pesikj Rok 2024 2 months ago

2857 lines (2857 loc) · 307 KB

Preview Code Blame Raw

Lekce 9

In [2]:

```
import pandas

from sklearn.svm import LinearSVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorize
from sklearn.metrics import ConfusionMatrixDisplay, accuracy_score
```

Popis importů

- `LinearSVC` je klasifikátor používající lineární verzi algoritmu Support Vector Machine a One-to-Rest postup pro klasifikaci do více tříd, dokumentace je zde
- `KNeighborsClassifier` - klasifikátor, používá algoritmus K Nearest Neigbors, dokumentace je zde
- `train_test_split` - funkce pro rozdělení dat na trénovací a testovací, dokumentace je zde
- `CountVectorizer` provádí konverzi textů na číselné pole, dokumentace je zde
- `TfidfVectorizer` provádí konverzi textů na číselné pole s využitím algoritmu TF-IDF, dokumentace je zde
- `ConfusionMatrixDisplay` - vizualizace matice záměn, dokumentace je zde
- `accuracy_score` - funkce pro vyhodnocení výsledků modelu, dokumentace je zde

Co to je NLP

Práce s přirozeným jazykem je odlišná od práce s jinými daty, a proto si zaslouží vlastní skupinu technik a algoritmů - NLP (Natural Language Processing). Ne všechny úlohy NLP se musí řešit pomocí strojového učení, ale v dnešní době tomu tak většinou je. Příklady úloh, které spadají do NLP jsou například:

- Strojový překlad (text v angličtině => text v češtině)
- Rozpoznávání řeči (řeč v češtině => text v češtině)
- Syntéza řeči (text v češtině > řeč v češtině)
- Klasifikace textu

Tyto úlohy určitě znáte i z běžného života. Další úlohy, které řeší NLP, a nacházejí se často "pod kapotou" jiných systémů, jsou například:

- Rozpoznávání pojmenovaných entit (NER, Named Entity Recognition): Pojmenované entity jsou výrazy, které označují jména, místa, data, názvy, ... Jaké pojmenované entity bychom mohli označit například ve větě *Král Karel si v Londýně včera zašel do kavárny Starbucks.* ?
- Určování slovních druhů (POS tagging, Part-of-speech tagging)

- Zjednodušení textu, shrnutí textu
- Určení významu slova (WSD, Word Sense Disambiguation): Při dvou významech slova *kolej* (studentské ubytování, železniční dráha), který z nich je zachycený větou *Včera proběhl na koleji večírek.* ?

Reprezentace textových dat

Naše datasety doposud obsahovaly proměnné, a datové body (pozorování) reprezentované pomocí těchto proměnných. Například u vzorku vody jsme dostali proměnné na základě chemického rozboru. U textových dat většinou dostaneme syrovější podobu dat, nikoliv proměnné. Jako kdybychom dostali vzorky vody, a sami museli provést chemickou analýzu.

Obecně řečeno, i v případě textu budeme jednotlivá pozorování nebo datové body reprezentovat pomocí číselných hodnot vstupních proměnných. Co ale budou tyto proměnné, označené v obrázku jako barvy, reprezentovat?

K reprezentování dat můžeme využít `CountVectorizer`. Ten vytvoří matici, kde každé slovo je reprezentováno jedním sloupcem a každý text ve vstupních datech jedním řádkem. Jeho fungování si nejprve ukážeme na jednoduchých datech, kde 4 uživatelé a uživatelky diskusního fóra vyjádřili své názory na jazyk Python. Celkem máme 4 diskusní příspěvky. Naším úkolem by bylo provést analýzu textu, která zjistí, kolik uživatelů (uživatelek) má k Pythonu kladný vztah a kolik záporný. Tento typ úloh je často označován jako analýza sentimentu (*sentiment analysis*).

Níže vidíme pole, které má 8 sloupců (v datech je totiž 8 unikátních slov) a 4 řádky (v datech jsou 4 řetězce). Pomocí metody `vec.get_feature_names_out()` si můžeme zobrazit popisky sloupců.

In [3]:
```
X = ["Python is great", "I like Python", "Python is the best language", "I
vec = CountVectorizer()
X = vec.fit_transform(X)
X = X.toarray()
X
```

Out[3]:
```
array([[0, 1, 0, 1, 0, 0, 1, 0],
       [0, 0, 0, 0, 0, 1, 1, 0],
       [1, 0, 0, 1, 1, 0, 1, 1],
       [0, 0, 1, 0, 0, 0, 1, 0]], dtype=int64)
```

In [4]:
```
vec.get_feature_names_out()
```

Out[4]:
```
array(['best', 'great', 'hate', 'is', 'language', 'like', 'python', 'the'],
      dtype=object)
```

Pro větší přehlednost si můžeme převést data do pandas tabulky. Vidíme například, že slovo `great` má hodnotu `1` pro řádek `0` (dokument na nulté pozici při počítání od 0). V datech vidíme, že slovo `great` se v něm skutečně vyskytuje. Ve všech ostatních příspěvcích se toto slovo nevyskytuje, proto mají ostatní řádky v tom sloupci hodnotu `0`.

In [5]:
```
df = pandas.DataFrame(X, columns=vec.get_feature_names_out())
df
```

Out[5]:

	best	great	hate	is	language	like	python	the
0	0	1	0	1	0	0	1	0
1	0	0	0	0	0	1	1	0
2	1	0	0	1	1	0	1	1
3	0	0	1	0	0	0	1	0

Pokud se na slova podíváme, dokázali bychom je rozdělit na několik skupin:

- slova, která naznačují kladný vztah (best, great, v tomto případě i slovo like, ale u něj to tak nemusí být vždy),
- slova, která naznačují negativní vztah (hate),
- slova, která nenaznačují ani jedno (is, language, python, the).

