

Podatkovno rudarjenje: zbirka vaj in nalog

Martin Stražar, Rok Gomišček, Tomaž Curk

29. februar 2020

Kazalo

Delovno okolje	9
Namestitev	9
Conda	9
Virtualenv	10
Docker	10
Git in GitHub	10
Nekaj osnovnih ukazov	10
Namestitev	10
1 Priprava podatkov	11
1.1 Knjižnica <code>numpy</code>	11
1.1.1 Pretvorba seznamov v večdimenzionalna polja	11
Vprašanje 1-1-1	12
1.1.2 Razlike med seznamami in polji	12
1.1.3 Uporaba polj	13
1.1.3.1 Naslavljanje	13
1.1.3.2 Rezanje	14
1.1.3.3 Naslavljanje polja s pomočjo druge strukture	15
Vprašanje 1-1-2	16
1.1.3.4 Funkcije za ustvarjanje polj	16
1.1.4 Osnovne računske operacije	17
1.1.4.1 Operacije polja s skalarjem	18
1.1.4.2 Operacije polje-polje (po elementih)	18
1.1.5 Iteracija po elementih polja	19
1.2 Primer: statistika temperatur v Stockholmu	20
Vprašanje 1-2-1	21
1.2.1 Procesiranje podatkov	21
1.2.1.1 Povprečje, aritmetična sredina	21
Vprašanje 1-2-2	21
1.2.1.2 Standardni odklon in varianca	21
Vprašanje 1-2-3	21
1.2.1.3 Najmanjša in največja vrednost	21
Vprašanje 1-2-4	22
1.2.1.4 Vsota, produkt	22
1.2.2 Globalno segrevanje?	22
Vprašanje 1-2-5	22
Vprašanje 1-2-6	25
2 Prikazovanje podatkov	27
2.1 Knjižnica <code>matplotlib</code>	27
Vprašanje 2-1-1	29
2.1.1 Objektno usmerjen način dela z <code>matplotlib</code>	30

2.1.2	Barvanje	32
2.1.3	Stili	33
2.1.4	Vizualizacije različnih tipov podatkov	34
2.1.5	Verjetnostne porazdelitve	35
	Vprašanje 2-1-2	36
	Vprašanje 2-1-3	36
2.1.6	Toplotne karte in konture	36
	2.1.6.1 Funkcija <code>pcolor</code>	36
	2.1.6.2 Funkcija <code>imshow</code>	37
	2.1.6.3 Funkcija <code>contour</code>	38
2.1.7	Nadzor nad velikostjo osi	38
	2.1.7.1 Domet	38
	2.1.7.2 Logaritemska lestvica	39
2.1.8	Nastavitev oznak na oseh	40
2.1.9	Velikost, razmerje in ločljivost	41
2.1.10	Legenda, oznake in naslovi	41
2.1.11	Shranjevanje slike	42
2.2	Primer: zimske olimpijske igre, Soči 2014	42
2.2.1	Predstavitve podatkov	44
	Vprašanje 2-2-1	44
	Vprašanje 2-2-2	44
2.2.2	Programski paket <code>Orange</code>	44
2.2.3	Izbira podmnožice vrstic	46
2.2.4	Prikaz točk v prostoru	46
	Vprašanje 2-2-3	47
2.2.5	Prikaz porazdelitev	47
	Vprašanje 2-2-4	48
2.2.6	Nagrade za dosego najvišjih mest	48
2.2.7	Spol udeležencev	49
	Vprašanje 2-2-5	50
	Vprašanje 2-2-6	50
2.2.8	Najuspešnejše države	51
	Vprašanje 2-2-7	51
2.2.9	Sestavljene vizualizacije	51
	Vprašanje 2-2-8	52
	Vprašanje 2-2-9	52
3	Porazdelitve in osamelci	53
3.1	Gaussova (normalna) porazdelitev	54
3.1.1	Učenje parametrov	56
	Vprašanje 3-1-1	57
3.2	Studentova porazdelitev	57
3.2.1	Učenje parametrov iz vzorca	58
	Vprašanje 3-1-2	58
3.3	Porazdelitev Beta	59
	Vprašanje 3-1-3	60
3.3.1	Učenje parametrov iz vzorca	60
	Vprašanje 3-1-4	60
3.4	Primer: iskanje neslanih šal	61
	Vprašanje 3-2-1	64
	Vprašanje 3-2-2	67
	Vprašanje 3-2-3	68
	Vprašanje 3-2-4	68

4	Odkrivanje skupin	69
4.1	Metoda voditeljev	70
	Vprašanje 4-1-1	70
	Vprašanje 4-1-2	71
4.1.1	Podatki	72
	Vprašanje 4-1-3	73
4.1.2	Ocenjevanje uspešnosti odkrivanja skupin	73
	Vprašanje 4-1-4	73
	Vprašanje 4-1-5	73
4.1.3	Metoda DBSCAN	74
	Vprašanje 4-1-6	74
4.2	Hierarhično gručenje	74
4.2.1	Podatki	76
4.2.2	Metode povezovanja	78
	Vprašanje 4-2-1	78
4.2.3	Mere razdalje	78
4.2.4	Določanje števila gruč	79
	4.2.4.1 Skupna deljena informacija	80
	4.2.4.2 Koeficent silhuete	80
	Vprašanje 4-2-2	80
	Vprašanje 4-2-3	81
4.3	Primer: genomiški podatki	81
	Vprašanje 4-3-1	83
5	Nadzorovano učenje	85
5.1	Linearna regresija	85
5.2	Polinomska regresija	89
5.3	Model polinomske regresije	91
	Vprašanje 5-1-1	92
5.4	Pretirano prileganje	92
	Vprašanje 5-1-2	93
5.5	Rešitev: kaznovanje pretirano kompleksnih modelov	93
	Vprašanje 5-1-3	94
	Vprašanje 5-1-4	94
5.6	Uporaba v praksi: analiza sentimenta	95
	Vprašanje 5-1-5	115
	Vprašanje 5-1-6	115
5.7	Naivni Bayesov klasifikator	115
	5.7.1 Primer za ogrevanje	115
5.8	Bayesov obrazec	117
5.9	Implementacija Naivnega Bayesovega klasifikatorja	118
	Vprašanje 5-2-1	118
	5.9.1 Sklepanje o podatkih	118
	5.9.2 Napovedovanje	119
	5.9.3 Log-transformacija	119
5.10	Uporaba klasifikatorja	122
5.11	Ocenjevanje uspešnosti klasifikacije	144
	5.11.1 Delež pravilno razvrščenih razredov (ang. classification accuracy)	144
	5.11.2 Natančnost, priklic (ang. precision, recall)	144
6	Nenegativna matrična faktorizacija in priporočilni sistemi	147
6.1	Uvodne definicije	147
6.2	Definicija problema	148
6.3	Stohastični gradientni sestop	148

