AnetaPopelova new file organisation 3 weeks ago

2086 lines (2086 loc) · 1.72 MB

Preview Code Blame Raw

Lekce 5

Úvod do strojového učení

Strojové učení (angl. *machine learning*) pod sebou sdružuje všechny systémy, které se učí z dat bez toho, aby potřebovali explicitní instrukce nebo pravidla. Informace, které se tyto systémy naučí z dat, umí pak generalizovat na nová, neznámá data.

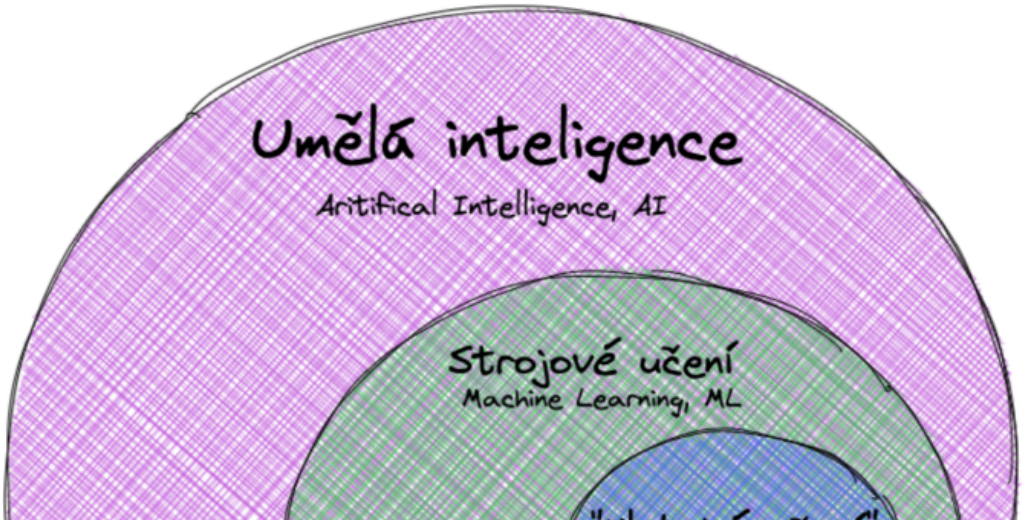
Když z analýzy dat našeho e-shopu zjistíme, že mladí studenti z Brna nakupují často boty přes internet a nabídneme proto všem budoucím uživatelům z Brna s věkem pod 26 let slevu na online nákup bot, tak to není strojové učení. Jedná se o jednoduché pravidlo, které jsme odvodili sami (i když třeba s využitím nějakých nástrojů datové analýzy).

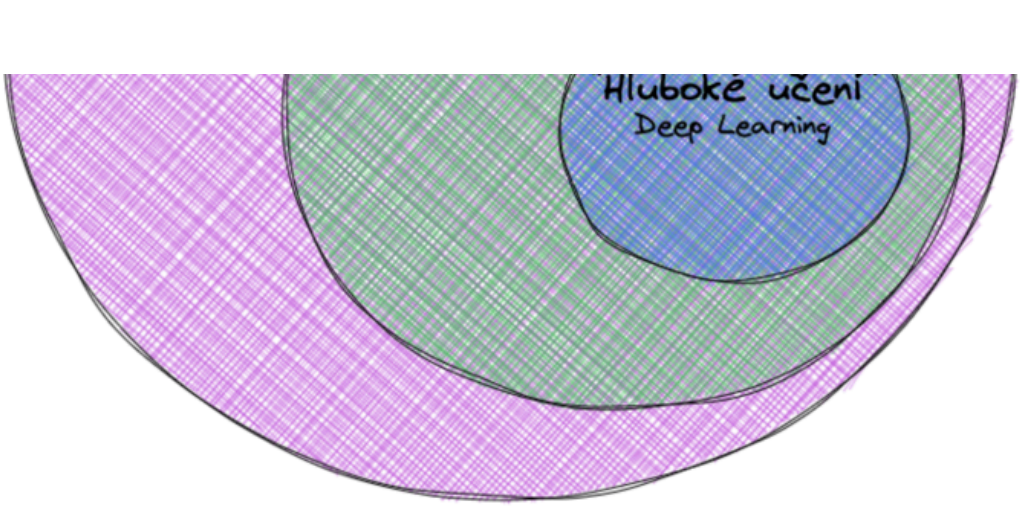
Když náš chatbot telefonního operátora při detekování slova "tarif" vždy odpoví tak, že pošle odkaz na přehled všech tarifů, tak to také není strojové učení. Zase se jedná jen o nějaké pravidlo.

V oblasti strojového učení panuje mírné zmatení pojmů, které se v různých odvětvích vykládají různě. Neberte proto definice z tohoto kurzu jako úplně závazné :-)

Strojové učení je podoblast umělé inteligence. Pojem umělé inteligence je často zaměňován právě za strojové učení. Můžete se také setkat s pojmem *deep learning* nebo s pojmem *hluboké neuronové sítě*. Ty označují podskupinu algoritmů strojového učení, které vzrostly v popularitě a zájmu veřejnosti i akademiků z důvodu výrazně lepších a zobecnitelnějších výsledků než takzvané klasické strojové učení. Vyžadují ale velký výpočetní výkon (větší než má běžný osobní počítač), a také větší objem dat.

V tomto kurzu se "hlubokému" strojovému učení věnovat nebudeme. Pro pochopení jak neuronová síť funguje bychom potřebovali i podstatnou část matematiky, konkrétně lineární algebru. Základní principy jsou ale stejné jako pro klasické strojové učení, a to je důležité si uvědomit.