Obecně bychom pak mohli říci, že vysoký počet "pozitivní slov" vede spíše k tomu, že celý komentář je zamýšlen pozitivně, a vysoký počet "negativních slov" vede k tomu, že celý komentář je negativní. Tento postup určitě není dokonalý (např. nerozpozná sarkasmus), ale umožní nám využít algoritmy, které už známe.

Uvažujme dále, že v jazyce je obrovské množství slov a my je nechceme ručně třídit. Na internetu ale můžeme najít datové sady, které obsahují nějaký text a k tomu označení, zda je text celkově pozitivní nebo negativní. Například u recenzí často lidé vyplňují textový komentář i hodnocení na nějaké škále (např. 1 až 5 hvězd). Můžeme pak použít supervised learning (učení s učitelem) a "ohodnotit" jednotlivá slova (nebo skupiny slov) jako pozitivní nebo negativní.

Analýza sentimentu ale není jediná úloha, která funguje na tomto principu. Obecně můžeme třídit dokumenty do skupin například v systémech uživatelské podpory (stěžuje si uživatel na problémy s internetem, chce levnější tarif, chce si aktivovat roaming?), při třídění článků do rubrik, třídění zboží do kategorií atd.

Pojďme si nyní načíst dataset, se kterým budeme dneska pracovat. Jedná se o databázi popisů filmů ze serveru IMDB. K dispozici máme název filmu, jeho žánr (to bude naše výstupní proměnná), a text popisku filmu. Text popisku budeme chtít převést na naše vstupní proměnné. Pokud bychom dobře ohodnotili jednotlivé slova, můžeme pak podle jejich počtu v dokumentu odhadnout, jaké je celkové vyznění zprávy.

Naším úkolem bude odhadnout žánr filmu, ke kterému popis patří. V datech máme název filmu, žánr a textový popis. Naším úkolem bude vytvořit model, který dokáže zařadit film do žánru, i když oficiální informaci od tvůrců filmu nemáme.

In [6]:
```
data = pandas.read_csv("movies.csv")
data.head()
```

Out[6]:

	title	genre	text
0	Santa Barbara (2014)	romance	Because of his close friend's betrayal, music ...
1	Behen Hogi Teri (2017)	comedy	Gattu (Rajkummar Rao) has a problem, he loves ...
2	The Last Coffin (2003)	horror	In the movie, a strange crossword puzzle start...
3	Lady Krampus (2016)	horror	It's Christmas time in Cleveland, Ohio and fou...
4	"Pandemic" (2016)	thriller	A deadly virus is unleashed on the fictional n...

Rozdělíme data na vstupní a výstupní proměnnou a poté na trénovací a testovací data.

In [7]:

```
X = data["text"]
y = data["genre"]

X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, rando
```

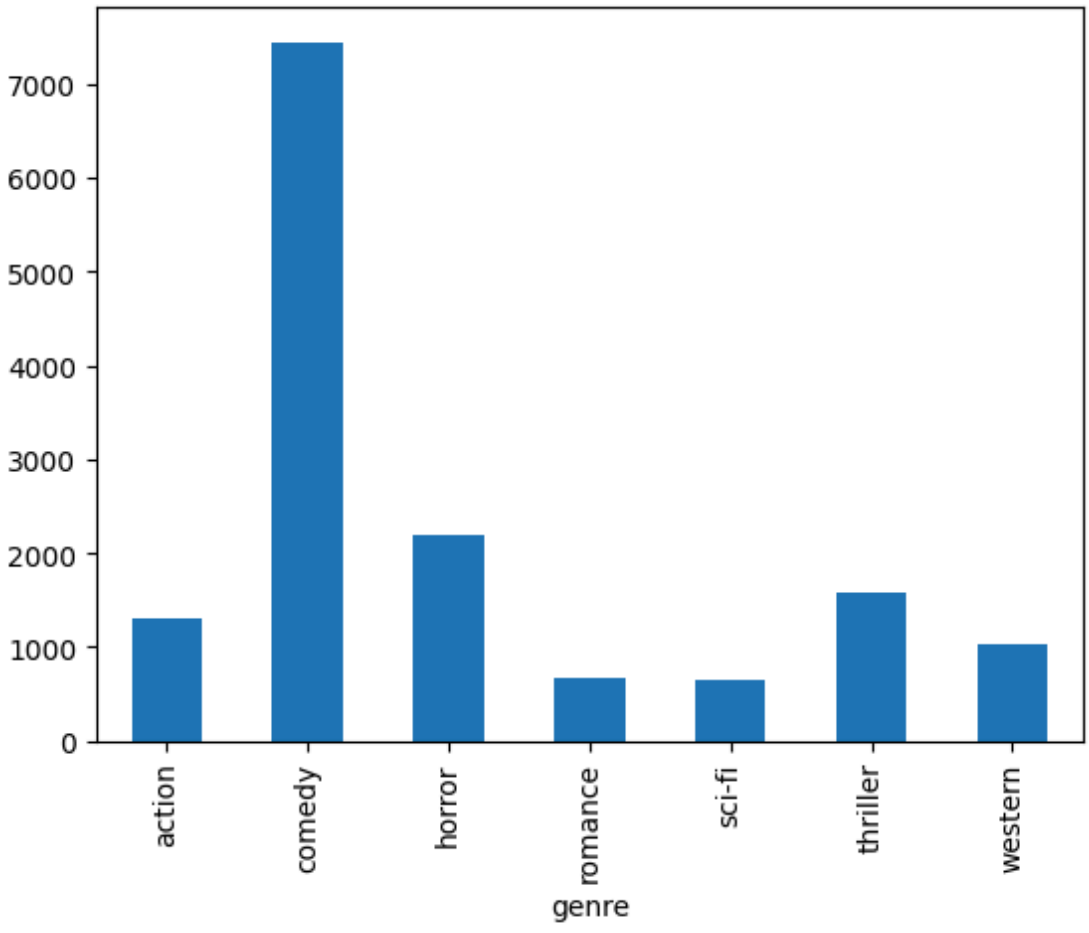
Musíme myslet na to, že rozložení jednotlivých skupin je velmi nerovnoměrné, například komedií je mnohem více než sci-fi filmů. Parametry strafify nám zařizuje, že funkce `train_test_split` zachová poměr jednotlivých skupin v testovacích i trénovacích datech stejný.