Vprašanje 6-1-1	149
Vprašanje 6-1-2	152
Vprašanje 6-1-3	153
Vprašanje 6-1-4	155
Vprašanje 6-1-5	155
Vprašanje 6-1-6	155
7 Omrežja	157
7.1 Knjižnica <code>networkx</code>	157
7.1.1 Gradnja grafa	157
7.1.2 Prikaz grafa	158
7.1.3 Segmentacija omrežja	159
7.2 Primer: analiza in vizualizacija omrežja elektronskih sporočil	161
8 Zaporedja	163
8.1 Skriti Markovi modeli	163
8.1.1 Generiranje zaporedij	164
8.1.2 Učenje parametrov modela iz podatkov	166
8.1.3 Viterbijev algoritem	168
8.2 Časovne vrste	170
8.2.1 Primerjava časovnih vrst	171
8.2.2 Dinamična poravnava signalov	172
8.3 Napovedovanje trendov	175
8.3.1 Gaussovi procesi	175
8.3.2 Primer	176
8.3.3 Kovariančne funkcije	177
Odgovori	181
1.1 Knjižnica <code>numpy</code>	181
Odgovor 1-1-1	181
Odgovor 1-1-2	181
1.2 Primer: statistika temperatur na severu	183
Odgovor 1-2-1	183
Odgovor 1-2-2	183
Odgovor 1-2-3	183
Odgovor 1-2-4	183
Odgovor 1-2-5	184
Odgovor 1-2-6	184
2.1 Knjižnica <code>matplotlib</code>	186
Odgovor 2-1-1	187
Odgovor 2-1-2	187
Odgovor 2-1-3	188
2.2 Primer: zimske olimpijske igre, Soči 2014	189
Odgovor 2-2-1	189
Odgovor 2-2-2	189
Odgovor 2-2-3	189
Odgovor 2-2-4	189
Odgovor 2-2-5	190
Odgovor 2-2-6	191
Odgovor 2-2-7	192
Odgovor 2-2-8	193
Odgovor 2-2-9	194
3.1 Pogoste verjetnostne porazdelitve	196
Odgovor 3-1-1	196

Odgovor 3-1-2	196
Odgovor 3-1-3	196
Odgovor 3-1-4	196
3.2 Primer: iskanje neslanih šal	197
Odgovor 3-2-1	197
Odgovor 3-2-2	197
Odgovor 3-2-3	197
Odgovor 3-2-4	197
4.1 Odkrivanje skupin	198
Odgovor 4-1-1	198
Odgovor 4-1-2	199
Odgovor 4-1-3	199
Odgovor 4-1-4	199
Odgovor 4-1-5	199
Odgovor 4-1-6	199
4.2 Hierarhično gručenje	200
Odgovor 4-2-1	200
Odgovor 4-2-2	200
Odgovor 4-2-3	200
4.3 Primer: genomski podatki v obliki nizov znakov	201
Odgovor 4-3-1	201
5.1 Linearna regresija	203
Odgovor 5-1-1	203
Odgovor 5-1-2	203
Odgovor 5-1-3	203
Odgovor 5-1-4	204
Odgovor 5-1-5	204
Odgovor 5-1-6	205
5.2 Naivni Bayesov klasifikator	206
6 Nenegativna matrična faktorizacija in priporočilni sistemi	209
7.1 Knjižica <code>networkx</code>	211
7.2 Primer: analiza in vizualizacija omrežja elektronskih sporočil	212
8.1 Skriti Markovi modeli	213
8.2 Modeliranje časovnih vrst	214
8.3 Nelinearna regresija ali napovedovanje trendov	215
Naloge	217
Priprava podatkov, osnovne statistike in vizualizacija	218
Iskanje strukture v podatkih	221
Napovedovanje vrednosti	224
Uporaba matrične faktorizacije za napovedovanje	228
Implementacija priporočilnega sistema	230
Literatura	231

Delovno okolje

Namestitev

Načinov namestitve delovnega okolja je več. Izberite tisto, ki vam najbolj ustreza.

```
In [1]: !cat skripte/pip_install.sh

# !/bin/bash

# Essentials
pip install --upgrade numpy
pip install --upgrade scipy
pip install --upgrade Pillow
pip install --upgrade matplotlib
#pip install --upgrade GPy

# Orange and requirements
pip install --upgrade Orange3

# Scikit-learn
pip install --upgrade sklearn

# iPython notebook and requirements
pip install --upgrade terminado
pip install --upgrade functools32
pip install --upgrade jupyter
pip install --upgrade jupyter_contrib_nbextensions

# Test installations
python -c "import Orange"
python -c "import sklearn"
python -c "import numpy"
python -c "import scipy"
python -c "import matplotlib"
#python -c "import GPy"
python -c "import jupyter"
```

Conda

```
In [2]: !cat skripte/conda_install.sh
```

```
conda install -c conda-forge jupyter_contrib_nbextensions
conda install -c conda-forge jupyter_nbextensions_configurator
jupyter contrib nbextension install --user
jupyter nbextensions_configurator enable --user

jupyter nbextension install https://rawgit.com/jfbercher/latex_envs/master/latex_envs.zip --user
jupyter nbextension enable latex_envs/latex_envs

pip install jupyter_latex_envs
jupyter nbextension install --py latex_envs
jupyter nbextension enable --py latex_envs

jupyter nbextension install https://rawgit.com/jfbercher/jupyter_nbTranslate/master/nbTranslate.zip --user
jupyter nbextension enable nbTranslate/main
```

Virtualenv

Docker

Git in GitHub

Git je sistem za upravljanje z različicami (Version Control System - VCS). Uporablja se ga tako za sodelovanje med razvijalci, kot za nadzor nad lastnim delom. Git upravlja z datotekami v repozitoriju - mapa, kjer se hrani zgodovina sprememb.

GitHub je spletna platforma za gostovanje Git repozitorijev in omogoča sodelovanje več ljudi na istem projektu.

Več o osnovah Gita in GitHuba si lahko preberete na <https://guides.github.com/>

Nekaj osnovnih ukazov

- `git init`: inicializira nov repozitorij v izbrani mapi
- `git clone`: na disk skopiramo vsebino oddaljenega repozitorija
- `git add`: ukaz, s katerim povemo, katere spremenjene datoteke želimo shraniti
 - `git add .`: doda vse datoteke
 - `git add datoteka.ipynb`: doda določeno datoteko
- `git commit -m šporočilo`: ukaz, s katerim shranimo izbrane datoteteke
- `git status`: ukaz, ki nam pokaže, katere datoteke so bile spremenjene in katere spremenjene datoteke bomo shranili
- `git pull`: osvežimo lokalni repozitorij s spremembami na oddaljenem repozitoriju
- `git push`: pošljemo svoje spremembe na oddaljen repozitorij
- `git branch`: pokaže lokalne veje, jih ustvari, ali zamenja aktivno vejo
- `git merge`: združevanje dveh vej

Celoten seznam ukazov in njihovih parametrov je dosegljiv na <https://git-scm.com/docs>.