Čtení na doma: Příklady využití strojového učení

- Detekce podvodů v bankovnictví: Strojové učení se používá k analýze transakcí a určení, zda jsou některé z nich podezřelé a mohou představovat podvod.
- Doporučovací systémy: Firmy jako Netflix a Amazon používají strojové učení k doporučení filmů, televizních pořadů nebo produktů založených na chování uživatelů.
- Rozpoznávání obličeje: Strojové učení se používá k vytvoření systémů rozpoznávání obličeje, které mohou být použity v mobilních telefonech pro odemykání nebo v bezpečnostních systémech.
- Autonomní vozidla: Strojové učení je klíčové pro vývoj autonomních vozidel, které se učí navigovat na základě analýzy obrazu a dalších senzorických dat.
- Prediktivní údržba: Výrobní společnosti mohou používat strojové učení k předvídání, kdy bude pravděpodobně potřeba údržba strojů a zařízení, což umožňuje minimalizovat dobu odstávky.
- Zdravotnictví a medicína: Strojové učení se využívá pro diagnostiku chorob, předvídání výsledků léčby a vývoj nových léků. Může také analyzovat obrazové data, například rentgenové snímky nebo snímky z magnetické rezonance.
- Zpracování přirozeného jazyka (NLP): Strojové učení se používá pro automatický překlad jazyků, generování textu, rozpoznávání hlasu a další úkoly spojené se zpracováním a generováním přirozeného jazyka.
- Bezpečnostní systémy: Strojové učení se také využívá k detekci neobvyklých aktivit nebo útoků na počítačové systémy a sítě.
- Energetika: Strojové učení může pomoci optimalizovat výrobu a distribuci energie, předvídat spotřebu energie a výkon solárních panelů.
- Marketing a prodej: Strojové učení se využívá pro segmentaci zákazníků, předpověď chování zákazníků a personalizaci reklamních kampaní.
- Zemědělství: Strojové učení může pomoci předpovědět úrodu, optimalizovat zavlažování a detekovat nemoci rostlin.
- Hudba a umění: Algoritmy strojového učení mohou generovat hudbu nebo umělecká díla a také doporučovat obsah na základě uživatelských preferencí.
- Meteorologie: Strojové učení se může využít k předpovědi počasí a klimatických změn.
- Social media: Strojové učení je hojně využíváno v social mediích pro filtraci a personalizaci obsahu, rozpoznávání a sledování trendů, detekci hate speech, fake news, a další.

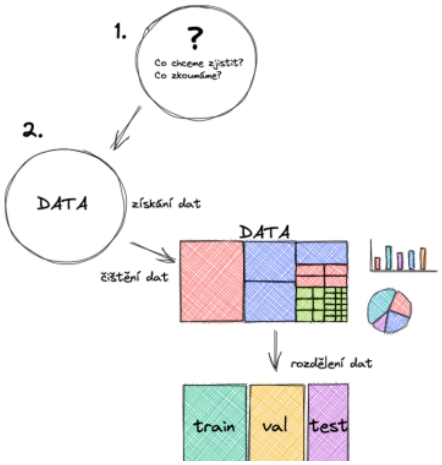
- Realitní trh: Strojové učení může být využito k odhadu cen nemovitostí na základě různých faktorů, jako je umístění, velikost, stáří a další.
- Hry: V oblasti herního průmyslu se strojové učení používá k vytváření inteligentních NPC (non-player characters), k návrhu nových úrovní nebo k optimalizaci herních strategií.

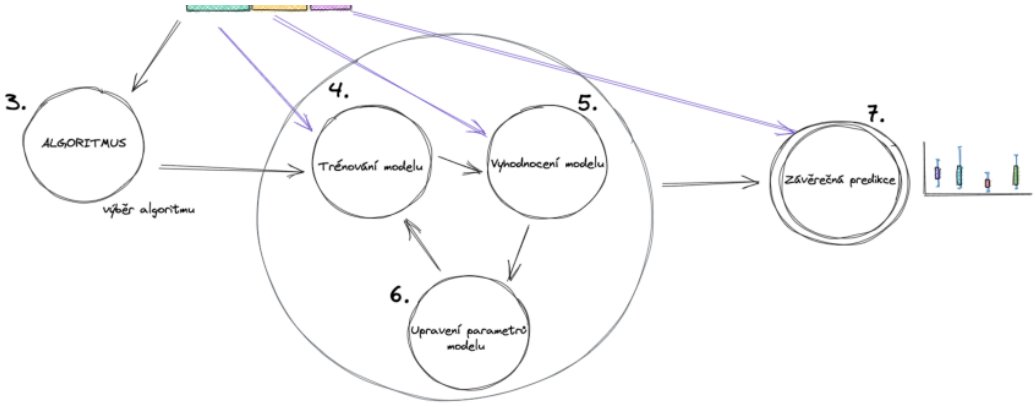
Obecný postup při využití strojového učení

Pokud chceme nějakou otázku zodpovědět pomocí strojového učení, můžeme následovat obecný postup, který je společný pro všechny algoritmy. Podobný postup jsme měli i u testování hypotéz.

1. V první řadě je potřeba si **definovat, čím se zabýváme, a jaký je problém, který řešíme**. Typicky budeme chtít získat nějaké předpovědi nebo odhady na základě dat. Také je dobré si rozmyslet, jestli vůbec strojové učení potřebujeme. Není náš problém řešitelný nějakým jednodušším způsobem?
2. Následně potřebujeme **získat data**, s pomocí kterých budeme naší úlohu řešit. Data musíme takzvaně připravit. Je potřeba se rozhodnout, co budeme dělat v případě chybějících hodnot, zamyslet se nad případnou normalizací dat (co když jsou některé záznamy o vzdálenosti v kilometrech a jiné v mílích?) a také data náhodně rozdělit na trénovací a testovací sadu.
3. Když máme připravená data, můžeme si **vybrat jeden nebo více algoritmů** strojového učení. Algoritmus vybereme podle typu úlohy a charakteru dat.
4. Teď už můžeme takzvaně **"natrénovat"** (*train*) náš model. Spustíme algoritmus na sadu trénovacích dat a výsledkem je natrénovaný nebo naučený model.
5. Na základě validační sady dat **ověříme**, jak dobře umí náš model předvídat na datech, která nezná.
6. Podle výsledků evaluace můžeme **upravit některé parametry modelu**. V tom případě jdeme zpět na krok trénování modelu, a takto postupujeme v cyklu, dokud nejsme s výsledky spokojení.
7. Na závěr spustíme náš model na **závěrečnou testovací sadu dat**. Tuto sadu algoritmus neviděl při trénování, ale ani jsme jí nepoužívali na upravování parametrů. Výsledky, které dostaneme pro tuto sadu dat by měly být ty, které napíšeme do akademického článku, do pracovní prezentace, nebo do zprávy o našem projektu.