In [8]:

```
data.groupby("genre").size().plot(kind="bar")
```

Out[8]:

```
<Axes: xlabel='genre'>
```



In [ ]:
```
sns
```

Jako první formu reprezentace popisků vyzkoušíme jednotlivá slova a jejich počty. V podstatě vytvoříme slovní zásobu, která bude obsahovat všechna slova, co se v našich trénovacích datech objeví. Jedno slovo bude jedna proměnná a hodnota proměnné bude počet, kolikrát se slovo v dokumentu (zde popisku filmu) objeví. Opět použijeme `CountVectorizer` .

In [9]:
```
vec = CountVectorizer()
X_train = vec.fit_transform(X_train)
X_test = vec.transform(X_test)
```

Jak teď naše data vypadají?

In [10]:
```
X_train.toarray()
```

Out[10]:
```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 1, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], dtype=int64)
```

Podívejme se na rozměry tabulky.

In [11]:
```
X_train.toarray().shape
```

Out[11]:
```
(11181, 49445)
```

In [12]:
```
vec.get_feature_names_out()
```

Out[12]:
```
array(['00', '000', '003', ..., 'ťthe', 'źs', 'žš'], dtype=object)
```

Vidíme, že data mají cca 49 tisíc unikátních slov. Bylo by pro nás jako pro lidi opravdu příliš pracné tato slova ručně projít a rozdělit na skupiny, jako jsme to provedli u imaginárních dat s hodnoceními jazyka Python. Můžeme ale využít některý z algoritmů supervised learning. Ten zařadí film do žánru v závislosti na tom, do jakých žánrů patří filmy s popisy, které obsahuje podobná slova.

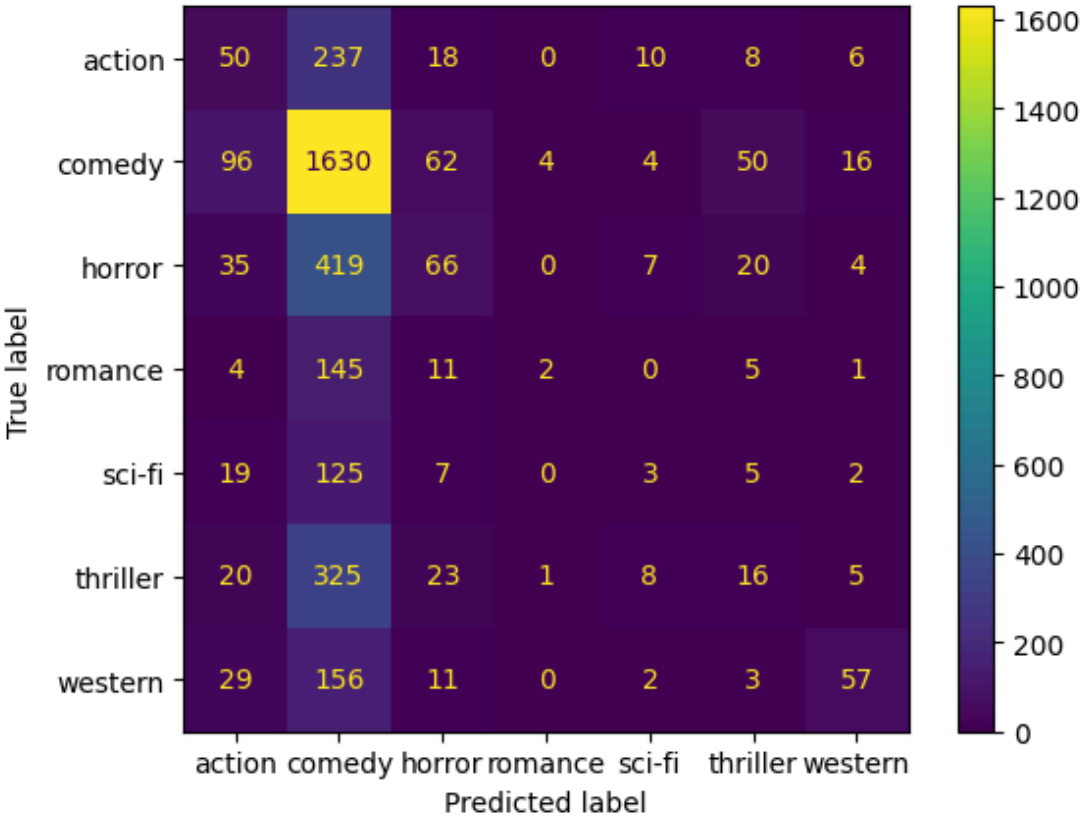
Pojďme tedy zkusit tyto vstupní proměnné předat klasifikačnímu algoritmu K Nearest Neighbours. Uvažujme například 5 nejbližších sousedů, tj. výchozí hodnotu.

