Unsupervised Learning

Učení bez učitele

Pokyny k online schůzce

1. Používejte nejlépe desktopovou aplikaci Teams.

2. Pokud nekladete dotaz nebo se neúčastníte diskuze, mějte prosím vypnutý mikrofon.

3. Pokud jen trochu můžete, mějte puštěnou kameru. Vás výraz tváře pomůže vyučujícímu :)

4. Jak můžete položit dotaz:
   a. Zapněte si mikrofon a rovnou se zeptejte.
      nebo:
   b. Napište dotaz do chatu. nebo:
   c. Použijte tlačítko zvednout ruku.
      (Po vyvolání ruku sundejte)

Agenda

- Redukce dimenzionality
- Shlukování
- Detekce anomalit
- Doporučovací systémy

Učení s učitelem vs. bez učitele

- většina praktických aplikací strojového učení je založena na učení s učitelem
  - ale většina dostupných dat je k dispozici bez labelů
  - tedy: máme příznaky, ale nemáme cílové hodnoty

- příklad: systém pro detekci vadných výrobků na produkční lince
  - automatizovaně získávat fotky každého produktu je jednoduché
    - během krátké doby můžeme mít dataset o mnoha tisících fotkách
    - ale nemáme labely, který výrobek je vadný a který ne
  - ručně přiřazovat labely mnoha fotkám, jestli je výrobek vadný nebo ne, je nákladné časově i finančně, protože je to lidská práce
    - pokud se zaměříme jen na malou (časově a finančně zvládnutelnou) podmnožinu fotek, které vybereme náhodně, pak patrně nebude výkon klasifikátoru nikterak oslnivý
      - navíc každá změna produktu/výroby by znamenala nutnost práci opakovat
  - metody učení bez učitele nám dokážou výrazně pomoci, abychom dosáhli co nejlepších výsledků při těchto úlohách s co nejmenším množstvím lidské práce

Redukce dimenzionality

Redukce dimenzionality

- pro určité problémy máme tisíce nebo dokonce miliony příznaků pro každou instanci
  - každý příznak představuje další dimenzi
  - trénování modelu pak trvá dlouhou dobu
  - pro řadu algoritmů může být mnohem obtížnější najít dobré řešení, než kdybychom měli méně (více vypovídajících) příznaků
  - “prokletí dimenzionality”
- v praxi je často možné výrazně zredukovat počet příznaků, čímž se dříve neřešitelný problém může stát řešitelným
  - např. dataset číslic MNIST
    - řada pixelů zejména na okraji obrázků nenese žádnou užitečnou hodnotu, takže se jich můžeme zbavit
    - dva pixely vedle sebe jsou většinou silně korelované, takže je můžeme sloučit do jednoho, aniž bychom ztratili velkou část informace

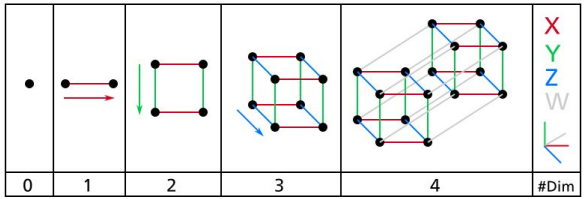
Redukce dimenzionality

- redukce počtu dimenzí vede typicky k rychlejšímu trénování, ale ztrácíme určitou část informací
  - může se pak stát, že systém bude mít o něco horší výsledky
  - pipeline pro zpracování dat je složitější (další krok navíc)
  - => nejprve zkusit bez redukce dimenzionality

- redukce na 2D/3D
  - snížíme-li počet dimenzí na 2, resp. 3, je možné data velmi dobře vizualizovat a získat další informace o datasetu
    - prezentace vizualizovaných dat pro nezúčastněné osoby je pak také mnohem snazší

Prokletí dimenzionality (proč dimenze vadí)

- příklad: vzdálenost dvou náhodných bodů v jednotkové hyperkrychli různého počtu dimenzí
  - čtverec: dva náhodné body jsou průměrně vzdálené 0,52
  - krychle: dva náhodné body jsou průměrně vzdálené 0,66
  - 1M-D hyperkrychle: dva body průměrně vzdálené 408



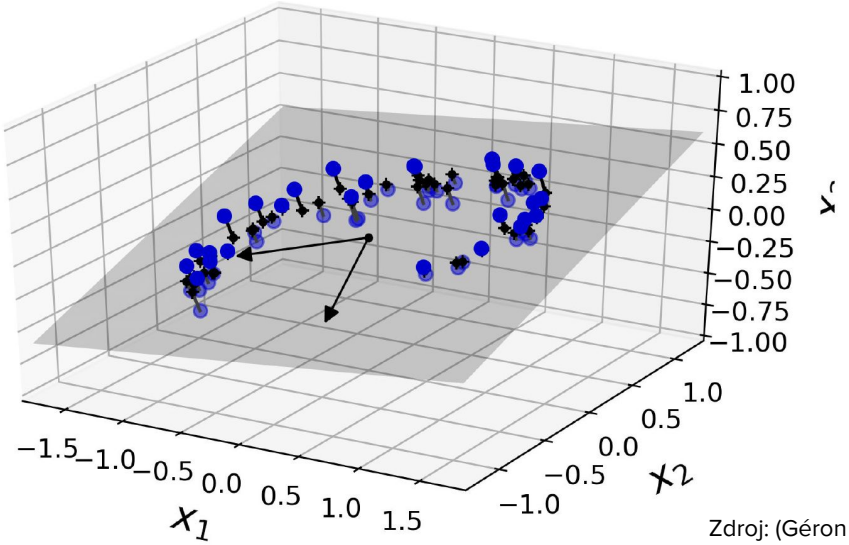
- => vysokodimenzionální datasety jsou velmi řídké (body od sebe velmi vzdálené)
  - nové instance pak budou pravděpodobně daleko od trénovacích instancí => špatné predikce na nových datech (musejí se dělat velké extrapolace)
  - výrazně větší náchylnost na přeučení modelu
- teoretické řešení: získat více dat, aby se dosáhlo požadované hustoty
  - 100 D problém (MNIST - 784 D) by potřeboval více trénovacích instancí, než je atomů v pozorovatelné části vesmíru, aby se dosáhlo průměrné vzdálenosti 0,1 mezi instancemi