Pro účely kurzu budeme často využívat jen dvě sady dat: trénovací a testovací.





Supervised learning

Existuje velká třída úloh, kterým se říká "supervised learning", nebo "s učitelem". Existují dvě velké podtřídy těchto úloh.

Klasifikační úlohy

Cílem klasifikační úlohy (*classification*) je rozdělit data do dvou či více skupin.

Příklady klasifikačních úloh jsou:

- Detekce spamu: Model strojového učení je natrénován na detekci spamových e-mailů na základě předchozích zkušeností. E-maily mohou být klasifikovány jako "spam" nebo "ne-spam".
- Diagnostika nemocí: Na základě různých diagnostických testů může být model strojového učení použit k diagnostice různých typů nemocí. Například na základě snímků z magnetické rezonance mohou být pacienti klasifikováni jako "má rakovinu" nebo "nemá rakovinu".
- Rozpoznávání ručně psaných číslic: Klasickým příkladem je MNIST dataset ručně psaných číslic, kde úkolem je klasifikovat obrázky do 10 tříd odpovídajících číslicím 0 až 9.
- Predikce úvěrové bonity: Model může být vytvořen pro klasifikaci jednotlivců do kategorií "dobrý úvěrový riziko" a "špatné úvěrové riziko" na základě finančních a osobních údajů.
- Klasifikace genů: V bioinformatice může být model použit k třídění genů do různých tříd na základě sekvence DNA.
- Detekce sociálních botů: Na sociálních médiích může být model použit k rozpoznání účtů, které jsou pravděpodobně ovládány boty, na základě jejich chování a vzorců aktivity.

Regresní úlohy

Cílem regresní (*regression*) úlohy je predikce číselné hodnoty. Strojové učení nabízí algoritmy podobné klasické regresi i řadu jiných.

Příklady regresních úloh jsou:

- Predikce spotřeby energie: Model může být natrénován na předpovědění spotřeby energie budov nebo celých měst na základě historických dat a různých vlivů, jako je počasí, doba dne, atd.

- Predikce akciových cen: I když je tato úloha velmi obtížná kvůli nestabilitě trhů, modely strojového učení mohou být použity pro předpovědění trendů akciových cen.
- Predikce délky hospitalizace: V zdravotnictví může být model použit k předpovědění doby, po kterou bude pacient potřebovat hospitalizaci.
- Predikce teploty: Modely mohou být využity pro předpovědění teploty v konkrétních časových úsecích nebo lokalitách na základě historických dat a vzorců.

In [1]:
```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import accuracy_score, precision_score, recall_score
```

In [2]:
```
import warnings

warnings.filterwarnings("ignore")
```

Popis importů

- `StandardScaler` - objekt pro normalizaci dat, dokumentace je zde
- `train_test_split` - funkce pro rozdělení dat na trénovací a testovací, dokumentace je zde
- `KNeighborsClassifier` - klasifikátor, používá algoritmus K Nearest Neigbors, dokumentace je zde
- `ConfusionMatrixDisplay` - vizualizace matice záměn, dokumentace je zde
- `accuracy_score`, `precision_score` a `recall_score` - funkce pro vyhodnocení výsledků modelu, dokumentace je zde

Strojové učení v Pythonu

V našem kurzu budeme pracovat s modulem `scikit-learn`. Tento modul je velice často využíván v takzvaném klasickém strojovém učení (tj. strojové učení, které *nevyužívá* hlubokých neuronových sítí). Používá se jak pro výuku (má k dispozici velké množství tutoriálů a příkladů), tak pro praktické využití v akademickém nebo soukromém sektoru.

Data si můžeš stáhnout zde.

Zdroj dat: https://archive.ics.uci.edu/ml/datasets/wine+quality

Definice problému a data

Máme data o chemickém složení vína a ohodnocení vín do kategorií dobré a špatné. Naším úkolem je vytvořit model, který bude klasifikovat víno na základě jeho složení do jedné ze dvou těchto skupin.

In [27]:
```
data = pd.read_csv("data/wine.csv")
data.head()
```

Out[27]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulp
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	

Konkrétně víme následující údaje:

- fixed acidity (stálá kyselost),
- volatile acidity (těkavá kyselost),
- citric acid (kyselina citronová),
- residual sugar (zbytkový cukr),
- chlorides (chloridy),
- free sulfur dioxide (volný oxid siřičitý),
- total sulfur dioxide (celkový oxid siřičitý),
- density (hustota),
- pH (pH),
- sulphates (sírany),
- alcohol (alkohol),
- quality (kvalita).

In [4]:
```
data.value_counts("quality")
```

Out[4]:
```
quality
good    855
bad     744
Name: count, dtype: int64
```

Rozdělíme si vstupní proměnné a cílovou hodnotu:

In [5]:
```
X = data.drop(columns=["quality"])
y = data["quality"]
```

Data si teď rozdělíme na trénovací a testovací sadu. Není to tak těžké udělat přímo pomocí pandasu (například mohli bychom data zpřeházet pomocí metody `.sample()`, a pak si vzít prvních x řádek pro testovací sadu a zbytek pro trénovací), ale modul `scikit-learn` má také pěknou metodu, kterou si ukážeme.