In [13]:
```
clf = KNeighborsClassifier()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

In [14]:
```
ConfusionMatrixDisplay.from_estimator(
    clf,
    X_test
```

```
    y_test,
)
```

Out[14]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1ed440cb830>

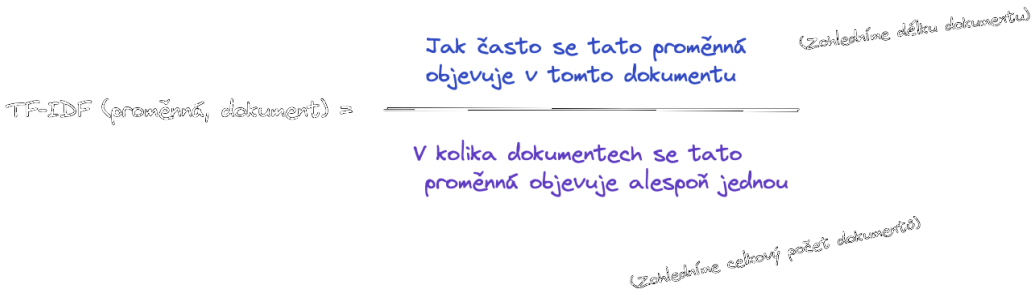


In [15]:
```
accuracy_score(y_test, y_pred)
```

Out[15]: 0.48940166353635634

Jak můžeme úspěšnost modelu zlepšit? V podstatě jsou dvě úrovně: Úroveň dat, a úroveň klasifikačního algoritmu. Pojďme začít u dat, protože když nebudeme mít čistá data, žádný algoritmus nás nezachrání (toto se dá také shrnout pořekadlem "garbage in, garbage out").

- Všimněme si, že nejčastější slova jsou taková, která se nacházejí skoro ve všech popiskách. Těmto častým slovům, která nenesou žádný význam, se říka *stop words*. Každý jazyk má svůj blacklist těchto slov, která se většinou z dat úplně vyfiltrují. `CountVectorizer` má parametr `stop_words`, který můžeme nastavit na hodnotu `"english"`.
- Dále se zamysleme nad tím, co znamenají hodnoty našich vstupních proměnných. Možná by bylo lepší tyto hodnoty nějak normalizovat (co kdybychom například měli veliké rozdíly mezi délkou vstupních dokumentů?). Populární metoda pro normalizaci četností slov je TF-IDF (Term Frequency Inverse Document Frequency). Tato normalizace zohledňuje jak četnost slova v dokumentu, tak i to, jak často se objevuje vůbec v celých vstupních datech. Takže slovo, které by se velmi často objevovalo jen v několika dokumentech, by mělo větší váhu, než jiné slovo, které se objevuje v mnoha dokumentech.



Můžeme tedy `CountVectorizer` vyměnit za `TfidfVectorizer` . Jak moc je teď důležitý parametr `stop_words` ?

In [16]:
```
X = ["Python is great", "I like Python", "Python is the best language", "I
vec = TfidfVectorizer()
X = vec.fit_transform(X)
X = X.toarray()
X
```

Out[16]:
```
array([[0.        , 0.72664149, 0.        , 0.5728925 , 0.        ,
        0.        , 0.37919167, 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.88654763, 0.46263733, 0.        ],
       [0.50676543, 0.        , 0.        , 0.39953968, 0.50676543,
        0.        , 0.26445122, 0.50676543],
       [0.        , 0.        , 0.88654763, 0.        , 0.        ,
        0.        , 0.46263733, 0.        ]])
```

In [17]:
```
vec.get_feature_names_out()
```

Out[17]:
```
array(['best', 'great', 'hate', 'is', 'language', 'like', 'python', 'the'],
      dtype=object)
```

In [18]:
```
df = pandas.DataFrame(X, columns=vec.get_feature_names_out())
df
```

Out[18]:

	best	great	hate	is	language	like	python	the
0	0.000000	0.726641	0.000000	0.572892	0.000000	0.000000	0.379192	0.000000
1	0.000000	0.000000	0.000000	0.000000	0.000000	0.886548	0.462637	0.000000
2	0.506765	0.000000	0.000000	0.399540	0.506765	0.000000	0.264451	0.506765
3	0.000000	0.000000	0.886548	0.000000	0.000000	0.000000	0.462637	0.000000

Nyní zkusíme přidat parametr `stop_words` . Která slova nám z dat zmizela?

In [19]:
```
X = ["Python is great", "I like Python", "Python is the best language", "I
vec = TfidfVectorizer(stop_words="english")
X = vec.fit_transform(X)
X = X.toarray()
X
```

Out[19]:
```
array([[0.        , 0.88654763, 0.        , 0.        , 0.        ,
```

