

# Machine Learning

## Strojové učení

Ing. Lukáš Bureš  
Ing. Petr Zimmermann  
Katedra kybernetiky

Plzeň  
3. listopadu 2014

# Obsah

<b>1</b>	<b>TODO</b>	<b>1</b>
<b>2</b>	<b>Přednáška 07</b>	<b>2</b>
2.1	Princip klasifikace . . . . .	2
2.1.1	Porovnání příznakových vektorů . . . . .	3
2.1.2	Míra podobnosti . . . . .	4
2.1.3	Míra odlišnosti . . . . .	4
2.1.4	Porovnání vektoru a třídy . . . . .	5
2.1.5	Reprezentant třídy . . . . .	6
2.1.6	Co je strojové učení? . . . . .	7
2.2	Lineární regrese . . . . .	8
2.2.1	Motivace . . . . .	8
2.2.2	Učení s učitelem . . . . .	8
2.2.3	Ztrátová funkce . . . . .	10
2.2.4	Gradientní algoritmus . . . . .	11
2.2.5	Vícerozměrná lineární regrese . . . . .	13
2.2.6	Gradientní algoritmus - vícerozměrná verze . . . . .	14
2.2.7	Features scaling . . . . .	15
2.2.8	Analytické řešení . . . . .	15
2.2.9	Gradientní algoritmus vs. analytické řešení . . . . .	16
2.3	Binární regrese - klasifikace . . . . .	17
2.3.1	Klasifikace . . . . .	17
2.3.2	Reprezentace hypotézy . . . . .	19
2.3.3	Rozhodovací hranice . . . . .	21
2.3.4	Ztrátová funkce . . . . .	24
2.3.5	Zjednodušená ztrátová funkce a gradientní metoda . . . . .	25
2.3.6	Pokročilé algoritmy optimalizace . . . . .	27
2.3.7	Klasifikace do více tříd . . . . .	29
2.4	Regularizace . . . . .	31
2.4.1	Problém přetrénování . . . . .	31
2.4.2	Ztrátová funkce . . . . .	33
2.4.3	Regularizace lineární regrese . . . . .	35
2.4.4	Regularizace binární regrese . . . . .	37
2.5	Učení s učitelem . . . . .	39
2.6	Učení bez učitele . . . . .	40

2.7	AdaBoost . . . . .	41
2.8	Kaskádní AdaBoost . . . . .	46
2.9	SVM . . . . .	50
2.10	SVM - Kernel trick . . . . .	55
2.11	Zdroje . . . . .	56
<b>3</b>	<b>Přednáška 08</b>	<b>57</b>
3.1	Bayesovská klasifikace . . . . .	57
3.2	Rozhodovací strom . . . . .	58
3.3	Rozhodovací les . . . . .	61
3.4	Náhodný les . . . . .	61
3.5	Neuronové sítě . . . . .	62
3.6	Konvoluční neuronové sítě . . . . .	65
3.7	Zdroje . . . . .	67

# Kapitola 1

## TODO

Tuně psané si vezme na starost Lukáš.

- Motivace - proč klasifikovat, co je to klasifikace, pojem experta,
- **Příznakový vektor** - co to je, jak se sestavuje, (nevím jestli hovořit o volbě počtu příznaků a tím pádem o redukci příznaků, dále neví zda zmiňovat, že vektor nemusí obsahovat pouze reálná čísla
- Učení s učitelem a **bez učitele**(Volba počtu tříd)
- Porovnávání vektorů II - na základě pravděpodobnostního rozdělení trénovací množiny, lineární/nelineární dělicí rovina
- **Ověření výsledku klasifikace - recall, precision, F1 skóre**

Update: 14. srpna 2014

- rozhodovací les - co ta reference ?
- Odstraneny jednopísmenkove spojky z koncu radek v kapitole 3.2-3.3
- odstraněny warningy

Update: 28. srpna 2014

- Dodelany reference od Lukase
- Opraveny Petrovo uvozovky ""na „“

Update: 9.9.2014

- Lukáš: Bayese, Bez ucitele - Kmeans, mean-shift, aglomerativni, divizni, (two threshold ?)
- Petr: Neuronovky, NN, KNN, Bayesuv nejak inteligentne - ne moc do hloubky, Perceptron
- Motivace - koho napadne nejakz blaf, tak to tam napise.

# Kapitola 2

## Přednáška 07

### 2.1 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.

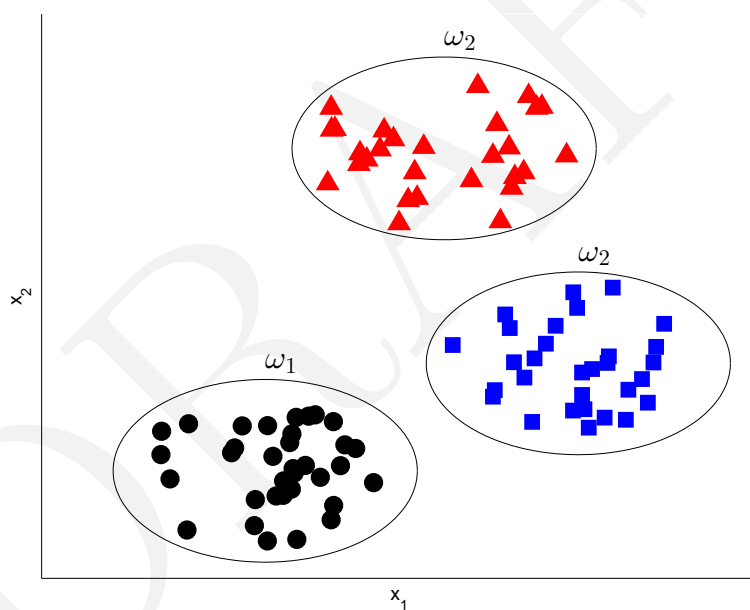
Terminologie skrze vědecké oblasti je velice pestrá. Ve statistice, kde se klasifikace často provádí binární regresí nebo obdobným postupem se pozorované proměnné nazývají „nezávislé proměnné“ a predikované kategorie jsou známy jako výsledky, které je možné považovat za závislé proměnné. Ve strojovém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

### 2.1.1 Porovnání příznakových vektorů

Jednotlivé příznakové vektory mohou být chápány jako body v mnohdimenzionálním 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

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

Třídy (shluky) v sobě sdružují vektory, které jsou si navzájem podobné (viz Obr. 2.1). 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 2.1: 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.

### 2.1.2 Míra podobnosti

Míra podobnosti  $s$  dvou vektorů z množiny  $X$  je definována jako

$$s : X \times X \longrightarrow \mathcal{R}, \quad (2.1)$$

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

$$\exists s_0 \in \mathcal{R} : -\infty < s(\mathbf{x}, \mathbf{y}) \geq s_0 < +\infty, \quad \forall \mathbf{x}, \mathbf{y} \in X, \quad (2.2)$$

$$s(\mathbf{x}, \mathbf{y}) = s(\mathbf{y}, \mathbf{x}), \quad \forall \mathbf{x}, \mathbf{y} \in X, \quad (2.3)$$

$$s(\mathbf{x}, \mathbf{x}) = s_0, \quad \forall \mathbf{x} \in X. \quad (2.4)$$

V praxi nejčastěji používané míry podobnosti jsou:

- Skalární součin

$$s_{\text{skal.souč.}}(\mathbf{x}, \mathbf{y}) = \mathbf{x}^\top \mathbf{y} = \sum_{i=1}^l x_i y_i, \quad (2.5)$$

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

- Kosinová podobnost

$$s_{\text{cos}}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x}^\top \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|}, \quad (2.6)$$

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

- Tanimotova míra

$$s_T(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x}^\top \mathbf{y}}{\|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 - \mathbf{x}^\top \mathbf{y}}. \quad (2.7)$$

### 2.1.3 Míra odlišnosti

Míra podobnosti  $d$  dvou vektorů z množiny  $X$  je definována jako

$$d : X \times X \longrightarrow \mathcal{R}, \quad (2.8)$$

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

$$\exists d_0 \in \mathcal{R} : -\infty \leq d_0 < d(\mathbf{x}, \mathbf{y}) < +\infty, \quad \forall \mathbf{x}, \mathbf{y} \in X, \quad (2.9)$$

$$d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x}), \quad \forall \mathbf{x}, \mathbf{y} \in X, \quad (2.10)$$

$$d(\mathbf{x}, \mathbf{x}) = d_0, \quad \forall \mathbf{x} \in X. \quad (2.11)$$

Mezi nejčastěji používané míry odlišnosti patří:

- $l_p$ -norma

$$d_p(\mathbf{x}, \mathbf{y}) = \left( \sum_{i=1}^l |x_i - y_i|^p \right)^{\frac{1}{p}}. \quad (2.12)$$

- $l_1$ -norma (tzv. norma Manhattan)

$$d_1(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^l |x_i - y_i|. \quad (2.13)$$

- $l_\infty$ -norma

$$d_\infty(\mathbf{x}, \mathbf{y}) = \max_{1 \leq i \leq l} |x_i - y_i|. \quad (2.14)$$

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. Známa Euklidova vzdálenost je získána pouhou modifikací  $l_p$ -normy.

### 2.1.4 Porovnání vektoru a třídy

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

- Maximální podobnostní funkce

$$\rho_{\max}(\mathbf{x}, \omega) = \max_{\mathbf{y} \in \omega} \rho(\mathbf{x}, \mathbf{y}). \quad (2.15)$$

- Minimální podobnostní funkce

$$\rho_{\min}(\mathbf{x}, \omega) = \min_{\mathbf{y} \in \omega} \rho(\mathbf{x}, \mathbf{y}). \quad (2.16)$$

- Průměrná podobnostní funkce

$$\rho_{\text{prům.}}(\mathbf{x}, \omega) = \frac{1}{n_\omega} \sum_{\mathbf{y} \in \omega} \rho(\mathbf{x}, \mathbf{y}), \quad (2.17)$$

kde  $n_\omega$  značí počet vektorů tvořících třídu  $\omega$ .

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



### 2.1.5 Reprezentant třídy

Mezi typické volby reprezentanta třídy patří:

- Průměr

$$\omega_p = \frac{1}{n_\omega} \sum_{\mathbf{y} \in \omega} \mathbf{y}, \quad (2.18)$$

kde  $n_\omega$  je počet prvků uvnitř třídy  $\omega$ . 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  $\omega_c \in \omega$  je vektor splňující:

$$\sum_{\mathbf{y} \in \omega} d(\omega_c, \mathbf{y}) \leq \sum_{\mathbf{y} \in \omega} d(\mathbf{z}, \mathbf{y}), \quad \forall \mathbf{z} \in \omega, \quad (2.19)$$

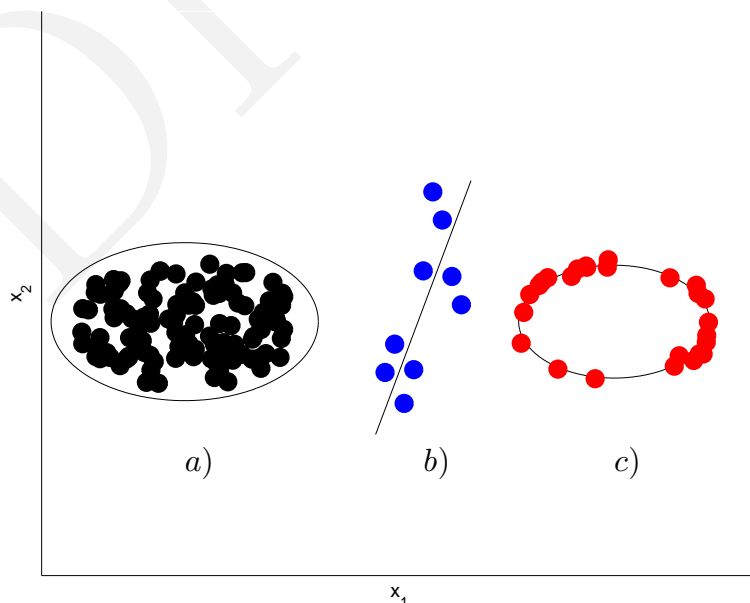
kde  $d$  značí míru odlišnosti (pokud by byla použita míra podobnosti  $s$  je nutné v předpisu otočit znaménka nerovnosti).

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

$$med(d(\omega_{med}, \mathbf{y}) | \mathbf{y} \in \omega) \leq med(d(\mathbf{z}, \mathbf{y}) | \mathbf{y} \in \omega), \quad \forall \mathbf{z} \in \omega, \quad (2.20)$$

kde  $d$  je míra odlišnosti.

Uvedené reprezentanty třídy lze použít pouze v případě, že je třída kompaktní (viz Obr. 2.2). 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 2.2: Příklad třídy a) kompaktní, b) lineární a c) sférické..

### 2.1.6 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 měříčem 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

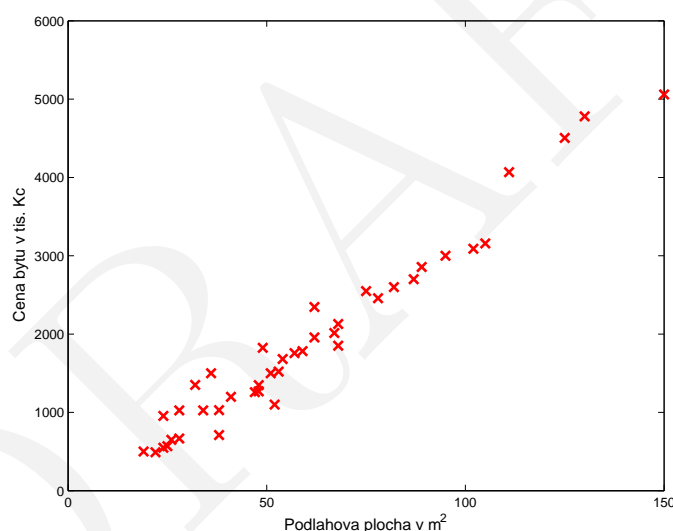
- učení s učitelem a
- učení bez učitele.

## 2.2 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  $y$ -ové 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.

### 2.2.1 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. 2.3 lze vidět vývoj ceny bytu v závislosti na podlahové ploše.



Obrázek 2.3: Vývoj ceny bytu v závislosti na podlahové ploše.

### 2.2.2 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 diskretního vstupu predikovat diskretní výstup.

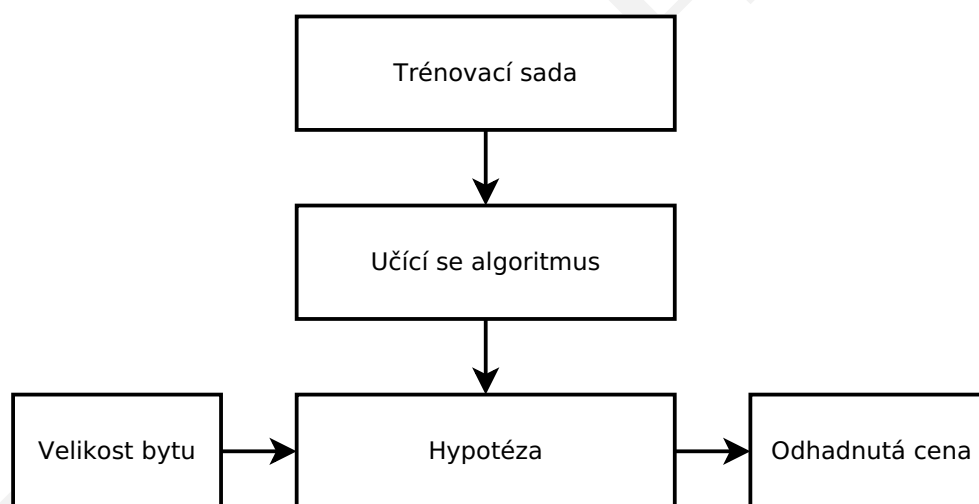
Více formálně v případě učení s učitelem potřebujeme množinu dat nazývanou trénovací množina. V našem případě predikce ceny bytu máme trénovací množinu cen bytů a k nim odpovídající podlahovou plochu. Naším úkolem je naučit se z těchto dat jak predikovat ceny bytů.

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  $(\mathbf{x}, y)$  a odpovídá jednomu řádku v tabulce 2.1. Dále  $(x^{(i)}, y^{(i)})$  označuje  $i$ -tý trénovací vzorek.

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

Tabulka 2.1: 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. 2.4.



Obrázek 2.4: 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:

$$h_{\Theta}(x) = \vartheta_0 + \vartheta_1 x, \quad (2.21)$$

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

$\vartheta$  je malá théta a  $\Theta$  je velká théta.

### 2.2.3 Ztrátová funkce

Pro nejlepší napasování našeho lineárního modelu na trénovací data budeme potřebovat určit parametr  $\vartheta_0$  a  $\vartheta_1$  v naší hypotéze:

$$h_{\Theta}(x) = \vartheta_0 + \vartheta_1 x, \quad (2.22)$$

Snažíme se zvolit parametry  $\vartheta_0$  a  $\vartheta_1$  tak, aby hypotéza  $h_{\Theta}(x)$  byla co nejblíže k  $y$  pro naše trénovací vzorky  $(x, y)$ . Jinými slovy snažíme se minimalizovat kvadratickou chybu:

$$\min_{\vartheta_0, \vartheta_1} \frac{1}{2m} \sum_{i=1}^m \left( h_{\Theta}(x^{(i)}) - y^{(i)} \right)^2, \quad (2.23)$$

kde

$$h_{\Theta}(x^{(i)}) = \vartheta_0 + \vartheta_1 x^{(i)}. \quad (2.24)$$

Přesněji označíme ztrátovou funkci:

$$J(\vartheta_0, \vartheta_1) = \frac{1}{2m} \sum_{i=1}^m \left( h_{\Theta}(x^{(i)}) - y^{(i)} \right)^2 \quad (2.25)$$

a minimalizujeme

$$\min_{\vartheta_0, \vartheta_1} J(\vartheta_0, \vartheta_1). \quad (2.26)$$

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.