In [6]:
```
X_train, X_test, y_train, y_test = train_test_split(
```

```
    x, y, test_size=0.3, random_state=42
)
```

Ještě nás čeká jeden důležitý krok. Všimněme si, že data nejsou normalizovaná. Například sloupec `total sulfur dioxide` se pohybuje řádově ve desítkách, zatímco `ph` se pohybuje v jednotkách. Data normalizujeme metodou z-scores. Odečteme průměr a vydělíme standardní odchylkou:

$$z = \frac{x - \mu}{\sigma}$$

Mohli bychom opět využít čistě pandas (například bychom si mohli napsat funkci, která data normalizuje, a pomocí `apply` jí uplatnit na každý sloupec). Ukážeme si ale jak postupovat pomocí knihovny `sklearn`.

In [7]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Proč napřed voláme metodu `fit_transform` na trénovací data a pak jen `transform` na testovací? `fit_transform` vlastně dělá dvě věci: napřed spočítá pro každý sloupec průměr a směrodatnou odchylku. Pak trénovací data normalizuje. Pro testovací data už nepočítá nové hodnoty průměru a odchylky, ale použije ty z trénovacích dat.

Jedná se o standardní postup: snažíme se maximálně zachovat odstup trénovacích dat od testovacích. Kdybychom využili testovací data pro spočítání průměru, už tím zprostředkováváme nějaké informace, které při trénování modelu nemáme znát.

Naše data už teď nejsou ve formátu `pandas.DataFrame`, ale jsou uložené jako matice knihovny `numpy`.

In [8]:
```
X_train
```

Out[8]:
```
array([[ 1.69536131e-01, -1.72107140e+00,  4.59303345e-01, ...,
         1.01180685e+00,  1.22661179e+00,  5.50057013e-01],
       [ 2.44606730e+00, -4.01957443e-01,  1.84105501e+00, ...,
        -2.10687612e+00,  1.22661179e+00, -2.05174641e-01],
       [-6.47680186e-01,  3.77472102e-02, -1.28054303e-03, ...,
         4.92026353e-01,  2.97270776e-01,  5.50057013e-01],
       ...,
       [-6.47680186e-01,  4.77451864e-01, -1.07597628e+00, ...,
         1.27169710e+00, -6.90154049e-01, -8.66002338e-01],
       [-2.39072027e-01, -1.83099757e+00,  4.08127357e-01, ...,
         3.72184202e-02,  8.20025095e-01,  1.39969262e+00],
       [-1.46489650e+00, -1.33632983e+00, -5.24565306e-02, ...,
         4.92026353e-01, -6.90154049e-01,  2.91015593e+00]])
```

Výběr algoritmu

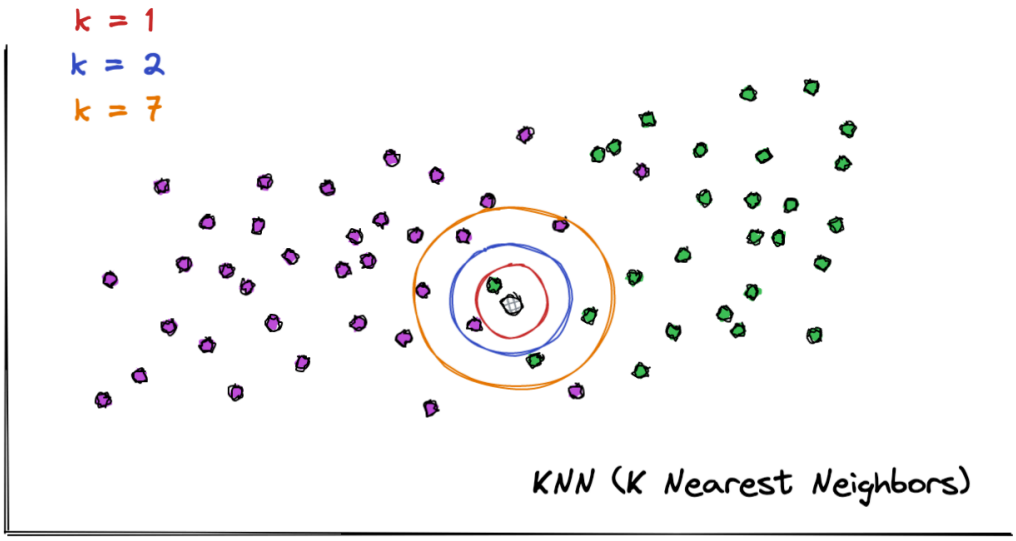
Pro začátek zvolíme jednoduchý algoritmus, který pracuje jen s jedním parametrem. Jmenuje se "K Nearest Neighbors", tedy se jedná o algoritmus, který pracuje s K nejbližšími "sousedy".

Tento algoritmus využívá úvahy, že stačí uložit si trénovací data, a pak každý nový testovací záznam určitým způsobem porovnat s nejbližšími sousedy v trénovacích datech. U trénovacích dat známe jejich cílovou hodnotu, takže na základě té rozhodneme i o hodnotě testovacího záznamu. Algoritmus se dá dobře znázornit i vizuálně, když si představíme, že pracujeme se dvěma vstupními proměnnými (osy x a y), a barva znázorňuje cílovou hodnotu. Červená tečka je nový testovací záznam, u kterého se rozhodujeme, jaká bude jeho hodnota.

Pro hodnotu parametru `k = 1` se prostě podíváme na nejbližšího souseda a přiřadíme stejnou hodnotu (zelenou) i našemu testovacímu záznamu.

Pro hodnotu `k = 2` se dostaneme do složité situace - není možné určit, jaká má být výsledná hodnota, protože jeden soused je fialový a druhý zelený. Z tohoto důvodu se typicky nepoužívají sudé hodnoty pro parametr `k`.

