{x}}}. \tag{1.94}$$

Poznámka

Při výpočtu parametrů $\boldsymbol{\Theta}$ pomocí gradientního algoritmu v případě binární regrese je také vhodné využít Features scaling, jako v případě lineární regrese (viz sekce 1.2.7).

1.3.6 Pokročilé algoritmy optimalizace

V této sekci si uvedeme příklady několika pokročilejších algoritmů optimalizace, které jsou lépe využitelné pro velké problémy strojového učení, kde je například velký počet příznaků.

Připomeňme náš optimalizační problém, máme ztrátovou funkci $J(\boldsymbol{\Theta})$ a chceme jí minimalizovat

$$\min_{\boldsymbol{\Theta}} J(\boldsymbol{\Theta}). \tag{1.95}$$

Musíme tedy znát rovnice pro výpočet $J(\boldsymbol{\Theta})$ a $\frac{\partial}{\partial \vartheta_j} J(\boldsymbol{\Theta})$ následně můžeme využít zmíněný gradientní algoritmus, nebo nějakou z pokročilejších optimalizačních metod jako jsou například

Název algoritmu	Výhody	Nevýhody
Conjugate gradient	Nemusí se volit	Více složité
BFGS	konstanta učení α.	
L-BFGS	Rychlejší než gradientní metoda.	

Tabulka 1.3: Příklady pokročilejších optimalizačních algoritmů.

Příklad

Nyní si uvedeme příklad společně s jeho implementací v MATLABu. Mějme vektor parametrů $\boldsymbol{\Theta} = [\vartheta_1,\ \vartheta_2]^\top$ a ztrátovou funkci $J(\boldsymbol{\Theta}) = (\vartheta_1 - 5)^2 + (\vartheta_2 - 5)^2$, dále vypočteme parciální derivace ztrátové funkce

$$\frac{\partial}{\partial \vartheta_1} J(\boldsymbol{\Theta}) = 2 \cdot (\vartheta_1 - 5), \tag{1.96}$$

$$\frac{\partial}{\partial \vartheta_2} J(\boldsymbol{\Theta}) = 2 \cdot (\vartheta_2 - 5). \tag{1.97}$$

Program v prostředí MATLAB by mohl vypadat následovně, nejprve je nutné si vytvořit funkci pro výpočet ztrátové funkce, která bude vracet hodnotu ztrátové funkce a jednotlivé gradienty.

```
1 function [jVal, grad] = costFunction(theta)
2
3 jVal = (theta(1, 1) - 5)^2 + (theta(2, 1) - 5)^2;
4
5 grad = zeros(size(theta, 1), 1);
6 grad(1, 1) = 2 * (theta(1, 1) - 5);
7 grad(2, 1) = 2 * (theta(2, 1) - 5);
8
9 end
```

Optimalizaci ztrátové funkce (hledání minima) můžeme spustit následovně.

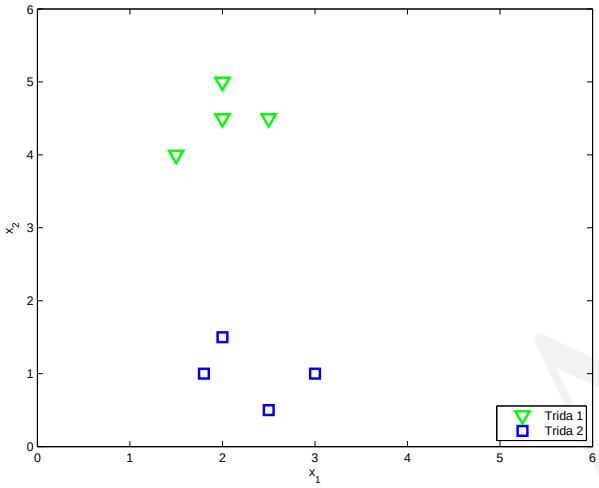
```
initialTheta = [1; 2];
options = optimset('GradObj', 'on', 'MaxIter', 100);
[optTheta, functionVal, exitFlag] = ...
    fminunc(@costFunction, initialTheta, options);
```

Výsledné hodnoty pro $\vartheta_{1,2} = 5$, což lze pro tento jednoduchý případ vidět z rovnic 1.96 a 1.97. Jak lze vidět z příkladu, nebylo nutné nastavovat parametr α, funkce *fminunc* v MATLABu si jej nastavila sama.

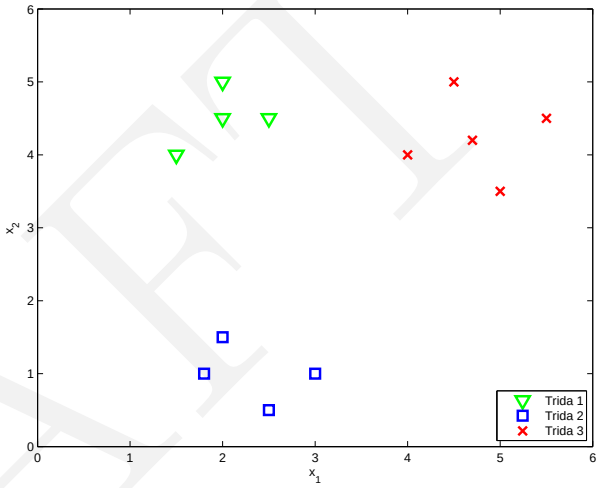
1.3.7 Klasifikace do více tříd

V této sekci si povíme o klasifikaci do více tříd. Tento problém lze uchopit tak, že budeme klasifikovat jednu třídu proti všem ostatním, náš klasifikační algoritmus by se dal nazvat jeden proti všem.

Nejdříve si popišme náš klasifikační problém, lze ho demonstrovat na příkladu klasifikace emailů. Představme si, že bychom rádi příchozí emaily rozdělovali do složek, nebo označovali podle skupiny lidí, od kterých nám email přišel, například pracovní emaily, emaily od kamarádů, emaily od rodiny a emaily od známých, s kterými sdílíme koníčky. Neboli chceme klasifikovat do tříd $y = \{1,\ 2,\ 3,\ 4\}$. Na Obr. 1.18 a 1.19 je vidět rozdíl v klasifikaci do dvou a více tříd.



Obrázek 1.18: Klasifikace do dvou tříd.



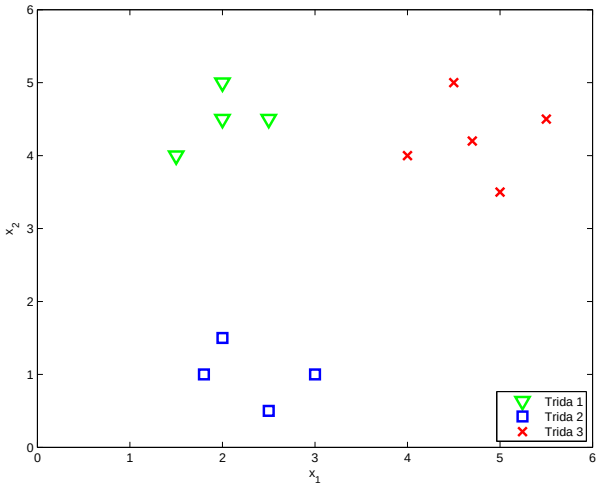
Obrázek 1.19: Klasifikace do více tříd.

Z předešlých sekcí víme jak klasifikovat do dvou tříd. Nyní tento přístup použijeme při řešení problému klasifikace do více tříd. Na Obr. 1.20 je znázorněno rozložení tříd a následně na Obr. 1.21 je ukázána klasifikace třídy 1 proti třídě, která je zkonstruována spojením všech ostatních tříd (v našem případě tříd 2 a 3). Na dalších Obr. 1.22 a 1.23 jsou naznačeny zbylé kombinace.Pro jednotlivé klasifikace je nutné vytvořit hypotézu

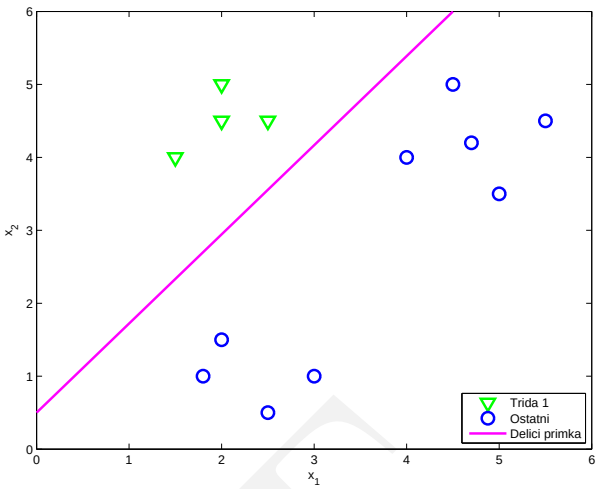
$$h_{\boldsymbol{\Theta}}^{(i)}(\boldsymbol{x}), \quad i \in \{1,\ 2,\ 3\}, \tag{1.98}$$

která odpovídá pravděpodobnostem

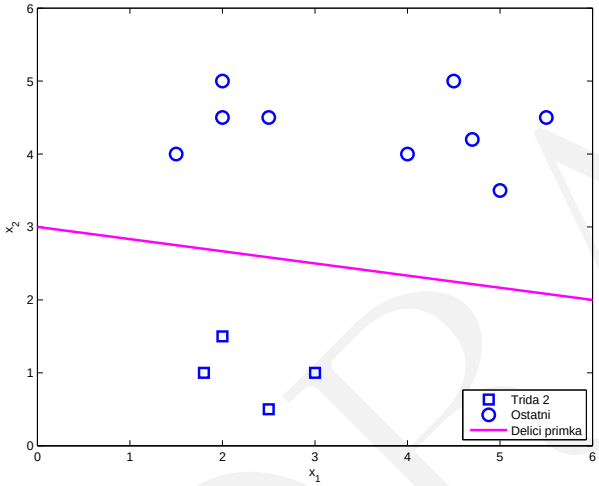
$$h_{\boldsymbol{\Theta}}^{(i)}(\boldsymbol{x}) = P(y = i|\boldsymbol{x}; \boldsymbol{\Theta}). \tag{1.99}$$



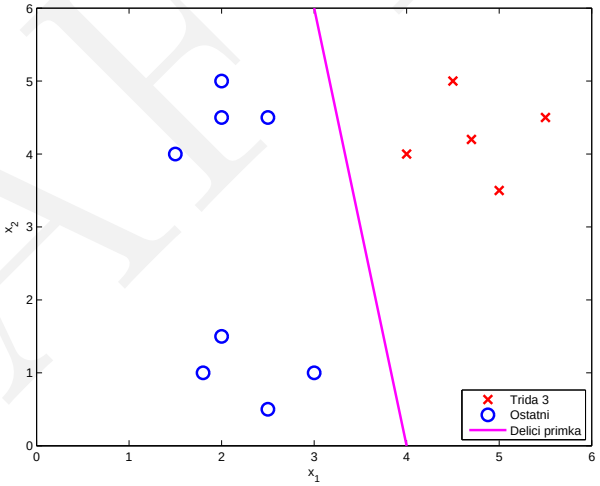
Obrázek 1.20: Rozložení tříd $y = \{1,\ 2,\ 3\}$.



Obrázek 1.21: Klasifikace třídy 1.



Obrázek 1.22: Klasifikace třídy 2.



Obrázek 1.23: Klasifikace třídy 3.

