

Kapitola 1

Metoda nejbližších sousedů

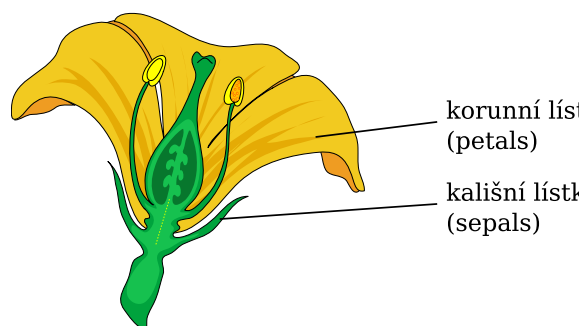
TM: doporučování filmů by byl realistický příklad na použití vzdálenosti

JL: Co do každé kapitoly přidat jednu zajímavou story – tady by to mohl být collaborative filtering, u prceptronu ty experimenty v padesátkách

V této kapitole si představíme jednu z nejjednodušších metod strojového učení, která je velice intuitivní a jmenuje se návodně *metoda nejbližších sousedů*. Pro začátek se podíváme na úlohu, který je poměrně jednoduchá a dobře poslouží k představení základních pojmů. V závěrečném programovacím cvičení se podíváme na zajímavější, ale o něco složitější úlohu.

1.1 Klasifikační úlohy

V této kapitole bude naší úlohou určování tří druhů kosatců podle šířky a délky jejich kališních a korunních lístků. Schéma květu najdete na obrázku 1.1. Pochopitelně to není tak, že by si někdo řekl, že potřebuje určovat druhy kosatců a bude to dělat tak, že pravítkem změří rozměry květu, ty zadá do počítače a dozví se, jaký druh má před sebou. To by bylo značně absurdní. Ve skutečnosti jen využíváme známou databázi měření, která pochází z roku 1936 a od druhé poloviny 20. století se často používá jako příklad v informatické literatuře pro ilustraci různých metod. Využívání dat, která původně vznikla pro jiné účely, ale obsahují zajímavé informace (např. účetní záznamy), je ve strojovém učení zcela běžné.



Obrázek 1.1: Schéma květu dvouděložných rostlin.

Úlohy tohoto typu, kdy se na základě nějakých vlastností objektů rozhodujeme, do jaké skupiny objekt patří, nazýváme obecně *klasifikace* (EN: *classification*). Klasifikaci obvykle popisujeme jako přiřazování *značky* (EN: *labels*), které označují jednotlivé skupiny – třídy (EN: *classes*).

Jiným příkladem klasifikace může být filtrování spamu v emailech. Zde se klasifikuje do dvou tříd – spam a běžný email. Složitějším příkladem klasifikace je identifikace obličejů na fotkách. Značky, které v tomto případě fotkám přiřazujeme jsou jména osob, které chceme na fotkách rozpoznat.

Metoda nejbližších sousedů se dá několika slovy popsat takto: k neklasifikovanému příkladu najdeme nejpodobnější – nejbližší známý příklad a neklasifikovaný příklad zařadíme do stejné třídy. Nejen, že se to snadno řekne, ale jak uvidíme později překvapivě snadno také naprogramuje.

K tomu, abychom se naučili kosatce rozpoznávat, máme k dispozici 100 příkladů, u nichž známe délku a šířku korunních lístků, délku a šířku kališních lístků, k jakému druhu která rostlina patří. Těmito příklady, ze kterých se naučíme rostliny rozpoznávat říkáme *trénovací data* (EN: *training data*). Jejich ukázkou najdete v tabulce 1.1.

koruna		kalich		druh
délka	šířka	délka	šířka	
6.4	3.2	4.5	1.5	Iris versicolor
5.3	3.7	1.5	0.2	Iris setosa
6.1	2.6	5.6	1.4	Iris virginica
7.7	3.0	6.1	2.3	Iris virginica
5.0	3.3	1.4	0.2	Iris setosa
7.0	3.2	4.7	1.4	Iris versicolor
...				

Tabulka 1.1: Ukázka trénovacích dat. Rozměry jsou uvedeny v centimetrech.

Vidíme tedy, že každý exemplář rostliny je popsán čtyřmi reálnými čísly a druhem, ke kterému patří. Číslo, které popisuje vlastnost instance, budeme říkat *rys* (EN: *feature*). Někdy se používá také pojem *příznak*, *vstupní veličina*, nebo slangově počestšelé „fíčura“. Jednotlivým exemplářům říkáme *instance* (EN: *instances*).