```
 0.46263733],
[0.        , 0.        , 0.        , 0.        , 0.88654763,
 0.46263733],
[0.66338461, 0.        , 0.        , 0.66338461, 0.        ,
 0.34618161],
[0.        , 0.        , 0.88654763, 0.        , 0.        ,
 0.46263733]])
```

In [20]:
```
vec.get_feature_names_out()
```

Out[20]:
```
array(['best', 'great', 'hate', 'language', 'like', 'python'],
      dtype=object)
```

In [21]:
```
df = pandas.DataFrame(X, columns=vec.get_feature_names_out())
df
```

Out[21]:

	best	great	hate	language	like	python
0	0.000000	0.886548	0.000000	0.000000	0.000000	0.462637
1	0.000000	0.000000	0.000000	0.000000	0.886548	0.462637
2	0.663385	0.000000	0.000000	0.663385	0.000000	0.346182
3	0.000000	0.000000	0.886548	0.000000	0.000000	0.462637

Čtení na doma - postup výpočtu

Podívejme se na způsob výpočtu. Vraťme se k tabulce, kterou vygeneruje `CountVectorizer`.

In [22]:
```
X = ["Python is great", "I like Python", "Python is the best language", "I

vec = CountVectorizer(stop_words="english")
X = vec.fit_transform(X)
X = X.toarray()
X
```

Out[22]:
```
array([[0, 1, 0, 0, 0, 1],
       [0, 0, 0, 0, 1, 1],
       [1, 0, 0, 1, 0, 1],
       [0, 0, 1, 0, 0, 1]], dtype=int64)
```

Nyní porovnejme matici s tím, co vygeneruje `TfidfVectorizer`. Dále přibyl parametr `smooth_idf`, jehož funkci si ještě objasníme.

In [23]:
```
X = ["Python is great", "I like Python", "Python is the best language", "I

vec = TfidfVectorizer(stop_words="english", smooth_idf=False)
X = vec.fit_transform(X)
X = X.toarray()
X
```

Out[23]:
```
array([[0.        , 0.9222914 , 0.        , 0.        , 0.        ,
        0.38649524],
```

```
       [0.        , 0.        , 0.        , 0.        , 0.9222914 ,
        0.38649524],
       [0.67796827, 0.        , 0.        , 0.67796827, 0.        ,
        0.28410924],
       [0.        , 0.        , 0.9222914 , 0.        , 0.        ,
        0.38649524]])
```

In [24]:
```
vec.get_feature_names_out()
```

Out[24]:
```
array(['best', 'great', 'hate', 'language', 'like', 'python'],
      dtype=object)
```

Pro přehlednost si zobrazíme výsledek jako pandas tabulku (`DataFrame`).

In [25]:
```
df = pandas.DataFrame(X, columns=vec.get_feature_names_out())
df
```

Out[25]:

	best	great	hate	language	like	python
0	0.000000	0.922291	0.000000	0.000000	0.000000	0.386495
1	0.000000	0.000000	0.000000	0.000000	0.922291	0.386495
2	0.677968	0.000000	0.000000	0.677968	0.000000	0.284109
3	0.000000	0.000000	0.922291	0.000000	0.000000	0.386495

Vypočítejme si nyní sami hodnoty v tabulce pro první řádek, abychom pochopili, jak `TfidfVectorizer` funguje. Začneme s výrazem great. Postupujeme podle vzorce.

$$\mathrm{idf}(t) = \log \frac{n}{1 + \mathrm{df}(t)}$$

kde df je počlet dokumentů, které obsahují výraz t, n je celkový počet dokumentů a výsledkem je idf (inverse document-frequency).

In [26]:
```
import numpy

n = 4
df_great = 1
idf_great = numpy.log(n / df_great) + 1
idf_great
```

Out[26]:
```
2.386294361119891
```

Dále dopočítáme hodnotu ukazatel tf-idf s využitím vzorce

$$\text{tf-idf}(t, d) = \mathrm{tf}(t, d) * \mathrm{tf}$$

kde $\mathrm{tf}(t, d)$ znamená term-frequency, tj. počet výskytů výrazu t, který vidíme v tabulce, kterou nám vygeneroval `CountVectorizer` .

In [27]:
```
tf_idf_great = 1 * idf_great
tf_idf_great
```

Out[27]:
```
2.386294361119891
```

Nyní postupujeme stejně pro výraz Python. Ten se objevuje ve všech 4 dokumentech.

In [28]:
```
df_python = 4
idf_python = numpy.log(n / df_python) + 1
idf_python
```

Out[28]:
```
1.0
```

In [29]:
```
tf_idf_python = 1 * idf_python
tf_idf_python
```

Out[29]:
```
1.0
```

Nakonec provedeme normalizaci, kterou získáme jako podíl tf-idf pro dané slovo a odmocninu druhých mocnin tf-idf pro všechna slova. Níže vidíme, že výsledek odpovídá hodnotě pro výraz great v dokumentu 0.

In [30]:
```
tf_idf_great / numpy.sqrt(tf_idf_great ** 2 + tf_idf_python ** 2)
```

Out[30]:
```
0.92229140312174
```

Dále si vyzkoušíme variantu bez parametru `smooth_idf` .

In [31]:
```
X = ["Python is great", "I like Python", "Python is the best language", "I