Pro hodnotu `k = 7` hlasování sousedů doapdne v prospěch fialové cílové hodnoty.



Trénování modelu

In [9]:
```
clf = KNeighborsClassifier()
clf.fit(X_train, y_train)
```

Out[9]:
```
KNeighborsClassifier()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Vyhodnocení modelu

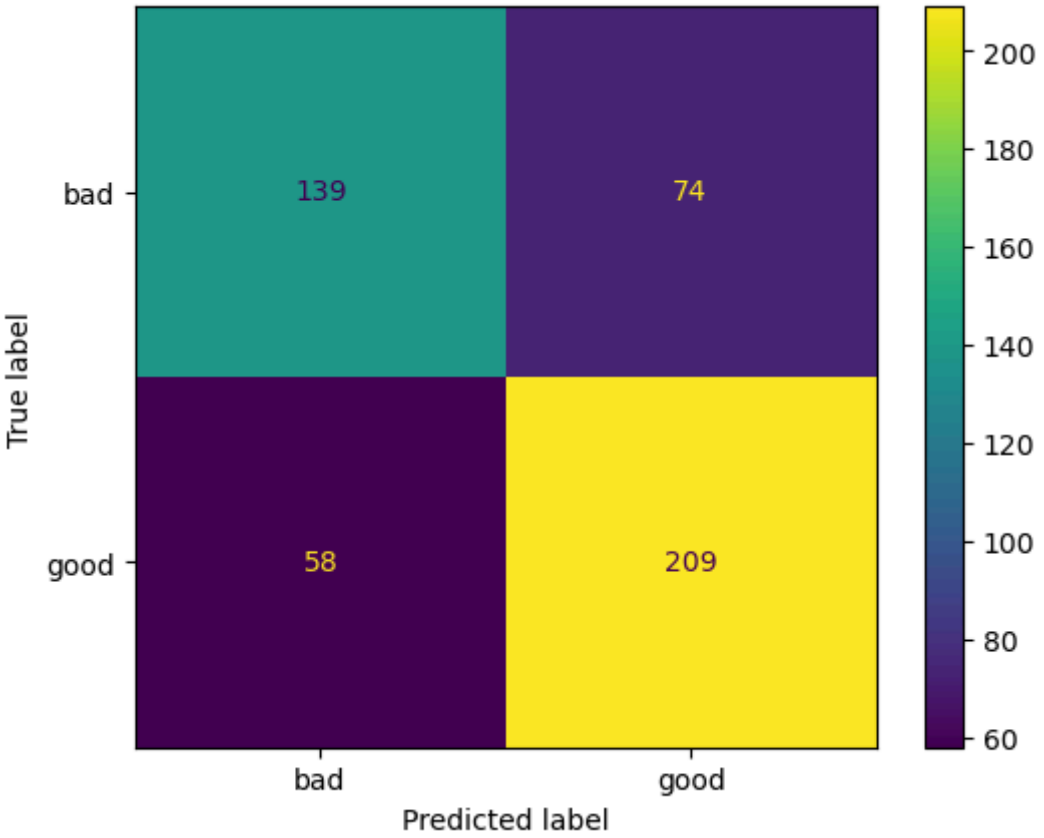
Získáme predikce našeho modelu a porovnáme je se skutečností.

In [10]:
```
y_pred = clf.predict(X_test)
```

Kvalitu modelu vyhodnotíme s využitím Matice záměn (Confusion Matrix).

In [11]:
```
ConfusionMatrixDisplay.from_estimator(
    clf,
    X_test,
    y_test,
)
```

Out[11]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x17553d150>
```



Existuje mnoho metrik, podle kterých můžeme měřit úspěšnost našeho modelu. Většina se dá odvodit z výsledků, které nám znázorňuje barevná matice. Každý ze čtverců si můžeme označit zkratkou a interpretovat, co znamená.

Terminologie	Popis	Počet
True Positives (**TP**)	Vína, která jsme správně klasifikovali jako dobrá.	209
True Negatives (**TN**)	Vína, která jsme správně označili jako špatná.	139
False Positives (**FP**)	Vína, která jsme chybně označili jako dobrá, ve skutečnosti jsou špatná.	74
False Negatives (**FN**)	Vína, která jsme chybně označili jako špatná, ve skutečnosti jsou dobrá.	58

Jak teď spočítat úspěšnost modelu? Nabízí se několik metrik:

- *Accuracy*: Poměr správně určených záznamů oproti celku.
  - $A = \frac{TP+TN}{TP+TN+FP+FN}$
- *Precision*: Tato metrika **penalizuje** označení **špatného vína za dobré**. Čím více špatných vín označíme za dobrá, tím má metrika menší hodnotu. Metrika

nepočítá s tím, kolik dobrých vín jsme označili za špatná.
    - $P = \frac{TP}{TP+FP}$
- *Recall*: Tato metrika **penalizuje** označení **dobrého vína za špatné**. Čím více dobrých vín označíme za špatná, tím má metrika menší hodnotu. Metrika nepočítá s tím, kolik špatných vín jsme označili za dobrá.
    - $R = \frac{TP}{TP+FN}$
- *F1 Score*: Metrika která zohlední jak Precision, tak Recall.