V případě klasifikace emailů na spamy a „nespamy“ se jako rysy používají informace o přítomnosti určitých slov nebo o tom, jak často uživatel na tuto adresu sám odesílá emaily. Instancemi jsou jednotlivé emaily.

Pro zjednodušení zápisu budeme s rysy zacházet jako s *vektory* reálných čísel. O vektorech většinou mluvíme v souvislosti s geometrií, ale pro nás bude zatím vektor jenom uspořádaná n -tice čísel. V případě úlohy s kosatci vypadá trénovací příklad vypadá takto:

$$(7.0, 3.2, 4.7, 1.4) \rightarrow \text{Iris versicolor}$$

1.2 Algoritmus hledání nejbližších sousedů

Algoritmus, který použijeme v případě, že potřebujeme klasifikovat novou, neznámou instanci je velice jednoduchý. Projdeme všechna trénovací data a pro každou trénovací instanci změříme její vzdálenost od té klasifikované. Vybereme tu, která má nejmenší vzdálenost a použijeme její třídu.

Algoritmus v Pythonu naprogramujeme například takto:

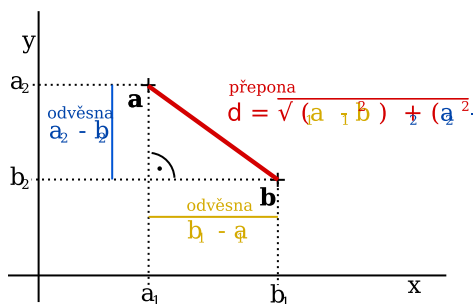
```
1 def nearest_neighbor(new_instance, training_data):
2     # Zdefinujeme funkci, která počítá vzdálenost nové instance
3     # od příkladu z trénovacích dat.
4     def distance_from_instance(training_example):
5         # Z trénovacího příkladu nás zajímá pouze vektor rysu.
6         feature_vector, _ = training_example
7         # Vratíme vzdálenost vektoru rysu trénovacího vektoru a nové instance.
8         return distance(feature_vector, new_instance)
9
10    # Definovanou funkci použijeme jako klíč pro hledání minima
11    # v trénovacích datech.
12    nearest_vector, nearest_label = \
13        min(training_data, key=distance_from_instance)
14    # Vratíme třídu nejbližšího souseda.
15    return nearest_label
```

Abychom mohli algoritmus spustit, chybí nám ještě implementovat funkci `distance`. Měřit vzdálenost mezi instancemi můžeme různým způsobem. My si nyní ukážeme ten nejběžnější způsob – *Eukleidovskou vzdálenost*.

Když pomocí vektorů označujeme body v ploše nebo prostoru, můžeme snadno měřit jejich vzdálenost pomocí Pythagorovy věty. Pro dva dvourozměrné vektory $\mathbf{a} = (a_1, a_2)$ a $\mathbf{b} = (b_1, b_2)$, můžeme spočítat jejich vzdálenost

$$d(\mathbf{a}, \mathbf{b}) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2}. \quad (1.1)$$

Geometrické odvození je celkem jednoduché a můžeme ho schématicky vidět na obrázku 1.2.



Obrázek 1.2: Odvození Eukleidovské vzdálenosti pomocí Pythagorovy věty. Rozdíly souřadnic určují velikosti přepon trojúhelníku. Odvěsna potom určuje jejich vzdálenost.

Pro třírozměrné vektory $\mathbf{c} = (c_1, c_2, c_3)$ a $\mathbf{d} = (d_1, d_2, d_3)$, máme Eukleidovskou vzdálenost

$$d(\mathbf{c}, \mathbf{d}) = \sqrt{(c_1 - d_1)^2 + (c_2 - d_2)^2 + (c_3 - d_3)^2}. \quad (1.2)$$

Znázornit její odvození jako v případě dvou rozměrů by bylo o něco složitější, ale stále možné.