vec = TfidfVectorizer(stop_words="english")
X = vec.fit_transform(X)
X = X.toarray()
X
```

Out[31]:
```
array([[0.        , 0.88654763, 0.        , 0.        , 0.        ,
        0.46263733],
       [0.        , 0.        , 0.        , 0.        , 0.88654763,
        0.46263733],
       [0.66338461, 0.        , 0.        , 0.66338461, 0.        ,
        0.34618161],
       [0.        , 0.        , 0.88654763, 0.        , 0.        ,
        0.46263733]])
```

In [32]:
```
df = pandas.DataFrame(X, columns=vec.get_feature_names_out())
df
```

Out[32]:

	best	great	hate	language	like	python
0	0.000000	0.886548	0.000000	0.000000	0.000000	0.462637
1	0.000000	0.000000	0.000000	0.000000	0.886548	0.462637
2	0.663385	0.000000	0.000000	0.663385	0.000000	0.346182
3	0.000000	0.000000	0.886548	0.000000	0.000000	0.462637

Rozdíl je v tom, že idf počítáme z mírně upraveného vzorce

$$\mathrm{idf}(t) = \log \frac{1+n}{1+\mathrm{df}(t)} + 1$$

In [33]:
```
n = 4
df_great = 1
idf_great = numpy.log((1 + n) / (1 + df_great)) + 1
tf_idf_great = 1 * idf_great
df_python = 4
idf_python = numpy.log((1 + n) / (1 + df_python)) + 1
tf_idf_python = 1 * idf_python
tf_idf_great / numpy.sqrt(tf_idf_great ** 2 + tf_idf_python ** 2)
```

Out[33]:
```
0.8865476297873808
```

Vlastní seznam stop words

Pokud nám nevyhovuje výchozí seznam stop words pro daný jazyk, můžeme si vytvořit svůj. To se může hodit pro češtinu nebo třeba v situaci, že se dané slovo vyskytuje často (např. v příspěvcích o Pythonu se bude často opakovat slovo Python).

In [34]:
```
X = [
    "Python je nejlepší",
    "Mám rád Python!",
    "Python je nejlepší jazyk",
    "Nesnáším Python!"
    ]
vec = TfidfVectorizer(stop_words=["je", "mám", "Python"])
X = vec.fit_transform(X)
X = X.toarray()
X
```

```
c:\Users\jiri.pesik.HULD\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\feature_extraction\text.py:408: UserWarning: Your stop_words may be inconsistent with your preprocessing. Tokenizing the stop words generated tokens ['python'] not in stop_words.
  warnings.warn(
```

Out[34]:
```
array([[0.        , 0.83388421, 0.        , 0.55193942, 0.        ],
       [0.        , 0.        , 0.        , 0.46263733, 0.88654763],
       [0.72664149, 0.5728925 , 0.        , 0.37919167, 0.        ],
       [0.        , 0.        , 0.88654763, 0.46263733, 0.        ]])
```

In [35]:
```
vec.get_feature_names_out()
```

Out[35]:
```
array(['jazyk', 'nejlepší', 'nesnáším', 'python', 'rád'], dtype=object)
```

In [36]:
```
df = pandas.DataFrame(X, columns=vec.get_feature_names_out())
df
```

Out[36]:

	jazyk	nejlepší	nesnáším	python	rád
0	0.000000	0.833884	0.000000	0.551939	0.000000
1	0.000000	0.000000	0.000000	0.462637	0.886548

2	0.726641	0.572892	0.000000	0.379192	0.000000
3	0.000000	0.000000	0.886548	0.462637	0.000000

Klasifikace dokumentů

Vyzoušejme nyní klasifikaci dokumentů (resp. filmů, o kterých dané dokumenty pojednávají). Využít můžeme například algoritmus `KNeighborsClassifier`.

In [37]:
```
X = data["text"]
y = data["genre"]

X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, rando

vec = TfidfVectorizer()
X_train = vec.fit_transform(X_train)
X_test = vec.transform(X_test)

clf = KNeighborsClassifier()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

accuracy_score(y_test, y_pred)
```

Out[37]: `0.5709686074590824`

Porovnáme accuracy s výsledkem stejného algoritmu při využití parametru `stop_words`.

In [38]:
```
X = data["text"]
y = data["genre"]

X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, rando

# Uvažujeme stop_words v angličtině
vec = TfidfVectorizer(stop_words="english")
X_train = vec.fit_transform(X_train)
X_test = vec.transform(X_test)

clf = KNeighborsClassifier()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