Namestitev

Ob namestitvi Gita na Windows izberite opcijo **Enable symbolic links**.

Poglavje 1

Priprava podatkov

1.1 Knjižnica numpy

Knjižnica `numpy` [1] omogoča numerično računanje v jeziku Python. Vsebuje učinkovite implementacije podatkovnih struktur kot so vektorji, matrike in polja. Vse podatkovne strukture izhajajo iz podatkovnega tipa polje (`array`). Večina računsko zahtevnih operacij je implementiranih v nižjenivojskih jezikih (Fortran, C). Polje lahko ustvarimo na različne načine:

- s pretvorbo Pythonovih seznamov ali terk,
- z uporabo funkcij `arange`, `linspace` in podobnih,
- z branjem podatkov iz datotek.

```
In [1]: import numpy as np
```

1.1.1 Pretvorba seznamov v večdimenzionalna polja

Konstruktor `array` uporabimo neposredno tako, da podamo seznam. Če podamo seznam števil, dobimo vektor:

```
In [2]: v = np.array([1, 2, 3, 4])
        v
```

```
Out[2]: array([1, 2, 3, 4])
```

Če podamo seznam seznamov, dobimo matriko:

```
In [3]: M = np.array([[1, 2], [3, 4]])
        M
```

```
Out[3]: array([[1, 2],
               [3, 4]])
```

Neglede na obliko, sta objekta `v` in `M` tipa `ndarray`.

```
In [4]: type(v), type(M)
```

```
Out[4]: (numpy.ndarray, numpy.ndarray)
```

Razlika je v njunih dimenzijah. Objekt `v` je vektor s štirimi elementi, `M` pa matrika 2×2 .

```
In [5]: v.shape
```

```
Out[5]: (4,)
```

```
In [6]: M.shape
```

```
Out[6]: (2, 2)
```

Podobno lahko izpišemo število elementov v celotnem seznamu.

```
In [7]: M.size
```

```
Out[7]: 4
```

Vprašanje 1-1-1 Sestavimo lahko polja poljubnih dimenzij. Poskusi sestaviti seznam-seznamov-seznamov(-seznamov, ...) in preveri, kakšne so njegove dimenzije!

```
In [8]: # Sestavi strukturo poljubnih dimenzij in preveri njeno dimenzijo in velikost
        # X =
```

Odgovor

1.1.2 Razlike med seznamami in polji

Struktura `numpy.ndarray` še vedno izgleda kot seznam-seznamov(-seznamov, ...). V čem je razlika?

Nekaj hitrih dejstev: * Pythonovi sezname lahko vsebujejo poljuben tip objektov, ki se znotraj seznama lahko razlikujejo (dinamično tipiziranje). Ne podpirajo matematičnih operacij, kot so matrično množenje. Implementacija takih operacij nad seznamom bi bila zaradi dinamičnega tipiziranja zelo neučinkovita. * Polja so **statično tipizirana** in **homogena**. Podatkovni tip elementov je določen ob nastanku. * Posledično so polja pomnilniško učinkovita, saj zasedajo zvezen prostor v pomnilniku.

Ugotovimo tip elementov v trenutnem polju:

```
In [9]: M.dtype
```

```
Out[9]: dtype('int64')
```

Vstavljanje podatkov poljubnih tipov v polje lahko vodi do težav. Poskusi:

```
In [10]: M[0,0] = "hello"
```

```
-----

ValueError                                Traceback (most recent call last)

<ipython-input-10-e1f336250f69> in <module>
----> 1 M[0,0] = "hello"

ValueError: invalid literal for int() with base 10: 'hello'
```

Nastavimo podatkovni tip ob ustvarjanju polja, n.pr., kompleksna števila:

```
In [11]: M = np.array([[1, 2, 3], [1, 4, 9]], dtype=complex)
```

M

```
Out[11]: array([[ 1.+0.j,  2.+0.j,  3.+0.j],
                [ 1.+0.j,  4.+0.j,  9.+0.j]])
```

Med izvanjanjem spremenimo tip zapisov v polju:

```
In [12]: M = M.astype(float)
         M
```

```
/Users/tomazc/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: ComplexWarning: Casting complex numbers to float results in a loss of information about the imaginary part, use `np.real` and `np.imag` to obtain
  """Entry point for launching an IPython kernel.
```

```
Out[12]: array([[ 1.,  2.,  3.],
                [ 1.,  4.,  9.]])
```

Uporabimo lahko podatkovne tipe: `int`, `float`, `complex`, `bool`, `object`.

Velikosti, v bitih, lahko podamo eksplicitno: `int64`, `int16`, `float128`, `complex128`.

1.1.3 Uporaba polj

Najprej si oglejmo načine uporabe polj.

1.1.3.1 Naslavljanje

Elemente naslavljammo z uporabo oglatih oklepajev, podobno kot pri seznamih.

```
In [13]: # v je vektor; naslavljammo ga po njegovi edini dimenziji
         v[0]
```

```
Out[13]: 1
```

```
In [14]: # matriko M naslavljammo z dvema podatkom - naslov je sedaj terka
         M[1,1]
```

```
Out[14]: 4.0
```

Naslavljanje po eni dimenziji vrne najprej vrstice.

```
In [15]: M[1]
```

```
Out[15]: array([ 1.,  4.,  9.])
```

Z uporabo `:` povemo, da bi radi vse elemente v pripadajoči dimenziji. Kako bi dostop do celotnega prvega stolpca implementirali s seznammi? Potrebni bi bilo nekaj zank `for`. Sintaksa naslavljanja to bistveno poenostavi.

```
In [16]: M[1, :] # Vrstica
```

```
Out[16]: array([ 1.,  4.,  9.])
```

```
In [17]: M[:, 1] # Stolpec, precej enostavno.
```

```
Out[17]: array([ 2.,  4.])
```

Posamezne elemente lahko spreminjamo s prireditvenimi stavki.

```
In [18]: M[0, 0] = 9
```

```
In [19]: M
```

```
Out[19]: array([[ 9.,  2.,  3.],
                [ 1.,  4.,  9.]])
```

Lahko jih nastavljammo po celotni dimenziji.

```
In [20]: M[1, :] = 0
         M[:, 2] = -1
```

```
In [21]: M
Out[21]: array([[ 9.,  2., -1.],
                [ 0.,  0., -1.]])
```

