ZK 2022

neděle 15. ledna 2023 11:39

1. (6b) Bayesovská klasifikace.

Předpokládejme, že máme dvě třídy hub, jedlé (E) a toxické (T). Automatický systém navržený tak, aby pomohl s rozpoznáním těchto tříd, poskytuje výstup, který je 0 (velmi nízké riziko otravy), 1 (střední riziko otravy) nebo 2 (vysoké riziko otravy).

Apriorní pravděpodobnosti pro uvedené dvě třídy jsou

$$p(E) = 0.99, p(T) = 0.01.$$

Podmíněné pravděpodobnosti jsou

$$\begin{array}{lll} p(0|E) = 0.89, & p(1|E) = 0.10, & p(2|E) = 0.01 \\ p(0|T) = 0.01, & p(1|T) = 0.20, & p(2|T) = 0.79. \end{array}$$

sdružená $P(E,0)$

	0	1	2	Σ
E	0.8811	0.099	0.0099	0.99
T	0.0001	0.002	0.0079	0.01
Σ		0.101		1

apriorní, marginální

(a) (2b) Vypočítejte podmíněnou pravděpodobnost $p(T|1)$, že houba je toxická za předpokladu, že systém hlásí střední riziko.

Bayesův vzorec $P(x|k) = \frac{P(k|x) \cdot P(x)}{P(k)}$

$$\rightarrow P(T|1) = \frac{P(1|T) \cdot P(T)}{P(1)} = \frac{P(1 \cap T)}{P(1)} = \frac{0.002}{0.101} = 0.0198 = P(T|1)$$

$$P(1|T) \cdot P(T) = P(T|1) \cdot P(1)$$

$$0.2 \times 0.01 = \frac{1}{500} = 2 \cdot 10^{-3} = 0.002$$

(b) (2b) Cílem je učinit binární rozhodnutí Y (*yes*, houba je akceptována) nebo N (*no*, houba se vyhodí). Penalizační funkce je:

$$\begin{array}{ll} W(E,Y) = 0, & W(E,N) = 1, \\ W(T,Y) = 1000, & W(T,N) = 0. \end{array}$$

	Y	N
E	0	1
T	1000	0

Určete bayesovskou optimální strategii, tedy optimální rozhodnutí pro všechny výstupy systému (0, 1, 2).

Když je houba... 0 → velmi ? Y/N ← $q^*(0)$

1

2

	Y	N
E	0	1
T	1000	0

	0	1	2	Σ
E	0.8811	0.099	0.0099	0.99
T	0.0001	0.002	0.0079	0.01
Σ		0.101		1

$q^*(0)$: ~~$0 \cdot P(E,Y) + 1000 \cdot P(E,N)$~~ (dá minimalizovat nižší beru to hned jako rozhodnutí)

$0 \cdot 0.8811 + 1000 \cdot 0.0001 = 0.1$

$1 \cdot 0.8811 + 0 \cdot 0 \cdot = 0.1811$ $\rightarrow q^*(0) = YES$

$q^*(1)$: $d=y$: ~~$0 \cdot 0.099$~~ $+ 1000 \cdot 0.002 = 2$

$d=n$: $1 \cdot 0.099 + 0 \cdot$ ~~0~~ $= 0.099$ $\rightarrow q^*(1) = NO$

$q^*(2)$: $d=y$: $1000 \cdot 0.0079 = 7.9$

$d=n$: $1 \cdot 0.0079 = 0.0079$ $\rightarrow q^*(2) = NO$

(c) (2b) Popište krátce formálně (pomocí vzorců) jak se optimální bayesovská strategie spočítá.

minimalizace Bayesovského rizika $R = \sum_{x \in X} \sum_{k \in K} p(x,k) \cdot W(k, q(x))$ (třída, rozhodnutí; sdružená pst, ztrátová funkce)

---

2. (6b) Odhad parametrů.

Kostkou se hází opakovaně tolikrát, dokud se na horní straně neobjeví „šestka“. Pravděpodobnost, že se to stane při n-tém hodu, je

$$p(n) = (1-q)^{n-1} q, \qquad (2)$$

kde q je pravděpodonost, že na kostce padne „šestka“.

(a) (1b) Zdůvodněte správnost vzorce pro $p(n)$.

vychází z geometrického rozdělení $X \sim Geom(p)$ (pst úspěchu)

↳ „počet pokusů do prvního úspěchu“

$p_k = p \cdot (1-p)^k$

nebo spíš binomické $p_k = \binom{n}{k} p^k (1-p)^{n-k}$

$P(n) = (1-q)^{n-1} \cdot q^1$

?