accuracy_score(y_test, y_pred)
```

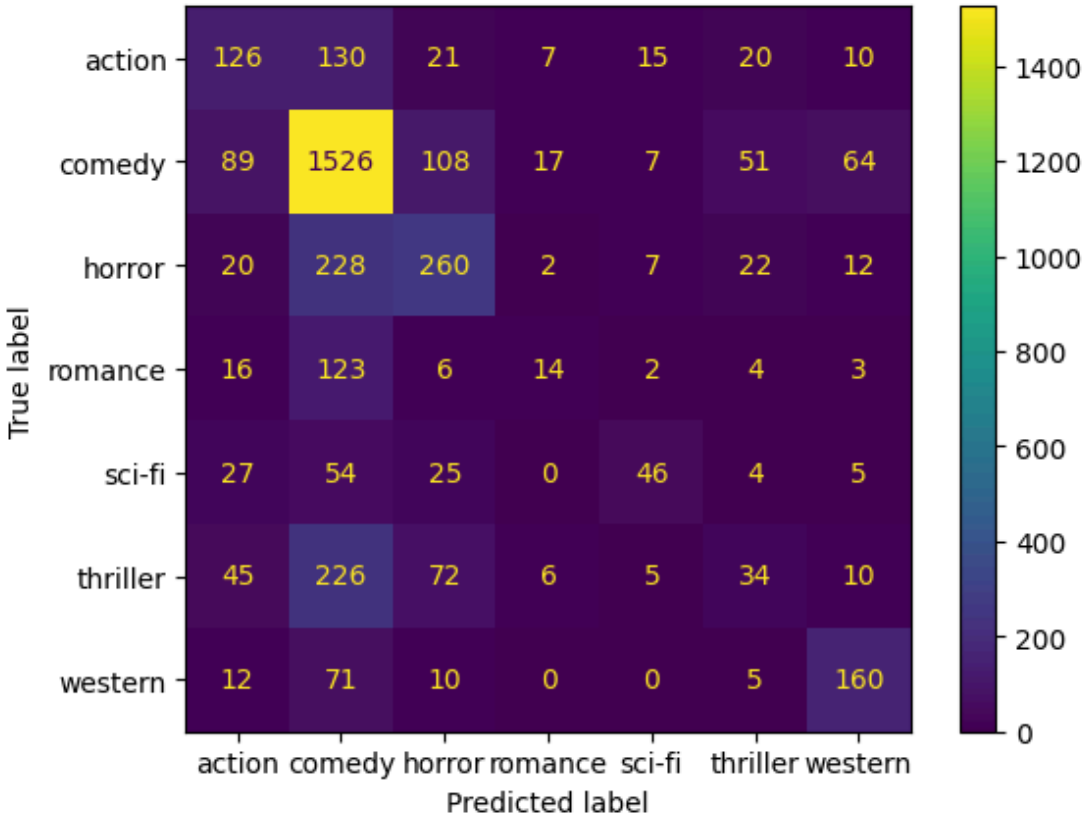
Out[38]: `0.5811644754494232`

Níže si zobrazíme matici záměn. Vidíme například, že náš model označil 228 hororů za komedii a naopak 108 komedií za horor.

In [39]:
```
ConfusionMatrixDisplay.from_estimator(
    clf,
    X_test,
```

```
    y_test,
)
```

Out[39]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1ed222e6c00>



Zkusme nyní přidat i dvojice slov. K tomu slouží parametr `ngram_range` .

In [40]:
```
X = data["text"]
y = data["genre"]

X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, rando

# Uvažujeme stop words v angličtině, samostatná slova a dvojice slov
vec = TfidfVectorizer(stop_words="english", ngram_range=(1, 2))
X_train = vec.fit_transform(X_train)
X_test = vec.transform(X_test)

clf = KNeighborsClassifier()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

accuracy_score(y_test, y_pred)
```

Out[40]: 0.594311778910652

Dále můžeme využít i lineární verzi algoritmu Support Vector Machine. Ten očividně dosahuje výrazně lepších výsledků.

Protože je počet filmů v různých žánrech různý, jedná se o další nevyvážený (*unballanced*) dataset. Aby to algoritmus SVM zohlednil, přidáme parametr `class_weight=balanced` .

In [41]:
```
X = data["text"]
y = data["genre"]

X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, rando

vec = TfidfVectorizer(stop_words="english", ngram_range=(1, 2))
X_train = vec.fit_transform(X_train)
X_test = vec.transform(X_test)

clf = LinearSVC(class_weight="balanced")
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