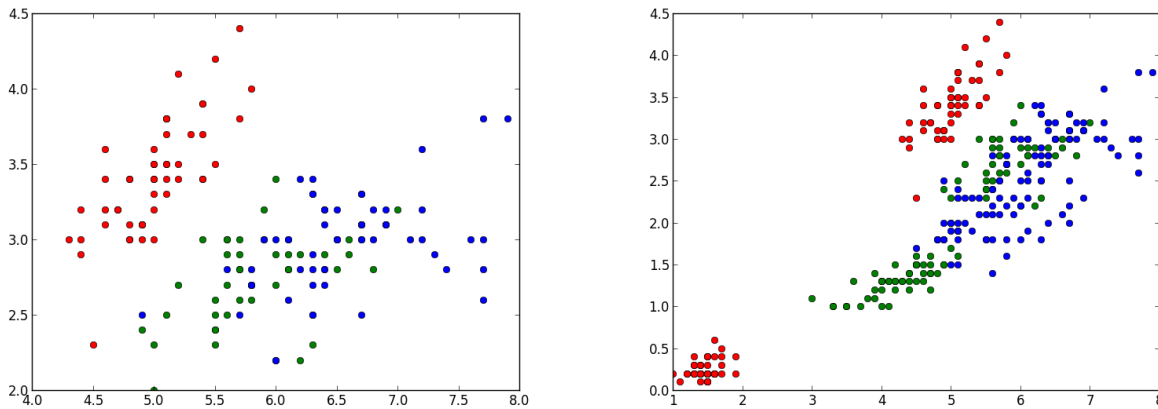
Eukleidovská vzdálenost se dá zobecnit pro vektory, které mají n rozměrů, ovšem bez rozumné možnosti postup vizualizovat. Pro vektory $\mathbf{e} = (e_1, \dots, e_n)$ a $\mathbf{f} = (f_1, \dots, f_n)$. Eukleidovskou vzdálenost mezi nimi spočítáme takto:

$$d(\mathbf{e}, \mathbf{f}) = \sqrt{(e_1 - f_1)^2 + \dots + (e_n - f_n)^2} = \sqrt{\sum_{i=1}^n (e_i - f_i)^2}. \quad (1.3)$$

V Pythonu zapíšeme Eukleidovskou vzdálenost velice snadno a podobně matematickému zápisu:

```
1 def distance(vec_a, vec_b):
2     # pocitame odmocninu - np.sqrt(...)
3     # ze souctu - sum(...)
4     # operator "-" vektory po slozkach odecte,
5     # operator "**" vektory po slozkach umocni
6     return np.sqrt(sum((vec_a - vec_b) ** 2))
```

Ve cvičení si ukážeme ještě další způsob, jak se počítat vzdálenost mezi instancemi.



Obrázek 1.3: Dvourozměrná vizualizace dat – délka $\times$ šířka korunních lístků (vlevo) a délka $\times$ šířka kališních lístků (vpravo), jednotlivé třídy jsou barevně označeny.

Na obrázku 1.3 vidíme jednoduchou vizualizaci našich trénovacích dat. Bohužel do grafu nelze zakreslit čtyřrozměrné vektory a tak jsme zakreslili zvlášť rozměry kališních a korunních lístků. Pokud bychom se rozhodli používat pouze dva rysy, mohli bychom skutečně hledat nejbližší sousedy v grafu tak, že bychom pravítkem měřili, která instance je nejpodobnější novému exempláři.

Zároveň si v grafu můžeme všimnout toho, že často ten nejbližší soused nemusí patřit do třídy, kterou považujeme intuitivně za nejsprávnější. Vezměme například na bod $[7, 1; 3, 2]$ v levém grafu. K němu je nejbližší zeleně obarvený bod, přestože leží v oblasti, které dominují spíše modré body a asi bychom selským rozumem považovali za správnější, kdyby tento nový bod byl také modrý. To se dá vyřešit jednoduchou modifikací algoritmu – místo jednoho nejbližšího sousedního bodu se můžeme podívat na více sousedních bodů z nich vybrat

převažující značku. Počet sousedů, na které se díváme je v následujícím kódu označen jako k .

```
1 def k_nearest_neighbors(k, new_instance, training_data):
2     # Seradíme trénovací data podle vzdálenosti od nové instance.
3     # Funkce "distance_from_instance" z předchozí funkce je nahrazena
4     # stručnějším lambda zápisem.
5     training_data_by_distance = \
6         sorted(training_data, key=lambda x: distance(x[0], new_instance))[0:k]
7     # Z těch vezmeme k nejbližších.
8     k_nearest_instances = training_data_by_distance[0:k]
9     # Z nejbližších trénovacích příkladů vezmeme pouze jejich třídu.
10    k_nearest_labels = [y for x, y in k_nearest_instances]
11
12    # Nyní spočítáme, kolikrát se která třída vyskytuje v okolí nové instance.
13    # V následující proměnné je tabulka, do které budeme zapisovat počty tříd.
14    labels_counts = {}
15    # Projdeme třídu nejbližších příkladů jednu po druhé:
16    for label in k_nearest_labels:
17        # pokud ještě není v tabulce, zapiseme její výskyt 1-krát,
18        if label not in labels_counts:
19            labels_counts[label] = 1
20        # pokud už je v tabulce, zvýšíme počet výskytu o 1.
21        else:
22            labels_counts[label] += 1
23    # Vratíme značku, která se vyskytla nejvícekrát.
24    return max(labels_counts, key=lambda c: labels_counts[c])
```