Shrnutí

Nejdříve je nutné natrénovat binární klasifikátory $h_{\boldsymbol{\Theta}}^{(i)}(\boldsymbol{x})$ pro všechny třídy i pro predikci pravděpodobnosti, že vzorek bude patřit do třídy $y = i$. Následně pro nový vstup x klasifikovat podle všech hypotéz a vybrat takovou třídu i, která maximalizuje pravděpodobnost hypotézy

$$\max_i h_{\boldsymbol{\Theta}}^{(i)}(\boldsymbol{x})\,. \tag{1.100}$$

1.4 Regularizace

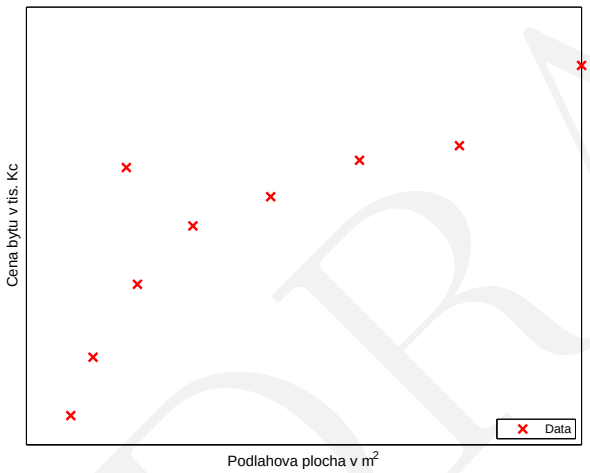
1.4.1 Problém přetrénování

Nyní když známe algoritmy lineární a binární regrese, které se hodí pro řešení mnoha úloh strojového učení, musíme si říci něco o problému zvaném přetrénování. Tento problém způsobuje, že dané algoritmy ve výsledku fungují velmi špatně.

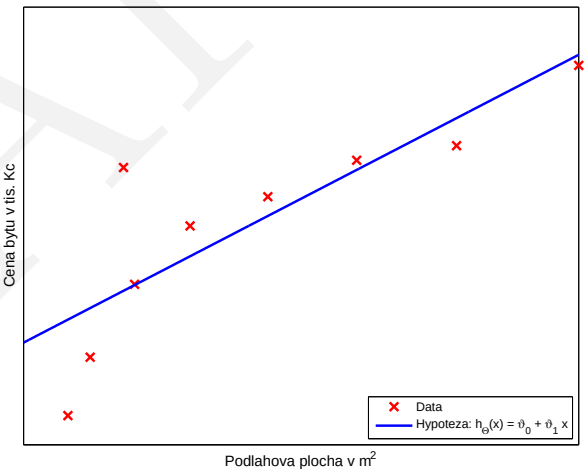
V této sekci si vysvětlíme v čem problém přetrénování spočívá a v následujících sekcích bude popsán postup jak se s daným problémem vypořádat.

Vraťme se k našemu problému predikce ceny velikosti bytu, ve kterém jsme využili algoritmu lineární regrese.

Na Obr. 1.24 jsou znázorněná trénovací data. Pokud využijeme lineární regresi společně s hypotézou $h_{\boldsymbol{\Theta}}(\boldsymbol{x}) = \vartheta_0 + \vartheta_1 x$, tak získáme predikci ve tvaru, který lze vidět na Obr. 1.25. V tomto případě naše zvolená hypotéza $h_{\boldsymbol{\Theta}}(\boldsymbol{x})$ nerespektuje dostatečně tvar našich trénovacích dat. Přesnější anglický termín pro takto nastalou situaci zní *underfit* nebo *high bias*, což se dá vysvětlit jako problém, kdy se snažíme použít regresi na trénovací data s málo komplexní hypotézou. Tedy hypotéza nemůže dostatečně přesně vystihovat tvar dat (nezobecňuje).



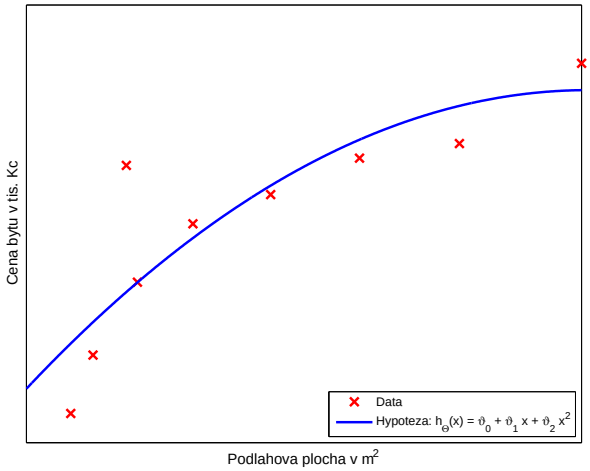
Obrázek 1.24: Trénovací data.



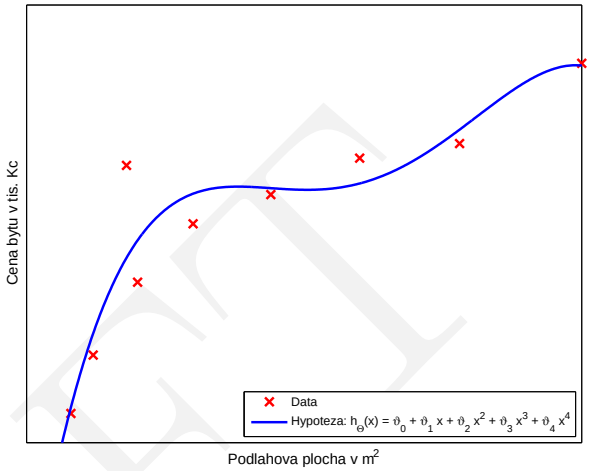
Obrázek 1.25: Lineární regrese (*underfit*, *high bias*).

Na Obr. 1.26 lze vidět regresi s hypotézou ve tvaru $h_{\boldsymbol{\Theta}}(\boldsymbol{x}) = \vartheta_0 + \vartheta_1 x + \vartheta_2 x^2$, která velmi dobře zobecňuje trénovací data a proto by byla v tomto případě nejlepší volbou.

Dále na Obr. 1.27 lze vidět regresi s hypotézou $h_{\boldsymbol{\Theta}}(\boldsymbol{x}) = \vartheta_0 + \vartheta_1 x + \vartheta_2 x^2 + \vartheta_2 x^3 + \vartheta_2 x^4$, jejíž chování je velmi „divoké“ a nezobecňuje trénovací data. V tomto případě se snaží hypotéza co nejvíce respektovat trénovací data za cenu ztráty obecnosti. Tento problém se nazývá přetrénování, anglicky *overfitting* nebo *high variance*.



Obrázek 1.26: Regrese pomocí hypotézy ve tvaru polynomu druhého stupně (*just right*).



Obrázek 1.27: Regrese pomocí hypotézy ve tvaru polynomu čtvrtého stupně (*overfitting*, *high variance*).

Přetrénování (*overfitting*):

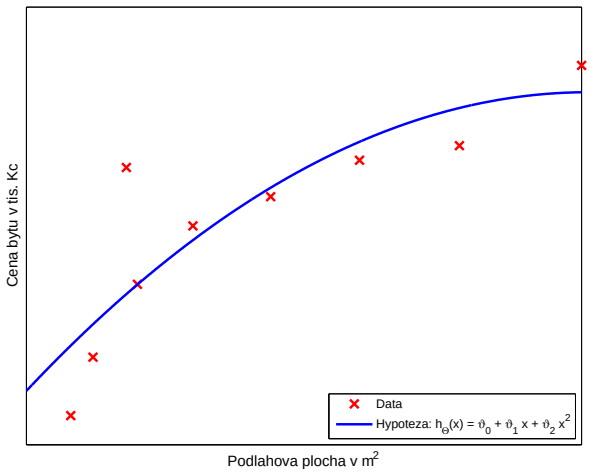
V případě, že máme příliš mnoho příznaků, hypotéza může velmi dobře respektovat trénovací data $\left(J(\boldsymbol{\Theta}) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2 \approx 0\right)$, ale nebude schopna správně klasifikovat nově příchozí vzorky. Nebude schopna generalizovat, tedy predikce na nově příchozích vzorcích selže.

Možnosti odstranění přetrénování:

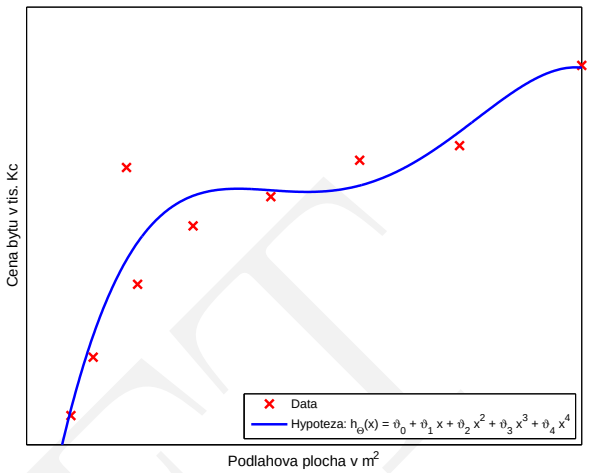
1. Redukovat počet příznaků.
   - Manuálně zvolit příznaky, které budou zachovány.
   - Automatická redukce příznaků například PCA.
2. Regularizace.
   - Zachovat všechny příznaky, ale redukovat velikost / váhu parametrů ϑ_j.
   - Funguje správně pokud máme velké množství příznaků, kde každý malou měrou přispívá k predikci y.

1.4.2 Ztrátová funkce

V sekci 1.4.1 jsme si ukázali jak lze pomocí kvadratické hypotézy $h_{\boldsymbol{\Theta}}(\boldsymbol{x}) = \vartheta_0 + \vartheta_1 x + \vartheta_2 x^2$ dobře zobecnit naše trénovací data. Dále již víme, že hypotéza ve tvaru $h_{\boldsymbol{\Theta}}(\boldsymbol{x}) = \vartheta_0 + \vartheta_1 x + \vartheta_2 x^2 + \vartheta_2 x^3 + \vartheta_2 x^4$ nezobecňuje naší trénovací množinu dat dobře.



Obrázek 1.28: Regrese pomocí hypotézy ve tvaru polynomu druhého stupně (*just right*).



Obrázek 1.29: Regrese pomocí hypotézy ve tvaru polynomu čtvrtého stupně (*overfitting, high variance*).

Předpokládejme, že budeme penalizovat parametry ϑ_3 a ϑ_4 (jejich hodnoty budou malé). V rovnici 1.101 je ukázána naše známá optimalizace

$$\min_{\boldsymbol{\Theta}} \frac{1}{2m} \sum_{i=1}^{m} \left(h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)} \right)^2, \tag{1.101}$$

kterou můžeme upravit na tvar

$$\min_{\boldsymbol{\Theta}} \frac{1}{2m} \sum_{i=1}^{m} \left(h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)} \right)^2 + 1000\vartheta_3^2 + 1000\vartheta_4^2, \tag{1.102}$$

kde hodnoty 1000 jsou pouze ilustrativní a reprezentují velká čísla (velké váhy). Tedy pokud bychom chtěli minimalizovat rovnici 1.102, tak to lze za předpokladu, že ϑ_3 a ϑ_4 budou malá čísla. V případě, že $\vartheta_3 \approx 0$ a $\vartheta_4 \approx 0$, tak získáme opět náš minimalizační problém jako v rovnici 1.101. Ještě větší dopad to má na tvar hypotézy $h_{\boldsymbol{\Theta}}(\boldsymbol{x}) = \vartheta_0 + \vartheta_1 x + \vartheta_2 x^2 + \vartheta_2 x^3 + \vartheta_2 x^4$, která dříve vedla k přetrénování. Ta v případě, že $\vartheta_3 \approx 0$ a $\vartheta_4 \approx 0$ přejde na tvar $h_{\boldsymbol{\Theta}}(\boldsymbol{x}) = \vartheta_0 + \vartheta_1 x + \vartheta_2 x^2$. Tímto se zbavíme problému přetrénování a získáme tvar hypotézy, který dobře zobecňuje trénovací data.