## 2.2.4 Gradientní algoritmus

V kapitole 2.2.3 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  $\vartheta_0$  a  $\vartheta_1$ , dále v každé iteraci měníme hodnoty  $\vartheta_0$  a  $\vartheta_1$ , 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:

$$J(\vartheta_0, \dots, \vartheta_n) \quad (2.27)$$

a snažíme se minimalizovat:

$$\min_{\vartheta_0, \dots, \vartheta_n} J(\vartheta_0, \dots, \vartheta_n) \quad (2.28)$$

Definice gradientního algoritmu: opakuj dokud není dosaženo konvergence

$$\vartheta_j = \vartheta_j - \alpha \frac{\partial}{\partial \vartheta_j} J(\vartheta_0, \vartheta_1), \quad (2.29)$$

pro  $j = \{0, 1\}$ , pro náš příklad (lineární regrese jedné proměnné) v sekci 2.2.1. V rovnici 2.29 proměnná  $\alpha$  určuje, jak velké kroky bude gradientní algoritmus dělat při hledání minima a nazývá se rychlost učení. V případě velké hodnoty  $\alpha$  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  $\alpha$  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  $\alpha$  konstantní. Proto se v reálných aplikacích volí nejdříve větší  $\alpha$ , které se s rostoucí hodnotou iterací nebo času zmenšuje.

### Poznámka

Pro implementaci: je nutné aktualizovat všechny  $\Theta$  současně

$$temp_0 = \vartheta_0 - \alpha \frac{\partial}{\partial \vartheta_0} J(\vartheta_0, \vartheta_1), \quad (2.30)$$

$$temp_1 = \vartheta_1 - \alpha \frac{\partial}{\partial \vartheta_1} J(\vartheta_0, \vartheta_1), \quad (2.31)$$

$$\vartheta_0 = temp_0, \quad (2.32)$$

$$\vartheta_1 = temp_1. \quad (2.33)$$

Nyní budeme aplikovat gradientní algoritmus na náš problém lineární regrese

$$\frac{\partial}{\partial \vartheta_j} J(\vartheta_0, \vartheta_1) = \frac{\partial}{\partial \vartheta_j} \cdot \frac{1}{2m} \sum_{i=1}^m \left( h_{\Theta}(x^{(i)}) - y^{(i)} \right)^2, \quad (2.34)$$

po dosazení z rovnice 2.24 dostáváme

$$\frac{\partial}{\partial \vartheta_j} J(\vartheta_0, \vartheta_1) = \frac{\partial}{\partial \vartheta_j} \cdot \frac{1}{2m} \sum_{i=1}^m (\vartheta_0 + \vartheta_1 x^{(i)} - y^{(i)})^2, \quad (2.35)$$

po zderivování podle jednotlivých parametrů  $\Theta$  získáme

$$\frac{\partial}{\partial \vartheta_0} J(\vartheta_0, \vartheta_1) = \frac{1}{m} \sum_{i=1}^m (h_{\Theta}(x^{(i)}) - y^{(i)}), \quad (2.36)$$

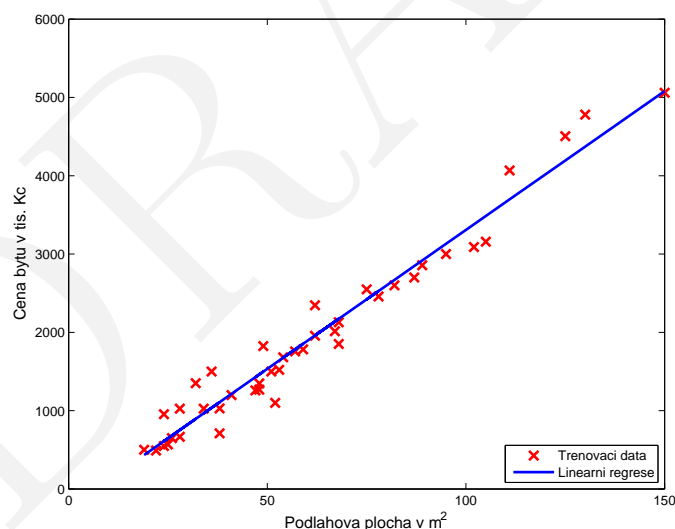
$$\frac{\partial}{\partial \vartheta_1} J(\vartheta_0, \vartheta_1) = \frac{1}{m} \sum_{i=1}^m (h_{\Theta}(x^{(i)}) - y^{(i)}) x^{(i)}. \quad (2.37)$$

Výsledný konkrétní tvar rovnice 2.29 pro  $\vartheta_0$  a  $\vartheta_1$

$$\vartheta_0 = \vartheta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\Theta}(x^{(i)}) - y^{(i)}), \quad (2.38)$$

$$\vartheta_1 = \vartheta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\Theta}(x^{(i)}) - y^{(i)}) x^{(i)}. \quad (2.39)$$

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

### 2.2.5 Vícerozměrná lineární regrese

V sekci 2.2.2 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 2.2.

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 2.2: Příklady vícerozměrných trénovacích vzorků.

Proměnnou  $n$  budeme označovat počet příznaků (v našem případě v tabulce 2.2  $x_{1,\dots,4}$ , proto  $n = 4$ .) Dále budeme stále jako u jednorozměrného případu označovat vstupní příznaky (všechny = sloupcový vektor)  $\mathbf{x}^{(i)}$  pro  $i$ -tý trénovací vzor. A hodnota jednotlivého  $j$ -tého příznaku  $i$ -tého trénovacího vzoru bude označována  $\mathbf{x}_j^{(i)}$ .

V případě jednorozměrné lineární regrese byla naše hypotéza  $h_{\Theta}$  reprezentována

$$h_{\Theta}(x) = \vartheta_0 + \vartheta_1 x, \quad (2.40)$$

v našem vícerozměrném případě nabývá tvaru

$$h_{\Theta}(\mathbf{x}) = \vartheta_0 + \vartheta_1 x_1 + \vartheta_2 x_2 + \vartheta_3 x_3 + \vartheta_4 x_4. \quad (2.41)$$

V případě 2.40 i 2.41 je možné si všimnout, že  $x_0 = 1$  čehož dále využijeme, přesněji  $\mathbf{x}_0^{(i)} = 1$  z důvodu dodržení konvence. Proto  $\mathbf{x} \in \mathbb{R}^{n+1}$  a to samé platí pro parametry  $\Theta \in \mathbb{R}^{n+1}$ .

#### Poznámka

V našem příkladě odpovídají

$$\mathbf{x} = [x_0, x_1, x_2, x_3, x_4]^{\top} \quad (2.42)$$

a

$$\Theta = [\vartheta_0, \vartheta_1, \vartheta_2, \vartheta_3, \vartheta_4]^{\top}. \quad (2.43)$$

Obecněji lze vícerozměrnou hypotézu zapsat jako

$$h_{\Theta}(\mathbf{x}) = \vartheta_0 + \vartheta_1 x_1 + \vartheta_2 x_2 + \dots + \vartheta_n x_n, \quad (2.44)$$

neboli jednoduše jako součin vektorů (vnitřní produkt)

$$h_{\Theta}(\mathbf{x}) = \Theta^{\top} \mathbf{x}. \quad (2.45)$$



### 2.2.6 Gradientní algoritmus - vícerozměrná verze

Budeme potřebovat hypotézu

$$h_{\Theta}(\mathbf{x}) = \Theta^{\top} \mathbf{x} = \vartheta_0 x_0 + \vartheta_1 x_1 + \vartheta_2 x_2 + \dots + \vartheta_n x_n, \quad (2.46)$$

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

$$J(\Theta) = \frac{1}{2m} \sum_{i=1}^m \left( h_{\Theta}(\mathbf{x}^{(i)}) - y^{(i)} \right)^2. \quad (2.47)$$

Gradientní algoritmus pak bude mít tvar: opakuj

$$\vartheta_j = \vartheta_j - \alpha \frac{\partial}{\partial \vartheta_j} J(\Theta), \quad (2.48)$$

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

Vztah z rovnice 2.48 lze zapsat jako

$$\frac{\partial}{\partial \vartheta_j} J(\Theta) = \frac{1}{m} \sum_{i=1}^m \left( h_{\Theta}(\mathbf{x}^{(i)}) - y^{(i)} \right) \mathbf{x}_j^{(i)} \quad (2.49)$$

a po spojení rovnic 2.48 a 2.49 získáváme výsledný tvar

$$\vartheta_j = \vartheta_j - \alpha \frac{1}{m} \sum_{i=1}^m \left( h_{\Theta}(\mathbf{x}^{(i)}) - y^{(i)} \right) \mathbf{x}_j^{(i)}, \quad (2.50)$$

pokud bychom výslednou rovnici 2.50 rozepsali pro první dva parametry  $\vartheta_0$  a  $\vartheta_1$  dostali bychom

$$\vartheta_0 = \vartheta_0 - \alpha \frac{1}{m} \sum_{i=1}^m \left( h_{\Theta}(\mathbf{x}^{(i)}) - y^{(i)} \right) \mathbf{x}_0^{(i)}, \quad \mathbf{x}_0^{(i)} = 1 \quad (2.51)$$

$$\vartheta_1 = \vartheta_1 - \alpha \frac{1}{m} \sum_{i=1}^m \left( h_{\Theta}(\mathbf{x}^{(i)}) - y^{(i)} \right) \mathbf{x}_1^{(i)} \quad (2.52)$$

$\vdots$

### 2.2.7 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 měly 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  $\mu$ , konkrétně pro  $i$ -tý příznak

$$x_i \leftarrow \frac{x_i - \mu_i}{\max - \min} \quad (2.53)$$

nebo

$$x_i \leftarrow \frac{x_i - \mu_i}{std} \quad (2.54)$$

( $std$  = Standard deviation).

### 2.2.8 Analytické řešení

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

#### Příklad:

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

$$J(\vartheta) = a\vartheta^2 + b\vartheta + c \quad (2.55)$$

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

$$\frac{d}{d\vartheta} J(\vartheta) = \dots \equiv 0 \quad (2.56)$$

a vyřešíme pro parametr  $\vartheta$ .

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

$$J(\vartheta_0, \vartheta_1, \dots, \vartheta_m) = \frac{1}{2m} \sum_{i=1}^m \left( h_{\Theta}(\mathbf{x}^{(i)}) - y^{(i)} \right)^2 \quad (2.57)$$

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

$$\frac{\partial}{\partial \vartheta_j} = \dots \equiv 0 \quad (2.58)$$

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

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

Nyní můžeme zapsat předpis pro výpočet parametrů  $\Theta$  analyticky

$$\Theta = (\mathbf{X}^{\top} \mathbf{X})^{-1} \mathbf{X}^{\top} \mathbf{y}. \quad (2.59)$$

Pro analytické řešení není nutné příznaky normovat, jako v případě gradientního algoritmu, na stejné měřítko, jak je popsáno v sekci 2.2.7.

### 2.2.9 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 $\alpha$ .	Není nutné volit konstantu učení $\alpha$ .
Potřebuje mnoho iterací.	Nepotřebuje iterovat.
Pracuje dobře, když je $n$ velké.	Pomalé pokud je $n$ velké.
	Nutnost vypočítat inverzi $(\mathbf{X}^{\top} \mathbf{X})^{-1}$ .
	(Složitost $\mathcal{O}(n^3)$ ).
Zvolit pokud $n > 10^6$ .	Zvolit pokud $n < 10^4$ .

## 2.3 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.

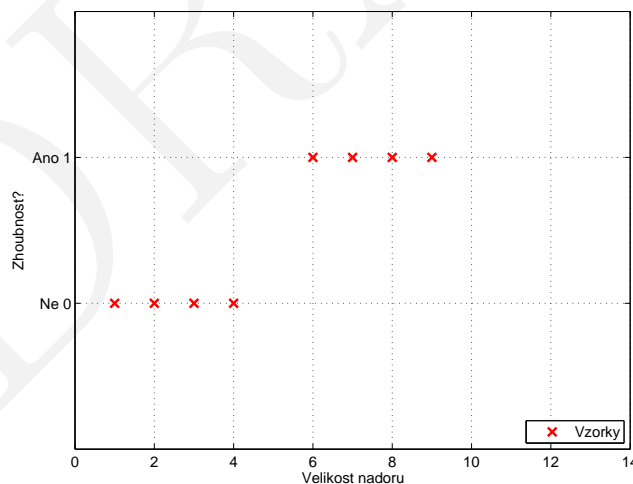
### 2.3.1 Klasifikace

Na úvod si uveďme několik klasifikačních příkladů:

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

Z příkladů je zřejmé, že se jedná o rozhodnutí mezi dvěma možnostmi, tedy  $y \in \{0, 1\}$ , kde 0 je „negativní třída“ (nezhoobný 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  $y \in \{0, 1, 2, 3, \dots\}$ . 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 nezhoobný, tedy buďto 1 nebo 0. Dostupná data pro náš příklad jsou na obrázku 2.6.



Obrázek 2.6: Trénovací data.

Jedna z věcí, kterou bychom mohli udělat, je aplikace algoritmu lineární regrese, který známe z kapitoly 2.2. Výsledek aplikace hypotézy

$$h_{\Theta}(x) = \Theta^{\top} x \quad (2.60)$$

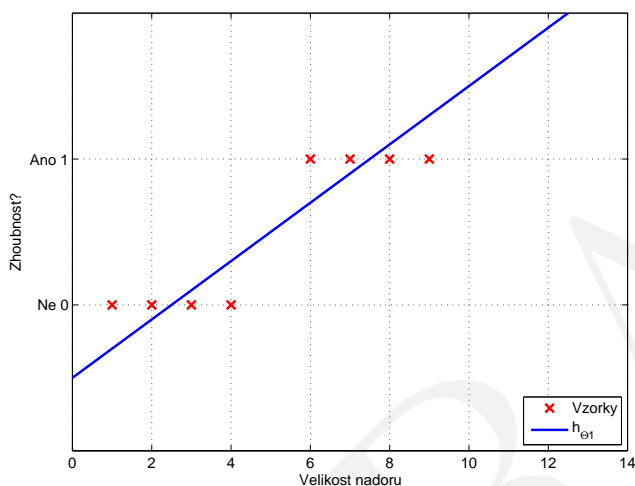
lze vidět na Obr. 2.7.

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

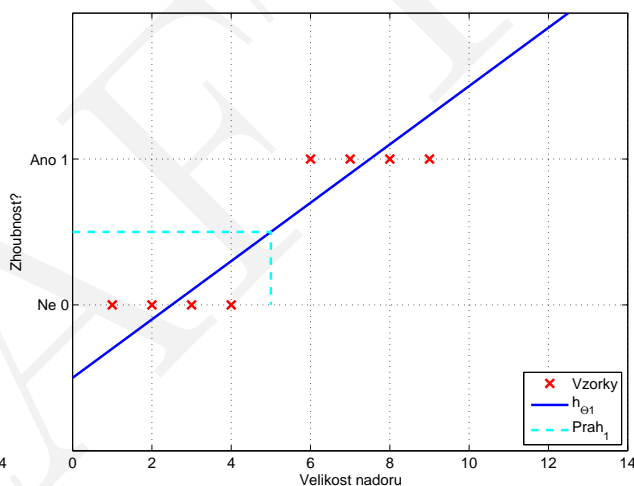
$$h_{\Theta}(x) \geq 0.5, \quad \text{predikuj } y = 1, \quad (2.61)$$

$$h_{\Theta}(x) < 0.5, \quad \text{predikuj } y = 0. \quad (2.62)$$

Tento poznatek je reprezentován na Obr. 2.8.



Obrázek 2.7: Lineární regrese.

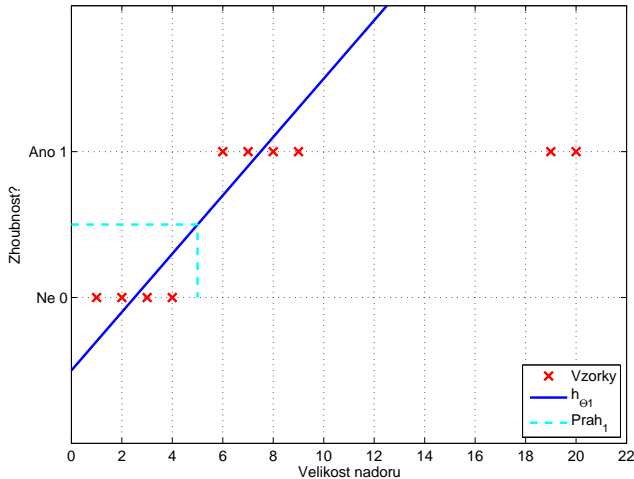


Obrázek 2.8: Lineární regrese - určení rozhodovacího prahu.

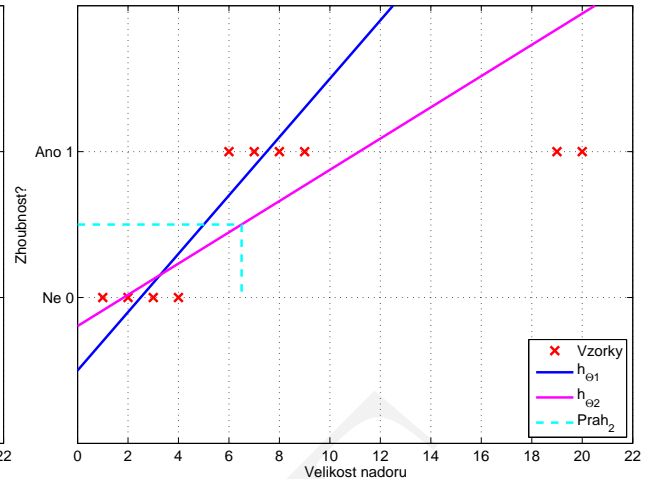
V tomto příkladě jsme si ukázali, že lineární regrese dělá něco co by se dalo považovat za řešení klasifikačního problému. Na následujícím rozšíření našeho příkladu bude ukázáno, že tomu tak není.