In [12]:
```
accuracy_score(y_test, y_pred)
```

Out[12]:
```
0.725
```

In [13]:
```
precision_score(y_test, y_pred, pos_label="good")
```

Out[13]:
```
0.7385159010600707
```

In [14]:
```
209 / (209 + 74)
```

Out[14]:
```
0.7385159010600707
```

In [15]:
```
recall_score(y_test, y_pred, pos_label="good")
```

Out[15]:
```
0.7827715355805244
```

In [16]:
```
209 / (209 + 58)
```

Out[16]:
```
0.7827715355805244
```

Úprava parametrů modelu

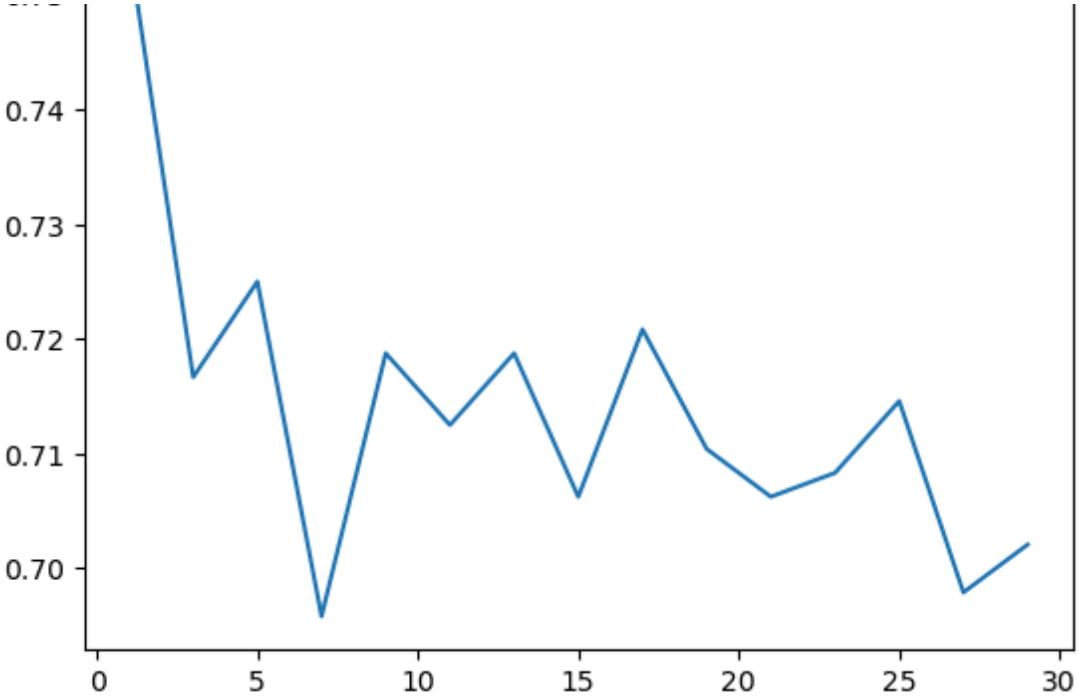
Máme tedy nějakou základní představu o tom, jak se chová náš model. Zkusíme upravit parametr `k`, který říká, na kolik "sousedů" se podíváme, abychom rozhodli o cílové hodnotě.

In [17]:
```
ks = range(1, 31, 2)
accuracy_scores = []

for k in ks:
    clf = KNeighborsClassifier(n_neighbors=k)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    accuracy_scores.append(accuracy_score(y_test, y_pred))
```

In [18]:
```
plt.plot(ks, accuracy_scores)
```

Out[18]:
```
[<matplotlib.lines.Line2D at 0x175729690>]
```



Závěrečná predikce

Nyní provedeme finální predikci

In [19]:
```
clf = KNeighborsClassifier(n_neighbors=1)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
accuracy_score(y_test, y_pred)
```

Out[19]:
```
0.7541666666666667
```

Spojení predikce s původními daty

Predikci můžeme spojit s původními daty, a to například převodem dat na pandas tabulku.

In [20]:
```
data_test = pd.DataFrame(X_test, columns=X.columns)
data_test["quality_prediction"] = y_pred
data_test["quality_actual"] = y_test.reset_index(drop=True)
data_test.head()
```

Out[20]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	de
0	-0.355817	0.147673	-0.973624	-0.033846	0.557887	-0.187023	-0.029268	0.17
1	-0.297445	-0.182105	-0.513040	-0.664499	-0.121204	0.494668	1.666575	-0.42
2	1.395361	0.752267	-0.257160	0.106299	0.409335	0.105130	-0.392663	1.96
3	0.111164	-0.401957	0.203423	-0.209028	-0.206090	1.565897	0.334127	0.66
4	-0.939543	-0.401957	-0.154809	-0.594427	-0.227312	0.202514	-0.392663	-1.07

Pro každý řádek si můžeme určit, jestli byla hodnota predikována správně.

In [21]:
```
def compare_values(row):
    if row["quality_prediction"] == row["quality_actual"]:
        return "correct"
    else:
        return "incorrect"


data_test["result"] = data_test.apply(compare_values, axis=1)
data_test.head()
```

Out[21]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	de
0	-0.355817	0.147673	-0.973624	-0.033846	0.557887	-0.187023	-0.029268	0.17
1	-0.297445	-0.182105	-0.513040	-0.664499	-0.121204	0.494668	1.666575	-0.42
2	1.395361	0.752267	-0.257160	0.106299	0.409335	0.105130	-0.392663	1.96
3	0.111164	-0.401957	0.203423	-0.209028	-0.206090	1.565897	0.334127	0.66
4	-0.939543	-0.401957	-0.154809	-0.594427	-0.227312	0.202514	-0.392663	-1.07

Níže jsou vybraná vína, pro které predikce dopadla nesprávně.