Regularizace

Čím jsou menší hodnoty parametrů $\vartheta_0, \vartheta_1, \ldots, \vartheta_n$, tím „jednodušší“ tvar hypotézy získáme a tím méně je tvar hypotézy náchylný k přetrénování. Zapsáno rovnicí

$$J(\boldsymbol{\Theta}) = \frac{1}{2m} \left(\sum_{i=1}^{m} \left(h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)} \right)^2 + \lambda \sum_{j=1}^{n} \vartheta_j^2 \right), \tag{1.103}$$

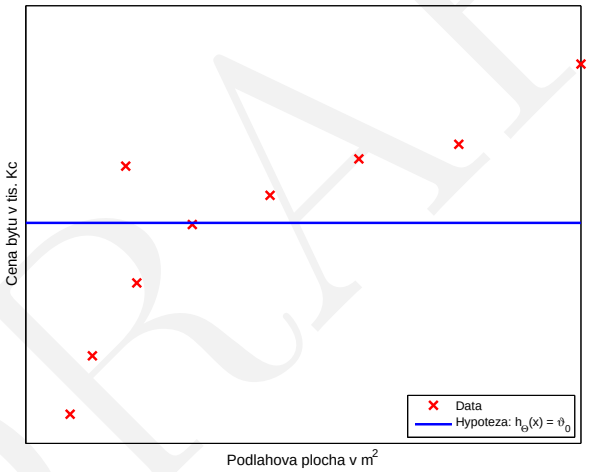
kde je nejdříve podotknout, že nová suma s indexem j má jinou mez n než první suma. To je způsobeno tím, že se regularizace vztahuje pouze na parametry $\vartheta_1, \vartheta_2, \ldots, \vartheta_m$, tedy parametr ϑ_0 není regularizován, protože je vždy násoben jedničkou, nepřijde do kontaktu s příznaky vzorků. Dále parametr λ je takzvaný regularizační parametr, který kontroluje vyvážení mezi tím, jak hodně bude regularizace ovlivňovat původní ztrátovou funkci $J(\boldsymbol{\Theta})$. Regularizace je dána vztahem

$$\lambda \sum_{j=1}^{n} \vartheta_j^2. \tag{1.104}$$

Našim cílem je jako dříve minimalizovat ztrátovou funkci

$$\min_{\boldsymbol{\Theta}} J(\boldsymbol{\Theta}) \tag{1.105}$$

Co se stane, pokud bude regularizační parametr λ příliš velký? Řekněme, že $\lambda = 10^{10}$. Poté hypotéza ve tvaru $h_{\boldsymbol{\Theta}}(\boldsymbol{x}) = \vartheta_0 + \vartheta_1 x + \vartheta_2 x^2 + \vartheta_2 x^3 + \vartheta_2 x^4$ přejde na tvar $h_{\boldsymbol{\Theta}}(\boldsymbol{x}) = \vartheta_0$ a výsledný průběh lze vidět na Obr. 1.30



Obrázek 1.30: Použití regularizace k získání *underfit* hypotézy.

Z uvedeného vyplývá, že je nutné volit regularizační parametr λ velmi opatrně. S velkým λ dosáhneme *underfit* problému a s malým λ neodstraníme *overfit* problém.

1.4.3 Regularizace lineární regrese

Pro lineární regresi máme odvozen gradientní algoritmus a analytické řešení. V této sekci oba algoritmy zobecníme s využitím regularizace.

Regularizace gradientního algoritmu lineární regrese

Máme naší optimalizační úlohu lineární regrese společně s regularizací

$$J\left(\boldsymbol{\Theta}\right) = \frac{1}{2m}\left(\sum_{i=1}^{m}\left(h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2 + \lambda\sum_{j=1}^{n}\vartheta_j^2\right), \tag{1.106}$$

snažíme se minimalizovat ztrátovou funkci

$$\min_{\boldsymbol{\Theta}} J\left(\boldsymbol{\Theta}\right) \tag{1.107}$$

Připomeňme si dříve představený gradientní algoritmus lineární regrese. Opakuj

$$\vartheta_j = \vartheta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)x_j^{(i)}, \quad j = 0, 1, \ldots, n. \tag{1.108}$$

Tento algoritmus je možno rozdělit do dvou kroků. Rozdělit výpočet pro ϑ_0 a $\vartheta_{1,\ldots,n}$. Opakuj

$$\vartheta_0 = \vartheta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)x_0^{(i)}, \tag{1.109}$$

$$\vartheta_j = \vartheta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)x_j^{(i)}, \quad j = 1, 2, \ldots, n. \tag{1.110}$$

Doposud na algoritmu nebylo nic změněno, pouze byl rozepsán do dvou kroků. Protože víme, že na parametr ϑ_0 se regularizace nevztahuje, proto můžeme algoritmus pozměnit a přidat regularizační člen. Následně bude mít gradientní algoritmus tvar. Opakuj

$$\vartheta_0 = \vartheta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)x_0^{(i)}, \tag{1.111}$$

$$\vartheta_j = \vartheta_j - \alpha\left(\frac{1}{m}\sum_{i=1}^{m}\left(h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)x_j^{(i)} + \frac{\lambda}{m}\vartheta_j\right), \quad j = 1, 2, \ldots, n, \tag{1.112}$$

kde platí

$$\frac{\partial}{\partial\vartheta_0} J\left(\boldsymbol{\Theta}\right) = \left(\frac{1}{m}\sum_{i=1}^{m}\left(h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)x_0^{(i)}\right), \tag{1.113}$$

$$\frac{\partial}{\partial\vartheta_j} J\left(\boldsymbol{\Theta}\right) = \left(\frac{1}{m}\sum_{i=1}^{m}\left(h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)x_j^{(i)} + \frac{\lambda}{m}\vartheta_j\right), \quad j = 1, 2, \ldots, n. \tag{1.114}$$

Rovnice 1.112 lze přepsat do tvaru

$$\vartheta_j = \vartheta_j\left(1 - \alpha\frac{\lambda}{m}\right) - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)x_j^{(i)}, \tag{1.115}$$

kde platí

$$1 - \alpha\frac{\lambda}{m} < 1. \tag{1.116}$$

Regularizace analytického řešení lineární regrese

Připomeňme si

$$\boldsymbol{X} = \left[\left(\boldsymbol{x}^{(1)}\right)^\top, \ldots, \left(\boldsymbol{x}^{(m)}\right)^\top\right]^\top, \quad \boldsymbol{X} \in \mathcal{R}^{m\times(n+1)} \tag{1.117}$$

a

$$\boldsymbol{y} = \left[y^{(1)}, \ldots, y^{(m)}\right]^\top, \quad \boldsymbol{y} \in \mathcal{R}^m. \tag{1.118}$$

Analytické řešení pro výpočet $\boldsymbol{\Theta}$ má tvar

$$\boldsymbol{\Theta} = \left(\boldsymbol{X}^\top \boldsymbol{X}\right)^{-1} \boldsymbol{X}^\top y. \tag{1.119}$$

Cílem je minimalizovat ztrátovou funkci

$$\min_{\boldsymbol{\Theta}} J\left(\boldsymbol{\Theta}\right). \tag{1.120}$$

Pokud využijeme regularizaci, tak naše analytické řešení přejde na tvar

$$\boldsymbol{\Theta} = \left(\boldsymbol{X}^\top \boldsymbol{X} + \lambda \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}\right)^{-1} \boldsymbol{X}^\top y \tag{1.121}$$

kde nová matice má velikost $\mathcal{R}^{(n+1)\times(n+1)}$

Rozšíření

Předpokládejme, že $m \leq n$, kde m je počet vzorků a n je počet příznaků a

$$\boldsymbol{\Theta} = \left(\boldsymbol{X}^\top \boldsymbol{X}\right)^{-1} \boldsymbol{X}^\top y, \tag{1.122}$$

pak je matice $\boldsymbol{X}^\top \boldsymbol{X}$ neinvertovatelná / singulární.

Pro výpočet lze v tomto případě využít pseudoinverze, konkrétně v MATLABu k tomuto účelu slouží funkce *pinv()*.

```
1 Theta = pinv(X' * X) * X' * y;
```

Pokud $\lambda > 0$

$$\boldsymbol{\Theta} = \left(\boldsymbol{X}^\top \boldsymbol{X} + \lambda \underbrace{\begin{bmatrix} 0 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{bmatrix}}_{\text{Matice musí být invertovatelná}}\right)^{-1} \boldsymbol{X}^\top y \tag{1.123}$$

1.4.4 Regularizace binární regrese

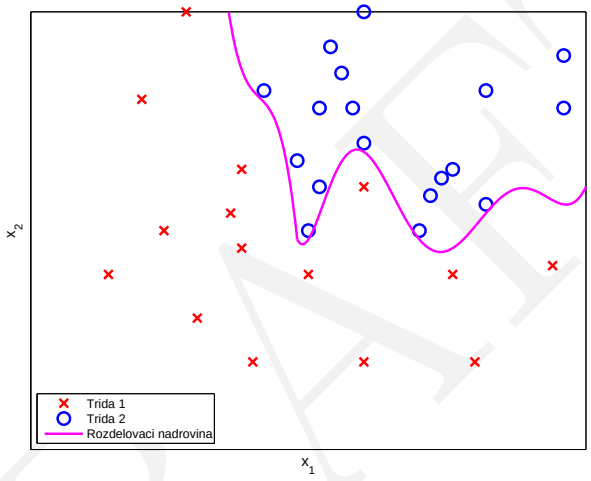
Pro binární regresi jsme dříve mluvili o dvou typech optimalizace. Mluvili jsme o tom, jak použít gradientní algoritmus a jak použít pokročilé metody optimalizace. V této sekci si ukážeme, jak tyto přístupy upravit, aby využily regularizace.

Regularizace binární regrese

Dříve jsme si ukázali, že binární regrese je náchylná na přetrénování v případě, kdy zvolíme vysoký stupeň polynomu reprezentující naší hypotézu, například

$$h_{\boldsymbol{\Theta}}(\boldsymbol{x}) = g\left(\vartheta_0 + \vartheta_1 x_1 + \vartheta_2 x_1^2 + \vartheta_3 x_1^2 x_2 + \vartheta_4 x_1^2 x_2^2 + \vartheta_5 x_1^2 x_2^3 + \ldots\right), \tag{1.124}$$

pro takovouto složitou hypotézu můžeme dostat dělící křivku velmi variabilní a velmi respektující rozdělení trénovacích dat, tedy křivku, která nedostatečně zobecňuje.



Obrázek 1.31: Možný výsledek s použitím hypotézy z rovnice 1.124.

Ztrátová funkce pro binární regresi má tvar

$$J(\boldsymbol{\Theta}) = -\left[\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log\left(h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \log\left(1 - h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}^{(i)}\right)\right)\right] \tag{1.125}$$

a její modifikací získáme regularizovaný tvar ztrátové funkce

$$J(\boldsymbol{\Theta}) = -\left[\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log\left(h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \log\left(1 - h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}^{(i)}\right)\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} \vartheta_j^2 \tag{1.126}$$

přidaná suma je jen do n, jelikož se neregularizuje složka ϑ_0, která je vždy rovna jedné a není v součinu se žádnou proměnnou x, y. Tvar ztrátové funkce společně s regularizací bude naší rozdělovací nadrovinu více zobecňovat a pomůže nám řešit problém přetrénování.