1.1.3.2 Rezanje

Rezanje polj je pogost koncept. Poljubno pod-polje dobimo z naslavljanjem `M[od:do:korak]`:

```
In [22]: A = np.array([1, 2, 3, 4, 5])
         A
```

```
Out[22]: array([1, 2, 3, 4, 5])
```

```
In [23]: A[1:3]
```

```
Out[23]: array([2, 3])
```

Naslovljena pod-polja lahko tudi spreminjamo.

```
In [24]: A[1:3] = [-2, -3]
         A
```

```
Out[24]: array([ 1, -2, -3,  4,  5])
```

Katerikoli od parametrov rezanja je lahko tudi izpuščen.

```
In [25]: A[:] # Privzete vrednosti parametrov od:do:korak.
```

```
Out[25]: array([ 1, -2, -3,  4,  5])
```

```
In [26]: A[:2] # korak velikosti 2
```

```
Out[26]: array([ 1, -3,  5])
```

```
In [27]: A[:3] # prvi trije elementi
```

```
Out[27]: array([ 1, -2, -3])
```

```
In [28]: A[3:] # elementi od tretjega naprej
```

```
Out[28]: array([4, 5])
```

Negativni indeksi se nanašajo na konec polja:

```
In [29]: A = np.array([1, 2, 3, 4, 5])
```

```
In [30]: A[-1]
```

```
Out[30]: 5
```

Zadnji trije elementi:

```
In [31]: A[-3:]
```

```
Out[31]: array([3, 4, 5])
```

Rezanje deluje tudi pri večdimenzionalnih poljih.

```
In [32]: A = np.array([[n+m*10 for n in range(5)] for m in range(5)])
         A
```

```
Out[32]: array([[ 0,  1,  2,  3,  4],
                [10, 11, 12, 13, 14],
                [20, 21, 22, 23, 24],
                [30, 31, 32, 33, 34],
                [40, 41, 42, 43, 44]])
```

```
In [33]: # pod-polje izvirnega polja A
         A[1:4, 1:4]
```

```
Out[33]: array([[11, 12, 13],
                [21, 22, 23],
                [31, 32, 33]])
```

Elemente lahko preskakujemo.

```
In [34]: A[:, :2, ::2]
```

```
Out[34]: array([[ 0,  2,  4],
                [20, 22, 24],
                [40, 42, 44]])
```

1.1.3.3 Naslavljanje polja s pomočjo druge strukture

Polje naslavljamo tudi s pomočjo drugih polj ali seznamov.

```
In [35]: row_indices = [1, 2, 3]
         A[row_indices]
```

```
Out[35]: array([[10, 11, 12, 13, 14],
                [20, 21, 22, 23, 24],
                [30, 31, 32, 33, 34]])
```

```
In [36]: col_indices = [1, 2, -1]
         A[row_indices, col_indices]
```

```
Out[36]: array([11, 22, 34])
```

Uporabljamo tudi *maske*. Le-te so strukture s podatki tipa `bool`, ki nakazujejo, ali bo element na pripadajočem mestu izbran ali ne.

```
In [37]: B = np.array([n for n in range(5)])
         B
```

```
Out[37]: array([0, 1, 2, 3, 4])
```

```
In [38]: row_mask = np.array([True, False, True, False, False])
         B[row_mask]
```

```
Out[38]: array([0, 2])
```

Malenkost drugačen način določanja maske.

```
In [39]: row_mask = np.array([1, 0, 1, 0, 0], dtype=bool)
         B[row_mask]
```

```
Out[39]: array([0, 2])
```

Način lahko uporabimo za pogojno naslavljanje elementov glede na njihovo vsebino.

```
In [40]: x = np.array([0, 4, 2, 2, 3, 7, 10, 12, 15, 28])
         x
```

```
Out[40]: array([ 0,  4,  2,  2,  3,  7, 10, 12, 15, 28])
```

```
In [41]: mask = (5 < x) * (x < 12.3)
         mask
```

```
Out[41]: array([False, False, False, False, False,  True,  True,  True, False, False], dtype=bool)
```

```
In [42]: x[mask]
```

```
Out[42]: array([ 7, 10, 12])
```

Vprašanje 1-1-2 Preizkusi kombinacije vseh do sedaj omenjenih načinov naslavljanja naenkrat. Hkrati naslavljaljaj, npr., vrstice z rezanjem, stolpce pa s pogojnim naslavljanjem. Ustvari več kot dvo-dimenzionalne strukture. Preveri, ali razumeš rezultat vsakega od naslavljanj.

```
In [43]: # Preizkusi več načinov naslavljanja hkrati.
         A[A[:, 0]>10, 0:2 ]
         # ...
         # ...
```

```
Out[43]: array([[20, 21],
               [30, 31],
               [40, 41]])
```

Odgovor

1.1.3.4 Funkcije za ustvarjanje polj

Knjižica `numpy` vsebuje funkcije za ustvarjanje pogostih tipov polj. Poglejmo nekaj primerov.

Razpon `arange`

```
In [44]: np.arange(0, 10, 1) # od, do, korak
```

```
Out[44]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In [45]: np.arange(-1, 1, 0.1)
```

```
Out[45]: array([ -1.00000000e+00,  -9.00000000e-01,  -8.00000000e-01,
                 -7.00000000e-01,  -6.00000000e-01,  -5.00000000e-01,
                 -4.00000000e-01,  -3.00000000e-01,  -2.00000000e-01,
                 -1.00000000e-01,  -2.22044605e-16,   1.00000000e-01,
                  2.00000000e-01,   3.00000000e-01,   4.00000000e-01,
                  5.00000000e-01,   6.00000000e-01,   7.00000000e-01,
                  8.00000000e-01,   9.00000000e-01])
```

Razpona `linspace` in `logspace`

Pozor: začetna in končna točka sta tudi vključeni.

```
In [46]: np.linspace(0, 10, 25) # od, do, stevilo med sabo enako oddaljenih točk
```

```
Out[46]: array([ 0.          ,  0.41666667,  0.83333333,  1.25          ,
                 1.66666667,  2.08333333,  2.5          ,  2.91666667,
                 3.33333333,  3.75          ,  4.16666667,  4.58333333,
                 5.          ,  5.41666667,  5.83333333,  6.25          ,
                 6.66666667,  7.08333333,  7.5          ,  7.91666667,
                 8.33333333,  8.75          ,  9.16666667,  9.58333333, 10.          ])
```

```
In [47]: np.logspace(0, 10, 11, base=np.e) # Poskusi z drugo osnovo (bazo): 2, 3, 10
```

```
Out[47]: array([ 1.00000000e+00,  2.71828183e+00,  7.38905610e+00,
                 2.00855369e+01,  5.45981500e+01,  1.48413159e+02,
                 4.03428793e+02,  1.09663316e+03,  2.98095799e+03,
                 8.10308393e+03,  2.20264658e+04])
```