Hlavní algoritmy pro řešení redukce dimenzionality

- Principal Component Analysis (metoda projekce)
- Locally Linear Embedding
- Multidimensional Scaling
- Isomap
- t-Distributed Stochastic Neighbor Embedding
- Linear Discriminant Analysis

Projekce

- instance většinou nejsou rozmístěny rovnoměrně ve všech dimenzích
  - některé příznaky mohou být (téměř) konstantní
  - jiné příznaky mohou být silně korelované s dalšími
- trénovací instance tak většinou leží poblíž nějakého méně dimenzionálního podprostoru
  - např. 3D dataset, kde instance leží poblíž 2D podprostoru (roviny)

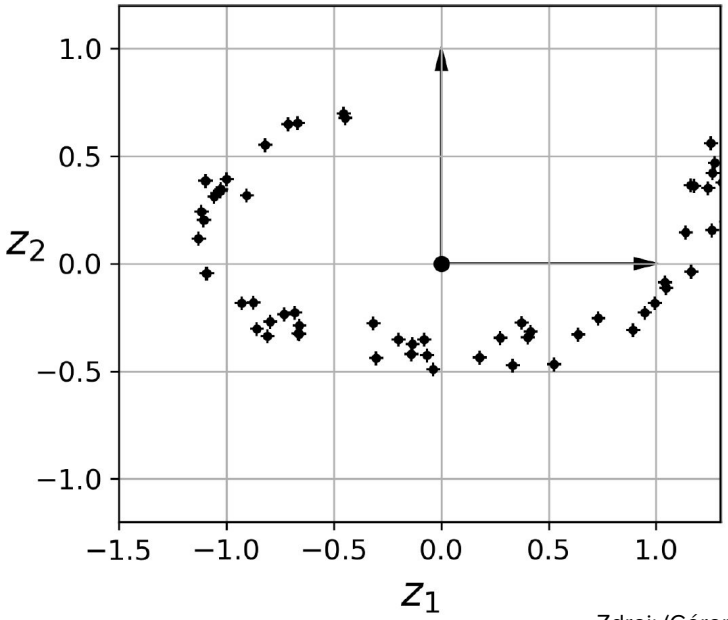


Zdroj: (Géron 2019)

Projekce

- můžeme promítnout instance z 3D datasetu kolmo na nejvhodnější 2D rovinu
  - z příznaků x_1, x_2, x_3 jsme nyní získali nové příznaky z_1 a z_2

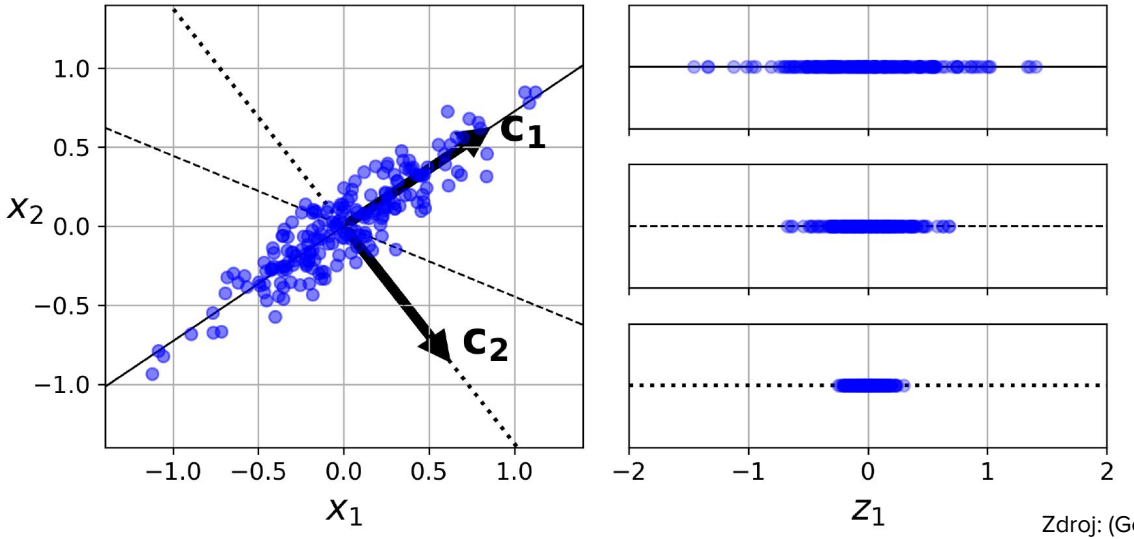
- nejpopulárnější algoritmus, který toto umí provést: PCA