Implementace bude provedena následovně: opakuj

$$\vartheta_0 = \vartheta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right) x_0^{(i)} \tag{1.127}$$

$$\vartheta_j = \vartheta_j - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}\left(h_{\boldsymbol{\Theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right) x_j^{(i)} + \frac{\lambda}{m}\vartheta_j\right], \qquad j = 1, 2, 3, \ldots, n, \tag{1.128}$$

kde

$$h_{\mathbf{\Theta}}(\boldsymbol{x}) = \frac{1}{1 + e^{-\mathbf{\Theta}^\top \boldsymbol{x}}}. \tag{1.129}$$

Z rovnice 1.128 lze vidět, že platí

$$\frac{\partial}{\partial \vartheta_j} J(\mathbf{\Theta}) = \frac{1}{m} \sum_{i=1}^{m} \left(h_{\mathbf{\Theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)} \right) x_j^{(i)} + \frac{\lambda}{m} \vartheta_j. \tag{1.130}$$

Regularizace pokročilé optimalizace

Nyní bude nastíněno jak využit pokročilé optimalizace v MATLABu. Nechť náš vektor parametrů $\mathbf{\Theta}$ odpovídá

$$\mathbf{\Theta} = [\vartheta_0, \vartheta_1, \ldots, \vartheta_n]^\top = [theta(1), theta(2), \ldots, theta(n+1)] \tag{1.131}$$

(z důvodu, že MATLAB indexuje od jedničky).

```
function [jVal, gradient] = costFunction(theta)
    jVal = [code to compute J(Theta)];
```

$$J(\mathbf{\Theta}) = \left[-\frac{1}{m} \sum_{i=1}^{m} y^{(i)} \log\left(h_{\mathbf{\Theta}}\left(\boldsymbol{x}^{(i)}\right) \right) + \left(1 - y^{(i)}\right) \log\left(1 - h_{\mathbf{\Theta}}\left(\boldsymbol{x}^{(i)}\right)\right) \right] + \frac{\lambda}{2m} \sum_{j=1}^{n} \vartheta_j^2 \tag{1.132}$$

dále výpočet gradientu

```
gradient(1) = [code to compute gradient(1)];
```

$$\frac{\partial}{\partial \vartheta_0} J(\mathbf{\Theta}) = \frac{1}{m} \sum_{i=1}^{m} \left(h_{\mathbf{\Theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)} \right) x_0^{(i)}, \tag{1.133}$$

```
gradient(2) = [code to compute gradient(2)];
```

$$\frac{\partial}{\partial \vartheta_1} J(\mathbf{\Theta}) = \left[\frac{1}{m} \sum_{i=1}^{m} \left(h_{\mathbf{\Theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)} \right) x_1^{(i)} \right] + \frac{\lambda}{m} \vartheta_1, \tag{1.134}$$

```
gradient(3) = [code to compute gradient(3)];
```

$$\frac{\partial}{\partial \vartheta_2} J(\mathbf{\Theta}) = \left[\frac{1}{m} \sum_{i=1}^{m} \left(h_{\mathbf{\Theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)} \right) x_2^{(i)} \right] + \frac{\lambda}{m} \vartheta_2, \tag{1.135}$$

```
gradient(n+1) = [code to compute gradient(n+1)];
```

$$\ldots \tag{1.136}$$

1.5 Učení s učitelem

Učení s učitelem je metoda strojového učení pro učení funkce z trénovacích dat. Trénovací data sestávají ze dvojic vstupních objektů (typicky vektorů příznaků) a požadovaného výstupu. Výstup funkce může být spojitá hodnota (při regresi) anebo může předpovídat označení třídy vstupního objektu (při klasifikaci). Úloha algoritmu učení je předpovídat výstupní hodnotu funkce pro každý platný vstupní objekt poté, co zpracuje trénovací příklady (tj. dvojice vstup a požadovaný výstup). Aby to dokázal, musí algoritmus zobecnit prezentovaná data na nové situace (vstupy) „smysluplným“ způsobem. Analogická úloha v lidské a zvířecí psychologii se často nazývá učení konceptů.

Přetrénování (anglicky *overfitting*) je stav, kdy je systém příliš přizpůsoben množině trénovacích dat, ale nemá schopnost generalizace a selhává na testovací (validační) množině dat. To se může stát například při malém rozsahu trénovací množiny nebo pokud je systém příliš komplexní (například příliš mnoho skrytých neuronů v neuronové síti). Řešením je zvětšení trénovací množiny, snížení složitosti systému nebo různé techniky regularizace, zavedení omezení na parametry systému, které v důsledku snižuje složitost popisu naučené funkce, nebo předčasné ukončení (průběžné testování na validační množině a konec učení ve chvíli, kdy se chyba na této množině dostane do svého minima).

Při učení se používají trénovací data (nebo trénovací množina), testovací data a často validační data.

Příklady algoritmů: rozhodovací stromy, AdaBoost, náhodné rozhodovací lesy, metoda nejbližšího souseda, metoda K-nejbližších sousedů, lineární regrese, Bayesův klasifikátor, neuronové sítě, binární regrese, support vector machine (SVM), atd.

Metoda minimální vzdálenosti

V každé třídě se snažíme najít typický prvek, který nazýváme etalon. Obraz klasifikovaného objektu pak porovnáme s etalony všech tříd a zařadíme jej do třídy, jejíž etalon je klasifikovanému obrazu nejpodobnější. Protože obrazy etalonu i klasifikovaného objektu jsou n-rozměrné vektory, můžeme podobnost definovat jako Euklidovu vzdálenost mezi nimi.

Metoda nejbližšího souseda

Klasifikace podle nejbližších sousedů spadá mezi neparametrické metody klasifikace. Tyto metody jsou založeny na podstatně slabších předpokladech než metody parametrické. Nepředpokládáme znalost tvaru pravděpodobnostních charakteristik tříd.

Metoda nejbližšího souseda je založena na hledání přímo aposteriorní pravděpodobnosti. Je myšlenkovým rozšířením metody klasifikace podle nejbližší vzdálenosti od etalonu.

Známe trénovací množinu $(x_i, \omega_i)_{i=1,\dots,K}$. Kde x_i je vzorek, kterému je přiřazena třída ω_i a K je velikost trénovací množiny. Pro neznámý prvek x hledáme x'_i takové, že $\|x'_k - x\| = \min \|x'_k - x\|_{i=1,\dots,K}$. Prvek zařadíme do stejné třídy, do jaké náleží x'_k.

Nejčastější je klasifikace podle jednoho souseda (1-NN), ale existuje i klasifikace pro obecně k sousedů.

Metoda k-nejbližších sousedů

Algoritmus k-nejbližších sousedů (neboli k-NN) je algoritmus strojového učení pro rozpoznávání vzorů. Jde o metodu pro učení s učitelem, kdy se klasifikují prvky reprezentované více dimenzionálními vektory do dvou nebo více tříd. Ve fázi učení se předzpracuje trénovací množina tak, aby všechny příznaky měly střední hodnotu 0 a rozptyl 1, toto umístí každý prvek trénovací množiny do některého místa v N-rozměrném prostoru. Ve fázi klasifikace je umístěn dotazovaný prvek do téhož prostoru a najde se k nejbližších sousedů. Objekt je pak klasifikován do té třídy, kam patří většina z těchto nejbližších sousedů.

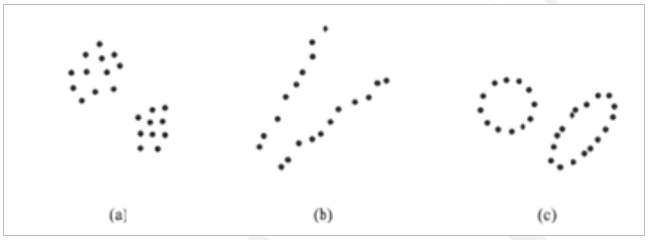
Pokud je $k = 1$, jde o speciální zjednodušený případ, metodu nejbližšího souseda. Pro hledání nejbližšího souseda v množině lze použít různé metriky. Nejobvyklejší je Euklidova metrika nebo Hammingova metrika.

1.6 Učení bez učitele

Učení bez učitele je další metoda strojového učení. Často je také nazývána **shlukování**. Jejím hlavním cílem je odhalit spojitosti či souvislosti mezi předloženými **obrazy** (daty) a podle nich je **klasifikovat** (rozdělit) do jednotlivých **shluků** (skupin).

Narozdíl od učení s učitelem **nemáme** během shlukování žádnou trénovací množinu dat. Před samotným procesem shlukováním je třeba zvolit vhodné příznaky, kterými budeme popisovat jednotlivé obrazy. Snažíme se samozřejmě počet potřebných příznaků minimalizovat (výpočetní náročnost, atd.) a proto vybíráme takové příznaky, které jsou co nejvíce informativní.

Po výběru příznaků je samozřejmě nutné zvolit vhodný shlukovací algoritmus. Tento výběr je ovlivněn mnoha faktory, jako například výpočetní náročnost, velikost klasifikované množiny, předpokládaný tvar shluků (viz obrázek 1.32), atd.



Obrázek 1.32: (a)Kompaktní shluky, (b) Podlouhlé shluky, (c) Sférické a elipsoidní shluky

Shlukování se využívá v celé škále oborů, například v biologii, ekologii, sociologii, geografii, geologii a grafové teorii. Shlukování je také jednou z nejpřirozenějších mentálních aktivit člověka, který také vjemy a informace klasifikuje do jednotlivých tříd.

Příklady algoritmů: Sekvenční algoritmy, K-means, Hierarchické algoritmy (aglomerativní, divizní), Fuzzy shlukování, Genetické shlukovací algoritmy, Algoritmy založené na morfologických operacích,...

K-means (MacQueenův algoritmus)

Jedná se o jeden z nejoblíbenějších a nejrozšířenějších shlukovacích algoritmů. Pro výpočet vzdálenosti mezi dvěma obrazy používá Euklidovu vzdálenost. Mezi jeho hlavní výhody patří jednoduchá implementace a nízká výpočetní náročnost. Mezi nevýhody naopak nutná znalost počtu tříd (shluků), do kterých budeme klasifikovat, a problematický výběr počátečního 'nástřelu' reprezentantů shluků. Algoritmus je vhodný pro kompaktní a sférické shluky a díky jeho rozšířenosti existuje mnoho jeho modifikací (K-methodois, PAM, CLARA, CLARANS).

Nyní bude uveden základní K-means algoritmus a dále podrobněji vysvětlen.

Algorithm 1 K-means algoritmus

1: Vstup: Vhodně zvolený odhad reprezentantů $\theta_j(0)$ pro j = 1,...,m.
2: for i = 1 to N: rozhodni nejbližšího reprezentanta θ_j pro dané x_i, zařaď x_i do shluku j.
3: for i = 1 to m: aktualizuj parametry: přepočti θ_j jako střední vektor j-tého shluku.
4: kroky 2 a 3 opakuj dokud se θ_j ve dvou po sobě jdoucích krocích nezmění.
5: Výstup: Středy shluků θ_j + kompletní shlukování pro všechny body

V prvním kroku algoritmu je po uživateli požadován počet shluků m a vhodný nástřel středů shluků $\theta_j(0)$. Tato část je v praxi často velmi problematická, často je řešena tak, že j počátečních nástřelů je zvoleno jako prvních j členů dané množiny. Algoritmus se dále snaží minimalizovat následující kritérium

$$J(\theta, U) = \sum_{i=1}^{N} \sum_{j=1}^{m} u_{ij} ||x_i - \theta_j||^2 \tag{1.137}$$

kde N je počet shluků, m počet bodů množiny, $u_{ij} = 1$ pokud daný bod patří do daného shluku, v opačném případě je $u_{ij} = 0$. Toho je docíleno tím, že každý bod je zařazen do do shluku od jehož reprezentanta má nejnižší Euklidovu vzdálenost. Po zařazení všech bodů do shluků jsou aktualizování jejich reprezentanti a celý algoritmus je spuštěn znovu. Algoritmus je ukončen ve chvíli, kdy reprezentanti nejsou změněni ve dvou po sobě jdoucích cyklech.