Rozšíříme náš příklad o další vzorky viz Obr. 2.9, kde nám přibyly dva vzorky více vpravo. Po aplikování lineární regrese, na rozšířená data, získáváme výsledek, který lze vidět na Obr. 2.10. Aktuální hypotéza  $h_{\Theta 2}$  již nesplňuje podmínky definované vztahy 2.61 a 2.62, 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  $h_{\Theta} > 1$  nebo  $h_{\Theta} < 0$ , přičemž výsledek binární klasifikace je pouze  $y = 0$  nebo  $y = 1$ . Proto bude nutné v dalších kapitolách přijít se sofistikovanějším postupem binární regrese (klasifikace), která splňuje podmínku  $0 \leq h_{\Theta} \leq 1$ .



Obrázek 2.9: Rozšíření trénovací množiny dat o dva vzorky vpravo.



Obrázek 2.10: Lineární regrese - určení rozhodovacího prahu a zobrazení jeho selhání při predikci.

### 2.3.2 Reprezentace hypotézy

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

V sekci 2.3.1 jsme zjistili, že bychom si pro binární regresní model přáli, aby platilo

$$0 \leq h_{\Theta}(x) \leq 1, \quad (2.63)$$

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

$$h_{\Theta}(x) = \Theta^T x, \quad (2.64)$$

tuto hypotézu upravíme na tvar

$$h_{\Theta}(x) = g(\Theta^T x), \quad (2.65)$$

kde funkce  $g$  je definována následovně

$$g(z) = \frac{1}{1 + e^{-z}}, \quad (2.66)$$

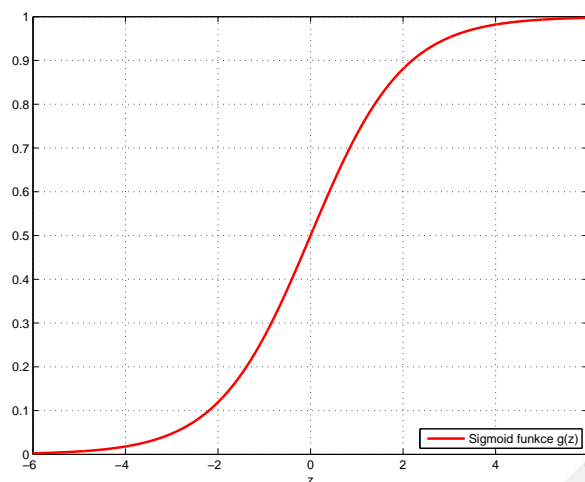
tato funkce se nazývá sigmoid funkce (Sigmoid function) nebo logistická/binární funkce (Logistic function). Jméno logistická funkce je důvod, proč vznikl termín logistická/binární regrese. Pokud spojíme rovnice 2.65 a 2.66 získáme tvar

$$h_{\Theta}(x) = \frac{1}{1 + e^{-\Theta^T x}} \quad (2.67)$$

a průběh grafu funkce  $g(z)$  lze vidět na Obr. 2.11.

Poznamenejme, že

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

Obrázek 2.11: Sigmoidní funkce  $g(z)$ .

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

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

### Příklad

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

$$h_{\Theta}(\mathbf{x}) = P(y = 1 | \mathbf{x}; \Theta), \quad (2.68)$$

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

$$P(y = 0 | \mathbf{x}; \Theta) + P(y = 1 | \mathbf{x}; \Theta) = 1, \quad (2.69)$$

neboli

$$P(y = 0 | \mathbf{x}; \Theta) = 1 - P(y = 1 | \mathbf{x}; \Theta). \quad (2.70)$$

### Poznámka

Pravděpodobnost může nabývat hodnot 0 – 1, neboli 0 – 100%.

### 2.3.3 Rozhodovací hranice

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

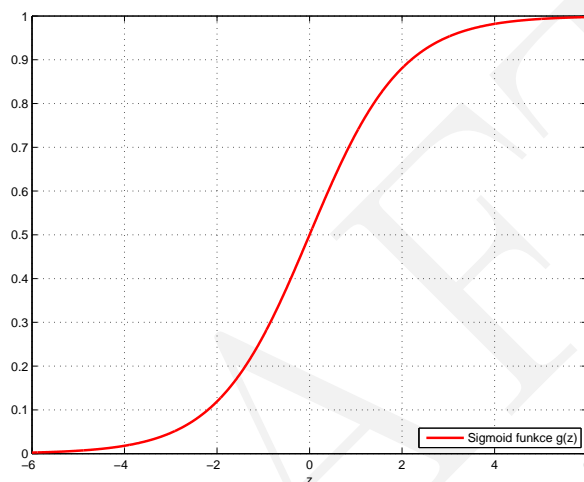
V kapitole 2.3.2 jsme si ukázali tvar hypotézy (sigmoid funkce)

$$h_{\Theta}(\mathbf{x}) = g(\Theta^{\top} \mathbf{x}), \quad (2.71)$$

kde

$$g(z) = \frac{1}{1 + e^{-z}} \quad (2.72)$$

a graf lze vidět na Obr. 2.12.



Obrázek 2.12: 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

$$h_{\Theta}(\mathbf{x}) = g(\Theta^{\top} \mathbf{x}) = P(y = 1 | \mathbf{x}; \Theta). \quad (2.73)$$

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

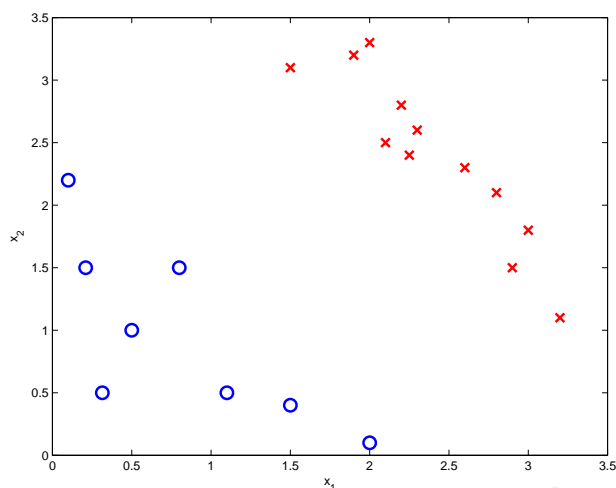


## Příklad - lineárně separabilní třídy

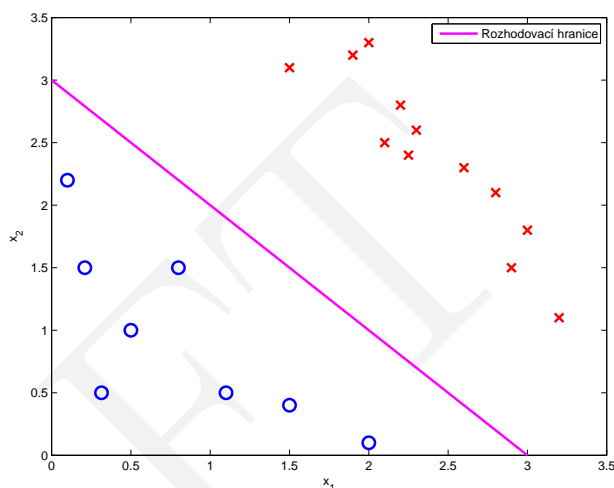
Předpokládejme, že máme trénovací sadu, která je znázorněna na Obr. 2.13 a naše hypotéza má tvar

$$h_{\Theta}(\mathbf{x}) = g(\vartheta_0 + \vartheta_1 x_1 + \vartheta_2 x_2). \quad (2.74)$$

Nyní se nebudeme zabývat tím, jak přizpůsobit parametry tomuto modelu. Předpokládejme, že  $\Theta = [\vartheta_0, \vartheta_1, \vartheta_2]^T = [-3, 1, 1]^T$ .



Obrázek 2.13: Vizualizace dvou tříd.



Obrázek 2.14: 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

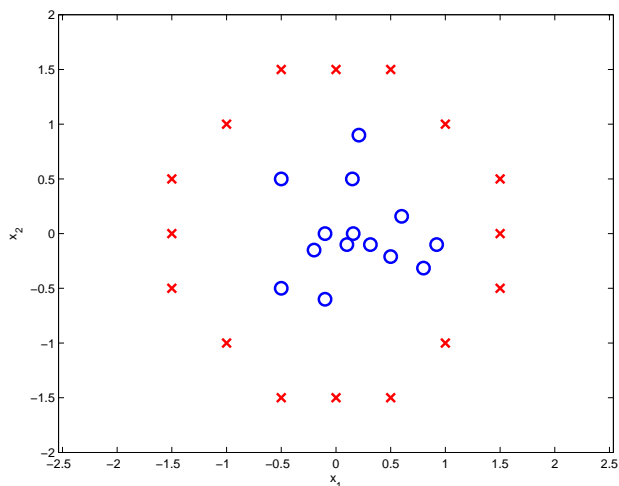
$$-3 + x_1 + x_2 \geq 0 \quad (2.75)$$

( $\Theta^T \mathbf{x} \geq 0$ ), jinými slovy pro všechna  $x_1$  a  $x_2$ , které splňují rovnici 2.75 bude naše hypotéza predikovat výstup  $y = 1$ . Řešení rovnice 2.75 je naznačeno na Obr. 2.14 a odpovídají mu červené křížky nad rozhodovací hranicí.

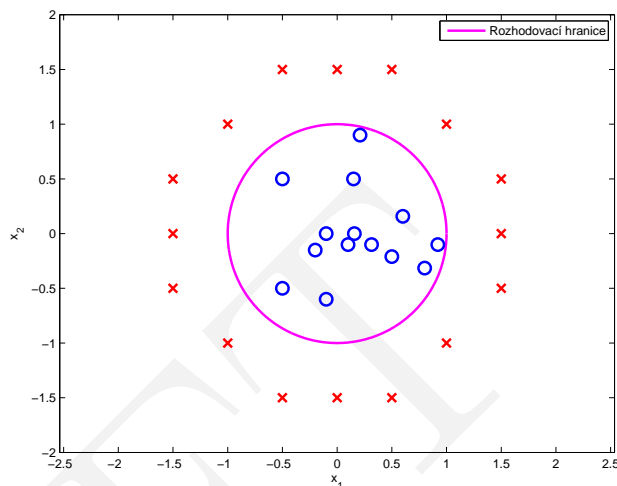
V případě, kdy  $x_1 + x_2 = 3$ , tak to odpovídá přesně  $h_{\Theta}(\mathbf{x}) = 0.5$ . Jinak řečeno řešení odpovídá přesně rozhodovací hranici mezi třídami.

## 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. 2.15.



Obrázek 2.15: Vizualizace lineárně neseparabilních tříd.



Obrázek 2.16: Vizualizace lineárně neseparabilních tříd společně s rozhodovací hranicí.

Naše hypotéza má tvar  $h_{\Theta}(\mathbf{x}) = g(\vartheta_0 + \vartheta_1 x_1 + \vartheta_2 x_2 + \vartheta_3 x_1^2 + \vartheta_4 x_2^2)$ , kde jsme oproti předcházejícímu příkladu (rovnice 2.74) přidali dva příznaky  $x_1^2$  a  $x_2^2$  a máme 5 parametrů  $\vartheta_0, \dots, \vartheta_4$ . Nebudeme zabývat tím, jak vypočítat parametry  $\Theta$  (bude popsáno v následujících sekcích), řekněme, že  $\Theta = [\vartheta_0, \vartheta_1, \vartheta_2, \vartheta_3, \vartheta_4]^T = [-1, 0, 0, 1, 1]^T$ . 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. 2.16.

## Poznámka

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

### 2.3.4 Ztrátová funkce

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

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

$$\{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \dots, (\mathbf{x}^{(m)}, y^{(m)})\} \quad (2.76)$$

o velikosti  $m$  vzorků, kde

$$\mathbf{x}^{(i)} = [x_0, x_1, \dots, x_n]^\top, \quad (2.77)$$

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

$$h_{\Theta}(\mathbf{x}) = \frac{1}{1 + e^{-\Theta^\top \mathbf{x}}}. \quad (2.78)$$

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

Připomeňme si ztrátovou funkci  $J(\Theta)$  pro lineární regresi (viz sekce 2.2.3), kterou zapíšeme v následujícím tvaru

$$J(\Theta) = \frac{1}{m} \sum_{i=1}^m \frac{1}{2} (h_{\Theta}(\mathbf{x}^{(i)}) - y^{(i)})^2, \quad (2.79)$$

dále můžeme ztrátovou funkci přepsat do tvaru

$$J(\Theta) = \frac{1}{m} \sum_{i=1}^m \text{Cena}(h_{\Theta}(\mathbf{x}^{(i)}), y^{(i)}), \quad (2.80)$$

kde

$$\text{Cena}(h_{\Theta}(\mathbf{x}^{(i)}), y^{(i)}) = \frac{1}{2} (h_{\Theta}(\mathbf{x}^{(i)}) - y^{(i)})^2. \quad (2.81)$$

Nyní lze z rovnice 2.80 lépe vidět, že ztrátová funkce je  $\frac{1}{m}$  krát součet  $\text{Cen}$  přes trénovací sadu. Rovnici 2.81 lze dále zjednodušit vynecháním horních indexů

$$\text{Cena}(h_{\Theta}(\mathbf{x}), y) = \frac{1}{2} (h_{\Theta}(\mathbf{x}) - y)^2 \quad (2.82)$$

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

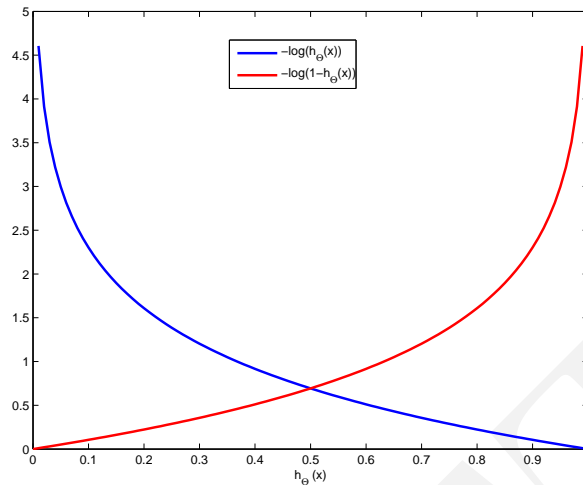
Tento předpis 2.81 pro výpočet  $\text{Ceny}$  platí pro lineární regresi. Avšak v této sekci se zajímáme o binární regresi. Pokud bychom chtěli minimalizovat tuto ztrátovou funkci v rovnici 2.80, tak tato funkce může být nekonvexní vzhledem k parametrům  $\Theta$  (není zaručeno, že dosáhneme globálního minima, jako v případě konvexní ztrátové funkce).

Uvedme si ztrátovou funkci, která se využívá při binární regresi

$$\text{Cena}(h_{\Theta}(\mathbf{x}), y) = \begin{cases} -\log(h_{\Theta}(\mathbf{x})), & \text{pro } y = 1 \\ -\log(1 - h_{\Theta}(\mathbf{x})), & \text{pro } y = 0 \end{cases} \quad (2.83)$$

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 2.83, rozebereme podrobněji. Na Obr. 2.17 lze vidět průběh ztrátové funkce.



Obrázek 2.17: Průběh ztrátové funkce.

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

Pro  $y = 0$  platí obdobně druhá část rovnice 2.83.

### 2.3.5 Zjednodušená ztrátová funkce a gradientní metoda

Naše ztrátová funkce  $J(\Theta)$  ve tvaru

$$J(\Theta) = \frac{1}{m} \sum_{i=1}^m Cena(h_{\Theta}(\mathbf{x}^{(i)}), y^{(i)}), \quad (2.84)$$

kde

$$Cena(h_{\Theta}(\mathbf{x}), y) = \begin{cases} -\log(h_{\Theta}(\mathbf{x})), & \text{pro } y = 1 \\ -\log(1 - h_{\Theta}(\mathbf{x})), & \text{pro } y = 0 \end{cases} \quad (2.85)$$

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

Naší  $Cenu$  z rovnice 2.85 můžeme zapsat v jedné rovnici, která má tvar

$$Cena(h_{\Theta}(\mathbf{x}), y) = -y \log(h_{\Theta}(\mathbf{x})) - (1 - y) \log(1 - h_{\Theta}(\mathbf{x})), \quad (2.86)$$

pokud  $y = 1$ , tak rovnice 2.86 přechází na první část rovnice 2.85 a pokud  $y = 0$ , tak rovnice 2.86 přechází na druhou část rovnice 2.85.

Po spojení a úpravě rovnic 2.86 a 2.84 získáme výsledný tvar

$$J(\Theta) = -\frac{1}{m} \left[ \sum_{i=1}^m y^{(i)} \log(h_{\Theta}(\mathbf{x}^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\Theta}(\mathbf{x}^{(i)})) \right]. \quad (2.87)$$

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

$$\min_{\Theta} J(\Theta), \quad (2.88)$$

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

$$h_{\Theta}(x) = \frac{1}{1 + e^{-\Theta^T x}}. \quad (2.89)$$

Nyní můžeme konečně zapsat celý gradientní algoritmus, opakuj

$$\vartheta_j = \vartheta_j - \alpha \frac{\partial}{\partial \vartheta_j} J(\Theta), \quad (2.90)$$

po každé iteraci aktualizuj všechny parametry  $\vartheta_j$  najednou dokud algoritmus nedokonverguje.

### Poznámka

V rovnici 2.90 bude využit tvar

$$\frac{\partial}{\partial \vartheta_j} J(\Theta) = \frac{1}{m} \sum_{i=1}^m (h_{\Theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}. \quad (2.91)$$

a proto je výhodnější psát rovnici 2.90 ve tvaru

$$\vartheta_j = \vartheta_j - \alpha \sum_{i=1}^m (h_{\Theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}. \quad (2.92)$$

Algoritmus vypadá totožně jako v případě lineární regrese (viz 2.50) s rozdílnou hypotézou  $h_{\Theta}(x)$ , v případě lineární regrese

$$h_{\Theta}(x) = \Theta^T x \quad (2.93)$$

a v případě binární regrese se jedná o

$$h_{\Theta}(x) = \frac{1}{1 + e^{-\Theta^T x}}. \quad (2.94)$$

### Poznámka

Při výpočtu parametrů  $\Theta$  pomocí gradientního algoritmu v případě binární regrese je také vhodné využít Features scaling, jako v případě lineární regrese (viz sekce 2.2.7).