Zdroj: (Géron 2019)

Principal Component Analysis

- zjistí, která nadrovina leží nejblíže instancím a následně instance na tuto nadrovinu promítne
  - hledají se osy, pro které je nejmenší střední kvadratická vzdálenost mezi datasetem a projekcí



Zdroj: (Géron 2019)

PCA

- metoda najde osu, která je zodpovědná za největší část variability v datasetu
  - následně najde druhou osu, kolmou na první, která je zodpovědná ze největší část zbývající variability v datasetu
  - a takto postupuje dále až do konce (kolik dimenzí, tolik os)
- zvolíme si takový počet dimenzí (hlavních komponent), abychom zachovali určitou požadovanou úroveň variability v datasetu (např. 95 % nebo 99 %)

```
from sklearn.decomposition import PCA

pca = PCA(n_components = 2)
X2D = pca.fit_transform(X)
```

```
>>> pca.explained_variance_ratio_
array([0.84248607, 0.14631839])
```

- pro 3D dataset jsme nechali spočítat první dvě hlavní komponenty a vidíme, že první osa dokáže zajistit 84 % variability, druhá 14 % a třetí zanedbatelný zbytek
- pokud chceme počet komponent automaticky dle požadované variability

```
pca = PCA(n_components=0.95)
X_reduced = pca.fit_transform(X_train)
```

PCA

- ukázka viz jupyter

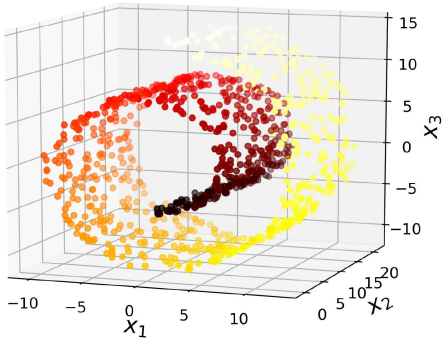
Kernel PCA

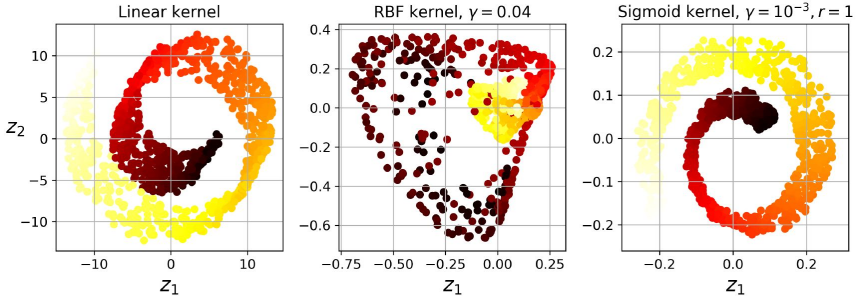
- kernel trick, který je využit u SVM, lze použít i pro projekce nelineárních datasetů pomocí PCA

```
from sklearn.decomposition import KernelPCA

rbf_pca = KernelPCA(n_components = 2, kernel="rbf", gamma=0.04)
X_reduced = rbf_pca.fit_transform(X)
```

- ukázka pro “Swiss Roll” dataset





Zdroj: (Géron 2019)

PCA - výběr kernelu a ladění hyperparametrů

- unsupervised learning algoritmus nemá metriku, kterou bychom mohli rovnou použít pro výběr nejlepších parametrů
- obvykle se ale jedná pro přípravu dat pro supervised algoritmus
  - můžeme tedy využít GridSearchCV a hledat takovou kombinaci hyperparametrů, která povede k nejlepšímu klasifikátoru

```
clf = Pipeline([
        ("kpca", KernelPCA(n_components=2)),
        ("log_reg", LogisticRegression())
    ])

param_grid = [{
        "kpca__gamma": np.linspace(0.03, 0.05, 10),
        "kpca__kernel": ["rbf", "sigmoid"]
    }]

grid_search = GridSearchCV(clf, param_grid, cv=3)
grid_search.fit(X, y)
```

```
>>> print(grid_search.best_params_)
{'kpca__gamma': 0.043333333333333335, 'kpca__kernel': 'rbf'}
```

