Kapitola 1

Metoda nejbližších sousedů

~~Dataset: Iris, Mushroom Data Set~~

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 ~~velice~~ 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.

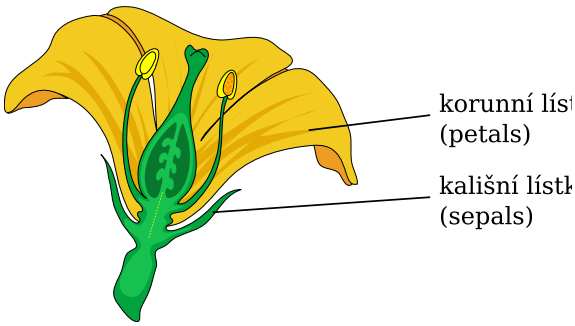
1.1 Klasifikační úlohy

V této kapitole ~~je~~ 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.

Úlohy tohoto typu, kdy se na základě nějakých vlastností objektů rozhodujeme, do jaké skupiny objekt patří, nazýváme obecně *klasifikace* (EN: *classification*). ~~Skupiny, do kterých objekty zařazujeme nazýváme *třídy*~~ 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 ~~.~~ ~~V~~na fotkách. Značky, které v tomto případě ~~třídy, do kterých klasifikujeme jsou lidé~~fotkám přiřazujeme jsou jména osob, které chceme na ~~fotografiích~~ fotkách rozpoznat.



Obrázek 1.1: Schéma květu dvouděložných rostlin.

Metoda nejbližších sousedů se dá několika slovy popsat takto: k neklasifikovanému příkladu najdeme nejpodobnější – nejbližší známý příklad a neklasifikovaný příklad zařadíme ho do stejné třídy. Nejen, že se to snadno řekne, ale jak uvidíme později překvapivě snadno také naprogramuje. Dříve, než se pustíme do samotného programování, zavedeme několik matematických pojmů, které nám vše velice usnadní.

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. Těmto příkladům, ze kterých se naučíme rostliny rozpoznávat říkáme *trénovací data* (EN: *training data*). Jejich ukázku 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. Číslu, které popisuje vlastnost instance, budeme říkat *rys* (EN: *feature*). Někdy se používá také pojem příznak, *příznak*, *vstupní veličina*, nebo slangově počeštělé „fíčura" „fíčura". Jednotlivým exemplářům exemplářům říkáme *instance*)(EN: *instances*).

V případě klasifikace emailů na spamy a „nespamy" „nespamy" se jako rysy používá 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 se čtveřicemi rysů s rysy zacházet jako se čtyřrozměrnými s *vektory* reálných čísel. O vektorech většinou mluvíme v souvislosti s nějakými geometrickými vlastnostmi. Pro geometrií, ale pro nás bude zatím vektor jenom uspořádaná n-tice čísel.