### 2.3.6 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  $J(\Theta)$  a chceme ji minimalizovat

$$\min_{\Theta} J(\Theta). \quad (2.95)$$

Musíme tedy znát rovnice pro výpočet  $J(\Theta)$  a  $\frac{\partial}{\partial \vartheta_j} J(\Theta)$  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 $\alpha$ .	
L-BFGS	Rychlejší než gradientní metoda.	

Tabulka 2.3: Příklady pokročilejších optimalizačních algoritmů.

#### Příklad

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

$$\frac{\partial}{\partial \vartheta_1} J(\Theta) = 2 \cdot (\vartheta_1 - 5), \quad (2.96)$$

$$\frac{\partial}{\partial \vartheta_2} J(\Theta) = 2 \cdot (\vartheta_2 - 5). \quad (2.97)$$

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.

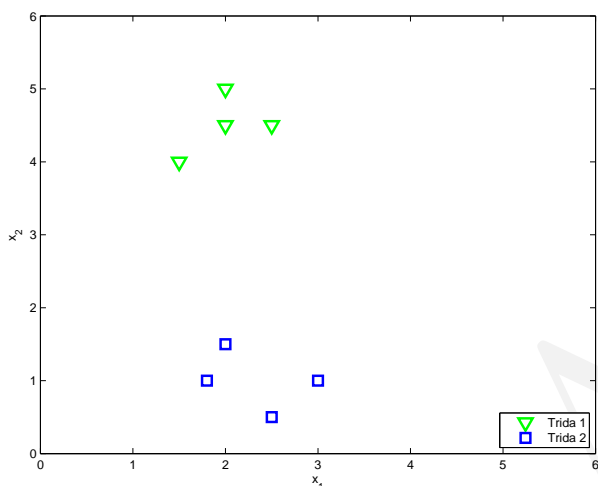
```
1 initialTheta = [1; 2];  
2 options = optimset('GradObj', 'on', 'MaxIter', 100);  
3 [optTheta, functionVal, exitFlag] = ...  
4     fminunc(@costFunction, initialTheta, options);
```

Výsledné hodnoty pro  $\vartheta_{1,2} = 5$ , což lze pro tento jednoduchý případ vidět z rovnic 2.96 a 2.97. Jak lze vidět z příkladu, nebylo nutné nastavovat parametr  $\alpha$ , funkce *fminunc* v MATLABu si jej nastavila sama.

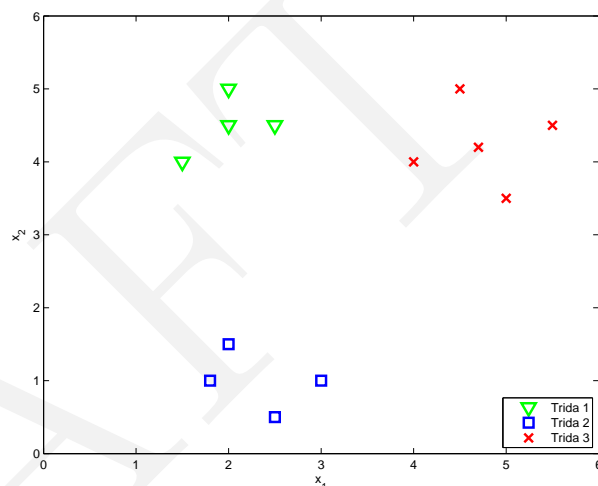
### 2.3.7 Klasifikace do více tříd

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 popíšme 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. 2.18 a 2.19 je vidět rozdíl v klasifikaci do dvou a více tříd.



Obrázek 2.18: Klasifikace do dvou tříd.



Obrázek 2.19: Klasifikace do více tříd.

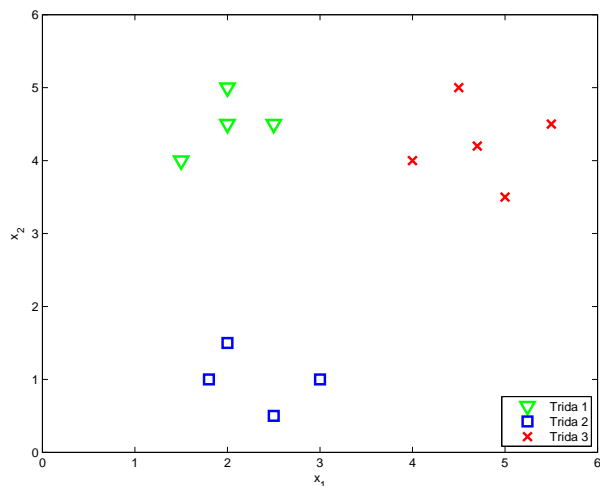
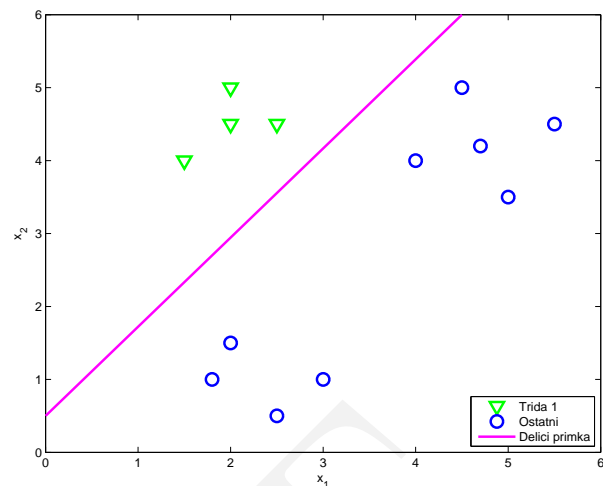
Z předešlých sekcí víme jak klasifikovat do dvou tříd. Nyní tento přístup použijeme při řešení problému klasifikace do více tříd. Na Obr. 2.20 je znázorněno rozložení tříd a následně na Obr. 2.21 je ukázána klasifikace třídy 1 proti třídě, která je zkonstruována spojením všech ostatních tříd (v našem případě tříd 2 a 3). Na dalších Obr. 2.22 a 2.23 jsou naznačeny zbylé kombinace. Pro jednotlivé klasifikace je nutné vytvořit hypotézu

$$h_{\Theta}^{(i)}(\mathbf{x}), \quad i \in \{1, 2, 3\}, \quad (2.98)$$

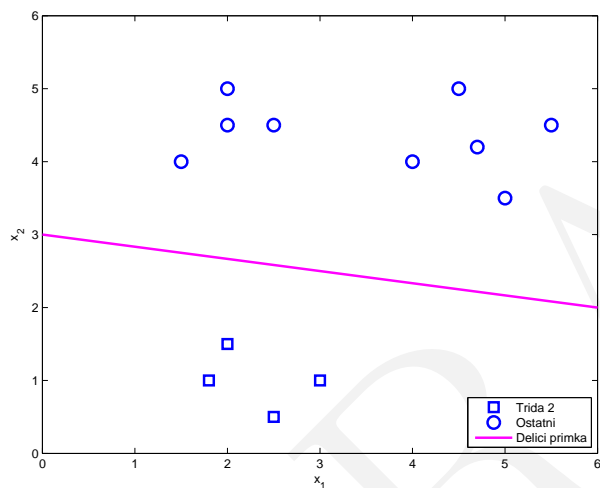
která odpovídá pravděpodobnostem

$$h_{\Theta}^{(i)}(\mathbf{x}) = P(y = i | \mathbf{x}; \Theta). \quad (2.99)$$

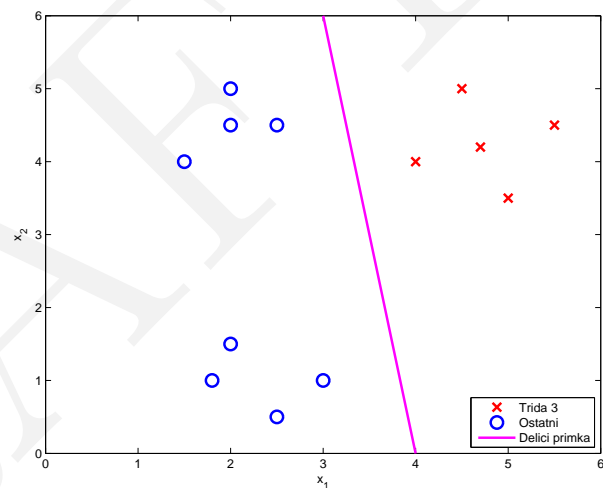



 Obrázek 2.20: Rozložení tříd  $y = \{1, 2, 3\}$ .


Obrázek 2.21: Klasifikace třídy 1.



Obrázek 2.22: Klasifikace třídy 2.



Obrázek 2.23: Klasifikace třídy 3.

## Shrnutí

Nejdříve je nutné natrénovat binární klasifikátory  $h_{\Theta}^{(i)}(\mathbf{x})$  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

$$\max_i h_{\Theta}^{(i)}(\mathbf{x}). \quad (2.100)$$

## 2.4 Regularizace

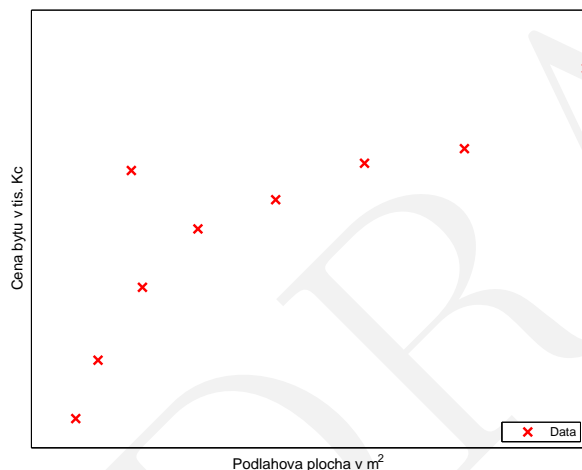
### 2.4.1 Problém přetrénování

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.

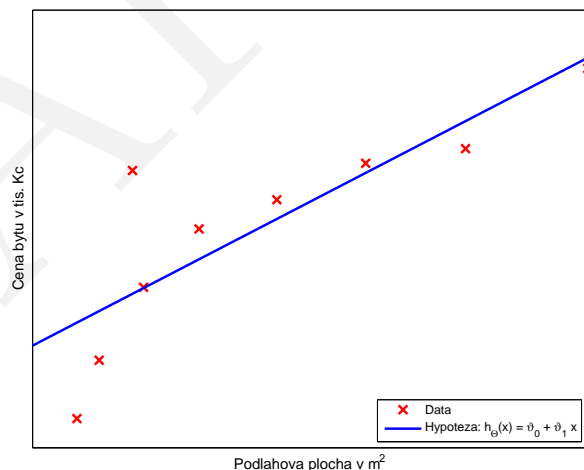
V této sekci si vysvětlíme v čem problém přetrénování spočívá a v následujících sekcích bude popsán postup jak se s daným problémem vypořádat.

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

Na Obr. 2.24 jsou znázorněná trénovací data. Pokud využijeme lineární regresi společně s hypotézou  $h_{\Theta}(\mathbf{x}) = \vartheta_0 + \vartheta_1 x$ , tak získáme predikci ve tvaru, který lze vidět na Obr. 2.25. V tomto případě naše zvolená hypotéza  $h_{\Theta}(\mathbf{x})$  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).



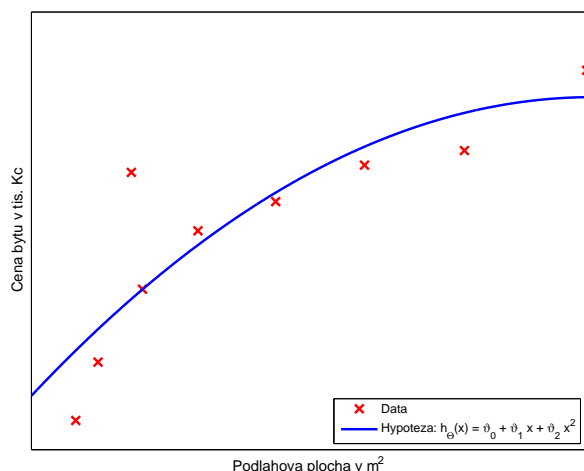
Obrázek 2.24: Trénovací data.



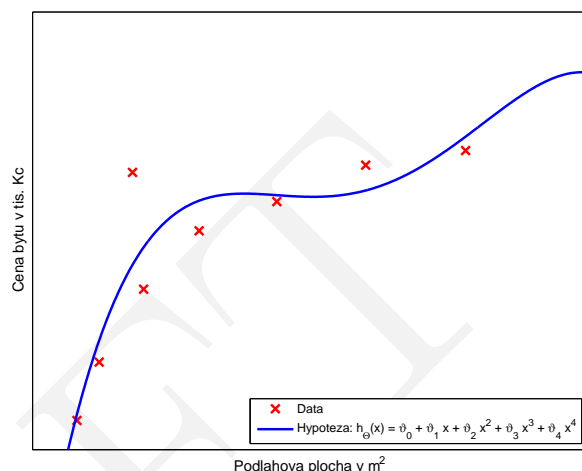
Obrázek 2.25: Lineární regrese (*underfit*, *high bias*).

Na Obr. 2.26 lze vidět regresi s hypotézou ve tvaru  $h_{\Theta}(\mathbf{x}) = \vartheta_0 + \vartheta_1 x + \vartheta_2 x^2$ , která velmi dobře zobecňuje trénovací data a proto by byla v tomto případě nejlepší volbou.

Dále na Obr. 2.27 lze vidět regresi s hypotézou  $h_{\Theta}(\mathbf{x}) = \vartheta_0 + \vartheta_1 x + \vartheta_2 x^2 + \vartheta_3 x^3 + \vartheta_4 x^4$ , 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 2.26: Regrese pomocí hypotézy ve tvaru polynomu druhého stupně (*just right*).



Obrázek 2.27: Regrese pomocí hypotézy ve tvaru polynomu čtvrtého stupně (*overfitting, high variance*).

### Přetrénování (*overfitting*):

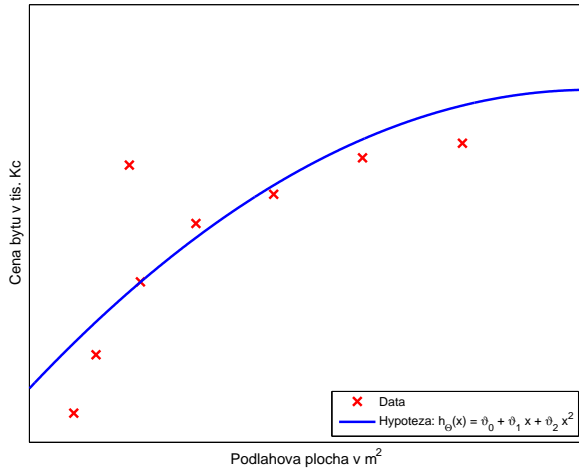
V případě, že máme příliš mnoho příznaků, hypotéza může velmi dobře respektovat trénovací data ( $J(\Theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\Theta}(\mathbf{x}^{(i)}) - y^{(i)})^2 \approx 0$ ), ale nebude schopna správně klasifikovat nově příchozí vzorky. Nebude schopna generalizovat, tedy predikce na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

### Možnosti odstranění přetrénování:

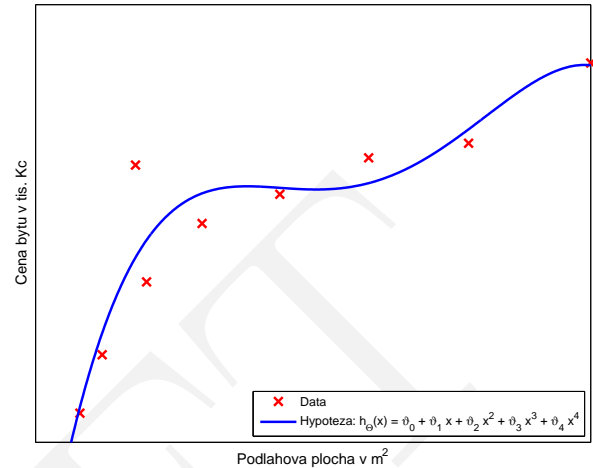
1. Redukovat počet příznaků.
  - Manuálně zvolit příznaky, které budou zachovány.
  - Automatická redukce příznaků například PCA.
2. Regularizace.
  - Zachovat všechny příznaky, ale redukovat velikost / váhu parametrů  $\vartheta_j$ .
  - Funguje správně pokud máme velké množství příznaků, kde každý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

### 2.4.2 Ztrátová funkce

V sekci 2.4.1 jsme si ukázali jak lze pomocí kvadratické hypotézy  $h_{\Theta}(\mathbf{x}) = \vartheta_0 + \vartheta_1 x + \vartheta_2 x^2$  dobře zobecnit naše trénovací data. Dále již víme, že hypotéza ve tvaru  $h_{\Theta}(\mathbf{x}) = \vartheta_0 + \vartheta_1 x + \vartheta_2 x^2 + \vartheta_3 x^3 + \vartheta_4 x^4$  nezobecňuje naší trénovací množinu dat dobře.



Obrázek 2.28: Regrese pomocí hypotézy ve tvaru polynomu druhého stupně (*just right*).



Obrázek 2.29: Regrese pomocí hypotézy ve tvaru polynomu čtvrtého stupně (*overfitting, high variance*).