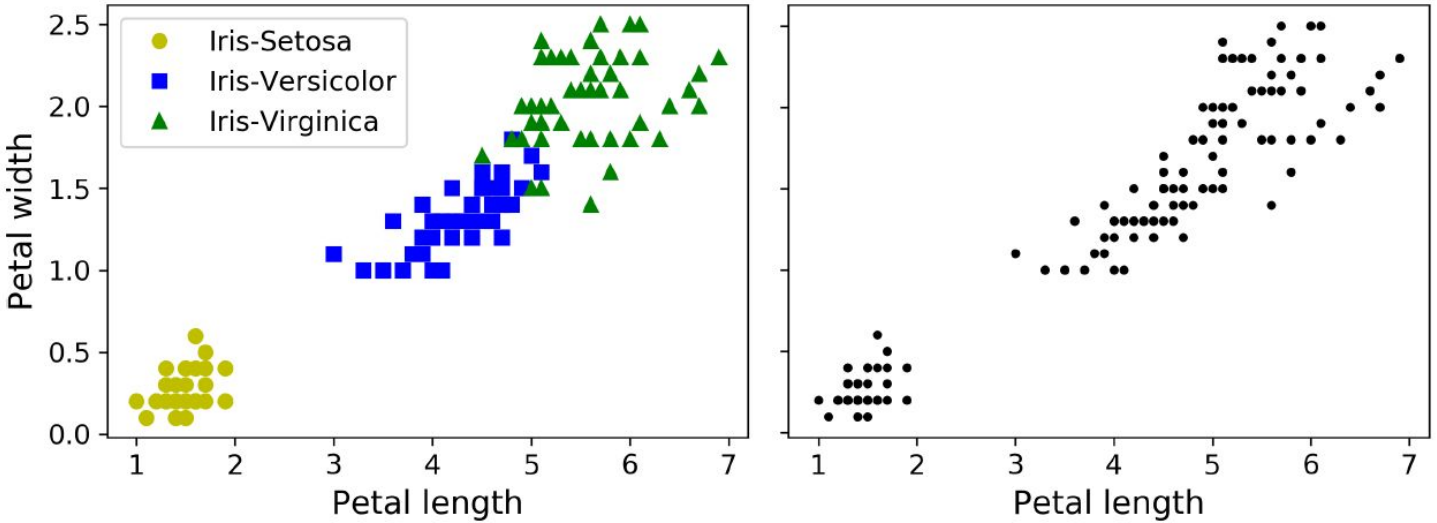
t-SNE (Stochastic Neighbor Embedding)

- velmi vhodné pro vizualizaci multidimenzionálního datasetu ve 2D/3D
- ukázka viz jupyter

Shlukování

Shlukování

- shlukování identifikuje podobné instance a přiřazuje je do shluků (clusterů)
  - nemáme ale na rozdíl od klasifikace žádné labely, jen příznaky



Zdroj: (Géron 2019)

Možné využití shlukování

- segmentace zákazníků
  - můžeme vytvořit skupiny zákazníků na základě jejich nákupů, chování na webu apod. a pak na ně lépe cílit
- analýza dat
  - při prozkoumávání datasetu můžeme zkusit najít shluky podobných instancí a ty pak analyzovat samostatně
- redukce dimenzionality
  - pokud máme vytvořené clustery, můžeme pro každou instanci měřit, jak moc patří do kterého shluku a tím nahradit mnohadimenzionální vektor příznaků
- detekce anomalit
  - instance, která má daleko do všech clusterů, je s vysokou pravděpodobností anomalita
  - využitelné např. při odhalování podvodných transakcí, škodlivého síťového provozu, vad při produkci výrobků apod.
- semi-supervised learning
  - pokud máme jen málo labelovaných dat, můžeme provést shlukování a propagovat známé labely na všechny instance z daného shluku
  - tím získáme mnohem více labelovaných dat a typicky také mnohem lepší výkon supervised algoritmu, který po tomto kroku následuje

Co znamená shluk

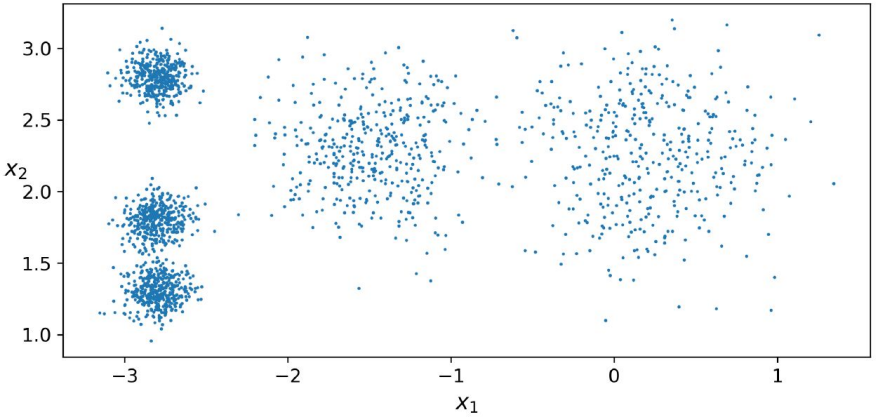
- záleží na konkrétním algoritmu, jak se shluky pracuje

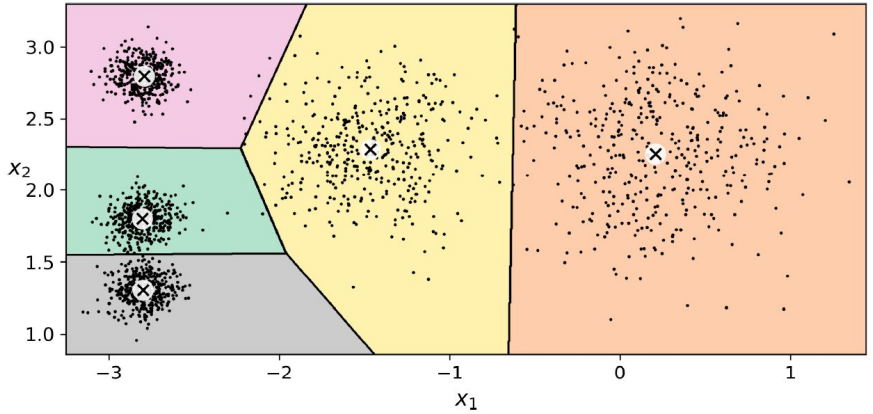
- některé algoritmy definují shluk pomocí bodu, který je uprostřed shluku a instance z tohoto shluku tak k němu mají blízko (centroid)