Hierarchické algoritmy

Na rozdíl od algoritmu K-means nepotřebujeme znát dopředu finální počet shluků. Jejich hlavní výhodou je, že po uživateli nejsou vyžadovány žádné počáteční parametry (na rozdíl od algoritmu K-means nepotřebujeme znát dopředu finální počet shluků, atd). Výsledkem těchto algoritmů není jen jediné shlukování, nýbrž jejich sekvence. Hierarchické algoritmy se dělí na dvě základní skupiny a to **aglomerativní** a **divizní**.

Základní myšlenkou aglomerativních algoritmů je v každém iteračním kroku spojit dva shluky (body) s nejvyšším kritériem podobnosti (nejnižší vzdáleností) do jediného shluku. Jestliže jsou dva vektory (obrazy) spojeny do stejného shluku v kroku t, setrvají ve stejném shluku až do konce algoritmu. To znamená, že na počátku každý bod tvoří vlastní shluk, tyto body jsou postupně spojovány do větších, až v posledním kroku algoritmu zbude pouze jediný shluk obsahující všechny prvky množiny.

Hierarchické algoritmy jsou vhodné pro kompaktní a podlouhlé shluky. Jejich výsledkem je, jak již bylo řečeno, sekvence shlukování. Tuto sekvenci lze přehledně zobrazit pomocí **dendogramu**. Jejich hlavní nevýhodou je nutnost vybrat správné shlukování ze získané sekvence. To lze zvolit pomocí několika přístupů, mezi ně patří například délka života shluků či definice maximálního prahu, po jehož překročení již shluky nebudou spojeny.

Divizní algoritmy postupují přesně opačným způsobem než aglomerativní, takzvaně v prvním kroku jsou všechny body členy jediného shluku a tento shluk je dále dělen až do

chvíle, kdy každý bod je ve svém shluku sám. Nejsou příliš využívány vzhledem k jejich vysoké výpočetní náročnosti.

1.7 AdaBoost

Adaptive Boosting (AdaBoost nebo AB) je algoritmus klasifikace a má tyto zajímavé vlastnosti:

- AdaBoost je lineární klasifikátor se všemi jeho potřebnými vlastnostmi.
- Výstup AdaBoost konverguje k logaritmu věrohodnosti (output converges to the logarithm of likelihood ratio).
- AdaBoost dobře zobecňuje.
- AdaBoost je příznakový selektor se zásadovou strategií (minimalizuje horní mez empirické chyby).
- AdaBoost je podobný sekvenčnímu rozhodování (produkuje posloupnost postupně složitějších klasifikátorů).

AdaBoost

AdaBoost je algoritmus sestavující „silný“ klasifikátor jako lineární kombinaci „slabých“ klasifikátorů.

Příklad: Nechť existuje imaginární závislý sázkař na dostihy, který se řídí nepsanými pravidly pro stanovení *výhry / prohry*: nejlepší kurz, nejrychlejší čas daného koně na kolo, nejvíce výher v poslední době (například za poslední měsíc). Z uvedeného lze konstatovat, že je velmi složité určit (na základě kombinace všech možných příznaků) na koho má gambler vsadit.

Příklad: Rozhodování o přijmutí na FAV, jedná se o binární klasifikaci (klasifikace do dvou tříd) *Přijat / Odmítnut*. V tabulce 1.4 jsou uvedeny, jak kvalitativní, tak kvantitativní trénovací vzorky.

ID	Jméno	Rozhodnutí	Kraj	Pohlaví	Dobrý v matematice	Dobrý ve sportu
1	Andrea	Přijat	Západočeský	Ž	Ano	Ne
2	Pepa	Přijat	Severočeský	M	Ano	Ne
3	Lukáš	Odmítnut	Severomoravský	M	Ne	Ne
4	Petr	Odmítnut	Středočeský	M	Ne	Ano
5	Lucie	Odmítnut	Západočeský	Ž	Ano	Ano
6	Ota	Přijat	Středočeský	M	Ne	Ano
7	Žofie	Odmítnut	Východočeský	Ž	Ne	Ne
8	Jaroslav	Odmítnut	Západočeský	M	Ne	Ano
9	Jan	Odmítnut	Západočeský	M	Ne	Ano
10	Marek	Přijat	Jihočeský	M	Ano	Ne

Tabulka 1.4: Imaginární výsledky přijmutí na FAV.

Je jednoduché vymyslet klasifikátor, který bude správně klasifikovat trénovací data lépe než náhodně (úspěšnost klasifikace, při problému klasifikace do dvou tříd, musí být

ostře větší než 50%). Slabý klasifikátor by bylo možné sestrojit z podmínky, pokud je uchazeč dobrý v matematice tak predikuj Přijat. Na druhou stranu je velmi složité najít jediný silný klasifikátor, který velmi přesně predikuje výslednou klasifikaci.

Co je slabý klasifikátor

Pro každé rozdělení pravděpodobnosti dané polynomiálním počtem vzorků a polynomiálním časem může být nalezen takový klasifikátor, který má obecně lepší chybu než při náhodné klasifikaci. Tedy, chyba

$$\varepsilon < 0.5, \tag{1.138}$$

což lze také zapsat jako $\gamma > 0$ pro obecnou chybu $(0.5 - \gamma)$.

Předpokladem je, že lze zkonstruovat slabý klasifikátor, tak aby trvale klasifikoval do správné třídy s lepší než 50% pravděpodobností.

Vzhledem k tomuto předpokladu je možné využít mnoho slabých klasifikátorů pro sestavení jednoho silného klasifikátoru, který bude správně klasifikovat $99 - 100\%$ vstupních vzorků.

AdaBoost algoritmus

Nyní bude představen diskrétní AdaBoost algoritmus a následně bude vysvětlen podrobněji.

Nechť existuje trénovací sada dat $S = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_N, y_N)\}$, kde $y_i \in \{-1, 1\}$ je výstupní klasifikace a $\boldsymbol{x}_i \in \boldsymbol{X}$ je vstupní příznakový vektor reálného objektu.

Algorithm 2 Diskrétní AdaBoost algoritmus

1: Vstup: $S = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_N, y_N)\}$. Počet iterací T.
2: Inicializace: $d_n^{(1)} = \frac{1}{N}$ pro všechny $n = 1, \cdots, N$.
3: **for** $t = 1, \cdots, T$ **do**
4: Natrénuj klasifikátor vzhledem k vážené množině vzorků $\{S, \boldsymbol{d^{(t)}}\}$ a získej hypotézu $h_t : \boldsymbol{x} \mapsto \{-1, +1\}$, tj. $h_t = L\left(S, \boldsymbol{d^{(t)}}\right)$.
5: Vypočti váženou chybu trénovací sady ε_t pro hypotézu h_t:

$$\varepsilon_t = \sum_{n=1}^{N} d_n^{(t)} \boldsymbol{I}\left(y_n \neq h_t\left(x\right)_n\right). \tag{1.139}$$

6: Nastav:

$$\alpha_t = \frac{1}{2} \log \frac{1 - \varepsilon_t}{\varepsilon_t}. \tag{1.140}$$

7: Aktualizuj váhy:

$$d_n^{(t+1)} = \frac{d_n^{(t)} \cdot e^{-\alpha_t y_n h_t(x_n)}}{Z_t}, \tag{1.141}$$

kde Z_t je normalizační konstanta taková, že platí $\sum_{n=1}^{N} d_n^{(t+1)} = 1$.
8: **if** $\varepsilon_t = 0$ **or** $\varepsilon_t \geq \frac{1}{2}$ **then**
9: Ukonči a nastav: $T = t - 1$
10: Výstup: $f_T\left(x\right) = \sum_{t=1}^{\top} \frac{\alpha_t}{\sum_{r=1}^{T} \alpha_r} h_t\left(x\right)$.

Vysvětlení diskrétního AdaBoost algoritmu

AdaBoost je agresivní algoritmus, který vybere jeden slabý klasifikátor v každém kroku. Váhy $\boldsymbol{d^{(t)}} = \left(d_1^{(t)}, \cdots, d_N^{(t)}\right)$ jsou přiřazeny k datům v kroku t a slabý klasifikátor h_t je založen na $\boldsymbol{d^{(t)}}$. Tyto váhy jsou aktualizovány každou iteraci. Váhy se zvětšují pro vzorky, které byly špatně klasifikovány v předchozí iteraci.

Váhy jsou inicializovány uniformě: $d_n^{(1)} = \frac{1}{N}$ pro obecnou verzi AdaBoost algoritmu. Pro odhad, pokud jsou nějaké vzorky klasifikovány dobře nebo špatně, slabý klasifikátor produkuje váženou empirickou chybu definovanou následovně

$$\varepsilon_t = \sum_{n=1}^{N} d_n^{(t)} \boldsymbol{I}\left(y_n \neq h_t\left(x\right)_n\right). \tag{1.142}$$

Když algoritmus vybere nejlepší hypotézu h_t, tak její váhy $\alpha_t = \frac{1}{2}\log\frac{1-\varepsilon_t}{\varepsilon_t}$ jsou vypočteny tak, aby minimalizovaly ztrátovou funkci. Jedna z možných ztrátových funkcí používaných v AdaBoost je

$$G^{AdaBoost}\left(\alpha\right) = \sum_{n=1}^{N} e^{-y_n(\alpha h_t(x_n) + f_{n-1}(x_n))}, \tag{1.143}$$

kde

$$f_{t-1}\left(x\right) = \sum_{r=1}^{t-1} \alpha_r h_r\left(x_n\right). \tag{1.144}$$

Iterační smyčka je zastavena pokud empirická chyba $\varepsilon_t = 0$ nebo $\varepsilon_t \geq \frac{1}{2}$. Pokud je $\varepsilon_t = 0$, tak je klasifikace optimální. Pokud $\varepsilon_t \geq \frac{1}{2}$, tak klasifikátor přestal respektovat podmínku slabých klasifikátorů a algoritmus AdaBoost nemůže fungovat.

Všechny slabé hypotézy, které jsou vybrány do stavu h_t jsou lineárně zkombinovány následovně

$$f_T\left(x\right) = \sum_{t=1}^{\top} \frac{\alpha_t}{\sum_{r=1}^{\top}\alpha_r} h_t\left(x\right). \tag{1.145}$$

Výsledný klasifikátor je založený na jednoduchém prahování, které rozhodne o tom, zda bude vzorek přijat nebo odmítnut.

Vliv slabých klasifikátorů

V každé iteraci AdaBoost vytvoří slabé klasifikátory založené na vážených vzorcích. Nyní bude diskutována výkonnost těchto slabých klasifikátorů založených na opětovném vážení vzorků. Pro pozorování vlivu slabých klasifikátorů nejprve nutné definovat několik základních nástrojů.

Základní klasifikátor: Nechť $\boldsymbol{d} = (d_1, \cdots, d_n)$ je váha (pravděpodobnost) vzorků S. Nechť S_+ je podmnožina vzorků, které jsou označeny kladně a podobně pro S_-. Dále $D_+ = \sum_{n:y_n=+1} d_n$ a podobně pro D_-. Základní klasifikátor f_{BL} (index BL značí „Base Line“) je definován jako

$$f_{BL}\left(x\right) = sign\left(D_+ - D_-\right), \quad \forall x. \tag{1.146}$$

Tedy predikuje $+1$ pokud $D_+ \geq D_-$ jinak -1. Je okamžitě zřejmé, že pro všechny váhy $\boldsymbol{d}$ je chyba základního klasifikátoru nejvýše $\frac{1}{2}$.

S touto definicí je možné lépe definovat slabý klasifikátor: Slabý klasifikátor vzorků S, pokud jsou dány váhy $\boldsymbol{d}$ vzorků S, je schopen dosáhnout vážené chyby klasifikace, která je striktně menší než $\frac{1}{2}$.