In [22]:
```
data_test_incorrect = data_test[data_test["result"] == "incorrect"]
data_test_incorrect.head()
```

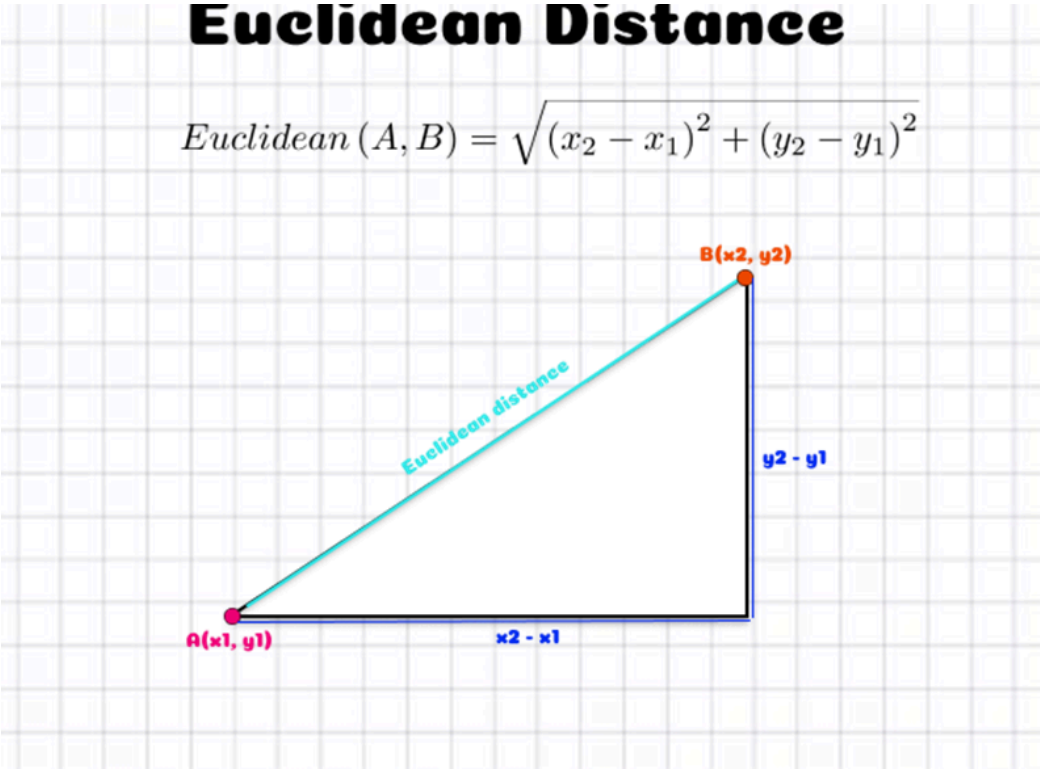
Out[22]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	d
0	-0.355817	0.147673	-0.973624	-0.033846	0.557887	-0.187023	-0.029268	0.1
8	-0.063954	-0.786699	0.561655	-0.734572	-0.630522	-0.576561	-0.544078	-0.3
11	-0.939543	1.246935	-1.280680	-0.524354	-0.206090	-0.966099	-1.058887	-0.4
12	-0.939543	-0.292031	-0.973624	-0.314137	-0.503192	0.202514	-0.392663	-0.6
14	-0.297445	-1.061514	0.510479	-0.384209	-0.121204	0.786821	0.334127	-0.2

Čtení na doma: Měření vzdálenosti

Jak je ale spočítána vzdálenost jednotlivých pozorování?

Ze školy si možná vzpomeneš na Euklidovskou vzdálenost, která počítá vzdálenost dvou bodů ve dvourozměrném prostoru.



Vzoreček, pomocí kterého je vzdálenost definována, je na obrázku. Tento vzoreček jde ale převést i do vícerozměrného prostoru. Napří pro třírozměrný prostor by byl:

$$Vzdálenost(A,B) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

Podobně můžeme přidat čtvrtý rozdměr, pátý rozměr atd. Tímto způsobem počítá vzdálenosti `scikit-learn` ve výchozím nastavení. Konkrétně používá Minkowskiho vzdálenost, která je trochu obecnější než Euklidovská. Místo druhé mocniny a druhé odmocniny umožňuje použít i jinou mocninu a odmocninu (např. třetí, čtvrtou atd.). Mocninu, kterou chceme použít, nastavíme pomocí parametru `p`. Pokud parametr nezadáme, je použita výchozí hodnota, což je 2.

Čtení na doma: Logistická regrese

Algoritmů na binární klasifikaci existuje velké množství. Dalším z nich je logistická regrese (*logistic regression*). Zatímco v lineární regresi jsme pracovali s vysvětlovanou proměnnou jako běžným číslem (neuvažovala jsme nějaké omezení rozsahu), logistická regrese predikuje pravděpodobnost, s jako pozorování patří do každých tříd.

Pro použití logistické regrese na naše data použijeme následující kód. Například níže vidíme, že pro první z testovacích vín model predikuje, že je dobré a pravděpodobností 35.2 % a že je špatné s pravděpodobností 64.9 %. U druhého vína je pravděpodobnost, že je dobré, pouze 15.5 % a špatné je s pravděpodobností 84.5 %.

In [23]:
```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression()
clf.fit(X_train, y_train)
y_pred = clf.predict_proba(X_test)
```

```
y_pred = clf.predict_proba(x_test)
y_pred
```