- jiné algoritmy definují shluk jako oblast, ve které spolu hustě sousedí instance
    - takový shluk pak může mít libovolný tvar, protože není definován jedním bodem

K-Means

- populární algoritmus pro řešení úlohy shlukování
- algoritmus najde centroidy a každý bod přiřadí do určitého shluku






```
from sklearn.cluster import KMeans
k = 5
kmeans = KMeans(n_clusters=k)
y_pred = kmeans.fit_predict(X)
```

Zdroj: (Géron 2019)

K-Means

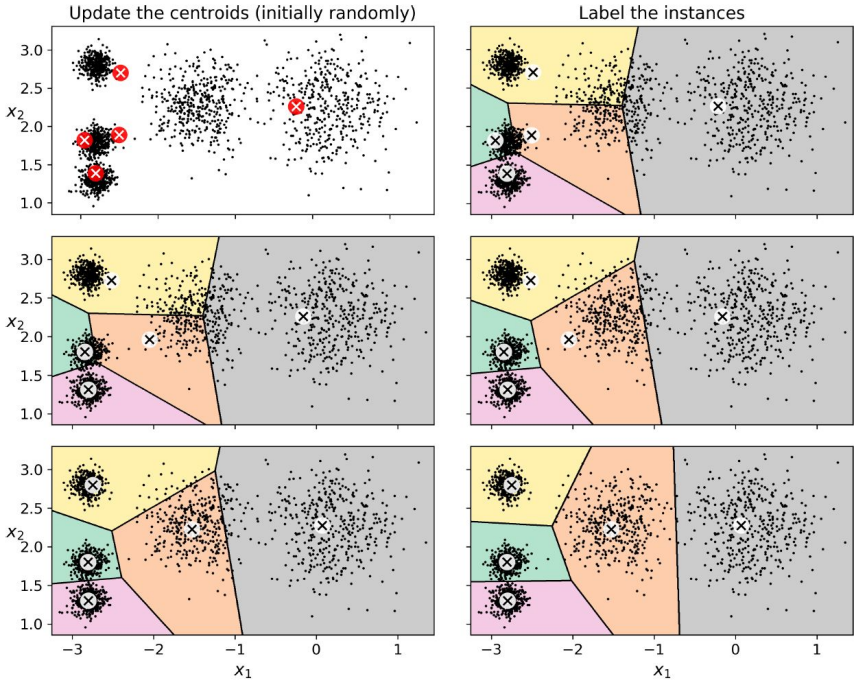
- predikce (přiřazení clusteru pro novou instanci) probíhá podle vzdálenosti od centroidu, takže pokud mají shluky různou velikost (viz předchozí slide), nemusí být výsledek zcela uspokojivý
- hard clustering (přiřazení do shluku) vs soft clustering (vzdálenosti k centroidům)

```
>>> kmeans.transform(X_new)
array([[2.81093633, 0.32995317, 2.9042344 , 1.49439034, 2.88633901],
       [5.80730058, 2.80290755, 5.84739223, 4.4759332 , 5.84236351],
       [1.21475352, 3.29399768, 0.29040966, 1.69136631, 1.71086031],
       [0.72581411, 3.21806371, 0.36159148, 1.54808703, 1.21567622]])
```

- výsledek soft clustering lze použít jako nástroj nelineární redukce dimenzionality
  - pokud máme mnohodimenzionální dataset, tak takto získáme transformovaná k-dimenzionální data

Princip K-means

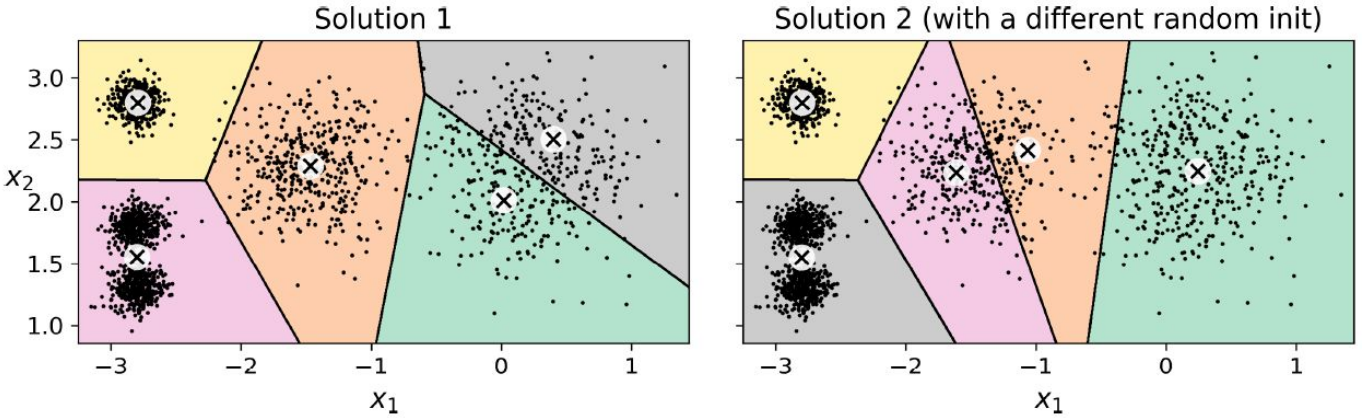
- na začátku se zvolí náhodná poloha centroidů podle zvoleného počtu clusterů
- instancím se přiřadí labely podle nejbližších centroidů
- aktualizuje se poloha centroidů podle průměru všech instancí, které do shluku patří
- takto se postupuje tak dlouho, dokud se mění poloha centroidů