Klíčová vlastnost Boosting algoritmu je splnění podmínky pro slabý klasifikátor, který je potřeba pro to, aby celkový algoritmus fungoval správně. Vážená empirická chyba každého slabého klasifikátoru musí být striktně menší než $\frac{1}{2} - \frac{1}{2}\gamma$, kde γ je parametr určující odchylku od od základního klasifikátoru představeného dříve. Je uvažován slabý binární klasifikátor h, vzhledem k množině vzorků $S = \{(x_n, y_n)\}_{n=1}^N$, kde pro každou dvojici (x_n, y_n) existuje nezáporná váha d_n. Pak je požadováno, aby

$$\varepsilon_t(h_t, \boldsymbol{d}) = \sum_{n=1}^{N} d_n \boldsymbol{I}(y_n \neq h(x)_n) \leq \frac{1}{2} - \frac{1}{2}\gamma, \quad (\gamma > 0). \tag{1.147}$$

Pro nějaké slabé klasifikátory nemusí existovat $\gamma > 0$, která respektuje předchozí podmínku aniž by porušila některé podmínky týkající se dat.

Příklad možného nalezení pozitivní γ, která respektuje podmínku empirické chyby 1.147. Předpokladem je mapování f z binární krychle $\boldsymbol{X} = \{+1, -1\}^d$ do $\boldsymbol{Y} = \{+1, -1\}$ a dále pozitivní labely y_n, které jsou dány $y_n = f(x_n)$. Funkce f je aproximována binární hypotézou h_t patřící do $\boldsymbol{H}$. Nechť H je třída binárních hypotéz a nechť D je rozložení nad $\boldsymbol{X}$. Korelace mezi f a H s respektováním D je dána $C_{H,D}(f) = \sup_{h \in H} \boldsymbol{E}_D\{f(x)\, h(x)\}$. Rozložení bez korelace mezi f a H je dáno $C_H(f) = \inf_D C_{H,D}(f)$. To může být dokázáno, pokud $T > 2 \cdot \log(2)\, dC_H(f)^{-2}$, pak f může být reprezentována přesně jako

$$f(x) = sign\left(\sum_{t=1}^{T} h_t(x)\right). \tag{1.148}$$

Jinými slovy, pokud je H silně korelována s cílovou funkcí f, pak f může být přesně reprezentována jako kombinace malých čísel funkce z H. Proto poté pro dostatečně velký počet iterací může být očekáváno, že se empirická chyba bude blížit nule.

Tento příklad ukázal binární klasifikaci s dvěma lineárně oddělitelnýma třídami. Ve více obecném případě můžou být třídy silně se překrývající, proto je nutné využít pokročilejší nástroje jako vytvořit pravidla týkající se slabých klasifikátorů.

Výhody AdaBoost algoritmu	**Nevýhody AdaBoost algoritmu**
Rychlost.	Slabé klasifikátory, které jsou současně komplexní vedou k přetrénování.
Prokazatelná efektivita vzhledem k předpokladu slabých klasifikátorů.	Pokud jsou slabé klasifikátory příliš slabé může vznikat „low margin“ problém i problém přetrénování.
Není potřeba volit žádný parametr (vyjma počtu iterací T).	Z empirických důkazů bylo ukázáno, že AdaBoost algoritmus je náchylný na uniformní šum.
Není potřebná žádná apriorní informace.	
Jednoduchá implementace.	
Univerzálnost.	

1.8 Kaskádní AdaBoost

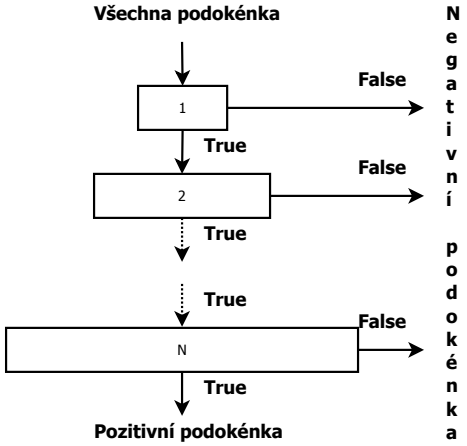
Je možné vybrat malé množství dobře odlišitelných příznaků a zkombinovat je do jednoho silného klasifikátoru, nicméně je potřeba představit několik hlavních myšlenek pro snížení výpočetního času. Rozšíření spočívá v tvorbě kaskády klasifikátorů, která navíc může dosahovat lepší klasifikační přesnosti. Tedy snaží se minimalizovat počet false negativ klasifikací namísto klasického trénování chyby, což představuje hlavní myšlenku, která bude dále sloužit pro vytvoření kaskády klasifikátorů.

Princip bude vysvětlen na praktickém příkladě hledání obličejů v obrázku.

Proč je to tak efektivní?

Princip spočívá v rychlém odmítnutí většiny negativních okének a ponechání co nejvíce pozitivních. Následně se využije více citlivějších (menších) podokének s více komplexnějšími klasifikátory. V první fázi kaskády, která obsahuje pouze několik málo příznaků se většinou dosahuje velmi vysokého detekčního poměru (kolem 100%), avšak na úkor false positive (přibližně 40%). Je zřejmé, že pro výslednou klasifikaci je takový počet chyb nepřijatelný. Proto se v případě problému detekce obličejů kombinuje mnoho za sebou navazujících klasifikátorů, což vede k více a více diskriminativní klasifikaci jejímž výsledkem je detekce obličejů ve scéně.

Tuto kaskádní strukturu lze přirovnat k degenerovanému rozhodovacímu stromu. Pokud je podokénko klasifikováno jako pozitivní v jednom stupni, tak postupuje dále v kaskádě a je ověřováno v každém dalším stupni znovu. Takto se pokračuje dále dokud není podokénko označeno negativně, nebo je po ověření poslední vrstvou prohlášeno za pozitivní (obsahující tvář). Na Obr. 1.33 je tento proces naznačen.



Obrázek 1.33: Schématický popis kaskádní detekce. Série klasifikátorů je aplikována na každé podokénko. Počáteční klasifikátor eliminuje velký počet negativních vzorků s malým processingem. Další vrstvy eliminují další negativní vzorky, tudíž je potřeba dalšího výpočetního času. Po několika stupních processingu je počet podokének radikálně redukován. Vzorky, které jsou klasifikovány jako pozitivní v posledním stupni jsou definitivně označeny jako pozitivní.

Tvorba více konzistentních klasifikátorů

Byla definována kaskáda jako množina na sebe navazujících klasifikátorů. První klasifikátor je velmi jednoduchý, ale postupem v kaskádě nabývá na složitosti.

V poslední vrstvě má klasifikátor více příznaků než v první. AdaBoost algoritmus generuje chybu trénování, která klesá teoreticky exponenciálně v závislosti na počtu iterací. Pokud je více příznaků (AdaBoost algoritmus běží více iterací) výsledný klasifikátor je více diskriminativní mezi pozitivními a negativními vzorky. Jinými slovy lze říci, že je klasifikátor „silnější“ než v případě kdy má k dispozici méně příznaků (běží menší počet iterací).

Další důležitou věcí kaskádního AdaBoost algoritmu je volba trénovací množiny. V každé vrstvě (učícím se kroku, iteraci) je i-tý klasifikátor testován na testovací množině negativních vzorků. Všechny špatně klasifikované negativní vzorky jsou předány dále do $(i+1)$-tého klasifikátoru, který se tedy musí zaměřit na složitější vzorky. Snaha spočívá v donucení následujícího klasifikátoru, aby měl poměr nalezených false positive vzorků menší než předchozí klasifikátor.

Trénování kaskádního AdaBoost algoritmu

Nechť F je míra detekovaných false positive vzorků kaskádního klasifikátoru, K je počet klasifikátorů a f_i je míra false positive vzorků i-tého klasifikátoru. Pro trénovanou kaskádu klasifikátorů je F dáno jako

$$F = \prod_{i=1}^{K} f_i, \tag{1.149}$$

následně míra detekce může být vypočtena jako

$$D = \prod_{i=1}^{K} d_i, \tag{1.150}$$

kde d_i je míra detekce i-tého klasifikátoru na vzorcích procházející daným klasifikátorem.

Počet příznaků, které musí byt hledány v reálném obrázku (například při hledání obličeje) nezbytný pravděpodobnostní proces. Většina podokének projdou kaskádou jedním nebo několika klasifikátory než jsou vyhodnoceny jako negativní vzorek a jen velmi malá část vzorků je na konci označena jako pozitivní vzorek. Chování tohoto procesu je závislé na pravděpodobnostním rozdělení obrázků v trénovací sadě. Hlavním nástrojem pro měření výkonu klasifikátorů je míra pozitivních vzorků na konci kaskády. Vzhledem k počtu vrstev v kaskádě K je míra pozitivních vzorků v i-té vrstvě (klasifikátoru) označena p_i. Nechť n_i je počet příznaků v i-té vrstvě. Očekávaný počet příznaků, který je ověřován, je dán

$$N = n_0 + \sum_{i=1}^{K} \left(n_i \prod_{j<i} p_i \right). \tag{1.151}$$

Jen několik málo příkladů je klasifikováno jako hledané objekty (obličeje), což je také důvod, proč je míra pozitivních vzorků přibližně rovna míře false positive vzorků.

Struktura kaskádního AdaBoost algoritmu vyžaduje znalost tří parametrů

1. Celkový počet klasifikátorů K.
2. Počet příznaků n_i v i-té vrstvě.
3. Práh θ_i v i-té vrstvě.

Nalezení těchto optimálních parametrů je obtížné, za předpokladu, že je nutné minimalizovat výpočetní čas celé klasifikace. Princip spočívá ve zvýšení počtu příznaků a počtu vrstev dokud není dosažená stanovená úspěšnost klasifikace.

Vzhledem k minimální přijatelné míře f_i (false positive míra pro i-tou vrstvu) a d_i (míra detekce pro i-tou vrstvu) je míra detekce d_i dosažena za pomoci snižování prahu θ_i, což přímo ovlivňuje f_i. Počet příznaků n_i v i-té vrstvě je zvyšován dokud není dosaženo f_i. Obecný kaskádový AdaBoost algoritmus je naznačen v algoritmu 3 a bude diskutován níže.

Jeden z hlavních faktorů efektivity kaskádového algoritmu je správa trénovací množiny během učení. Většinou je trénovací množina použita pro první vrstvu následně každou iteraci je aktuální klasifikátor ověřován na validační množině. Je velmi nevhodné ověřovat úspěšnost klasifikace na trénovací množině, jelikož bude pro tuto množinu bude úspěšnost vyšší. V každé další vrstvě je trénovací množina negativních vzorků znovu inicializována. Když je dosaženo F_i a D_i pro i-tou vrstvu, tak je aktuální model testován na velké množině negativních vzorků, které jsou zvoleny náhodně. Mnoho false positive vzorků je vloženo do trénovací sady negativních vzorků v $i-1$-té vrstvě. Nová trénovací množina negativních vzorků je vytvořena ze vzorků, které byly špatně detekovány v i-té vrstvě, tedy následující $i+1$-tá vrstva je trénována na vzorcích, které jsou pro předešlo vrstvu „složité“. Tedy, čím dále v kaskádě se klasifikátor nachází, tím lépe dokáže klasifikovat mezi pozitivními a negativními vzorky (podokýnky obsahujícími lidské tváře).