Předpokládejme, že budeme penalizovat parametry  $\vartheta_3$  a  $\vartheta_4$  (jejich hodnoty budou malé). V rovnici 2.101 je ukázána naše známá optimalizace

$$\min_{\Theta} \frac{1}{2m} \sum_{i=1}^m \left( h_{\Theta}(\mathbf{x}^{(i)}) - y^{(i)} \right)^2, \quad (2.101)$$

kterou můžeme upravit na tvar

$$\min_{\Theta} \frac{1}{2m} \sum_{i=1}^m \left( h_{\Theta}(\mathbf{x}^{(i)}) - y^{(i)} \right)^2 + 1000\vartheta_3^2 + 1000\vartheta_4^2, \quad (2.102)$$

kde hodnoty 1000 jsou pouze ilustrativní a reprezentují velká čísla (velké váhy). Tedy pokud bychom chtěli minimalizovat rovnici 2.102, tak to lze za předpokladu, že  $\vartheta_3$  a  $\vartheta_4$  budou malá čísla. V případě, že  $\vartheta_3 \approx 0$  a  $\vartheta_4 \approx 0$ , tak získáme opět náš minimalizační problém jako v rovnici 2.101. Ještě větší dopad to má na tvar hypotézy  $h_{\Theta}(\mathbf{x}) = \vartheta_0 + \vartheta_1 x + \vartheta_2 x^2 + \vartheta_3 x^3 + \vartheta_4 x^4$ , která dříve vedla k přetrénování. Ta v případě, že  $\vartheta_3 \approx 0$  a  $\vartheta_4 \approx 0$  přejde na tvar  $h_{\Theta}(\mathbf{x}) = \vartheta_0 + \vartheta_1 x + \vartheta_2 x^2$ . Tímto se zbavíme problému přetrénování a získáme tvar hypotézy, který dobře zobecňuje trénovací data.

### Regularizace

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

$$J(\Theta) = \frac{1}{2m} \left( \sum_{i=1}^m \left( h_{\Theta}(\mathbf{x}^{(i)}) - y^{(i)} \right)^2 + \lambda \sum_{j=1}^n \vartheta_j^2 \right), \quad (2.103)$$

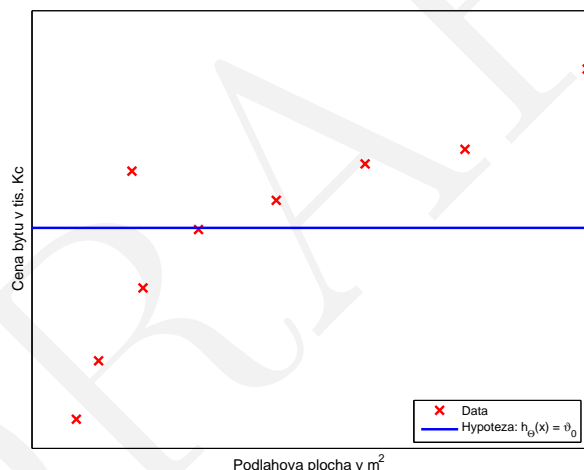
kde je nejdříve podotknout, že nová suma s indexem  $j$  má jinou mez  $n$  než první suma. To je způsobeno tím, že se regularizace vztahuje pouze na parametry  $\vartheta_1, \vartheta_2, \dots, \vartheta_m$ , tedy parametr  $\vartheta_0$  není regularizován, protože je vždy násoben jedničkou, nepříjde do kontaktu s příznaky vzorků. Dále parametr  $\lambda$  je takzvaný regularizační parametr, který kontroluje vyvážení mezi tím, jak hodně bude regularizace ovlivňovat původní ztrátovou funkci  $J(\Theta)$ . Regularizace je dána vztahem

$$\lambda \sum_{j=1}^n \vartheta_j^2. \quad (2.104)$$

Naším cílem je jako dříve minimalizovat ztrátovou funkci

$$\min_{\Theta} J(\Theta) \quad (2.105)$$

Co se stane, pokud bude regularizační parametr  $\lambda$  příliš velký? Řekněme, že  $\lambda = 10^{10}$ . Poté hypotéza ve tvaru  $h_{\Theta}(\mathbf{x}) = \vartheta_0 + \vartheta_1 x + \vartheta_2 x^2 + \vartheta_2 x^3 + \vartheta_2 x^4$  přejde na tvar  $h_{\Theta}(\mathbf{x}) = \vartheta_0$  a výsledný průběh lze vidět na Obr. 2.30



Obrázek 2.30: Použití regularizace k získání *underfit* hypotézy.

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

### 2.4.3 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

$$J(\Theta) = \frac{1}{2m} \left( \sum_{i=1}^m (h_{\Theta}(\mathbf{x}^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n \vartheta_j^2 \right), \quad (2.106)$$

snažíme se minimalizovat ztrátovou funkci

$$\min_{\Theta} J(\Theta) \quad (2.107)$$

Připomeňme si dříve představený gradientní algoritmus lineární regrese. Opakuj

$$\vartheta_j = \vartheta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\Theta}(\mathbf{x}^{(i)}) - y^{(i)}) x_j^{(i)}, \quad j = 0, 1, \dots, n. \quad (2.108)$$

Tento algoritmus je možno rozdělit do dvou kroků. Rozdělit výpočet pro  $\vartheta_0$  a  $\vartheta_{1,\dots,n}$ . Opakuj

$$\vartheta_0 = \vartheta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\Theta}(\mathbf{x}^{(i)}) - y^{(i)}) x_0^{(i)}, \quad (2.109)$$

$$\vartheta_j = \vartheta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\Theta}(\mathbf{x}^{(i)}) - y^{(i)}) x_j^{(i)}, \quad j = 1, 2, \dots, n. \quad (2.110)$$

Doposud na algoritmu nebylo nic změněno, pouze byl rozepsán do dvou kroků. Protože víme, že na parametr  $\vartheta_0$  se regularizace nevztahuje, proto můžeme algoritmus pozměnit a přidat regularizační člen. Následně bude mít gradientní algoritmus tvar. Opakuj

$$\vartheta_0 = \vartheta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\Theta}(\mathbf{x}^{(i)}) - y^{(i)}) x_0^{(i)}, \quad (2.111)$$

$$\vartheta_j = \vartheta_j - \alpha \left( \frac{1}{m} \sum_{i=1}^m (h_{\Theta}(\mathbf{x}^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \vartheta_j \right), \quad j = 1, 2, \dots, n, \quad (2.112)$$

kde platí

$$\frac{\partial}{\partial \vartheta_0} J(\Theta) = \left( \frac{1}{m} \sum_{i=1}^m (h_{\Theta}(\mathbf{x}^{(i)}) - y^{(i)}) x_0^{(i)} \right), \quad (2.113)$$

$$\frac{\partial}{\partial \vartheta_j} J(\Theta) = \left( \frac{1}{m} \sum_{i=1}^m (h_{\Theta}(\mathbf{x}^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \vartheta_j \right), \quad j = 1, 2, \dots, n. \quad (2.114)$$

Rovnice 2.112 lze přepsat do tvaru

$$\vartheta_j = \vartheta_j \left( 1 - \alpha \frac{\lambda}{m} \right) - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\Theta}(\mathbf{x}^{(i)}) - y^{(i)}) x_j^{(i)}, \quad (2.115)$$

kde platí

$$1 - \alpha \frac{\lambda}{m} < 1. \quad (2.116)$$

## Regularizace analytického řešení lineární regrese

Připomeňme si

$$\mathbf{X} = \left[ (\mathbf{x}^{(1)})^\top, \dots, (\mathbf{x}^{(m)})^\top \right]^\top, \quad \mathbf{X} \in \mathcal{R}^{m \times (n+1)} \quad (2.117)$$

a

$$\mathbf{y} = [y^{(1)}, \dots, y^{(m)}]^\top, \quad \mathbf{y} \in \mathcal{R}^m. \quad (2.118)$$

Analytické řešení pro výpočet  $\Theta$  má tvar

$$\Theta = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}. \quad (2.119)$$

Cílem je minimalizovat ztrátovou funkci

$$\min_{\Theta} J(\Theta). \quad (2.120)$$

Pokud využijeme regularizaci, tak naše analytické řešení přejde na tvar

$$\Theta = \left( \mathbf{X}^\top \mathbf{X} + \lambda \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix} \right)^{-1} \mathbf{X}^\top \mathbf{y} \quad (2.121)$$

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

$$\Theta = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}, \quad (2.122)$$

pak je matice  $\mathbf{X}^\top \mathbf{X}$  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  $\lambda > 0$

$$\Theta = \left( \mathbf{X}^\top \mathbf{X} + \lambda \underbrace{\begin{bmatrix} 0 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{bmatrix}}_{\text{Matice musí být invertovatelná}} \right)^{-1} \mathbf{X}^\top \mathbf{y} \quad (2.123)$$

### 2.4.4 Regularizace binární regrese

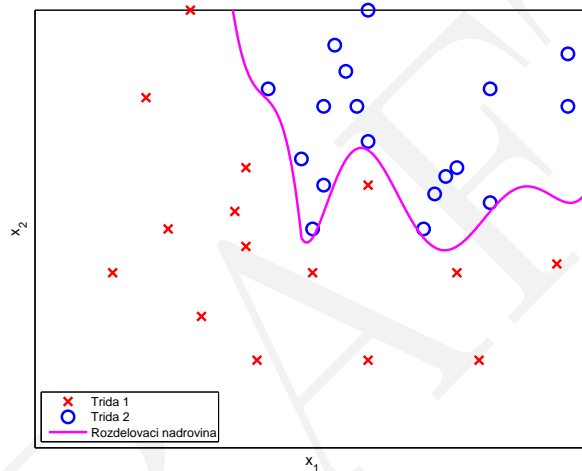
Pro binární regresi jsme dříve mluvili o dvou typech optimalizace. Mluvili jsme o tom, jak použít gradientní algoritmus a jak použít pokročilé metody optimalizace. V této sekci si ukážeme, jak tyto přístupy upravit, aby využily regularizace.

#### Regularizace binární regrese

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

$$h_{\Theta}(\mathbf{x}) = g\left(\vartheta_0 + \vartheta_1 x_1 + \vartheta_2 x_1^2 + \vartheta_3 x_1^2 x_2 + \vartheta_4 x_1^2 x_2^2 + \vartheta_5 x_1^2 x_2^3 + \dots\right), \quad (2.124)$$

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



Obrázek 2.31: Možný výsledek s použitím hypotézy z rovnice 2.124.

Ztrátová funkce pro binární regresi má tvar

$$J(\Theta) = - \left[ \frac{1}{m} \sum_{i=1}^m y^{(i)} \log(h_{\Theta}(\mathbf{x}^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\Theta}(\mathbf{x}^{(i)})) \right] \quad (2.125)$$

a její modifikací získáme regularizovaný tvar ztrátové funkce

$$J(\Theta) = - \left[ \frac{1}{m} \sum_{i=1}^m y^{(i)} \log(h_{\Theta}(\mathbf{x}^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\Theta}(\mathbf{x}^{(i)})) \right] + \frac{\lambda}{2m} \sum_{j=1}^n \vartheta_j^2 \quad (2.126)$$

přidaná suma je jen do  $n$ , jelikož se neregularizuje složka  $\vartheta_0$ , 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

$$\vartheta_0 = \vartheta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\Theta}(\mathbf{x}^{(i)}) - y^{(i)}) x_0^{(i)} \quad (2.127)$$

$$\vartheta_j = \vartheta_j - \alpha \left[ \frac{1}{m} \sum_{i=1}^m (h_{\Theta}(\mathbf{x}^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \vartheta_j \right], \quad j = 1, 2, 3, \dots, n, \quad (2.128)$$



kde

$$h_{\Theta}(\mathbf{x}) = \frac{1}{1 + e^{-\Theta^T \mathbf{x}}}. \quad (2.129)$$

Z rovnice 2.128 lze vidět, že platí

$$\frac{\partial}{\partial \vartheta_j} J(\Theta) = \frac{1}{m} \sum_{i=1}^m (h_{\Theta}(\mathbf{x}^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \vartheta_j. \quad (2.130)$$

### Regularizace pokročilé optimalizace

Nyní bude nastíněno jak využít pokročilé optimalizace v MATLABu. Nechť náš vektor parametrů  $\Theta$  odpovídá

$$\Theta = [\vartheta_0, \vartheta_1, \dots, \vartheta_n]^T = [\text{theta}(1), \text{theta}(2), \dots, \text{theta}(n+1)] \quad (2.131)$$

(z důvodu, že MATLAB indexuje od jedničky).

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

$$J(\Theta) = \left[ -\frac{1}{m} \sum_{i=1}^m y^{(i)} \log(h_{\Theta}(\mathbf{x}^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\Theta}(\mathbf{x}^{(i)})) \right] + \frac{\lambda}{2m} \sum_{j=1}^n \vartheta_j^2 \quad (2.132)$$

dále výpočet gradientu

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

$$\frac{\partial}{\partial \vartheta_0} J(\Theta) = \frac{1}{m} \sum_{i=1}^m (h_{\Theta}(\mathbf{x}^{(i)}) - y^{(i)}) x_0^{(i)}, \quad (2.133)$$

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

$$\frac{\partial}{\partial \vartheta_1} J(\Theta) = \left[ \frac{1}{m} \sum_{i=1}^m (h_{\Theta}(\mathbf{x}^{(i)}) - y^{(i)}) x_1^{(i)} \right] + \frac{\lambda}{m} \vartheta_1, \quad (2.134)$$

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

$$\frac{\partial}{\partial \vartheta_2} J(\Theta) = \left[ \frac{1}{m} \sum_{i=1}^m (h_{\Theta}(\mathbf{x}^{(i)}) - y^{(i)}) x_2^{(i)} \right] + \frac{\lambda}{m} \vartheta_2, \quad (2.135)$$

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

... (2.136)

## 2.5 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 nejbližšího souseda**

**Metoda K-nejbližších sousedů**

**Metoda minimální vzdálenosti**

**Bayesův klasifikátor**

**Perceptron**

## 2.6 Učení bez učitele

Učení bez učitele je ...

**Příklady algoritmů:** BIRCH, hierarchické algoritmy, divizní algoritmy, K-means (MacQueen algoritmus), expectation-maximization (EM) algoritmus, atd.

**K-means (MacQueenův algoritmus)**

**Aglomerativní algoritmus**

**Divizní algoritmus**

## 2.7 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:** Necht 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) *Přijít* / *Odmítnut*. V tabulce 2.4 jsou uvedeny, jak kvalitativní, tak kvantitativní trénovací vzorky.

ID	Jméno	Rozhodnutí	Kraj	Pohlaví	Dobrý v matematice	Dobrý ve sportu
1	Andrea	Přijít	Západočeský	Ž	Ano	Ne
2	Pepa	Přijít	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	Přijít	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	Přijít	Jihočeský	M	Ano	Ne

Tabulka 2.4: 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 sestavit z podmínky, pokud je uchazeč dobrý v matematice tak predikuj Přijít. 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

$$\varepsilon < 0.5, \quad (2.137)$$

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  $S = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)\}$ , kde  $y_i \in \{-1, 1\}$  je výstupní klasifikace a  $\mathbf{x}_i \in \mathbf{X}$  je vstupní příznakový vektor reálného objektu.

### Vysvětlení diskrétního AdaBoost algoritmu

AdaBoost je agresivní algoritmus, který vybere jeden slabý klasifikátor v každém kroku. Váhy  $\mathbf{d}^{(t)} = (d_1^{(t)}, \dots, d_N^{(t)})$  jsou přiřazeny k datům v kroku  $t$  a slabý klasifikátor  $h_t$  je založen na  $\mathbf{d}^{(t)}$ . Tyto váhy jsou aktualizovány každou iterací. 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

$$\varepsilon_t = \sum_{n=1}^N d_n^{(t)} \mathbf{I}(y_n \neq h_t(x_n)). \quad (2.141)$$

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

$$G^{AdaBoost}(\alpha) = \sum_{n=1}^N e^{-y_n(\alpha h_t(x_n) + f_{n-1}(x_n))}, \quad (2.142)$$

kde

$$f_{t-1}(x) = \sum_{r=1}^{t-1} \alpha_r h_r(x_n). \quad (2.143)$$

---

**Algorithm 1** Diskrétní AdaBoost algoritmus

---

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

$$\varepsilon_t = \sum_{n=1}^N d_n^{(t)} \mathbf{I}(y_n \neq h_t(x_n)). \quad (2.138)$$

- 6:     Nastav:

$$\alpha_t = \frac{1}{2} \log \frac{1 - \varepsilon_t}{\varepsilon_t}. \quad (2.139)$$

- 7:     Aktualizuj váhy:

$$d_n^{(t+1)} = \frac{d_n^{(t)} \cdot e^{-\alpha_t y_n h_t(x_n)}}{Z_t}, \quad (2.140)$$

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:  $f_T(x) = \sum_{t=1}^T \frac{\alpha_t}{\sum_{r=1}^T \alpha_r} h_t(x)$ .
- 

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

$$f_T(x) = \sum_{t=1}^T \frac{\alpha_t}{\sum_{r=1}^T \alpha_r} h_t(x). \quad (2.144)$$

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  $\mathbf{d} = (d_1, \dots, d_n)$  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

$$f_{BL}(x) = \text{sign}(D_+ - D_-), \quad \forall x. \quad (2.145)$$

Tedy predikuje  $+1$  pokud  $D_+ \geq D_-$  jinak  $-1$ . Je okamžitě zřejmé, že pro všechny váhy  $\mathbf{d}$  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  $\mathbf{d}$  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  $\gamma$  je parametr určující odchylku 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

$$\varepsilon_t(h_t, \mathbf{d}) = \sum_{n=1}^N d_n \mathbf{I}(y_n \neq h(x)_n) \leq \frac{1}{2} - \frac{1}{2}\gamma, \quad (\gamma > 0). \quad (2.146)$$

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

Příklad možného nalezení pozitivní  $\gamma$ , která respektuje podmínku empirické chyby 2.146. Předpokladem je mapování  $f$  z binární krychle  $\mathbf{X} = \{+1, -1\}^d$  do  $\mathbf{Y} = \{+1, -1\}$  a dále pozitivní labely  $y_n$ , které jsou dány  $y_n = f(x_n)$ . Funkce  $f$  je aproximována binární hypotézou  $h_t$  patřící do  $\mathbf{H}$ . Nechť  $H$  je třída binárních hypotéz a nechť  $D$  je rozložení nad  $\mathbf{X}$ . Korelace mezi  $f$  a  $H$  s respektováním  $D$  je dána  $C_{H,D}(f) = \sup_{h \in H} \mathbf{E}_D\{f(x)h(x)\}$ . Rozložení bez korelace mezi  $f$  a  $H$  je dáno  $C_H(f) = \inf_D C_{H,D}(f)$ . To může být dokázáno, pokud  $T > 2 \cdot \log(2) d C_H(f)^{-2}$ , pak  $f$  může být reprezentována přesně jako

$$f(x) = \text{sign}\left(\sum_{t=1}^T h_t(x)\right). \quad (2.147)$$

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 oddělitelnými třídami. Ve více obecném případě mohou 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.	