Zdroj: (Géron 2019)

K-means inicializace centroidů

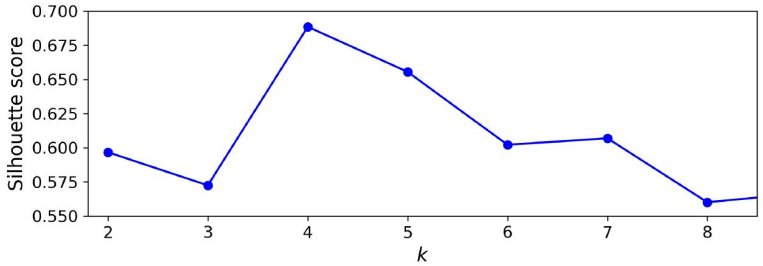
- nevhodná výchozí poloha centroidů může vést k suboptimálnímu řešení
- implementace algoritmu tedy pracuje tak, že celý shlukovací postup opakuje n-krát (standardně 10x) a volí nejlepší řešení
- výchozí poloha v reálné implementaci nebývá volena úplně náhodně, ale používají se (v sklearn automaticky) sofistikovanější přístupy



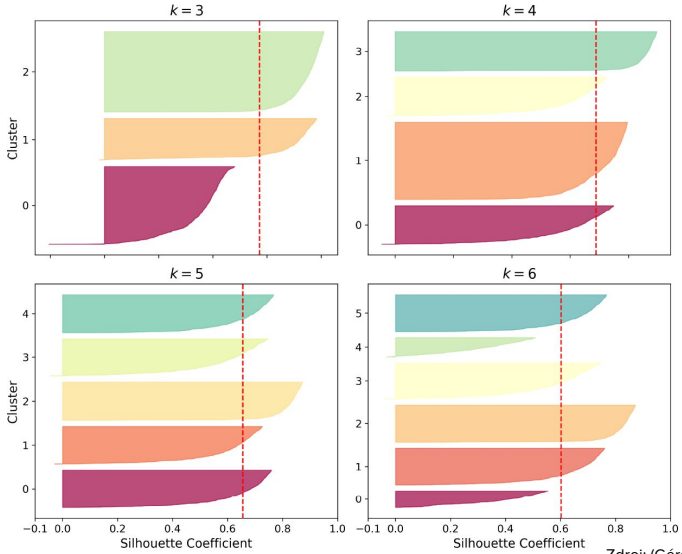
Zdroj: (Géron 2019)

Nalezení optimálního počtu clusterů

- koeficient siluety (silhouette coef.)
  - porovnává, jaká je průměrná vzdálenost mezi instancí a ostatními instancemi ve stejném clusteru a mezi instancí a instancemi v nejbližším sousedním clusteru
  - hodnoty -1 (instance patrně patří do špatného clusteru) až 1 (instance je dobře umístěna ve svém clusteru), hodnoty okolo 0 pak značí, že instance je poblíž hranice clusteru



- slibné hodnoty koeficientu siluety lze prozkoumat pomocí diagramu siluety (instance seřazené podle clusteru a hodnoty koeficientu)
  - pokud instance v clusteru mají nízkou hodnotu (poblíž čáry, nebo nalevo od ní, pak to značí, že daný cluster je špatný, protože instance jsou blízko jiného clusteru)
  - vidíme také velikost clusteru (výška) => k=5 patrně nejlepší



Zdroj: (Géron 2019)