Algorithm 3 Učení kaskádního AdaBoost algoritmu

Vstup: f maximální přijatelná míra false positive pro každou vrstvu, d minimální přijatelná míra detekce pro každou vrstvu, F_{target} počet false positive na konci procesu, P množina pozitivních vzorků, N množina negativních vzorků.
Inicializace: $F_0 = D_0 = 1$, $i = 0$ číslo aktuální vrstvy.
while $F_i > F_{target}$ **do**
    $i = i + 1$
    $n_i = 0$
    $F_i = F_{i-1}$
    while $F_i > f \cdot F_{i-1}$ **do**
        $n_i = n_i + 1$
        Natrénuj AdaBoost klasifikátor (algoritmus 2) s P a N jako trénovací množinou.
        Vypočti F_i a D_i pro aktuální klasifikátor na základě validační množiny.
        Snižuj práh klasifikátoru dokud není poměr i-tého klasifikátoru alespoň

$$d \cdot \frac{D_{i-1}}{D_i} \geq D_{i-1} \cdot d \tag{1.152}$$

    Smaž negativní vzorky trénovací množiny.
    if $F_i > F_{target}$ **then**
        Ověř aktuální kaskádu klasifikátorů na množině negativních vzorků a vlož nějaké špatně detekované vzorky do množiny N.

1.9 SVM

Tato metoda hledá v prostoru nejširší možný pruh, který rozděluje množinu vektorů do dvou nadrovin. Nadrovina je následně popsána pomocí vektorů ležících nejblíže její hranice, kterých je obvykle malý počet a nazývají se support vectors. Tyto vektory daly za vznik názvu celé metody SVM (z ang. *Support Vectors Machine*). Princip klasifikační metody SVM bude nejprve vysvětlen na případu jednoznačně oddělitelné množiny (lineárně separovatelná množina) vektorů a následně se text zaměří na mnohem obecnější případ množiny, kterou nelze jednoznačně rozdělit (nelineárně separovatelná množina) do dvou tříd.

Lineárně separovatelná množina

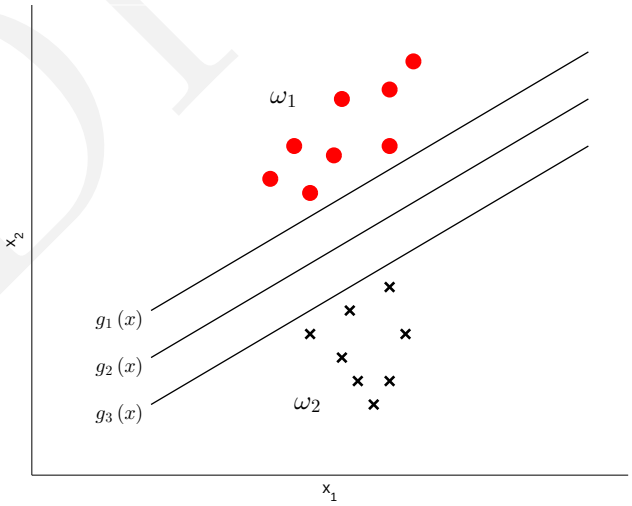
Nechť $\boldsymbol{x}_i, i = 1, 2, \ldots, N$, jsou vektory tvořící trénovací množinu X. Vektory náleží do jedné ze dvou tříd ω_1, ω_2 a jsou lineárně separovatelné. To znamená, že existuje parametr ω, pro který platí

$$\begin{aligned} \boldsymbol{\omega}^\top \boldsymbol{x} > 0, &\quad \text{pro } \forall \boldsymbol{x} \in \omega_1, \\ \boldsymbol{\omega}^\top \boldsymbol{x} < 0, &\quad \text{pro } \forall \boldsymbol{x} \in \omega_2. \end{aligned}$$

Jinými slovy, úkolem klasifikační metody je nalézt nadrovinu $g(\boldsymbol{x})$

$$g(\boldsymbol{x}) = \boldsymbol{\omega}^\top \boldsymbol{x} + \omega_0 = 0, \tag{1.153}$$

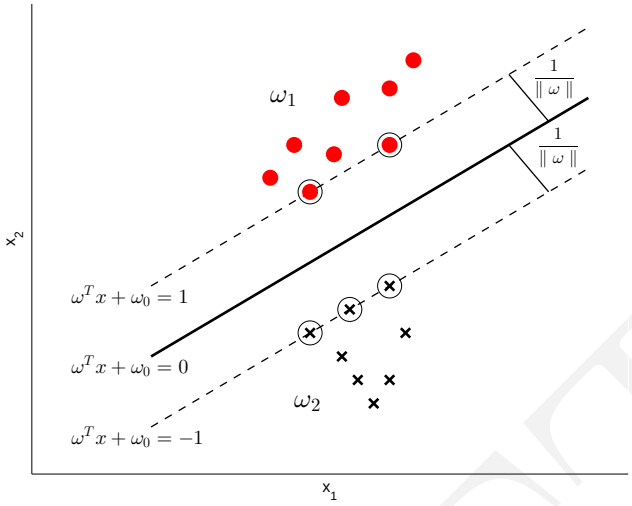
která rozděluje vektory $\boldsymbol{x}_i$ do příslušných tříd. Koeficient ω_0 z předpisu 1.153 je nulový pouze v případě že nadrovina prochází počátkem souřadnicového systému. Jak je patrné z Obr. 1.34, rozdělujících nadrovin je obvykle celá řada. Úkolem klasifikační metody SVM je zvolit takovou, která minimalizuje chybu klasifikace tím, že má maximální možný odstup od obou tříd.



Obrázek 1.34: Příklad lineárně separovatelných tříd třemi možnými nadrovinami.

Obecně je nadrovina definována svým směrem (parametr $\boldsymbol{\omega}$) a polohou v prostoru (koeficient ω_0). SVM vedle těchto dvou faktorů zohledňuje i odstup nadroviny od klasifikačních tříd. Jelikož je nežádoucí, aby během určování nadroviny byla některé z tříd

zvýhodňována, SVM hledá takový směr $\boldsymbol{\omega}$, pro který má nadrovina stejnou vzdálenost k nejbližším bodům z obou tříd ω_1 a ω_2 (viz Obr. 1.35).



Obrázek 1.35: Optimální nadrovina dělící vektory do dvou tříd; kroužkem jsou znázorněny support vectors.

Vzdálenost mezi bodem a nadrovinou lze vyjádřit pomocí vztahu

$$z = \frac{\mid g\left(\boldsymbol{x}\right) \mid}{\parallel \boldsymbol{\omega} \parallel}, \tag{1.154}$$

kde $\parallel \boldsymbol{\omega} \parallel = \sqrt{\boldsymbol{\omega}^2}$. Nejblíže položené body (na obr. znázorněny kroužkem) mají vzdálenost od nadroviny rovnu 1, pokud se nacházejí ve třídě ω_1 a -1, náleží-li do třídy ω_2. To vede k závěru, že SVM hledá nadrovinu:

- mající odstup $\frac{1}{\|\boldsymbol{\omega}\|} + \frac{1}{\|\boldsymbol{\omega}\|} = \frac{2}{\|\boldsymbol{\omega}\|}$.
- splňující podmínky

$$\begin{aligned} \boldsymbol{\omega}^\top \boldsymbol{x} + \omega_0 &\geq +1, \quad \text{pro } \forall \boldsymbol{x} \in \omega_1, \\ \boldsymbol{\omega}^\top \boldsymbol{x} + \omega_0 &\leq -1, \quad \text{pro } \forall \boldsymbol{x} \in \omega_2. \end{aligned}$$

Dále je definován tzv. indikátor třídy y_i ($+1$ pro ω_1 a -1 pro ω_2) a cíl metody SVM může být shrnut následovně:

$$\min J\left(\boldsymbol{\omega}, \omega_0\right) = \frac{1}{2} \parallel \boldsymbol{\omega} \parallel^2, \tag{1.155}$$

za podmínky

$$y_i \left(\boldsymbol{\omega}^\top \boldsymbol{x}_i + \omega_0\right) \geq 1, \quad i = 1, 2, \ldots, N. \tag{1.156}$$

Vektory, pro které platí $y_i \left(\boldsymbol{\omega}^\top \boldsymbol{x}_i + \omega_0\right) = 1$, jsou označovány jako *support vectors* (viz Obr. 1.35).

Matematické řešení

Z výše uvedených faktů vyplývá, že konstrukce nadroviny, která maximalizuje odstup od obou klasifikačních tříd, je kvadratickou optimalizační úlohou a její řešení vede na Lagrangeovu funkci, kde λ značí Lagrangeův multiplikátor

$$\mathcal{L}(\boldsymbol{\omega}, \omega_0, \boldsymbol{\lambda}) = \frac{1}{2}\boldsymbol{\omega}^\top\boldsymbol{\omega} - \sum_{i=1}^{N} \lambda_i \left[y_i \left(\boldsymbol{\omega}^\top \boldsymbol{x}_i + \omega_0 \right) - 1 \right], \tag{1.157}$$

s podmínkami

$$\begin{aligned} \frac{\partial}{\partial \boldsymbol{\omega}} \mathcal{L}(\boldsymbol{\omega}, \omega_0, \boldsymbol{\lambda}) &= 0, & (1.158)\\ \frac{\partial}{\partial \omega_0} \mathcal{L}(\boldsymbol{\omega}, \omega_0, \boldsymbol{\lambda}) &= 0, & (1.159)\\ \lambda_i &\geq 0, \quad i = 1, 2, \ldots, N, & (1.160)\\ \left[y_i \left(\boldsymbol{\omega}^\top \boldsymbol{x}_i + \omega_0 \right) - 1 \right] &= 0, \quad i = 1, 2, \ldots, N. & (1.161) \end{aligned}$$

Kombinací rovnice 1.157 s 1.159 a 1.160 je získáno řešení

$$\begin{aligned} \boldsymbol{\omega} &= \sum_{i=1}^{N} \lambda_i y_i \boldsymbol{x}_i, & (1.162)\\ 0 &= \sum_{i=1}^{N} \lambda_i y_i. & (1.163) \end{aligned}$$

Takto zkonstruovaná nadrovina se označuje za nadrovinu optimální. Optimalizační úloha definovaná předpisy 1.155 a 1.156 patří mezi úlohy konvexního programování a jako takovou ji lze řešit pomocí Lagrangeovy duality

$$\max \mathcal{L}(\boldsymbol{\omega}, \omega_0, \boldsymbol{\lambda}), \tag{1.164}$$

za podmínek

$$\boldsymbol{\omega} = \sum_{i=1}^{N} \lambda_i y_i \boldsymbol{x}_i, \quad \sum_{i=1}^{N} \lambda_i y_i = 0, \quad \lambda_i \geq 0, \quad i = 1, 2, \ldots, N. \tag{1.165}$$

Dosazením rovnic 1.165 do 1.164 je získána optimalizační úloha ekvivalentní k úloze popsané předpisy 1.155 a 1.156

$$\max_{\lambda} \left(\sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j \right), \tag{1.166}$$

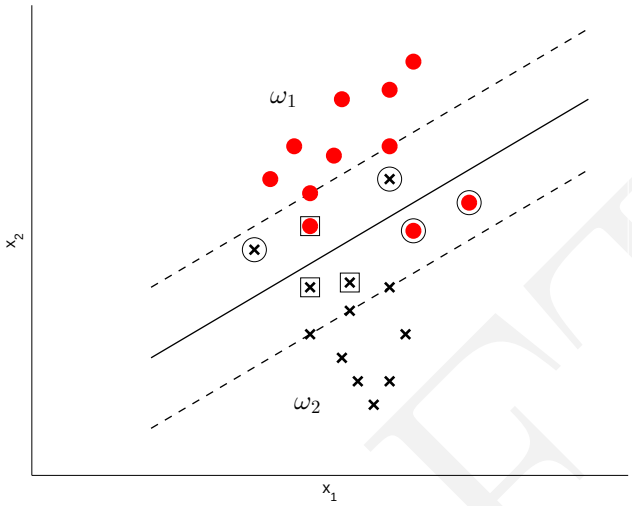
s podmínkou

$$\sum_{i=1}^{N} \lambda_i y_i = 0, \quad \lambda_i \geq 0, \quad i = 1, 2, \ldots, N. \tag{1.167}$$

Maximalizováním výrazu 1.166 jsou vypočteny optimální Lagrangeovy multiplikátory a optimální nadrovina je získána jejich dosazením do rovnice 1.165. Koeficient ω_0 je vypočten z podmínky 1.161.