## 2.8 Kaskádní AdaBoost

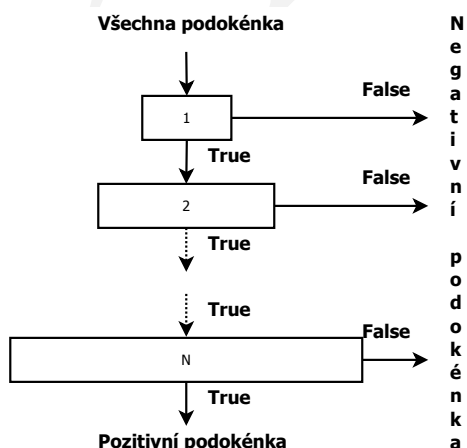
Je možné vybrat malé množství dobře odlišitelných příznaků a zkombinovat je do jednoho silného klasifikátoru, nicméně je potřeba představit několik hlavních myšlenek pro snížení výpočetního času. Rozšíření spočívá v tvorbě kaskády klasifikátorů, která navíc může dosahovat lepší klasifikační přesnosti. 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. 2.32 je tento proces naznačen.



Obrázek 2.32: 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

$$F = \prod_{i=1}^K f_i, \quad (2.148)$$

následně míra detekce může být vypočtena jako

$$D = \prod_{i=1}^K d_i, \quad (2.149)$$

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

Počet příznaků, které musí být 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

$$N = n_0 + \sum_{i=1}^K \left( n_i \prod_{j<i} p_j \right). \quad (2.150)$$

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  $\theta_i$  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 dosažena 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  $\theta_i$ , 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 2 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 předešlou vrstvu „složitější“. Tedy, čím dále v kaskádě se klasifikátor nachází, tím lépe dokáže klasifikovat mezi pozitivními a negativními vzorky (podokýnky obsahujícími lidské tváře).

**Algorithm 2** Učení kaskádního AdaBoost algoritmu

---

- 1: 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.
- 2: Inicializace:  $F_0 = D_0 = 1$ ,  $i = 0$  číslo aktuální vrstvy.
- 3: **while**  $F_i > F_{target}$  **do**
- 4:      $i = i + 1$
- 5:      $n_i = 0$
- 6:      $F_i = F_{i-1}$
- 7:     **while**  $F_i > f \cdot F_{i-1}$  **do**
- 8:          $n_i = n_i + 1$
- 9:         Natrénuj AdaBoost klasifikátor (algoritmus 1) s  $P$  a  $N$  jako trénovací množinou.
- 10:         Vypočti  $F_i$  a  $D_i$  pro aktuální klasifikátor na základě validační množiny.
- 11:         Snižuj práh klasifikátoru dokud není poměr  $i$ -tého klasifikátoru alespoň

$$d \cdot \frac{D_{i-1}}{D_i} \geq D_{i-1} \cdot d \quad (2.151)$$

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

## 2.9 SVM

Tato metoda hledá v prostoru nejširší možný pruh, který rozděluje množinu vektorů do dvou nadrovin. Nadrovina je následně popsána pomocí vektorů ležících nejbližší 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 případě 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

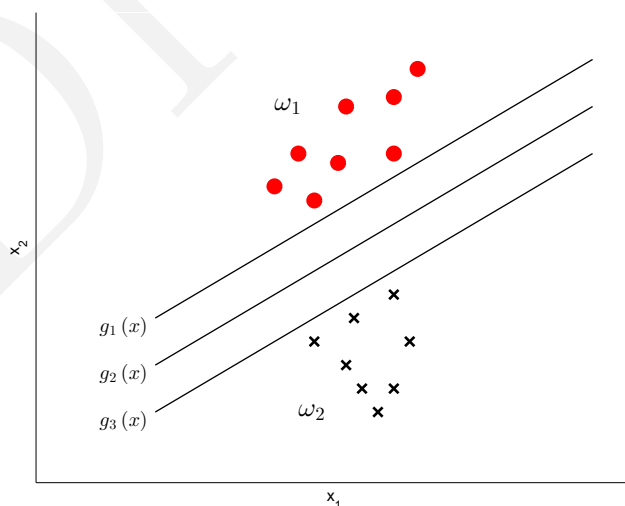
Nechť  $\mathbf{x}_i, i = 1, 2, \dots, N$ , jsou vektory tvořící trénovací množinu  $X$ . Vektory náleží do jedné ze dvou tříd  $\omega_1, \omega_2$  a jsou lineárně separovatelné. To znamená, že existuje parametr  $\omega$ , pro který platí

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

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

$$g(\mathbf{x}) = \omega^\top \mathbf{x} + \omega_0 = 0, \quad (2.152)$$

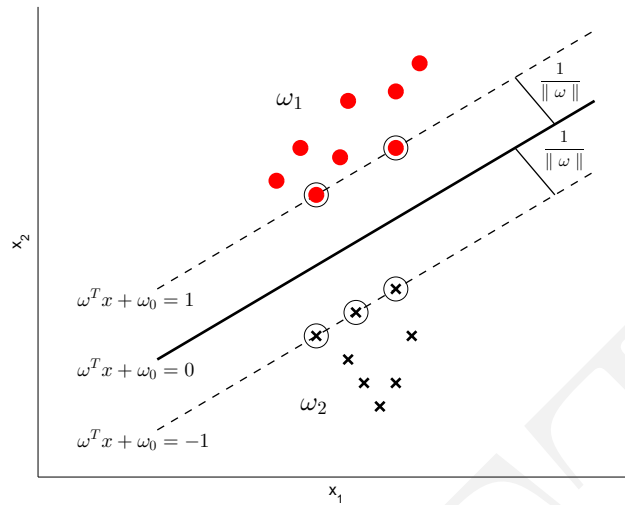
která rozděluje vektory  $\mathbf{x}_i$  do příslušných tříd. Koeficient  $\omega_0$  z předpisu 2.152 je nulový pouze v případě že nadrovina prochází počátkem souřadnicového systému. Jak je patrné z Obr. 2.33, rozdělujících nadrovin je obvykle celá řada. Úkolem klasifikační metody SVM je zvolit takovou, která minimalizuje chybu klasifikace tím, že má maximální možný odstup od obou tříd.



Obrázek 2.33: Příklad lineárně separovatelných tříd třemi možnými nadrovinami.

Obecně je nadrovina definována svým směrem (parametr  $\omega$ ) a polohou v prostoru (koeficient  $\omega_0$ ). 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  $\omega$ , pro který má nadrovina stejnou vzdálenost k nejbližším bodům z obou tříd  $\omega_1$  a  $\omega_2$  (viz Obr. 2.34).



Obrázek 2.34: Optimální nadrovina dělící vektory do dvou tříd; kroužkem jsou znázorněny support vectors.

Vzdálenost mezi bodem a nadrovinou lze vyjádřit pomocí vztahu

$$z = \frac{|g(\mathbf{x})|}{\|\omega\|}, \quad (2.153)$$

kde  $\|\omega\| = \sqrt{\omega^2}$ . Nejbližše položené body (na obr. znázorněny kroužkem) mají vzdálenost od nadroviny rovnu 1, pokud se nacházejí ve třídě  $\omega_1$  a  $-1$ , náleží-li do třídy  $\omega_2$ . To vede k závěru, že SVM hledá nadrovinu:

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

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

Dále je definován tzv. indikátor třídy  $y_i$  ( $+1$  pro  $\omega_1$  a  $-1$  pro  $\omega_2$ ) a cíl metody SVM může být shrnut následovně:

$$\min J(\omega, \omega_0) = \frac{1}{2} \|\omega\|^2, \quad (2.154)$$

za podmínky

$$y_i (\omega^\top \mathbf{x}_i + \omega_0) \geq 1, \quad i = 1, 2, \dots, N. \quad (2.155)$$

Vektory, pro které platí  $y_i (\omega^\top \mathbf{x}_i + \omega_0) = 1$ , jsou označovány jako *support vectors* (viz Obr. 2.34).

### 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  $\lambda$  značí Lagrangeův multiplikátor

$$\mathcal{L}(\boldsymbol{\omega}, \omega_0, \boldsymbol{\lambda}) = \frac{1}{2} \boldsymbol{\omega}^\top \boldsymbol{\omega} - \sum_{i=1}^N \lambda_i \left[ y_i (\boldsymbol{\omega}^\top \mathbf{x}_i + \omega_0) - 1 \right], \quad (2.156)$$

s podmínkami

$$\frac{\partial}{\partial \boldsymbol{\omega}} \mathcal{L}(\boldsymbol{\omega}, \omega_0, \boldsymbol{\lambda}) = 0, \quad (2.157)$$

$$\frac{\partial}{\partial \omega_0} \mathcal{L}(\boldsymbol{\omega}, \omega_0, \boldsymbol{\lambda}) = 0, \quad (2.158)$$

$$\lambda_i \geq 0, \quad i = 1, 2, \dots, N, \quad (2.159)$$

$$\left[ y_i (\boldsymbol{\omega}^\top \mathbf{x}_i + \omega_0) - 1 \right] = 0, \quad i = 1, 2, \dots, N. \quad (2.160)$$

Kombinací rovnice 2.156 s 2.158 a 2.159 je získáno řešení

$$\boldsymbol{\omega} = \sum_{i=1}^N \lambda_i y_i \mathbf{x}_i, \quad (2.161)$$

$$0 = \sum_{i=1}^N \lambda_i y_i. \quad (2.162)$$

Takto zkonstruovaná nadrovina se označuje za nadrovinu optimální. Optimalizační úloha definovaná předpisy 2.154 a 2.155 patří mezi úlohy konvexního programování a jako takovou ji lze řešit pomocí Lagrangeovy duality

$$\max \mathcal{L}(\boldsymbol{\omega}, \omega_0, \boldsymbol{\lambda}), \quad (2.163)$$

za podmínek

$$\boldsymbol{\omega} = \sum_{i=1}^N \lambda_i y_i \mathbf{x}_i, \quad \sum_{i=1}^N \lambda_i y_i = 0, \quad \lambda_i \geq 0, \quad i = 1, 2, \dots, N. \quad (2.164)$$

Dosazením rovnic 2.164 do 2.163 je získána optimalizační úloha ekvivalentní k úloze popsané předpisy 2.154 a 2.155

$$\max_{\boldsymbol{\lambda}} \left( \sum_{i=1}^N \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j \right), \quad (2.165)$$

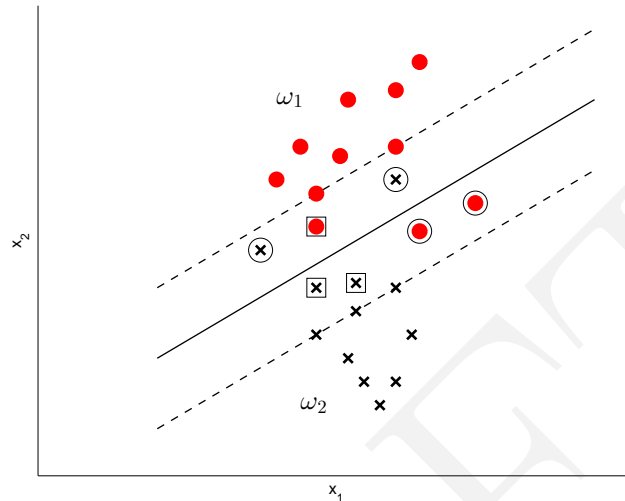
s podmínkou

$$\sum_{i=1}^N \lambda_i y_i = 0, \quad \lambda_i \geq 0, \quad i = 1, 2, \dots, N. \quad (2.166)$$

Maximalizováním výrazu 2.165 jsou vypočteny optimální Lagrangeovy multiplikátory a optimální nadrovina je získána jejich dosazením do rovnice 2.164. Koeficient  $\omega_0$  je vypočten z podmínky 2.160.

### Nelineárně separovatelná množina

Jestliže prvky trénovací množiny  $X$  nelze lineárně oddělit, nemůžeme být výše zmíněný postup konstrukce dělicí nadroviny použít. Příklad 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 2.35: Příklad lineárně neseparovatelné množiny.

Šířka dělicího pásu je definovaná vzdáleností dvou paralelních nadrovin s předpisem

$$\omega^\top \mathbf{x} + \omega_0 \pm 1 \quad (2.167)$$

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

- vektory ležící mimo dělicí pás jsou považovány za správně klasifikované a splňují tak podmínku 2.155.
- vektory správně klasifikované, které leží uvnitř pásu, splňují nerovnost (na Obr. 2.35 znázorněny čtvercem)

$$0 \leq y_i (\omega^\top \mathbf{x} + \omega_0) < 1.$$

- vektory chybně klasifikované, které leží uvnitř pásu, splňují nerovnost (na Obr. 2.35 znázorněny kroužkem)

$$y_i (\omega^\top \mathbf{x} + \omega_0) < 0.$$

Všechny tři kategorie mohou být popsány podmínkou

$$y_i (\omega^\top \mathbf{x} + \omega_0) \geq 1 - \xi_i, \quad (2.168)$$

kde  $\xi_i$  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

$$J(\omega, \omega_0, \xi) = \frac{1}{2} \|\omega\|^2 + C \sum_{i=1}^N I(\xi_i), \quad (2.169)$$



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

$$I(\xi_i) = \begin{cases} 1, & \xi_i > 0, \\ 0, & \xi_i = 0. \end{cases} \quad (2.170)$$

### Matematické řešení

Optimalizace funkce 2.169 je matematicky velice složitá, jelikož obsahuje nespojitou funkci  $I(\bullet)$ . Z tohoto důvodu se SVM zaměřuje na optimalizaci tzv. blízke funkce a jejím cílem se tak stává

$$\min J(\boldsymbol{\omega}, \omega_0, \boldsymbol{\xi}) = \frac{1}{2} \|\boldsymbol{\omega}\|^2 + C \sum_{i=1}^N \xi_i, \quad (2.171)$$

za podmínek

$$y_i (\boldsymbol{\omega}^\top \mathbf{x}_i + \omega_0) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad i = 1, 2, \dots, N. \quad (2.172)$$

Řešení opět vede na Lagrangeovu funkci

$$\mathcal{L}(\boldsymbol{\omega}, \omega_0, \boldsymbol{\lambda}, \boldsymbol{\xi}, \boldsymbol{\mu}) = \frac{1}{2} \|\boldsymbol{\omega}\|^2 + C \sum_{i=1}^N \xi_i - \sum_{i=1}^N \mu_i \xi_i - \sum_{i=1}^N \lambda_i [y_i (\boldsymbol{\omega}^\top \mathbf{x}_i + \omega_0) - 1 + \xi_i], \quad (2.173)$$

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

$$\boldsymbol{\omega} = \sum_{i=1}^N \lambda_i y_i \mathbf{x}_i, \quad \sum_{i=1}^N \lambda_i y_i = 0, \quad C - \mu_i - \lambda_i = 0 \quad (2.174)$$

společně s podmínkami

$$\lambda_i [y_i (\boldsymbol{\omega}^\top \mathbf{x}_i + \omega_0) - 1 + \xi_i] = 0, \quad \mu_i \xi_i = 0, \quad \mu_i \geq 0, \quad \lambda_i \geq 0, \quad i = 1, 2, \dots, N. \quad (2.175)$$

Použitím Lagrangeovy duality je úkolem SVM

$$\max \mathcal{L}(\boldsymbol{\omega}, \omega_0, \boldsymbol{\lambda}, \boldsymbol{\xi}, \boldsymbol{\mu}), \quad (2.176)$$

za podmínek

$$\boldsymbol{\omega} = \sum_{i=1}^N \lambda_i y_i \mathbf{x}_i, \quad \sum_{i=1}^N \lambda_i y_i = 0, \quad C - \mu_i - \lambda_i = 0, \quad \mu_i \geq 0, \quad \lambda_i \geq 0, \quad i = 1, 2, \dots, N. \quad (2.177)$$

Po dosazení výrazů 2.177 do 2.176 je získána ekvivalentní optimalizační úloha

$$\max_{\boldsymbol{\lambda}} \left( \sum_{i=1}^N \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j \right), \quad (2.178)$$

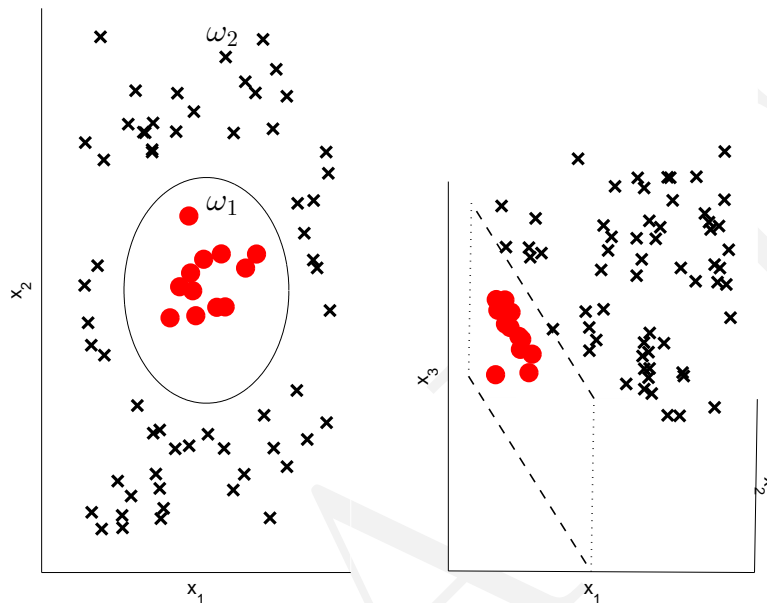
s podmínkou

$$\sum_{i=1}^N \lambda_i y_i = 0, \quad 0 \leq \lambda_i \leq C, \quad i = 1, 2, \dots, N, \quad (2.179)$$

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  $\xi_i > 0$ .

## 2.10 SVM - Kernel trick

Doposud bylo cílem metody SVM lineárně rozdělit trénovací množinu  $X$  do dvou tříd  $\omega_1$  a  $\omega_2$ . 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. 2.36).