Out[23]:
```
array([[0.64871874, 0.35128126],
       [0.84527832, 0.15472168],
       [0.39259006, 0.60740994],
       [0.56960806, 0.43039194],
       [0.33390177, 0.66609823],
       [0.69003356, 0.30996644],
       [0.90894771, 0.09105229],
       [0.73750588, 0.26249412],
       [0.40535795, 0.59464205],
       [0.3052027 , 0.6947973 ],
       [0.17922099, 0.82077901],
       [0.62545334, 0.37454666],
       [0.47504301, 0.52495699],
       [0.69555898, 0.30444102],
       [0.60465993, 0.39534007],
       [0.05077332, 0.94922668],
       [0.79560857, 0.20439143],
       [0.52472594, 0.47527406],
       [0.05488478, 0.94511522],
       [0.72400306, 0.27599694],
       [0.53057489, 0.46942511],
       [0.75375473, 0.24624527],
       [0.15562035, 0.84437965],
       [0.06105802, 0.93894198],
       [0.56525403, 0.43474597],
       [0.53961399, 0.46038601],
       [0.08495893, 0.91504107],
       [0.58406597, 0.41593403],
       [0.80669681, 0.19330319],
       [0.12855047, 0.87144953],
       [0.74530976, 0.25469024],
       [0.52435414, 0.47564586],
       [0.37503665, 0.62496335],
       [0.68128233, 0.31871767],
       [0.48954955, 0.51045045],
       [0.88114178, 0.11885822],
       [0.12375297, 0.87624703],
       [0.30985635, 0.69014365],
       [0.52132652, 0.47867348],
       [0.12495663, 0.87504337],
       [0.66190721, 0.33809279],
       [0.77353549, 0.22646451],
       [0.17568482, 0.82431518],
       [0.85509053, 0.14490947],
       [0.34317228, 0.65682772],
       [0.26953795, 0.73046205],
       [0.06802205, 0.93197795],
       [0.57034946, 0.42965054],
       [0.80614612, 0.19385388],
       [0.42938215, 0.57061785],
       [0.85502233, 0.14497767],
       [0.77744685, 0.22255315],
       [0.37242345, 0.62757655],
       [0.07198402, 0.92801598],
       [0.86909983, 0.13090017],
       [0.91682696, 0.08317304],
       [0.16539111, 0.83460889],
       [0.67123651, 0.32876349],
       [0.2486585 , 0.7513415 ],
       [0.6546869 , 0.3453131 ],
       [0.46820044, 0.53179956],
       [0.22290427, 0.77709573],
```

```
[0.22209427, 0.77790573],
[0.71911824, 0.28088176],
[0.69520257, 0.30479743],
[0.04036   , 0.95964   ],
[0.62892827, 0.37107173],
[0.09106115, 0.90893885],
[0.60378919, 0.39621081],
[0.0614756 , 0.9385244 ],
[0.71183771, 0.28816229],
[0.04933377, 0.95066623],
[0.93428533, 0.06571467],
[0.37664042, 0.62335958],
[0.21972455, 0.78027545],
[0.10665905, 0.89334095],
[0.75068355, 0.24931645],
[0.01738589, 0.98261411],
[0.32373263, 0.67626737],
[0.16888153, 0.83111847],
[0.06026156, 0.93973844],
[0.56495318, 0.43504682],
[0.05815059, 0.94184941],
[0.53556545, 0.46443455],
[0.37855038, 0.62144962],
[0.32074268, 0.67925732],
[0.07353607, 0.92646393],
[0.74194277, 0.25805723],
[0.1975862 , 0.8024138 ],
[0.08460542, 0.91539458],
[0.65296391, 0.34703609],
[0.15118938, 0.84881062],
[0.59066017, 0.40933983],
[0.19317615, 0.80682385],
[0.2653379 , 0.7346621 ],
[0.8500647 , 0.1499353 ],
[0.40784334, 0.59215666],
[0.8186879 , 0.1813121 ],
[0.40073399, 0.59926601],
[0.76729013, 0.23270987],
[0.46191126, 0.53808874],
[0.75959783, 0.24040217],
[0.71589557, 0.28410443],
[0.30002901, 0.69997099],
[0.30589754, 0.69410246],
[0.5203971 , 0.4796029 ],
[0.1474708 , 0.8525292 ],
[0.47702912, 0.52297088],
[0.60163423, 0.39836577],
[0.16609276, 0.83390724],
[0.78039254, 0.21960746],
[0.01635522, 0.98364478],
[0.75743781, 0.24256219],
[0.10491081, 0.89508919],
[0.96838793, 0.03161207],
[0.29290776, 0.70709224],
[0.27014   , 0.72986   ],
[0.07730341, 0.92269659],
[0.50414499, 0.49585501],
[0.86683791, 0.13316209],
[0.22625758, 0.77374242],
[0.13498565, 0.86501435],
[0.65909557, 0.34090443],
[0.35078496, 0.64921504],
[0.64363823, 0.35636177],
[0.55529001, 0.44470999],
[0.68120906, 0.31879094],
```

```
[0.14516101, 0.85483899],
[0.3826224 , 0.6173776 ],
[0.47746641, 0.52253359],
[0.20374118, 0.79625882],
[0.37664042, 0.62335958],
[0.67770879, 0.32229121],
[0.83784903, 0.16215097],
[0.09268485, 0.90731515],
[0.42938215, 0.57061785],
[0.76012947, 0.23987053],
[0.81694657, 0.18305343],
[0.73940637, 0.26059363],
[0.75332702, 0.24667298],
[0.34325929, 0.65674071],
[0.20729506, 0.79270494],
[0.15695461, 0.84304539],
[0.11698962, 0.88301038],
[0.48018602, 0.51981398],
[0.35147071, 0.64852929],
[0.75277939, 0.24722061],
[0.11176   , 0.88824   ],
[0.54357448, 0.45642552],
[0.23050729, 0.76949271],
[0.88439198, 0.11560802],
[0.35977287, 0.64022713],
[0.11170622, 0.88829378],
[0.78543767, 0.21456233],
[0.32866809, 0.67133191],
[0.37664042, 0.62335958],
[0.16149755, 0.83850245],
[0.6253081 , 0.3746919 ],
[0.26333586, 0.73666414],
[0.53395237, 0.46604763],
[0.2346051 , 0.7653949 ],
[0.09589379, 0.90410621],
[0.34420535, 0.65579465],
[0.17359956, 0.82640044],
[0.89350422, 0.10649578],
[0.6381387 , 0.3618613 ],
[0.4451807 , 0.5548193 ],
[0.93975094, 0.06024906],
[0.74172974, 0.25827026],
[0.86462378, 0.13537622],
[0.89870515, 0.10129485],
[0.78532598, 0.21467402],
[0.75332709, 0.24667291],
[0.09519608, 0.90480392],
[0.60280356, 0.39719644],
[0.80161059, 0.19838941],
[0.35813902, 0.64186098],
[0.15802567, 0.84197433],
[0.70218866, 0.29781134],
[0.7031536 , 0.2968464 ],
```