Naključna polja, modul `numpy.random`

```
In [48]: from numpy import random
         random.seed(42) # zagotovi ponovljivost naključnih rezultatov
```


Enakomerno (uniformno) porazdeljene vrednosti v intervalu [0,1]:

```
In [49]: random.rand(5, 5)
```

```
Out[49]: array([[ 0.37454012,  0.95071431,  0.73199394,  0.59865848,  0.15601864],
 [ 0.15599452,  0.05808361,  0.86617615,  0.60111501,  0.70807258],
 [ 0.02058449,  0.96990985,  0.83244264,  0.21233911,  0.18182497],
 [ 0.18340451,  0.30424224,  0.52475643,  0.43194502,  0.29122914],
 [ 0.61185289,  0.13949386,  0.29214465,  0.36636184,  0.45606998]])
```

Normalno porazdeljene vrednosti s povprečno vrednostjo 0 in odklonom 1:

```
In [50]: random.randn(5, 5)
```

```
Out[50]: array([[ -0.62947496,  0.59772047,  2.55948803,  0.39423302,  0.12221917],
 [ -0.51543566, -0.60025385,  0.94743982,  0.291034  , -0.63555974],
 [ -1.02155219, -0.16175539, -0.5336488  , -0.00552786, -0.22945045],
 [ 0.38934891, -1.26511911,  1.09199226,  2.77831304,  1.19363972],
 [ 0.21863832,  0.88176104, -1.00908534, -1.58329421,  0.77370042]])
```

Diagonalna matrika diag

Na diagonalni naj bodo vrednosti 1, 2 in 3.

```
In [51]: np.diag([1, 2, 3])
```

```
Out[51]: array([[1, 0, 0],
 [0, 2, 0],
 [0, 0, 3]])
```

Diagonala naj bo odmaknjena od glavne diagonale za k mest. Pozor, dimenzija matrike se temu ustrezno poveča.

```
In [52]: np.diag([1, 2, 3], k=1)
```

```
Out[52]: array([[0, 1, 0, 0],
 [0, 0, 2, 0],
 [0, 0, 0, 3],
 [0, 0, 0, 0]])
```

Ničle in enice - zeros, ones

```
In [53]: np.zeros((3, 3))
```

```
Out[53]: array([[ 0.,  0.,  0.],
 [ 0.,  0.,  0.],
 [ 0.,  0.,  0.]])
```

```
In [54]: np.ones((3, 3))
```

```
Out[54]: array([[ 1.,  1.,  1.],
 [ 1.,  1.,  1.],
 [ 1.,  1.,  1.]])
```

1.1.4 Osnovne računske operacije

Ključno pri uporabi interpretiranih jezikov je, da kar najbolj izkoriščamo vektorske operacije. Izogibajmo se odvečni uporabi zank. Karseda veliko operacij implementiramo kot operacije med matrikami in vektorji, npr., kot vektorsko ali matrično množenje.

1.1.4.1 Operacije polja s skalarjem

Uporabimo običajne aritmetične operacije za množenje, seštevanje in deljenje s skalarjem.

```
In [55]: v1 = np.arange(0, 5)
```

```
In [56]: v1 * 2
```

```
Out[56]: array([0, 2, 4, 6, 8])
```

```
In [57]: v1 + 2
```

```
Out[57]: array([2, 3, 4, 5, 6])
```

```
In [58]: A * 2, A + 2
```

```
Out[58]: (array([[ 0,  2,  4,  6,  8],
                 [20, 22, 24, 26, 28],
                 [40, 42, 44, 46, 48],
                 [60, 62, 64, 66, 68],
                 [80, 82, 84, 86, 88]]), array([[ 2,  3,  4,  5,  6],
                 [12, 13, 14, 15, 16],
                 [22, 23, 24, 25, 26],
                 [32, 33, 34, 35, 36],
                 [42, 43, 44, 45, 46]]))
```

1.1.4.2 Operacije polje-polje (po elementih)

Operacije med več polji se privzeto obravnavajo po elementih. Na primer, množenje po elementih dosežemo z uporabo operatorja `*`.

```
In [59]: A * A
```

```
Out[59]: array([[ 0,  1,  4,  9, 16],
                 [100, 121, 144, 169, 196],
                 [400, 441, 484, 529, 576],
                 [900, 961, 1024, 1089, 1156],
                 [1600, 1681, 1764, 1849, 1936]])
```

```
In [60]: v1 * v1
```

```
Out[60]: array([ 0,  1,  4,  9, 16])
```

Pozor, dimenzije polj se morajo ujemati.

```
In [61]: A.shape, v1.shape
```

```
Out[61]: ((5, 5), (5,))
```

```
In [62]: A * v1
```

```
Out[62]: array([[ 0,  1,  4,  9, 16],
                 [ 0, 11, 24, 39, 56],
                 [ 0, 21, 44, 69, 96],
                 [ 0, 31, 64, 99, 136],
                 [ 0, 41, 84, 129, 176]])
```

1.1.5 Iteracija po elementih polja

Skušamo se držati načela, da se izogibamo uporabi zank preko elementov polja. Razlog je počasna implementacija zank v intepretiranih jezikih, kot je Python. Včasih pa se zankam ne moremo izogniti. Zanka `for` je smiselna rešitev.

```
In [63]: v = np.array([1,2,3,4])
```

```
    for element in v:
        print(element)
```

```
1
2
3
4
```

```
In [64]: M = np.array([[1,2], [3,4]])
```

```
    for row in M:
        print("row", row)

        for element in row:
            print(element)
```

```
row [1 2]
1
2
row [3 4]
3
4
```

Generator `enumerate` uporabimo kadar želimo iteracijo po elementih in morebitno spreminjanje njihovih vrednosti.

```
In [65]: for i, row in enumerate(M):
        print("row index", i, "row", row)

        for j, element in enumerate(row):
            print("col index", j, "element", element)

            # Kvadriramo vsakega od elementov
            M[i, j] = element ** 2
```

```
row index 0 row [1 2]
col index 0 element 1
col index 1 element 2
row index 1 row [3 4]
col index 0 element 3
col index 1 element 4
```

Dobimo polje, kjer je vsak element kvadrat prvotne vrednosti.

```
In [66]: M
```

```
Out[66]: array([[ 1,  4],
               [ 9, 16]])
```

Več o knjižnici `numpy` lahko preberete v [1, 2, 3, 4].