Obrázek 2.36: Příklad namapování nelineárně separovatelné množiny z prostoru  $\mathcal{R}^2$  do prostoru  $\mathcal{R}^3$ , kde lze množinu lineárně rozdělit.

V rovnici 2.165 je jádro (ang. *kernel*) popsáno vztahem  $\mathbf{x}_i^\top \mathbf{x}_j$  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:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^\top \mathbf{x}_j. \quad (2.180)$$

Po dosazení předpisu jádra zpět do rovnice 2.165 se cílem metody SVM stává:

$$\max_{\lambda} \left( \sum_{i=1}^N \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \right), \quad (2.181)$$

s podmínkou

$$\sum_{i=1}^N \lambda_i y_i = 0, \quad \lambda_i \geq 0, \quad i = 1, 2, \dots, N. \quad (2.182)$$

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  $\omega_1$  a  $\omega_2$ . 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 2.5.

Typ jádra	Předpis	Poznámka
Polynomiální jádro	$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \mathbf{x}_j + \theta)^d$	Parametr $d$ a práh $\theta$ je volen uživatelem.
Sigmoidové jádro	$K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\eta \mathbf{x}_i \mathbf{x}_j + \theta)$	Parametr $\eta$ a práh $\theta$ je volen uživatelem.
Gaussovo jádro <i>Radial Basis Function</i> (RBF)	$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{1}{2\sigma^2} \ \mathbf{x}_i - \mathbf{x}_j\ ^2\right)$	Parametr $\sigma$ je volen uživatelem.

Tabulka 2.5: 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.

## 2.11 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.

# Kapitola 3

## Přednáška 08

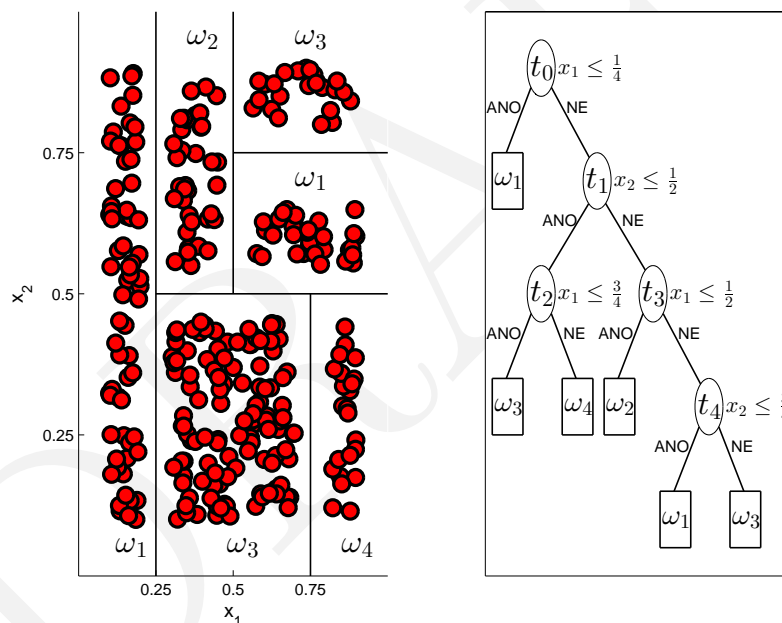
### 3.1 Bayesovská klasifikace

Bayes classifier - pridame uz aj pravdepodobnost = 30 min, bylo v USK

## 3.2 Rozhodovací strom

Rozhodovací strom je příkladem nelineární klasifikační metody. Představuje vícepatrový rozhodovací systém, kterým příznakové vektory postupně procházejí až do bodu, kdy jsou zařazeny do odpovídající třídy.

Sestavení rozhodovacího stromu se provádí postupným dělením prostoru trénovací množiny  $X$  na navzájem se neprolínající regiony, přičemž každý region odpovídá právě jedné klasifikační třídě. Rozhodovací strom  $T$  se skládá z uzlů, uvnitř kterých dochází k porovnávání jednotlivých prvků příznakového vektoru  $\mathbf{x} \in X$ . Na základě těchto porovnání vektor postupně propadne až do jednoho z listů stromu, které odpovídají jednotlivým klasifikačním třídám. Obecně nejpoužívanější forma rozhodovacího stromu je ta, která rozděluje prostor vstupní množiny na kvádry, jejichž hrany jsou rovnoběžné se souřadnicovými osami tohoto prostoru. Takový strom se nazývá stromem binárním a prvky příznakového vektoru jsou porovnávány na základě splnění nerovnosti  $x_i \leq \alpha$ , kde  $x_i$  je  $i$ -tý prvek příznakového vektoru  $\mathbf{x}$  a  $\alpha$  značí zvolený práh. Nerovnost  $x_i \leq \alpha$  je označována jako rozhodovací pravidlo.



Obrázek 3.1: Ukázka rozdělení prostoru trénovací množiny (vpravo) pomocí rozhodovacího stromu (vlevo).

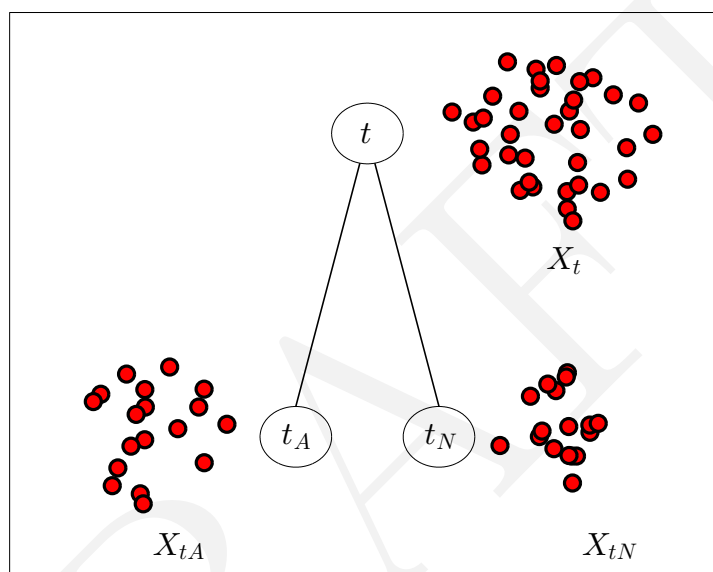
Příklad zobrazený na Obr. 3.1 ukazuje princip konstrukce rozhodovacího stromu. Je zřejmé, že výsledný tvar stromu je ovlivněn volbou rozhodovacího pravidla a pořadím, ve kterém jsou prvky příznakových vektorů porovnávány. Konstrukce každého stromu musí splňovat následující podmínky:

- Každý uzel  $t$  odpovídá podmnožině  $X_t \subset X$ , která vznikla na základě rozhodovacího pravidla předešlého uzlu (výjimku tvoří první uzel, tzv. kořen stromu, pro který  $X_t = X$ ). Rozhodovacím pravidlem uzlu  $t$  je podmnožina  $X_t$  dále rozdělena na  $X_{tA}$  a  $X_{tN}$ .  $X_{tA}$  v sobě slučuje vektory, které splnily rozhodovací pravidlo uzlu  $t$  a  $X_{tN}$  naopak slučuje ty, které pravidlo nesplnily (viz Obr. 3.2). Každé takové rozdělení

musí splňovat:

$$\begin{aligned} X_{tA} \cap X_{tN} &= \emptyset, \\ X_{tA} \cup X_{tN} &= X_t. \end{aligned}$$

- Ze všech možných rozdělení  $X_t$  je zvoleno to, které nejlépe splňuje rozdělovací kritérium.
- Je nutné zvolit zastavující podmínku, která reguluje narůstání stromu. Uzel, který tuto podmínku splňuje je označena za list stromu.
- Každému listu musí být přiřazena odpovídající klasifikační třída.



Obrázek 3.2: Příklad dělení uzlu  $t$  a jemu odpovídající množina vektorů  $X_t$

Pojmy rozdělovací kritérium, zastavující podmínka a princip přiřazení klasifikačních tříd listům stromu jsou vysvětleny v následujících podkapitolách.

## Volba rozhodovacího pravidla

Jak bylo výše uvedeno, v případě binárního stromu má rozhodovací pravidlo tvar  $x_i \leq \alpha_i$  ( $\alpha_i$  je práh rozdělovací vektory na základě porovnání jejich  $i$ -tého prvku). Úkolem je tedy zvolit jak hodnotu prahu  $\alpha_i$ , tak i index prvku  $i$ . Prah však může nabývat nekonečně mnoha hodnot, jelikož  $\alpha_i \in \mathbb{R}$  a otestovat všechny z nich je tak nemyslitelné. Díky existenci trénovací množiny  $X$  lze ovšem vymezit konečnou množinu hodnot, které je pro získání  $\alpha_i$  nutné otestovat. Pokud je porovnáván  $i$ -tý prvek,  $x_i$ , vektoru  $\mathbf{x}$ , jsou veškeré  $i$ -té prvky všech příznakových vektorů uvnitř trénovací množiny vzestupně seřazeny a tato řada pak tvoří množinu variant, které je nutné následně podrobit testování. Obdobně se postupuje pro všechny prvky  $x_i, i = 1, 2, \dots, l$  všech vektorů  $\mathbf{x} \in X$ . Postupně tak vznikne několik možností rozdělení podmnožiny  $X_t$  a rozdělovacím kritériem je vybráno to nejlepší z nich. Tímto způsobem je uzlu  $t$  určeno, který prvek vektoru  $\mathbf{x}$  se bude uvnitř něj porovnávat s nalezeným prahem (dojde ke konkretizaci rozhodovacího pravidla  $x_i \leq \alpha_i$ ).

## Rozdělovací kritérium

Z důvodu, aby bylo možné kvalitativně porovnávat vzniklá rozdělení  $X_t$  z předchozího kroku, je definována tzv. míra rozmanitosti uzlu.

Nechť  $P(\omega_i | t)$  značí pravděpodobnost jevu, že vektory uvnitř podmnožiny  $X_t$ , náležící uzlu  $t$ , patří do třídy  $\omega_i$ ,  $i = 1, 2, \dots, M$ . Míra rozmanitosti uzlu je definována jako:

$$I(t) = - \sum_{i=1}^M P(\omega_i | t) \log_2 P(\omega_i | t). \quad (3.1)$$

Je vhodné poznamenat, že předpis 3.1 představuje entropii z teorie informace spjatou s uzlem  $t$ . Ta nabývá svého maxima pokud se všechny pravděpodobnosti  $P(\omega_i | t) = \frac{1}{M}$ , naopak je rovna nule, pokud všechny vektory uvnitř množiny spadají do jedné třídy, což znamená, že právě jedna pravděpodobnost  $P(\omega_i | t) = 1$  a ostatní jsou nulové. V praxi jsou pravděpodobnosti  $P(\omega_i | t)$  odhadovány pomocí vztahu  $\frac{N_t^i}{N_t}$ , kde  $N_t^i$  je počet vektorů v  $X_t$  patřících do třídy  $\omega_i$  a  $N_t$  je celkový počet vektorů uvnitř podmnožiny  $X_t$ .

Dalším rozdělením  $X_t$  na dvě podmnožiny  $X_{tA}$  a  $X_{tN}$ , kde  $X_{tA}$  má  $N_{tA}$  počet vektorů a  $X_{tN}$  má  $N_{tN}$  počet vektorů, dojde k nárůstu rozmanitosti uzlu definovaného jako

$$\Delta I(t) = I(t) - \frac{N_{tA}}{N_t} I(t_A) - \frac{N_{tN}}{N_t} I(t_N), \quad (3.2)$$

kde  $I(t_A)$ ,  $I(t_N)$  jsou míry rozmanitosti uzlů  $t_A$  a  $t_N$ . Cílem rozdělovacího kritéria tedy je zvolit takové rozdělení uzlu  $t$  (resp. podmnožiny  $X_t$ ), pro které dojde k nejvyššímu nárůstu rozmanitosti uzlu,  $\Delta I(t)$ .

## Zastavující podmínka

Dojde-li k naplnění zastavující podmínky pro některý z uzlů rozhodovacího stromu, je takový uzel prohlášen za list stromu a jeho dělení je ukončeno. Zastavující podmínka může být definována prahem  $Tr$ , který určuje minimální potřebný nárůst rozmanitosti uzlu,  $\Delta I(t)$ . Pokud je po rozdělení uzlu  $t$  maximální hodnota  $\Delta I(t)$  přes všechna možná rozdělení uzlu menší než práh  $Tr$ , je uzel  $t$  prohlášen za list. Další možností volby zastavující podmínky je stanovení minimálního počtu vektorů uvnitř podmnožin vzniklých dělením uzlu  $t$ .

## Princip přiřazení třídy listům

Jakmile je uzel rozhodovacího stromu prohlášen za list, musí mu být přiřazena jedna z klasifikačních tříd. Listu je přiřazena třída  $\omega_j$ , kde index  $j$  odpovídá

$$j = \arg \min_i P(\omega_i | t). \quad (3.3)$$

Jinými slovy, listu  $t$  je přiřazena taková třída, jenž je uvnitř  $X_t$  nejčastěji zastoupena.

Vedle binárních rozhodovacích stromů lze v odborné literatuře dohledat i stromy, jejichž rozhodovací pravidlo má tvar  $\sum_{i=1}^l c_i x_i \leq \alpha$ . Jedná se tedy o nalezení přímek, nikoli

pouze prahů, dělících jednotlivé prvky příznakových vektorů. Tímto rozhodovacím pravidlem lze sice v některých případech vhodněji rozdělit prostor trénovací množiny  $X$ , ovšem konstrukce stromu je mnohem složitější.

Nevýhodou klasifikační metody založené na rozhodovacím stromu zůstává princip sestavování samotného stromu. Pokud dojde k malé změně trénovací množiny  $X$ , dojde zároveň ke změně tvaru rozhodovacího stromu  $T$ , což může vést k nárůstu chyby klasifikace. Tento nedostatek lze odstranit zapojením více než jednoho rozhodovacího stromu do procesu klasifikace. Taková klasifikační metoda je označována za rozhodovací les.

### 3.3 Rozhodovací les

Hlavní předností rozhodovacího lesa je to, že účinně potlačuje citlivost výsledného tvaru rozhodovacího stromu  $T$  na změny v trénovací množině  $X$ . Toho je docíleno tak, že množina  $X$  obsahující  $N$  prvků je algoritmem *bootstrap aggregating* rozdělena na  $n$  množin,  $X_{(n)}$ , obsahujících  $N_{(n)}$  prvků, přičemž  $N_{(n)} \leq N$ . A pro každou množinu  $X_{(n)}$  je následně zkonstruován rozhodovací strom  $T_{(n)}$  postupem, který je popsán v kapitole 3.2.

Klasifikace neznámého vektoru  $\mathbf{y}$  probíhá tak, že je tento vektor přiveden na vstupy všech rozhodovacích stromů  $T_{(n)}$ . Každý rozhodovací strom poté určí příslušnost vektoru  $\mathbf{y}$  do některé třídy  $\omega_i$ ,  $i = 1, 2, \dots, M$ . Finální index třídy  $i$  je vybrán na základě nejčastějšího výsledku mezi všemi provedenými klasifikacemi (tj. výstupy stromů  $T_{(n)}$ ) neznámého vektoru  $\mathbf{y}$ .

Dalším faktorem, jenž ovlivňuje výsledný tvar rozhodovacího stromu, je pořadí prvků příznakových vektorů, které je během nalézání rozhodovacích pravidel pro jednotlivé uzly uvažováno. Doposud byly v tomto textu představeny rozhodovací stromy využívající ke své konstrukci prvky vektorů popořadě. Následující kapitola popisuje modifikaci rozhodovacích lesů, jež stanoví toto pořadí náhodně.

### 3.4 Náhodný les

Náhodný les je modifikací metody rozhodovacího lesa, která vznikla za účelem ještě většího zpřesnění výsledku klasifikace. Jsou-li během konstrukce stromů porovnávány prvky příznakových vektorů popořadě, zvyšuje se tím riziko toho, že velká část takto vzniklých stromů bude vzájemně korelovaná. Ovšem vysoká korelace stromů, jak bylo prokázáno, uvnitř rozhodovacího lesa zvyšuje chybu klasifikace. Z tohoto důvodu používá náhodný les zcela nový princip konstrukce jednotlivých rozhodovacích stromů.

Prvním krokem této metody zůstává rozdělení trénovací množiny  $X$  na  $n$  množin pomocí algoritmu *bootstrap aggregating*. Rozdíl nastává v bodě konstrukce jednotlivých stromů. Během té je nejprve uživatelem zvolen parametr  $m$  ( $m \ll l$ , kde  $l$  je dimenze vektoru  $\mathbf{x} \in X$ ), který ovlivňuje volbu rozhodovacího pravidla pro jednotlivé uzly konstruovaných stromů. Jako rozhodovací pravidlo je totiž zvoleno to, které vykáže nejlepší výsledek mezi  $m$  náhodně vybranými prvky příznakového vektoru. Tímto způsobem vznikají všechny stromy tvoří finální náhodný les. Klasifikace neznámého vektoru  $\mathbf{y}$  pak probíhá stejně, jako tomu bylo v případě klasického rozhodovacího lesa.



### 3.5 Neuronové sítě

Dříve popsaná klasifikační metoda SVM usiluje o nalezení takové nadroviny  $g(x)$ , která rozděluje trénovací množinu  $X$  do dvou tříd  $\omega_1$ ,  $\omega_2$ . V případě lineárně separovatelné množiny nalezená nadrovina  $g(x)$  klasifikovala trénovací vektory,  $\mathbf{x} \in X$ , bez chyby. Pokud ovšem trénovací množina lineárně separovatelná nebyla, cílem metody SVM bylo zkonstruovat nadrovinu  $g(x)$  minimalizující chybu klasifikace. Klasifikátory pracující na principu neuronových sítí byly primárně navrženy právě k řešení lineárně neseparovatelných problémů. Řešení je nalezeno pomocí transformace těchto problémů do prostoru, ve kterém již existuje adekvátní lineární řešení.