Nelineárně separovatelná množina

Jestliže prvky trénovací množiny X nelze lineárně oddělit, nemůžu být výše zmíněný postup konstrukce dělící nadroviny použit. Případ neoddělitelné množiny je zobrazen na obr. Jakýkoli pokus o vykreslení pásu rozdělujícího vektory do dvou tříd skončí neúspěchem. Vždy se totiž uvnitř takového pásu budou vyskytovat vektory trénovací množiny.



Obrázek 1.36: Příklad lineárně neseparovatelné množiny.

Šířka dělícího pásu je definovaná vzdáleností dvou paralelních nadrovin s předpisem

$$\boldsymbol{\omega}^\top \boldsymbol{x} + \omega_0 \pm 1 \tag{1.168}$$

vektory $\boldsymbol{x}_i$, $\boldsymbol{x}_i \in X$, nyní náleží do jedné ze tří kategorií:

- vektory ležící mimo dělící pás jsou považovány za správně klasifikované a splňují tak podmínku 1.156.
- vektory správně klasifikované, které leží uvnitř pásu, splňují nerovnost (na Obr. 1.36 znázorněny čtvercem)
$$0 \leq y_i \left(\boldsymbol{\omega}^\top \boldsymbol{x} + \omega_0 \right) < 1.$$
- vektory chybně klasifikované, které leží uvnitř pásu, splňují nerovnost (na Obr. 1.36 znázorněny kroužkem)
$$y_i \left(\boldsymbol{\omega}^\top \boldsymbol{x} + \omega_0 \right) < 0.$$

Všechny tři kategorie mohou být popsány podmínkou

$$y_i \left(\boldsymbol{\omega}^\top \boldsymbol{x} + \omega_0 \right) \geq 1 - \xi_i, \tag{1.169}$$

kde ξ_i představuje tzv. klouzavou proměnou. Ta dosahuje hodnoty $\xi_i = 0$ pro první kategorii, $0 < \xi_i \leq 1$ pro druhou kategorii a $\xi_i > 1$ pro třetí kategorii.

Úkolem klasifikační metody SVM je opět konstrukce nadroviny, s co největším odstupem od obou tříd, a zároveň minimalizovat počet bodů, pro které je $\xi_i > 1$. Jde tedy o úkol minimalizace ztrátové funkce

$$J\left(\boldsymbol{\omega}, \omega_0, \boldsymbol{\xi}\right) = \frac{1}{2} \parallel \boldsymbol{\omega} \parallel^2 + C \sum_{i=1}^{N} I\left(\xi_i\right), \tag{1.170}$$

kde C je pozitivní konstanta, $\boldsymbol{\xi}$ je vektor parametrů ξ_i a

$$I\left(\xi_i\right) = \begin{cases} 1, & \xi_i > 0, \\ 0, & \xi_i = 0. \end{cases} \tag{1.171}$$

Matematické řešení

Optimalizace funkce 1.170 je matematicky velice složitá, jelikož obsahuje nespojitou funkci $I\left(\bullet\right)$. Z tohoto důvodu se SVM zaměřuje na optimalizaci tzv. blízké funkce a jejím cílem se tak stává

$$\min J\left(\boldsymbol{\omega}, \omega_0, \boldsymbol{\xi}\right) = \frac{1}{2} \parallel \boldsymbol{\omega} \parallel^2 + C \sum_{i=1}^{N} \xi_i, \tag{1.172}$$

za podmínek

$$y_i \left(\boldsymbol{\omega}^\top \boldsymbol{x} + \omega_0\right) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad i = 1, 2, \ldots, N. \tag{1.173}$$

Řešení opět vede na Lagrangeovu funkci

$$\mathcal{L}\left(\boldsymbol{\omega}, \omega_0, \boldsymbol{\lambda}, \boldsymbol{\xi}, \boldsymbol{\mu}\right) = \frac{1}{2} \parallel \boldsymbol{\omega} \parallel^2 + C \sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \mu_i \xi_i - \sum_{i=1}^{N} \lambda_i \left[y_i \left(\boldsymbol{\omega}^\top \boldsymbol{x}_i + \omega_0\right) - 1 + \xi_i\right], \tag{1.174}$$

za podmínek $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\omega}}$, $\frac{\partial \mathcal{L}}{\partial \omega_0}$ a $\frac{\partial \mathcal{L}}{\partial \xi_i}$ nebo

$$\boldsymbol{\omega} = \sum_{i=1}^{N} \lambda_i y_i \boldsymbol{x}_i, \quad \sum_{i=1}^{N} \lambda_i y_i = 0, \quad C - \mu_i - \lambda_i = 0 \tag{1.175}$$

společně s podmínkami

$$\lambda_i \left[y_i \left(\boldsymbol{\omega}^\top \boldsymbol{x}_i + \omega_0\right) - 1 + \xi_i\right] = 0, \quad \mu_i \xi_i = 0, \quad \mu_i \geq 0, \quad \lambda_i \geq 0, \quad i = 1, 2, \ldots, N. \tag{1.176}$$

Použitím Lagrangeovy duality je úkolem SVM

$$\max \boldsymbol{\mathcal{L}}\left(\boldsymbol{\omega}, \omega_0, \boldsymbol{\lambda}, \boldsymbol{\xi}, \boldsymbol{\mu}\right), \tag{1.177}$$

za podmínek

$$\boldsymbol{\omega} = \sum_{i=1}^{N} \lambda_i y_i \boldsymbol{x}_i, \quad \sum_{i=1}^{N} \lambda_i y_i = 0, \quad C - \mu_i - \lambda_i = 0, \quad \mu_i \geq 0, \quad \lambda_i \geq 0, \quad i = 1, 2, \ldots, N. \tag{1.178}$$

Po dosazení výrazů 1.178 do 1.177 je získána ekvivalentní optimalizační úloha

$$\max_{\lambda} \left(\sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j \right), \tag{1.179}$$

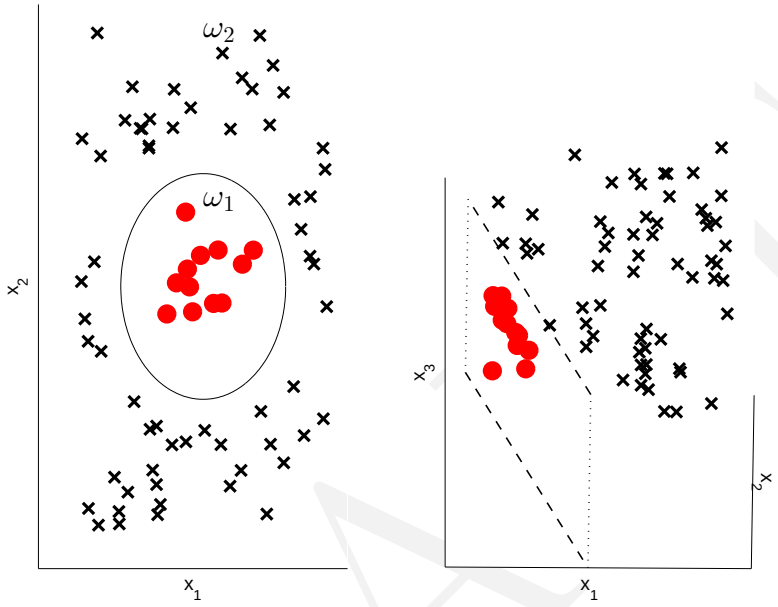
s podmínkou

$$\sum_{i=1}^{N} \lambda_i y_i = 0, \quad 0 \leq \lambda_i \leq C, \quad i = 1, 2, \ldots, N, \tag{1.180}$$

jejímž řešením jsou vypočteny Lagrangeovy multiplikátory. Ty vedou k získání nadroviny, která má maximální možný odstup od obou tříd a zároveň minimalizuje počet vektorů, pro něž je $\xi_i > 0$.

1.10 SVM - Kernel trick

Doposud bylo cílem metody SVM lineárně rozdělit trénovací množinu X do dvou tříd ω_1 a ω_2. Mnohdy je ale takový postup řešení neefektivní a vhodnější by bylo v prostoru trénovací množiny zkonstruovat hranici s nelineárním popisem. Za tímto účelem se využívá tzv. *Kernel Trick*, kdy dochází k namapování vstupní množiny X do prostoru vyšší dimenze, v kterém je ji možné lineárně rozdělit (viz Obr. 1.37).



Obrázek 1.37: Příklad namapování nelineárně separovatelné množiny z prostoru $\mathcal{R}^2$ do prostoru $\mathcal{R}^3$, kde lze množinu lineárně rozdělit.

V rovnici 1.166 je jádro (ang. *kernel*) popsáno vztahem $\boldsymbol{x}_i^\top \boldsymbol{x}_j$ a představuje míru podobnosti mezi i-tým a j-tým vektorem. Tato míra je označována jako skalární součin (ang. *inner product*). Předpis jádra metody SVM tedy vypadá následovně:

$$K\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) = \boldsymbol{x}_i^\top \boldsymbol{x}_j. \tag{1.181}$$

Po dosazení předpisu jádra zpět do rovnice 1.166 se cílem metody SVM stává:

$$\max_{\lambda} \left(\sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j K\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) \right), \tag{1.182}$$

s podmínkou

$$\sum_{i=1}^{N} \lambda_i y_i = 0, \quad \lambda_i \geq 0, \quad i = 1, 2, \ldots, N. \tag{1.183}$$

Vhodnou volbou jádra metody SVM lze docílit namapování trénovacích vektorů do prostoru, ve kterém jsou následně aplikovány principy hledání klasifikační nadroviny, jež má maximální možný odstup od nově namapovaných tříd ω_1 a ω_2. Jinými slovy, pomocí *Kernel Trick* je možné metodu SVM, která je metodou lineární, transformovat v metodu nelineární (nelineární klasifikátor). Nejčastěji využívaná nelineární jádra jsou uvedena v tabulce 1.5.

Typ jádra	Předpis	Poznámka
Polynomiální jádro	$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i \boldsymbol{x}_j + \theta)^d$	Parametr d a práh θ je volen uživatelem.
Sigmoidové jádro	$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = tahh(\eta \boldsymbol{x}_i \boldsymbol{x}_j + \theta)$	Parametr η a práh θ je volen uživatelem.
Gaussovo jádro *Radial Basis Function* (RBF)	$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = exp\left(-\frac{1}{2\sigma^2} \parallel \boldsymbol{x}_i - \boldsymbol{x}_j \parallel^2\right)$	Parametr σ je volen uživatelem.

Tabulka 1.5: Nejčastěji využívaná nelineární jádra pro metodu SVM.

Volbu jádra pro *Kernel Trick* nelze zobecnit. Obvykle se jádra iterativně testují a vybráno je to, které vykáže nejmenší chybu klasifikace. Často se ovšem doporučuje toto testování odstartovat s RBF jádrem.

1.11 Zdroje

[1] G. Rätsch, T. Onoda K-R. Müller. „Soft Margins for AdaBoost“, *Machine Learning*, 1-35, August 1998
[2] P. Viola and M. Jones, „Fast and Robust Classification using Asymmetric AdaBoost and a Detector Cascade.“ Mitsubishi Electric Research Lab, Cambridge, MA. 2001
[3] Cortes C., Vapnik V., „Support Vector Networks“, *Machine Learning*, vol. 20, pp. 273-297, 1995.
[4] Theodoridis S., Koutroubas K., *Pattern Recognition*, Elsevier Inc., 2008.