Praktická aplikace K-Means

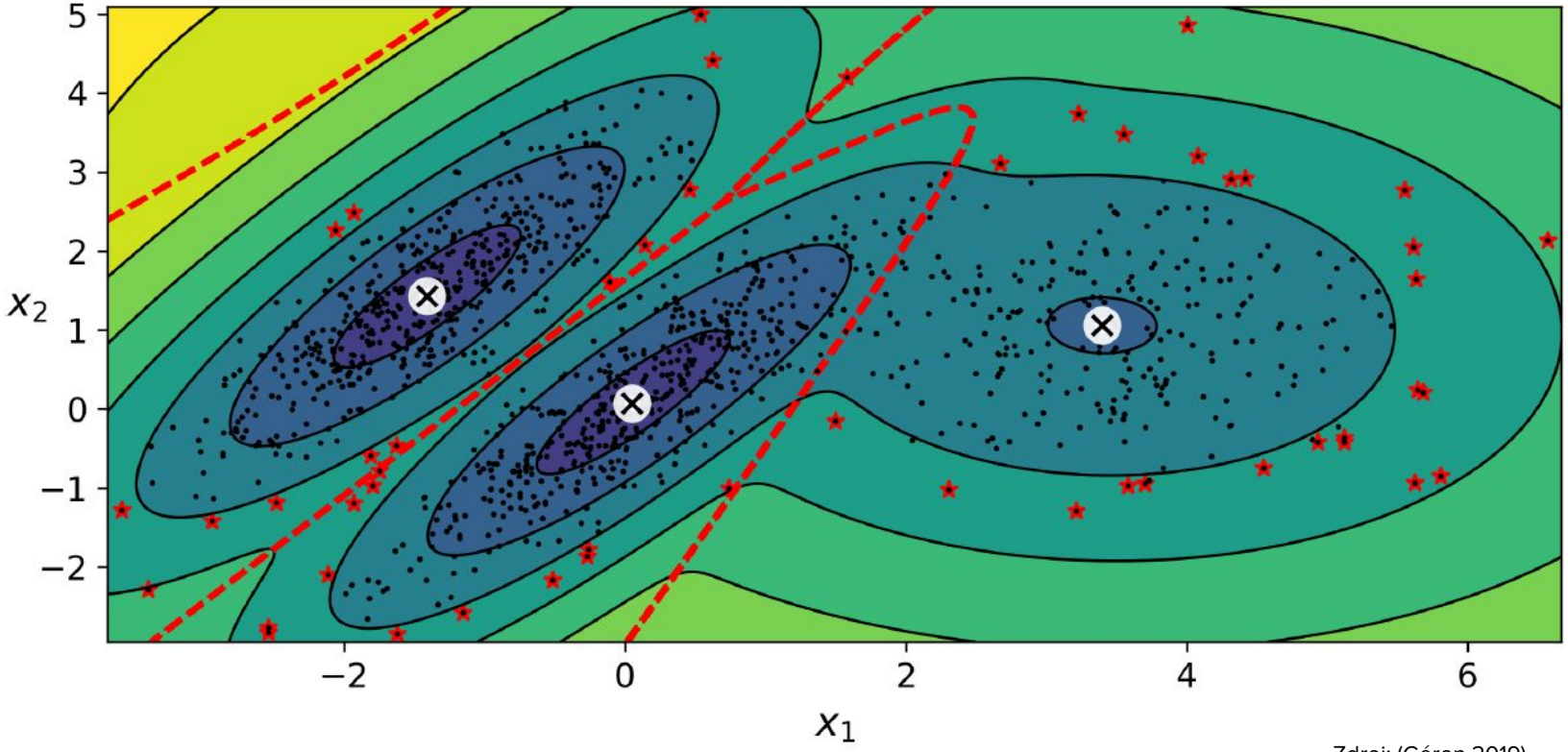
- chceme natrénovat klasifikátor, ale nemáme labely
- viz jupyter

Detekce anomalit

Detekce anomalit

- hledáme instance, které se značně odlišují od normálu
- kromě dříve uvedených způsobů využití lze tyto postupy použít pro odstranění chybných odlehlých hodnot (outliers) z trénovacího datasetu, což může vést ke zlepšení modelu
- algoritmy:
  - Gaussian Mixtures
    - shlukovací algoritmus, ale je vhodný i pro detekci anomalit
    - pracuje s hustotou instancí; instance v oblasti s nízkou hustotou je anomalita
  - Isolation Forest
    - buduje náhodné lesy, kde každý strom vzniká z náhodně vybraných příznaků a náhodných prahových hodnot a pokračuje tak dlouho, dokud každá instance není na samostatném listu
    - anomalita je typicky daleko od ostatních instancí, proto v průměru celého lesu skončí anomalita oddělena dříve do samostatného listu, než ostatní instance
  - One-class SVM
    - vhodné spíše pro novelty detection - máme čistý dataset a chceme upozornit na novou instanci, která je odlišná (oproti tomu anomality detection předpokládá, že anomality jsou již v trénovacím datasetu)
    - dokáže udělat hranici okolo původního datasetu a označit pak instance, které spadají mimo tyto hranice

Detekce anomalit pomocí Gaussian Mixtures



Zdroj: (Géron 2019)

Detekce anomalit pomocí One-class SVM

- viz jupyter

Doporučovací systémy

Doporučovací systémy

- složité téma, které lze řešit řadou velmi rozdílných způsobů

- memory based
  - content filtering
    - TF-IDF, míry podobnosti, shlukování
  - collaborative filtering
    - míry podobnosti, KNN, shlukování

- model based
  - content filtering
    - neuronové sítě
  - collaborative filtering
    - neuronové sítě

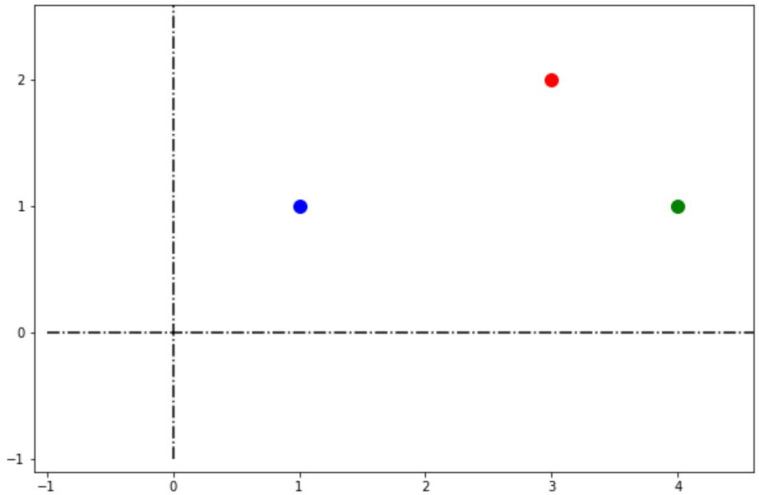
Content vs. Collaborative filtering

- content filtering
  - doporučení je řešeno na základě podobnosti obsahu
  - item vs. user content
    - item - hledáme instance, které jsou podle obsahu podobné jiné instanci
    - user - hledáme uživatele, kteří jsou podobní našemu uživateli a pak nabídneme to, co měli rádi podobní uživatelé
  - příklad u databáze filmů:
    - item-content filtering: žánr, herecké obsazení, režie, produkce, klíčová slova a jakákoliv další metadata o filmu
    - user-content filtering: věk, pohlaví a podobně
- collaborative filtering
  - máme k dispozici hodnocení položek od různých uživatelů a hledáme podobnou položku na základě těchto hodnocení