Základním stavebním kamenem neuronové sítě je neuron, na jehož vstup je přiveden vektor  $\mathbf{x}$  a výstupem je pak 1 (je-li  $\mathbf{x} \in \omega_1$ ) či 0 (je-li  $\mathbf{x} \in \omega_2$ ). Neuron s takovýmto binárním výstupem je označován jako perceptron. Neuronová síť, skládající se výhradně z perceptronů, se nazývá perceptronová síť a pokud v ní dochází k řazení perceptronů za sebe, je taková síť označována jako vícevrstvá perceptronová síť (MLP - *Multilayer perceptron*). Princip fungování MLP bude popsán v následující části textu na příkladu dvouvrstvé perceptronové sítě.

#### Dvouvrstvá perceptronová síť

Úkolem této neuronové sítě je klasifikovat  $l$ -dimenzionální vektor  $\mathbf{x} \in \mathcal{R}^l$  pomocí  $p$  perceptronů (viz Obr. 3.3a). Tyto perceptrony se nacházejí v tzv. skryté vrstvě a jejich úkolem je namapovat  $l$ -dimenzionální vstupní vektor do  $p$ -dimenzionálního prostoru  $H_p$ , který si lze představit jako vrcholy prostorového mnohoúhelníku definovaného:

$$H_p = \{[y_1, \dots, y_p]^T \in \mathcal{R}^p, y_i \in \{0, 1\}, 1 \leq i \leq p\}.$$

Každý perceptron ve skryté vrstvě sítě vlastně představuje jednu klasifikační nadrovinu a převádí lineárně neseparovatelnou trénovací množinu  $X$  do prostoru, v němž je ji možné bez chyby rozdělit do dvou tříd (viz Obr. 3.3b).

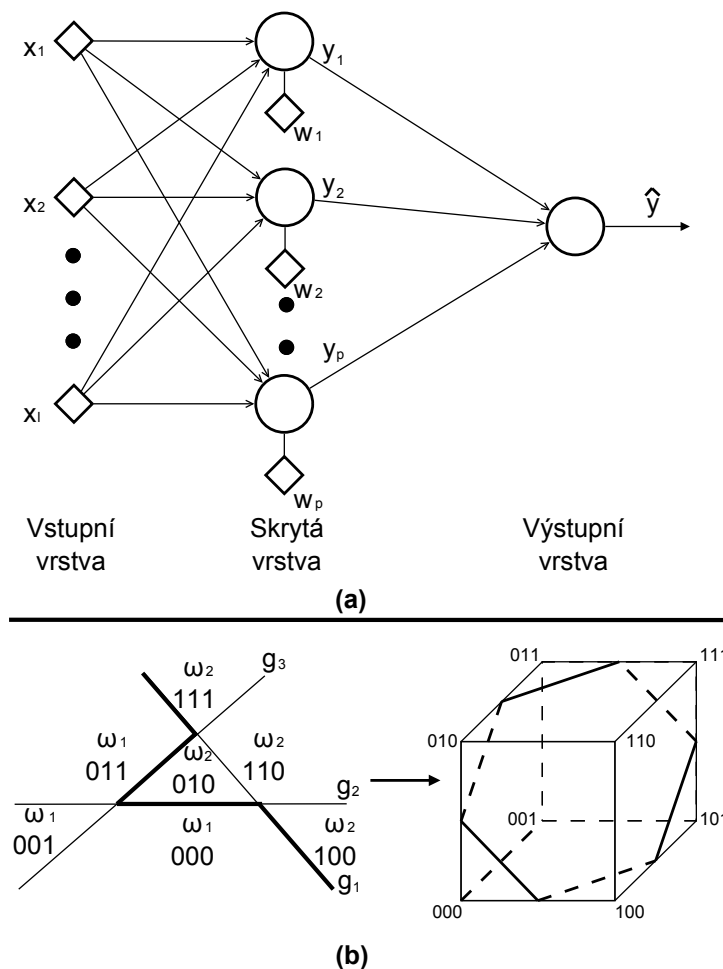
Vícevrstvé perceptronové sítě se používají v případě komplikovanějších klasifikačních úloh. Jejich konstrukce obsahuje více skrytých vrstev a pracují na principu rozložení složitějšího problému na menší mnohem jednodušší problémy. Kombinací řešení těchto dílčích problémů je následně dosaženo hledaného výsledku.

#### Aktivační funkce

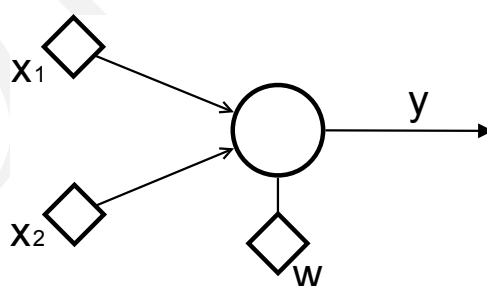
Uvažujme jednoduchý perceptron s dvěma vstupy  $x_1, x_2$  a jedním výstupem  $y$  (viz Obr. 3.4). Výstup takového perceptronu odpovídá předpisu:

$$y = f(a), \text{ kde } a = \sum_{i=1}^2 (wx_i).$$

Funkce  $f(a)$  je označována jako aktivační funkce neuronu a  $w$  je jeho synaptická váha. Mezi nejčastěji používané aktivační funkce patří:



Obrázek 3.3: (a) Dvouvrstvá perceptronová síť, (b) Příklad transformace nelineárně separovatelného problému do prostoru, ve kterém existuje lineární řešení; neuronová síť je tvořena třemi perceptrony ve skryté vrstvě (perceptrony představují klasifikační nadrovinu  $g_1, g_2, g_3$ ).



Obrázek 3.4: Perceptron.

- Skoková funkce

$$f(a) = \begin{cases} 1 & \text{pro } a > 0 \\ 0 & \text{pro } a < 0 \end{cases} \quad (3.4)$$

- Sigmoida

$$f(a) = \frac{1}{1 + \exp(-\lambda a)} \quad (3.5)$$

kde  $\lambda$  je parametr sklonu.

- Hyperbolický tangens

$$f(a) = \frac{1 + \exp(-\lambda a)}{1 - \exp(-\lambda a)} \quad (3.6)$$

Proces trénování neuronové sítě spočívá v nastavení synaptických vah jednotlivých neuronů. Jelikož je k dispozici trénovací množina jedná se opět o tzv. učení s učitelem, které je realizováno např. algoritmem zpětného šíření (*Backpropagation Algorithm*).

## Algoritmus zpětného šíření

Algoritmus vychází z trénovací množiny, která obsahuje  $N$  dvojic  $(y(i), \mathbf{x}(i))$ ,  $i = 1, 2, \dots, N$ , kde  $\mathbf{x}(i)$  představuje  $i$ -tý vektor trénovací množiny a  $y(i)$  určuje jeho příslušnost do jedné ze tříd. Během trénování jsou v jednotlivých iteracích získávány výstupy neuronové sítě  $\hat{y}(i)$ , které se liší od požadovaného  $y(i)$ . Cílem učení je nastavit synaptické váhy jednotlivých vrstev sítě tak, aby rozdíl mezi aktuálním výstupem  $\hat{y}(i)$  a požadovaným výstupem  $y(i)$  byl minimální.

Nové synaptické váhy jsou vypočítávány jako

$$w_j^r(\text{nova}) = w_j^r(\text{stara}) + \Delta w_j^r, \quad (3.7)$$

kde  $w_j^r$  je váha  $j$ -tého neuronu v  $r$ -té vrstvě sítě a  $\Delta w_j^r$  značí přírůstek, který je definován

$$\Delta w_j^r = -\mu \frac{\partial J}{\partial w_j^r}, \quad (3.8)$$

kde  $\mu$  je konstanta učení (parametr algoritmu). Cílem algoritmu zpětného šíření je minimalizace nákladové funkce  $J$  s předpisem:

$$J = \sum_{i=1}^N \mathcal{E}(i), \quad (3.9)$$

kde lze za  $\mathcal{E}(i)$  zvolit rozdíl čtverců

$$\mathcal{E}(i) = \frac{1}{2} \sum_{i=1}^N (y(i) - \hat{y}(i))^2. \quad (3.10)$$

Přírůstek  $\Delta w_j^r$  ze vzorce 3.7 je stanoven vyřešením rovnice

$$\frac{\partial \mathcal{E}(i)}{\partial w_j^r} = 0. \quad (3.11)$$

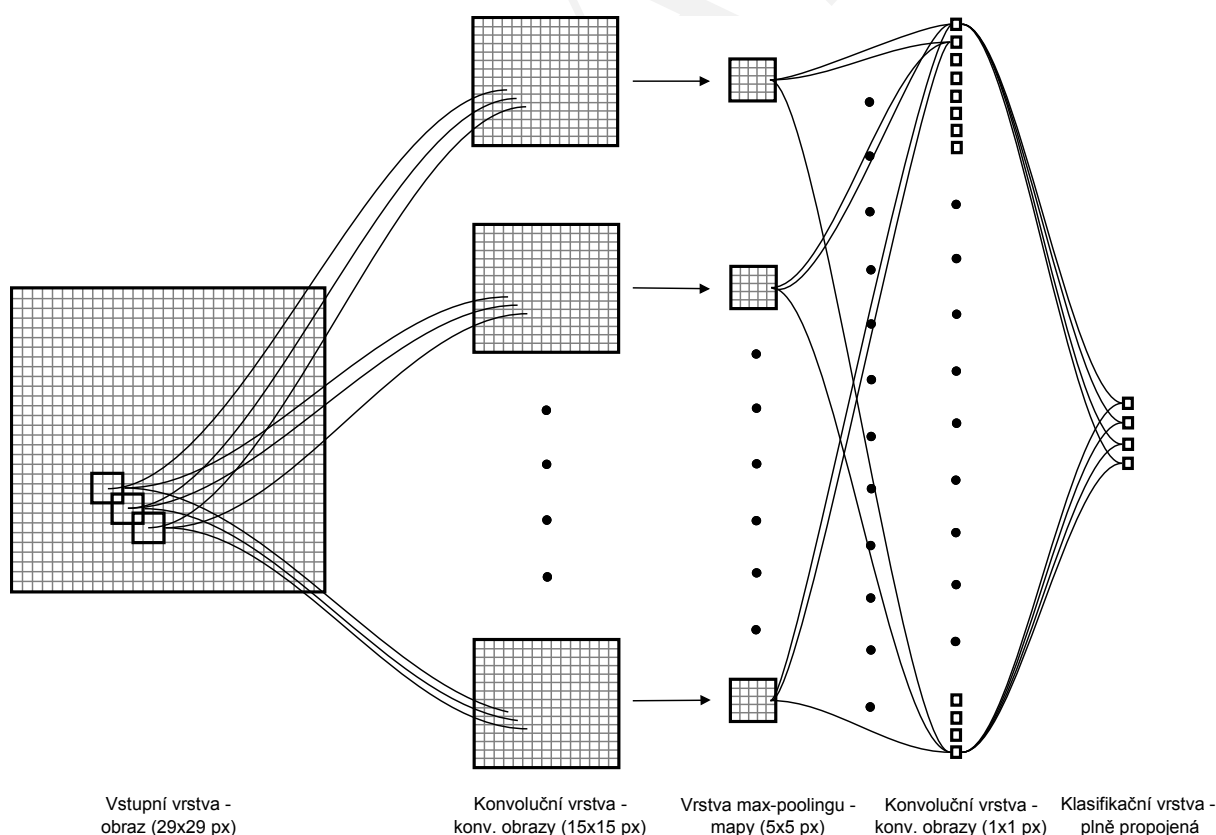
Algoritmus učení neuronové sítě probíhá v několika krocích. Nejprve jsou náhodně nastaveny synaptické váhy  $w_j^r$  pro jednotlivé neurony (obvykle se doporučuje váhy volit v intervalu  $\langle -1, 1 \rangle$ ) a je zvolena hodnota konstanty učení  $\mu > 0$  a maximální chyby  $\mathcal{E}_{max} > 0$ . Tento krok se nazývá inicializace. Poté je spočtena odezva vstupního vektoru  $\hat{y}(i)$  a dojde k aktualizaci vah podle vzorce 3.7. Obdobně se postupuje pro všechny prvky  $x(i) \in X$  z trénovací množiny. Jestliže je na konci trénovacího cyklu výsledná chyba  $\sum_{i=1}^N \mathcal{E}(i) \leq \mathcal{E}_{max}$  algoritmus skončí, v opačném případě dojde k nové inicializaci plynoucí z řešení rovnice 3.11 a algoritmus se zopakuje.

Algoritmus zpětného šíření představuje jednoduchý způsob učení neuronové sítě, který ovšem velmi kvalitně vystihuje princip takového učení. V odborné literatuře se lze setkat i se složitějšími algoritmy učení, mezi které patří algoritmy založené na určení směru gradientu funkce 3.9 či algoritmy využívající kvazi-Newtonovu metodu.

### 3.6 Konvoluční neuronové sítě

Za účelem klasifikace obrazových dat jsou momentálně v praxi nejhojněji využívány tzv. konvoluční neuronové sítě (CNN - *Convolutional Neural Networks*), které mají podobnou strukturu jako běžné neuronové sítě, ale výstupy jednotlivých vrstev nejsou získávány pomocí aktivační funkce neuronů, nýbrž pomocí konvoluce vstupu a příslušného konvolučního jádra.

CNN se od klasické neuronové sítě liší již svým vstupem. Vstupem totiž není příznakový vektor obrazu, ale obraz samotný (viz Obr. 3.5). Nad vstupním obrazem je provedena konvoluce, jejíž výsledek představuje výstup konvoluční vrstvy. Ta může být následována vrstvou max-poolingu, nebo další konvoluční vrstvou. Konvoluce probíhá pouze nad definovaným prostorem vstupu, tzn. že konvoluční jádro nesmí zasahovat mimo okraje vstupního obrazu. Zřetěžením těchto operací je postupně redukována velikost vstupního obrazu až do bodu, kdy je výstupem pouze jednodimenzionální příznakový vektor. Ten je v nejvyšší vrstvě sítě (v tzv. klasifikační vrstvě) zařazen do jedné klasifikační třídy.



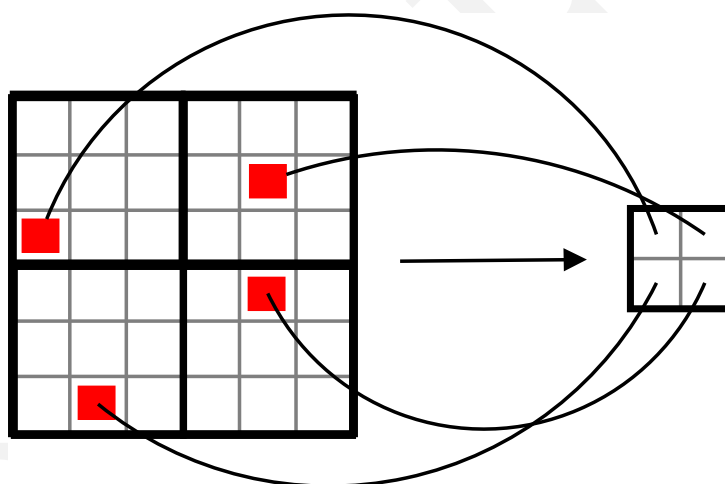
Obrázek 3.5: Architektura konvoluční neuronové sítě; velikost konvolučních jader je  $3 \times 3$ , hodnota posunu je nastavena na 1 a klasifikace probíhá do čtyř tříd.

## Konvoluční vrstva

Konvoluční vrstva je určena počtem a velikostí konvolučních jader a hodnotou posunu. Počet konvolučních jader  $M$  určuje počet výstupů vrstvy, které posléze vstupují do vrstvy následující. Hodnota posunu,  $S_x$  a  $S_y$ , definuje o kolik pixelů se jádro během konvoluce po vstupním obraze posouvá ve směru osy  $x$  a  $y$ . Velikost konvolučního jádra  $(K_x, K_y)$  spolu s hodnotou posunu určují velikost výstupních konvolučních obrazů  $(M_x, M_y)$ .

## Vrstva max-poolingu

Max-pooling představuje proces podvzorkování vstupního konvolučního obrazu. Tento proces je do CNN zařazen z důvodu, že urychluje konvergenci sítě a vede k nalezení robustnějších obrazových příznaků. Max-pooling pracuje na principu detekce maxim v nepřekrývajících se regionech vstupního obrazu. Tyto regiony jsou definovány velikostí konvolučního jádra  $(K_x, K_y)$ . Výstupem vrstvy max-poolingu je tedy mapa obsahující nalezená maxima a rozměr mapy odpovídá podvzorkování vstupního obrazu o faktor  $K_x$ , resp.  $K_y$ , podél obou os (viz Obr. 3.6).



Obrázek 3.6: Ukázka max-poolingu; maxima dílčích regionů vstupního obrazu jsou vyznačena červeně a regiony mají rozměr  $3 \times 3$ .

## Klasifikační vrstva

Tato vrstva představuje nejvyšší vrstvu CNN a obsahuje přesně tolik neuronů, kolik je klasifikačních tříd. Bývá také označována jako plně propojená vrstva, jelikož na vstup každého jejího neuronu jsou přivedeny všechny výstupní konvoluční obrazy vrstvy předešlé. Jak již bylo poznamenáno, výstupy předcházející vrstvy představují jednodimenzionální příznakový vektor vstupního obrazu, u kterého je následně rozhodnuto, do jaké náleží třídy.

Učení CNN probíhá obdobným způsobem jako učení klasické neuronové sítě. Jen s tím rozdílem, že nedochází k aktualizaci vah, ale jsou upravovány formy jednotlivých konvolučních jader.

## 3.7 Zdroje

- [1] Breiman L., „Bagging Predictors“, *Machine Learning*, Vol. 24, pp. 123-140, 1996.
- [2] Breiman L., „Random Forests“, *Machine Learning*, Vol. 45, pp. 5-32, 2001.
- [3] Quinlan J.R., *C4.5: Programs for Machine Learning*, Morgan Kaufmann, 1993.
- [4] Theodoridis S., Koutroubas K., *Pattern Recognition*, Elsevier Inc., 2008.