Největší praktickou nevýhodou metody nejbližších sousedů je to, že vyžaduje, aby při klasifikaci nových instancí, byla všechna trénovací data načtena v paměti. To může být v případě velkých datových sad problematické. V ukázkách kódu vždy procházíme všechny trénovací instance. V praxi se používají metody, které umožňují procházet trénovací data chytřeji bez nutnosti vždy procházet celá trénovací data.

1.3 Doporučování filmů

Na principu této metody je založeno také doporučování obsahů na různých webových službách, např. Youtube nebo Netflix (největší světová „půjčovna“ filmů). Základní algoritmus je velice jednoduchý. Pro každého uživatele zaznamenáváme, jak hodnotil jaké filmy. Každý film tedy můžeme popsat vektorem, který říká, jak se líbil kterému uživateli a zároveň každého uživatele můžeme popsat vektorem, který říká, jak se mu líbily které filmy.

Pokud tedy chceme uživateli doporučit něco, co se mu pravděpodobně bude líbit, doporučíme mu, který se líbil podobným lidem (uživatelům, kteří mají nejpodobnější vektory). Stejně tak můžeme snadno vytipovat podobné filmy – budou to zase filmy s nejpodobnějšími vektory.

JL: Mohl bych k tomu namalovat nějaký obrázek

Jako rysy nemusíme používat pouze hodnocení od uživatelů. V případě Youtube je cennou informací, jestli se uživatel na video díval až do konce, jestli ho komentoval, kolik času strávil

čtením komentářů a další podobná data.

Největším problémem této metody se skutečnost, že většina uživatelů většinu videí vůbec neviděla. Kdybychom tedy měli počítat Eukleidovskou vzdálenost jejich vektorů, téměř by se nelišila, protože uživatelé se shodují téměř ve všech dimenzích (videa, která nikdy neviděli). Mohli bychom také chtít porovnat každé dva uživatele pouze v těch dimenzích, které kódují videa, které oba viděli. Potom bychom narazili na problém, že vzdálenosti mezi různými páry uživatelů by byly vzájemně neporovnatelné, protože by používaly různé rysy. Zároveň je nutné umět tuto podobnost počítat velice efektivně, protože se obvykle pracuje s obrovskými daty (miliony uživatelů a videí).

Doporučování uživatelům na základě historie je pochopitelně i komerčně velice zajímavý problém. V letech v roce 2007 vypsala Netflix cenu jeden milion dolarů pro toho, kdo nalezne algoritmus, který bude umět doporučovat filmy s o 10 % větší úspěšností, než algoritmus, který používal Netflix sám. Soutěže se každý rok účastnili přes pět tisíc týmů, které odevzdaly více než 13 000 řešení. Hlavní cena byla udělena až v roce 2009. Přestože trénovací data do soutěže byla anonymizovaná, už v roce 2007 se dvěma vědci z Texaské univerzity podařilo odhalit ztotožnit uživatele s jejich profily v Internet Movie Database. Po sérii žalob na ochranu soukromí se Netflix rozhodl soutěž v roce 2010 ukončit.

1.4 Cvičení – rozpoznávání znaků

JL: OCR pomocí k-NN s Hamming Loss

Shrnutí

- *Klasifikace* je typ úlohy, jejímž cílem je rozdělit objekty (*instance*) do několika *tříd*.
- Jednotlivé *instance* popisujeme množinou *rysů*, které typicky zapisujeme jako *vektory* reálných čísel.
- Algoritmy strojového učení se učí z *trénovacích dat* – datové sady, kde jsou jednotlivým instancím správně přiřazené třídy.
- *Metoda nejbližších sousedů* pracuje tak, že k neznámé instanci nalezne nejpodobnější příklady a přiřadí jí převládající třídu.
- K počítání podobnosti se používá *vzdálenost* mezi vektory rysů, nejtypičtější příkladem je *Eukleidovská vzdálenost*.