Míry podobnosti

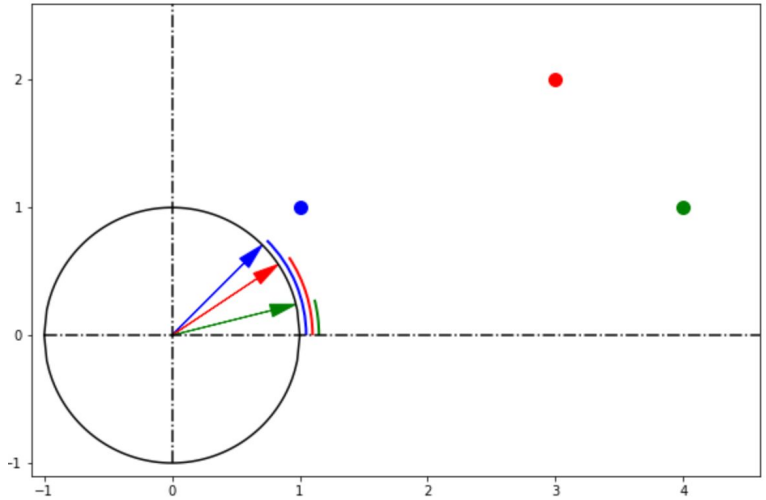
euklidovská vzdálenost (euclidean distance)

- určuje vzdálenost “pravítkem” mezi dvěma body

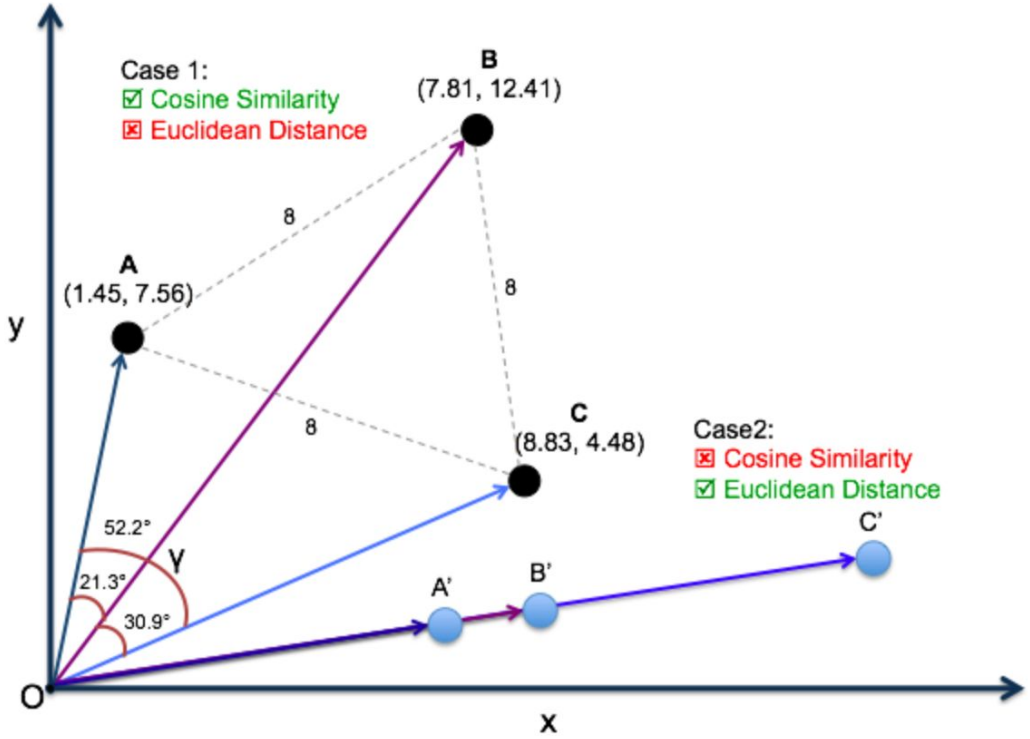


kosinová vzdálenost (cosine similarity)

- udává cosinus mezi dvěma vektory
  - pokud jsou stejné (úhel 0), pak cos = 1



Zdroj: https://www.baeldung.com/cs/euclidean-distance-vs-cosine-similarity



Zdroj: https://medium.com/@sasi24/cosine-similarity-vs-euclidean-distance-e5d9a9375fc8

Doporučovací systém pro filmy

- dataset MovieLens
- viz jupyter

Zdroje

- Coelho, L. P.; Richert, W. (2013) Building machine learning systems with Python. Birmingham: Packt Publishing. ISBN 978-1-78216-140-0.

- Géron, A. (2019) Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition. O'Reilly Media, Inc. ISBN 9781492032649.

- Chollet, F. (2019) Deep Learning v jazyku Python. Knihovny Keras, TensorFlow. Grada Publishing, a.s. ISBN 978-80-247-3100-1.

- Segaran, T. (2007) Programming collective intelligence: building smart web 2.0 applications. Beijing: O'Reilly Media. ISBN 0-596-52932-5.