$(1-q)^{n-1} \cdot q$

pst že nepadne $n-1$ krát

pst že padne

(b) (3b) Jaký je maximálně vědohodný odhad pro q pokud šestka padla v k-tém hodu?

maximum likelihood

k-tý hod → $P(k) = (1-q)^{k-1} \cdot q$

ML: $\hat{q} = \text{argmax } L(q)$

$L(q) = P(T|q) = (1-q)^{k-1} \cdot (1-q)^{k-1} \cdots q \cdot q \cdot q \cdots q = q^k \cdot (1-q)^{n-k}$

$$L(q) = \underbrace{(1-q)^{k-1} \cdot (1-q)^{k-1} \cdots}_{P(\mathcal{T}|q)} \underbrace{q \cdot q \cdot q \cdots q}_{q^k} = q^k \cdot (1-q)^{k^2-k}$$

$$(1-q)^{(k-1)+(k-1)+\cdots} \quad k \cdot (k-1) = k^2 - k$$

$$\log(L(q)) = l(q) = \log\left(q^k (1-q)^{k^2-k}\right) = \log(q^k) + \log\left((1-q)^{k^2-k}\right)$$

$$= k \cdot \log(q) + (k^2-k) \cdot \log(1-q)$$

$$\frac{\partial l(q)}{\partial q} = \frac{k}{q} + \frac{k^2-k}{1-q} \cdot (-1) = \frac{k}{q} - \frac{k^2-k}{1-q} = 0$$

$$\frac{k(1-q) - q(k^2-k)}{q(1-q)} = 0$$

$$k - kq - k^2 q + kq = 0$$

$$k - k^2 q = 0$$

$$q = \frac{k}{k^2} = \frac{1}{k}$$

(c) (2b) Předpokládejme, že kostkou hodíme maximálně n_{max}-krát. Jaký je maximálně věrohodný odhad pro q, když šestka ani v posledním hodu nepadne?

?

ani v n_{max} nepadne 6

$$p = (1-q)^{n_{max}}$$

$$L = (1-q)^k$$

$$\log(L) = l = k \cdot \log(1-q)$$

$$\frac{\partial l}{\partial q} = \frac{k}{1-q} \cdot (-1) = 0$$

$$\frac{-k}{1-q} = 0$$

---

3. (4b) Neuronové sítě.

(a) (1b) Vysvětlete, co je vanishing gradient a jak se tento problém řeší.

- **Problém mizejícího gradientu - vypropagovaný gradient** je příliš **malý**, takže **nejsvrchnější vrstvy** neuronové sítě **zůstanou beze změny**
- Může k němu dojít například **při použití aktivační funkce**, která **mapuje velká čísla na čísla z relativně malého intervalu** - např. **sigmoida**
- Když derivuju skoro **konstatní funkce** - derivace je pak 0
- Způsobí to, že **velká změna vstupu zapříčiní** pouze **malou změnu výstupu** a tím pádem deerivaci funkce budou malé hodnoty (kladné hodnoty nižší než 1)
  - **Při chain rule** se budou **násobit malé hodnoty** a vzniknou tak **ještě daleko menší hodnoty**
  - Po aplikaci gradientu na váhy přechodu sítě se jejich **hodnota téměř nezmění**
- dochází k **zaseknutí učení a trénování sítě**
- Lze řešit:
  - **vhodnou inicializací vah**
  - Použití **jiné aktivační funkce,** která nemapuje velká čísla do intervalu o malém rozptylu - např. Parametric **ReLU**

Backpropagation ale s sebou nese řadu problémů. Asi nejočividnější je vysoká výpočetní náročnost a s tím související rychlost učení [Hay09, Nie115]. To je ale problém umělých neuronových sítí obecně. Dalším problémem je mizející gradient (vanishing gradient problem). Toto je problém vícevrstvých sítí. Jedná se o situaci, kdy vypropagovaný gradient je příliš malý, a tudíž nejsvrchnější vrstvy zůstanou beze změny [BeSF94]. Tomuto problému lze částečně předejít vhodnou inicializací vah [Beng09, LBOM98]. Deep learningové metody tento problém řeší předtrénováním o kterém je řeč v kapitole [4] o hlubokých neuronových sítích. Posledním

Problémy při trénování neuronových sítí

Problém mizejícího gradientu

Tento problém vzniká například při použití aktivační funkce, která mapuje velká čísla na čísla z relativně malého intervalu. Příkladem může být tangens hyperbolický nebo sigmoida. Toto mapování způsobuje, že velká změna vstupu zapříčiní pouze malou změnu výstupu, takže derivace funkce jsou malé hodnoty. Při počítání gradientů pomocí řetězového pravidla

dochází k násobení jednotlivých parciálních derivací od poslední vrstvy směrem k počátční. Jelikož jsou tyto parciální derivace nízká čísla (kladné hodnoty nižší než 1), jejich násobením získáme gradient, který je ještě daleko nižší hodnota, než samotné parciální derivace z těchto vrstev. Po aplikaci gradientu na váhy přechodu sítě se jejich hodnota téměř nezmění, takže nedochází k učení a trénování sítě je zaseknuto na stejném místě.