1.2 Primer: statistika temperatur v Stockholmu

Knjižnico `numpy` uporabimo na primeru podatkov dnevne temperature v Stockholmu. Podatki obsegajo meritve za vsak dan med leti 1800 in 2011. Shranjeni so v datoteki, kjer vrstice predstavljajo meritve. Posamezni podatki - leto, mesec, dan in izmerjena temperatura - so ločeni z vejico.

```
In [1]: from csv import DictReader
```

```
fp = open('podatki/stockholm.csv', 'rt')
reader = DictReader(fp)
```

```
for row in reader:
    print(row)
    break # izpisi samo prvo vrstico
```

```
OrderedDict([('Year', '1800'), ('Month', '1'), ('Day', '1'), ('Temp', '-6.1')])
```

Predstavitev podatkov v obliki slovarja je koristna zaradi svoje jasnosti, vendar bo računanje bistveno hitrejše, če podatke naložimo kot polje.

```
In [2]: import numpy as np
```

```
np.set_printoptions(suppress=True)
```

```
data = np.loadtxt('podatki/stockholm.csv', delimiter=",", skiprows=1)
data
```

```
Out[2]: array([[ 1800. ,    1. ,    1. ,   -6.1],
               [ 1800. ,    1. ,    2. ,  -15.4],
               [ 1800. ,    1. ,    3. ,  -15. ],
               ...,
               [ 2011. ,   12. ,   29. ,    4.9],
               [ 2011. ,   12. ,   30. ,    0.6],
               [ 2011. ,   12. ,   31. ,   -2.6]])
```

Preverimo velikost podatkov: število vrstic (*meritve*, *vzorci*) in število stolpcev (*atributov*).

```
In [3]: data.shape
```

```
Out[3]: (77431, 4)
```

Stolpci hranijo podatke v tem vrstnem redu: `leto`, `mesec`, `dan` in `temperatura`.

Poglejmo si vse meritve, ki so bile narejene v letu 2011. Ustvarimo binarni vektor `data[:, 0] == 2011`, ki vsebuje vrednost `True` nas ustreznih mestih ter ga uporabimo za naslavljanje podatkov.

```
In [4]: data[data[:, 0] == 2011]
```

```
Out[4]: array([[ 2011. ,    1. ,    1. ,   -2.3],
               [ 2011. ,    1. ,    2. ,   -3.6],
               [ 2011. ,    1. ,    3. ,   -6.9],
               ...,
               [ 2011. ,   12. ,   29. ,    4.9],
               [ 2011. ,   12. ,   30. ,    0.6],
               [ 2011. ,   12. ,   31. ,   -2.6]])
```

Vprašanje 1-2-1 Izpišite temperaturo pred 200 leti, na primer, temperaturo dne 5. decembra 1817.

In [5]:

Odgovor

1.2.1 Procesiranje podatkov

Na tej točki nastopijo operacije, ki nam povedo nekaj o podatkih. Izračunali bomo nekaj osnovnih statistik.

1.2.1.1 Povprečje, aritmetična sredina

Dnevna temperatura je v stolpcu z indeksom 3 (četrti stolpec). Izračunamo povprečje vseh meritev.

In [5]: `np.mean(data[:, 3])`

Out[5]: 6.1971096847515854

Ugotovimo, da je bila povprečna dnevna temperatura v Stockholmu v preteklih 200 letih prijetnih 6.2 C.

Vprašanje 1-2-2 Kakšna je povprečna temperatura januarja (mesec s številko 1)?

In [6]:

Odgovor

1.2.1.2 Standardni odklon in varianca

In [6]: `np.std(data[:,3]), np.var(data[:,3])`

Out[6]: (8.2822716213405734, 68.596023209663414)

Vprašanje 1-2-3 V katerem mesecu je odklon temperature največji?

```
In [7]: # Poišči mesec z največjim odklonom oz. varianco v temperaturi.  
        # Namig: zgradi seznam terk oblike (odklon v temperaturi, mesec)  
        # ...
```

Odgovor

1.2.1.3 Najmanjša in največja vrednost

Poiščimo najnižjo dnevno temperaturo:

In [8]: `data[:,3].min()`

Out[8]: -25.800000000000001

Poiščimo najvišjo dnevno temperaturo:

In [9]: `data[:,3].max()`

Out[9]: 28.300000000000001

Vprašanje 1-2-4 Pošči mesec in leto, ko so zabeležili največjo temperaturo.

```
In [10]: # Poišči mesec in leto, kjer smo v povprečju beležili najvišjo temperaturo.
# Namig: zgradi seznam terk oblike (povprečna temperatura, (leto, mesec))
# ...
```

Odgovor

1.2.1.4 Vsota, produkt

Temperatur ponavadi ne seštevamo. Pa vendar, izkoristimo priložnost za prikaz funkcij vsote in produkta.

```
In [11]: data[:, 3].sum() # vsota vseh temperatur
Out[11]: 479848.40000000002

In [12]: data[:, 3].sum() / data.shape[0] # dobimo ravno aritmetično sredino
Out[12]: 6.1971096847515854

In [13]: # prva vrstica v podatkih ...
data[0, :]
Out[13]: array([ 1800. ,      1. ,      1. ,    -6.1])

In [14]: # ... in njen produkt
np.prod(data[0, :])
Out[14]: -10980.0
```

1.2.2 Globalno segrevanje?

Odgovorimo na še nekaj vprašanj. Po Stockholmu krožijo govorice, da se temperatura iz leta v leto povečuje.

```
In [15]: # Izračunajmo povprečno temperaturo za vsako leto posebej
letna_povprečja = dict()

for leto in range(1800, 2012):
    # Uporabimo pogojno naslavljanje polja
    letna_povprečja[leto] = data[data[:, 0] == leto, 3].mean()
```

Vprašanje 1-2-5 Izpiši leta, ko je povprečna temperatura višja od prejšnjega leta.

```
In [16]: # Izpiši vsako leto, ki ima večjo povprečno temperaturo od prejšnjega
#
```

Poišči 10 najtoplejših let.

```
In [17]: # Poišči 10 najtoplejših let
#
```

Odgovor

Zadnja leta so res sumljivo topla. Poskusimo prikazati podatke z uporabo knjižnice matplotlib.

```
In [18]: %matplotlib inline
%config InlineBackend.figure_formats = ['jpg']
import matplotlib
matplotlib.figure.Figure.__repr__ = lambda self: (
    f"<{self.__class__.__name__} size {self.bbox.size[0]:g}"
```

```

        f"x{self.bbox.size[1]:g} with {len(self.axes)} Axes>")
import matplotlib.pyplot as plt
plt.style.use('PR.mplstyle')

-----

FileNotFoundError                                Traceback (most recent call last)

~/anaconda3/lib/python3.6/site-packages/matplotlib/style/core.py in use(style)
    112         try:
--> 113             rc = rc_params_from_file(style, use_default_template=False)
    114             _apply_style(rc)

~/anaconda3/lib/python3.6/site-packages/matplotlib/__init__.py in rc_params_from_file(fname, fail
1028     """
-> 1029     config_from_file = _rc_params_in_file(fname, fail_on_error)
    1030

~/anaconda3/lib/python3.6/site-packages/matplotlib/__init__.py in _rc_params_in_file(fname, fail
    944     rc_temp = {}
--> 945     with _open_file_or_url(fname) as fd:
    946         try:

~/anaconda3/lib/python3.6/contextlib.py in __enter__(self)
     80         try:
---> 81             return next(self.gen)
     82         except StopIteration:

~/anaconda3/lib/python3.6/site-packages/matplotlib/__init__.py in _open_file_or_url(fname)
    929         encoding = "utf-8"
--> 930         with io.open(fname, encoding=encoding) as f:
    931             yield f