~~Trénovací~~ V případě úlohy s kosatci vypadá trénovací příklad ~~pro nás~~ 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 ~~je~~ 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~~:~~:

```
def nearest_neighbor(new_instance, training_data):
    # Zadefinujeme funkci, ktera pocita vzdalenost nove intance
    # od prikladu z trenovacich dat.
    def distance_from_instance(training_example):
        # Z trenovaciho prikladu nas zajima pouze vektor rysu.
        feature_vector, _ = training_example
        # Vratime vzdalenost vektoru rysu trenovaciho vektoru a nove instance.
        return distance(feature_vector, new_instance)

    # Definovanou funkci pouzije jako klic pro hledani minima
    # v trenovacich datech.
    nearest_vector, nearest_label = \
            min(training_data, key=distance_from_instance)
    # Vratime tridu nejbliziho souseda.
    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

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

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

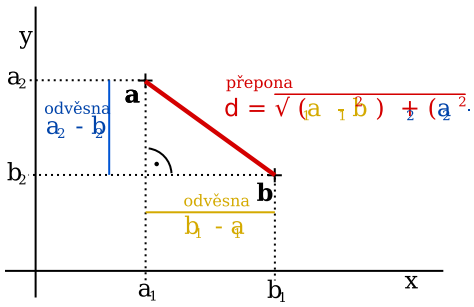
Pro třírozměrné vektory $\mathbf{c} = (c_1, c_2, c_3)$ a $\mathbf{d} = (d_1, d_2, d_3)$, máme Eukleidovskou vzdálenost

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

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 $\mathbf{e} = (e_1, \ldots, e_n)$ a $\mathbf{f} = (f_1, \ldots, f_n)$. Eukleidovskou vzdálenost mezi nimi spočítáme takto:

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

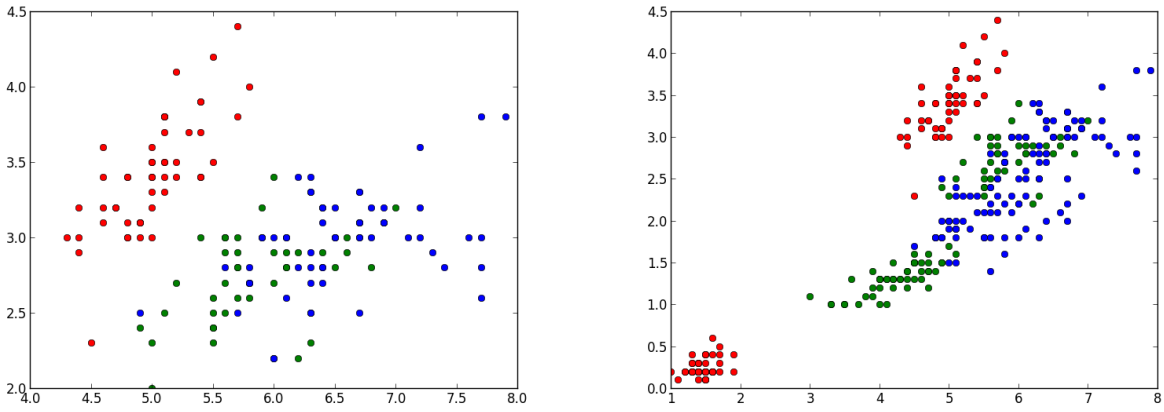


Obrázek 1.2: Odvození Eukleidovské vzdálenosti pomocí Pythagorovy věty. Rozdíly souřadnice určují velikosti přepon trojúhelníku. Odvěsna potom určuje jejich vzdálenost.

V Pythonu zapíšeme Eukleidovskou vzdálenost velice snadno a podobně matematickému zápisu: ~~Vidíme, že se jedná o odmocninu (np.sqrt(...) ze nějakého součtu (sum(...). Operátor -odečte po složkách vektory a operátor **je po složkách umocní.~~

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

Ve cvičení si ukážeme ještě další způsob, jak se počítat vzdálenost mezi instancemi. ~~Pro tyto účely máme další 50 příkladů, u nichž známe, ke~~



Obrázek 1.3: Dvourozměrná vizualizace dat – délka × šířka korunních lístků (vlevo) a délka × 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 nejblíže 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*.

```
def k_nearst_neighbors(k, new_instance, training_data):
    # Seradime trenovaci data podle vzdalenosti od nove instance.
    # Funkce "distance_from_instance" z predchozi funkce je nahrazena
    # strucnejsim lambda zapisem.
    training_data_by_distance = \
        sorted(training_data, key=lambda x: distance(x[0], new_instance))[0:k]
    # Z tech vezmeme k nejblich.
    k_nearest_instances = training_data_by_distance[0:k]
    # Z nejblizsich trenovacich prikladu vezmeme pouze jejich tridy.
    k_nearest_labels = [y for x, y in k_nearest_instances]

    # Nyni spocitame, kolikrat se ktera trida vyskytuje v okoli nove instance.
    # V nasledujici promenne je tabulka, do ktere budeme zapisovat pocty trid.
    labels_counts = {}
    # Projdeme tridy nejblizsich prikladu jednu po druhe:
    for label in k_nearest_labels:
        # pokud jeste neni v tabulce, zapiseme jeji vyskyt 1-krat,
        if label not in labels_count:
            labels_counts[label] = 1
        # pokud uz je v tabulce, zvysime pocet vyskytu o 1.
        else:
            labels_counts[label] += 1
    # Vratime znacku, ktera se vyskytla nejvickrat.
    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 ~~druhu patří. Takovým datům říkáme *testovací data.*~~ 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 vypsal 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 vědcům 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 ukonči.

1.4 Cvičení – rozpoznávání znaků

JL: OCR pomocí k-NN s Hamming Loss

Shrnutí

- ~~klasifikace~~ *Klasifikace* je typ úlohy, jejímž cílem je rozdělit objekty (*instance*) do několika *tříd*.
- ~~množina rysů~~ Jednotlivé *instance* popisujeme množinou *rysů*, které typicky zapisujeme jako *vektory* reálných čísel.
- ~~vektor, skalární součin, vzdálenost~~ Algoritmy strojového učení se učí z *trénovacích dat* – datové sady, kde jsou jednotlivým instancím správě přiřazené třídy.
- ~~trénovací a testovací data~~ *Metoda nejbližších sousedů* pracuje tak, že k neznámé instanci nalezne nejpodobnější příklady a přiřadí jí převládající třídu.

- ~~accuracy~~ K počítání podobnosti se používá *vzdálenost* mezi vektory rysů, nejtypičtější příkladem je *Eukleidovská vzdálenost*.