Problému mizejícího gradientu se dá předejít několika způsoby. Prvním z nich je použití aktivační funkce, která nemapuje velká čísla do intervalu o malém rozptylu, a jejíž derivace pak nejsou tak nízké hodnoty. Příkladem může být již zmíněná funkce *Parametric ReLU*. Dalším řešením může být použití vrstvy normalizace dávek před aktivační vrstvou, nebo použití reziduálních bloků s reziduálními spojeními, které jsou napojeny přímo na vstup do reziduálního bloku a neprochází aktivací. Tato spojení pak přivádí hodnotu vstupu do součtu s výstupem celého bloku, a tím zvyšují celkovou hodnotu derivace celého bloku. Princip reziduálních spojení bude více popsán v dalších kapitolách. Použití normalizace dávek před aktivační vrstvou zajistí, že vstup do aktivační vrstvy už je normalizován do potřebného intervalu, a tím pádem nedochází ke snížení hodnot derivací.

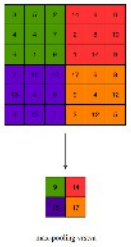
(b) (1b) Co je max-pooling? Proč se používá?

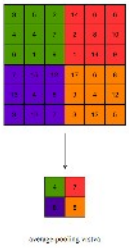
- **Sdružovací** vrstva (**pooling** layer)
- Může **provádět podvzorkování** --> umožňuje **méně náročný výpočet v dalších vrstvách**
- Konktrétně max-pooling vybere z dat v okolí prvek s největší hodnotou a ten pak propaguje jako jediný dál
- Může snižovat počet dimenzí

Sdružovací vrstva

Důležitou vrstvou je *sdružovací* vrstva (angl. *pooling layer*). Princip této vrstvy je velmi jednoduchý. Postupně prochází data a z okolí o předem dané velikosti vybere nebo vypočítá 1 hodnotu. Pokud se po datech pohybuje s posunem (angl. *stride*) většího než 1, dochází ke snížení počtu dimenzí výstupních dat oproti datům vstupním. Tato vrstva tedy může provádět podvzorkování (angl. *downsampling*), což umožňuje méně náročný výpočet v dalších vrstvách. Další výhodou této vrstvy je, že zabezpečuje invariaci proti malým posunům vstupních dat.

Sdružovací vrstva má několik verzí, které se používají v praxi, pro příklad uvedu nejjednodušší, kterými jsou *max-pooling* a *average-pooling*. První jmenovaná vybere z okolí prvek s největší hodnotou a ten propaguje jako jediný dál. Druhá v pořadí spočítá průměr všech hodnot prvků v daném okolí a tuto hodnotu posune na svůj výstup.





(c) (1b) Co je softmax?

- Jedna z **aktivačních výstupních** funkcí, která se často dává **nakonec NN**
- Aby **normalizovala výstup na pravděpodobnostní rozdělení** (převod na pravděpodobnostní distribuci)
  - Mapuje vstupní hodnoty na výstupní pravděpodobnostní rozdělení
  - Vstupem jsou reálné hodnoty výstupem je K - dimenzionální vektor pravděpodobostí (0-1)
  - Výsledek se musí sčítat do 1
- Počítá se zvlášť pro každou třídu
- Pravděpodobnost s jakou **bude náležet výstupní vektor** do dané třídy s daným vnitřním potenciálem - výstup je ta **třída** s největší pravděpodobností

$$p(k|X) = \frac{e^{w_k X}}{\sum_{i=1}^{K} e^{w_i X}}$$

(d) (1b) Co je metoda stochastic gradient descent?

- Jeden z **algoritmů gradientního sestupu - iterativní optimalizační** metoda
- Na rozdíl od základní verze algoritmu - která prochází celý dataset
- počítá průměrné hodnoty chybové funkce a hodnoty parametrů můžeme počítat pro každý jednotlivý vzorek v datasetu
- Je **rychlejší,** protože n**emusí procházet všechny vzorky dat** - zjednodušuje výpočet
- Přidává malou dávku šumu do učícího procesu

---

4. (4b) Vysvětlete, co je K-means++. Co přináší použití K-means++ ve srovnání se standardním K-means?

- **Chytřejší inicializace** centroidů clusterů
- Cílem je udělat k-means, které bude mít **větší šanci skončit v lepším lokálním minimu** (minimum s nižším J (nižší ztrátová funkce)
- Inicialiuje centry následujícím algoritmem:
  1) **Random vyberu jeden** centr c1
  2) Spočítám **vzdálenosti od bodů do clusteru c1** (obecně pro každý bod spočítám vzdálenost do nejbližšího clustru - k nejbližšímu centru)
  3) Jako **další cluster** vyberu bod **s pravděpodobností úměrné druhé mocnině této vzdálenosti**
  4) Opakuju **dokud nevyberu K centrů** clustrů
  5) Pokračuju stejně jako běžný K-means algoritmus