```

FileNotFoundError: [Errno 2] No such file or directory: 'PR.mplstyle'

During handling of the above exception, another exception occurred:

```

OSError                                Traceback (most recent call last)

<ipython-input-18-78ed3b6a1a07> in <module>
      6         f"x{self.bbox.size[1]:g} with {len(self.axes)} Axes>")
      7 import matplotlib.pyplot as plt
----> 8 plt.style.use('PR.mplstyle')

~/anaconda3/lib/python3.6/site-packages/matplotlib/style/core.py in use(style)

```

```

117             "not a valid URL or path. See `style.available` for "
118             "list of available styles.")
--> 119         raise IOError(msg % style)
120
121

```

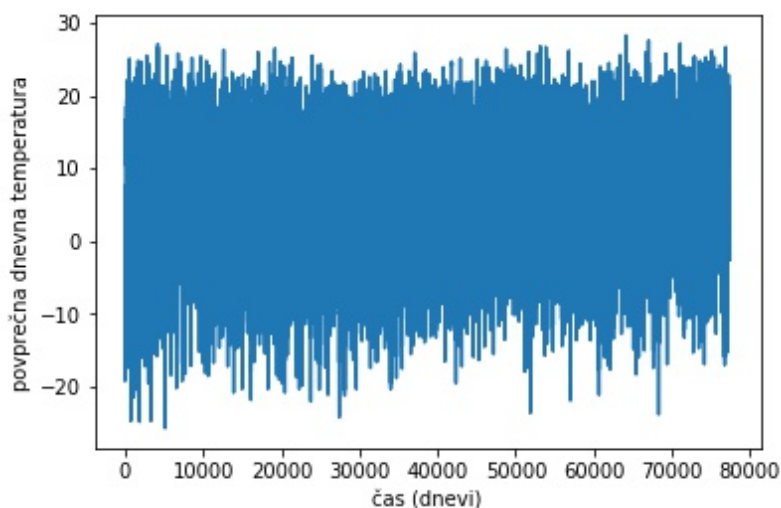
OSError: 'PR.mplstyle' not found in the style library and input is not a valid URL or path. See

Naredimo novo sliko (*figure*) in nanjo narišemo povprečne temperature v odvisnosti od časa.

```

In [19]: plt.figure()
         plt.plot(data[:, 3])
         plt.xlabel("čas (dnevi)") # Vedno označimo osi.
         plt.ylabel("povprečna dnevna temperatura");

```



Precej nepregledno. Poizkusite razširiti sliko tako da spremenite `plt.figure(figsize=(sirina, visina))`, kjer sta *visina* in *sirina* podani v palcih oz. inčah (privzeto (5, 3)).

Vseeno pa opazimo, da se pogostost dni s temperaturo nižjo od -20.0 C zmanjšuje. Poglejmo.

```

In [20]: plt.figure()

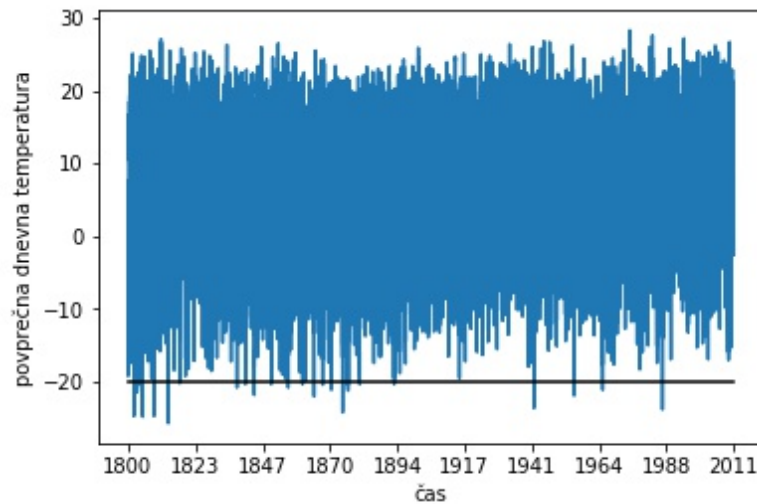
         # Narišimo izvirne podatke
         plt.plot(data[:, 3])

         # Z vodoravno črto označimo -20.0 C.
         plt.plot([0, len(data)], [-20, -20], color="black")

         # Spremenimo še oznako x-osi. Dodajmo 10 enako oddaljenih kazalcev.
         ticks = np.arange(0, len(data), len(data)//9, dtype=int)
         plt.xticks(ticks)
         plt.gca().set_xticklabels(data[ticks, 0].astype(int))

         # Vedno označimo osi.
         plt.xlabel("čas")
         plt.ylabel("povprečna dnevna temperatura")
         plt.show()

```

Od 80-tih let prejšnjega stoletja res nismo imeli kakšnega posebej hladnega dneva. Vseeno pa bi želeli še bolj poenostaviti prikaz. Prikažimo vsako leto z eno točko, ki naj prikazuje povprečno temperaturo leta.

Vprašanje 1-2-6 Nariši sliko povprečne letne temperature. Uporabi funkcijo `plt.plot(x, y)` kjer je `x` vektor let, `y` pa vektor pripadajočih povprečnih temperatur. Ali misliš, da se temperatura z leti res povečuje?

```
In [21]: # Pomagaj si s letna_povprečja.  
         # Os x: leto  
         # Os y: povprečna letna temperatura  
         # ...
```

Odgovor

Poglavje 2

Prikazovanje podatkov

2.1 Knjižnica matplotlib

Matplotlib je knjižnica za 2D in 3D risanje v programskem jeziku Python. Vključuje:

- Nadzor nad posameznimi elementi slike.
- Izvoz rezultatov v obliki PNG, PDF, SVG, EPS, in PGF.
- Podpora sintaksi \LaTeX

Bistven del uporabnosti knjižnice je, da lahko slike v celoti zgradimo z uporabo ukazov, kar odstrani potrebo po ročnem urejanju. Slednje jo dela zelo primerno za uporabo v znanstvenem delu, kjer lahko generiramo kompleksne prikaze na različnih podatkih brez potrebe po spremembi programske kode.