accuracy_score(y_test, y_pred)
```

```
c:\Users\jiri.pesik.HULD\AppData\Local\Programs\Python\Python312\Lib\site-pa
ckages\sklearn\svm\_classes.py:32: FutureWarning: The default value of `dual
` will change from `True` to `'auto'` in 1.5. Set the value of `dual` explic
itly to suppress the warning.
  warnings.warn(
```

Out[41]:
```
0.77247115642608
```

Zjistit můžeme seznam skupin, na které algoritmus třídí popisy (to jsou žárny filmů, které uvažujeme).

In [42]:
```
clf.classes_
```

Out[42]:
```
array(['action', 'comedy', 'horror', 'romance', 'sci-fi', 'thriller',
       'western'], dtype=object)
```

Pro každé slovo a každý žánr vygeneruje algoritmus SVM koeficient. Čím je koeficient vyšší, patří spíše koeficient k danému žánru.

In [43]:
```
df_coef = pandas.DataFrame(clf.coef_.T, columns=clf.classes_)
df_coef["words"] = vec.get_feature_names_out()
df_coef = df_coef.set_index("words")
df_coef
```

Out[43]:

words	action	comedy	horror	romance	sci-fi	thriller
00	-1.110178e-01	0.077759	0.119142	-1.969639e-02	-0.032717	-0.078708
00 31	-1.342387e-02	-0.092837	0.138789	2.168404e-19	-0.017770	0.000000
00 bonus	-2.052927e-03	0.012859	0.000000	-4.859085e-03	0.000000	-0.009856
00 breast	-2.052927e-03	0.012859	0.000000	-4.859085e-03	0.000000	-0.009856
00 cameraman	3.252607e-19	-0.073188	0.120041	0.000000e+00	-0.004287	-0.020664
...	...	...	...	...	...	...

źs	-4.337334e-02	0.109317	-0.055493	-6.234519e-03	-0.022747	-0.035536
źs 70	-2.168667e-02	0.054658	-0.027746	-3.117259e-03	-0.011374	-0.017768
źs sit	-2.168667e-02	0.054658	-0.027746	-3.117259e-03	-0.011374	-0.017768
žš	-1.295680e-02	-0.005874	-0.009363	-1.772791e-02	0.062392	-0.003743
žš brand	-1.295680e-02	-0.005874	-0.009363	-1.772791e-02	0.062392	-0.003743

521095 rows × 7 columns

Můžeme si to vyzkoušet na žánru sci-fi. Níže jsou slova, která jsou typická pro sci-fi (mají nejvyšší hodnoty koeficientů).

In [44]:
```
df_coef.sort_values("sci-fi", ascending=False).head(20)
```

Out[44]:

words	action	comedy	horror	romance	sci-fi	thriller	west
earth	-0.305544	-1.049080	-0.795856	-0.346325	3.565868	-0.865330	-0.390
alien	-0.625959	-0.950723	-0.753304	-0.100391	3.075054	-0.355362	-0.127
space	-0.849306	-0.839350	-0.310389	-0.170092	2.801740	-0.421741	-0.213
planet	0.026377	-0.833928	-0.553625	-0.235634	2.684631	-0.671789	-0.303
future	0.071173	-0.851809	-0.677676	0.229715	2.285674	-0.321754	-0.362
scientist	-0.235538	-0.749948	0.183006	-0.130906	1.858985	-0.329763	-0.172
humans	0.003954	-0.586366	-0.460290	-0.212928	1.786474	-0.135461	-0.108
universe	0.087262	-0.551388	-0.237815	-0.083737	1.657259	-0.496430	-0.156
science	-0.358140	-0.669996	0.076131	-0.024774	1.628298	-0.458513	0.010
government	0.398769	-0.931299	-0.350972	-0.436382	1.624148	-0.066905	0.118
fi	-0.272165	-0.646416	-0.135908	0.023320	1.616683	-0.316247	-0.111
sci	-0.247751	-0.609926	-0.128806	0.008115	1.543129	-0.299955	-0.123
humanity	-0.193607	-0.492137	0.089497	-0.160775	1.531281	-0.607235	0.025
human	-0.729257	-0.997938	0.868838	-0.004849	1.460665	-0.223509	-0.395
sci fi	-0.247449	-0.593126	-0.110345	0.026633	1.438116	-0.300091	-0.105
scientists	0.029895	-0.607623	-0.208912	0.000861	1.372411	-0.152152	-0.127
ship	-0.464275	-0.306491	-0.015037	0.120950	1.363643	-0.522264	-0.211
world	0.388411	-0.311812	-0.625560	-0.406121	1.313124	0.449006	-1.033
time travel	-0.256850	-0.364096	-0.162699	-0.150368	1.291275	-0.145794	-0.081

abilities	0.052012	-0.674116	-0.084266	-0.033214	1.244575	-0.043383	-0.126

Níže jsou slova, která která jsou typická pro jiné žárny než pro sci-fi (mají nejnižší hodnoty koeficientů).

In [45]:
```
df_coef.sort_values("sci-fi", ascending=False).tail(20)
```

Out[45]:

words	action	comedy	horror	romance	sci-fi	thriller	w
face	0.264577	-0.601017	3.105767e-01	0.221542	-0.456564	0.288235	-0.(
ghost	-0.458955	-0.924409	1.833619e+00	-0.342129	-0.460815	0.282566	-0.1
better	0.015046	0.827586	3.884094e-02	-0.322129	-0.461761	-0.431746	-0.1
brother	0.374421	-0.814076	1.517002e-01	0.113788	-0.471054	0.119607	0.1
gets	-0.148712	0.459813	-8.462203e-01	0.131542	-0.471515	0.356660	0.2
house	-1.184854	-0.571967	2.105994e+00	-0.386934	-0.480518	0.132942	-0.8
orbot	0.415281	0.000000	3.035766e-18	0.000000	-0.490230	0.000000	0.(
girlfriend	-0.286361	0.799962	4.686213e-02	-0.139815	-0.510474	0.034139	-0.4
decide	-0.189591	0.677621	2.287177e-01	-0.036119	-0.525864	-0.450979	-0.2
town	-0.395280	-0.778853	5.478209e-01	0.000069	-0.531709	-0.411699	1.5
gang	0.265154	-1.086324	-1.271547e+00	-0.295240	-0.578087	-0.719295	2.1
blood	0.080106	-1.657251	2.945607e+00	-0.331755	-0.594742	-0.309754	-0.3
killer	-0.471433	-2.237926	1.724066e+00	-0.409638	-0.630543	1.285854	0.2
money	-0.209301	1.132279	-1.142158e+00	-0.426854	-0.636594	-0.074310	0.5
daughter	-0.086703	-0.087383	-8.185474e-02	0.094638	-0.639733	0.339277	0.4
horror	-0.765436	-2.542597	5.083340e+00	-0.534896	-0.652962	-1.009266	-0.3
night	-0.738198	-0.893714	2.122809e+00	-0.051495	-0.657614	0.060568	-0.1
zombies	-0.563926	-0.766787	2.237238e+00	-0.123861	-0.687748	-0.351070	-0.2
love	-0.681247	-0.642391	-1.523869e+00	4.229292	-0.697383	-0.964504	-0.5
comedy	-1.749761	5.994937	-1.842959e+00	-1.132088	-1.157547	-1.983850	-0.8

Čtení na doma: Seznam stop words pro češtinu

I pro ostatní jazyky existují připravené seznamy stop words, pouze nejsou součástí modulu `scikit-learn`. Existuje například modul `stop-words`, který obsahuje stop words pro řadu jazyků (kromě češtiny třeba Ukrajinštinu, němčinu nebo katalánštinu). Modul je třeba nainstalovat.