Spletna stran knjižnice je tudi bogat vir dodatnih primerov: <http://matplotlib.org/>

```
In [1]: %matplotlib inline
```

```
import matplotlib
```

```
%config InlineBackend.figure_format = 'jpg'
```

```
matplotlib.figure.Figure.__repr__ = lambda self: (
```

```
    f"<{self.__class__.__name__} size {self.bbox.size[0]:g}"
```

```
    f"x{self.bbox.size[1]:g} with {len(self.axes)} Axes>")
```

```
import matplotlib.pyplot as plt
```

```
plt.style.use('PR.mplstyle')
```

```
matplotlib.__version__
```

```
-----  
FileNotFoundError
```

```
Traceback (most recent call last)
```

```
~/anaconda3/lib/python3.6/site-packages/matplotlib/style/core.py in use(style)
```

```
112         try:
```

```
--> 113             rc = rc_params_from_file(style, use_default_template=False)
```

```
114             _apply_style(rc)
```

```
~/anaconda3/lib/python3.6/site-packages/matplotlib/__init__.py in rc_params_from_file(fname, fail_on_error)
```

```
1028     """
```

```
-> 1029     config_from_file = _rc_params_in_file(fname, fail_on_error)
```

1030

```

~/anaconda3/lib/python3.6/site-packages/matplotlib/__init__.py in _rc_params_in_file(fname, fail
944     rc_temp = {}
--> 945     with _open_file_or_url(fname) as fd:
946         try:

~/anaconda3/lib/python3.6/contextlib.py in __enter__(self)
80         try:
---> 81             return next(self.gen)
82         except StopIteration:

~/anaconda3/lib/python3.6/site-packages/matplotlib/__init__.py in _open_file_or_url(fname)
929         encoding = "utf-8"
--> 930         with io.open(fname, encoding=encoding) as f:
931             yield f

```

FileNotFoundError: [Errno 2] No such file or directory: 'PR.mplstyle'

During handling of the above exception, another exception occurred:

```

OSError                                Traceback (most recent call last)

<ipython-input-1-6de4f0886ee3> in <module>
      6     f"x{self.bbox.size[1]:g} with {len(self.axes)} Axes>")
      7 import matplotlib.pyplot as plt
----> 8 plt.style.use('PR.mplstyle')
      9
     10 matplotlib.__version__

~/anaconda3/lib/python3.6/site-packages/matplotlib/style/core.py in use(style)
    117         "not a valid URL or path. See `style.available` for "
    118         "list of available styles.")
--> 119         raise IOError(msg % style)
    120
    121

```

OSError: 'PR.mplstyle' not found in the style library and input is not a valid URL or path. See

Enostavna slika v okolju matplotlib:

```

In [2]: import numpy as np
        np.random.seed(42)

        x = np.linspace(0, 5, 10)
        y = x ** 2

```

Funkcija `plot` sprejema parametre: * podatki na ordinati, * podatki na abscisi, * ostali parametri (oblikovanje, ...)

```
In [3]: plt.figure()
        plt.plot(x, y, 'r')
        plt.xlabel('x')
        plt.ylabel('y')
        plt.title('Kvadratna funkcija, $y=x^2$');
```



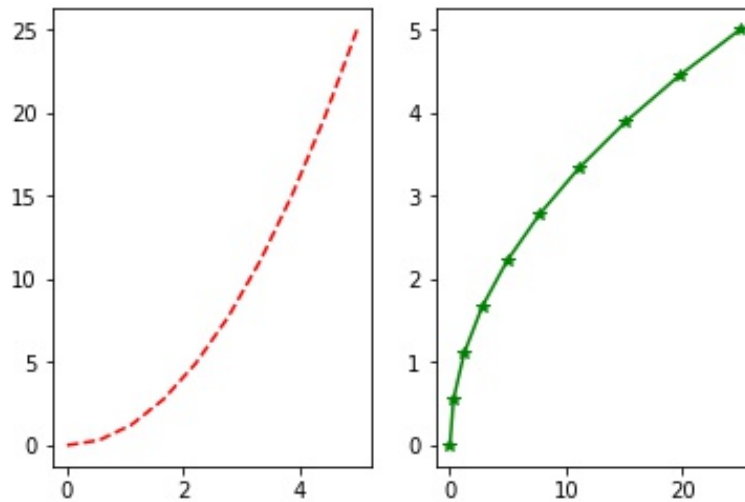
Vprašanje 2-1-1 Nariši funkcije x^{-3} , x^{-2} , x^{-1} , x^0 , x^1 , x^2 , x^3 na intervalu $(0,5]$. To dosežeš z večkratnim klicanjem funkcije `plot`, po enkrat za vsako krivuljo.

```
In [4]: # Nariši funkcije na isti graf
        # ...
```

Odgovor

Z uporabo okolja `subplot` lahko ustvarimo sliko z več platni.

```
In [5]: plt.subplot(1, 2, 1)    # ustvari sliko z 1 x 2 platnoma in izberi prvo platno
        plt.plot(x, y, 'r--')    # nariši podatke
        plt.subplot(1, 2, 2)    # na isti sliki izberi drugo platno
        plt.plot(y, x, 'g*-');  # ...
```



2.1.1 Objektno usmerjen način dela z matplotlib

V zgornjih primerih smo uporabljali vmesnik, kjer je bila vsaka slika del *globalnega* okolja. Ta način je uporaben za preprostejše slike. Zahtevnejše prikaze pa omogoča objektno usmerjen način, posebej v primerih, ko imamo opravka z več kot eno sliko naenkrat.

Bistveni del objektno-usmerjenega okolja sta objekta **slika** (ang. *figure*) in **os** (ang. *axis*). Ena slika vsebuje eno ali več osi. Os je vsebovalnik s koordinatnim sistemom, v katerega rišemo objekte (črte, stolpce, oblike, ...).

Ustvarimo novo sliko v spremenljivki **fig** ter ji dodajmo novo os, do katere dostopamo preko spremenljivke **axes**.

```
In [6]: fig = plt.figure()

# poskušaj spreminjati parametre
axes = fig.add_axes([0.1, 0.1, 0.8, 0.8]) # x koordinata osi relativno na sliko,
# y koordinata osi relativno na sliko,
# širina,
# višina

axes.plot(x, y, 'r')

axes.set_xlabel('x')
axes.set_ylabel('y')
axes.set_title('Ponovno kvadratna funkcija');
```



Sedaj imamo popoln nadzor nad vstavljanjem osi. Na sliko lahko dodamo poljubno število osi, ki se lahko tudi prekrivajo.

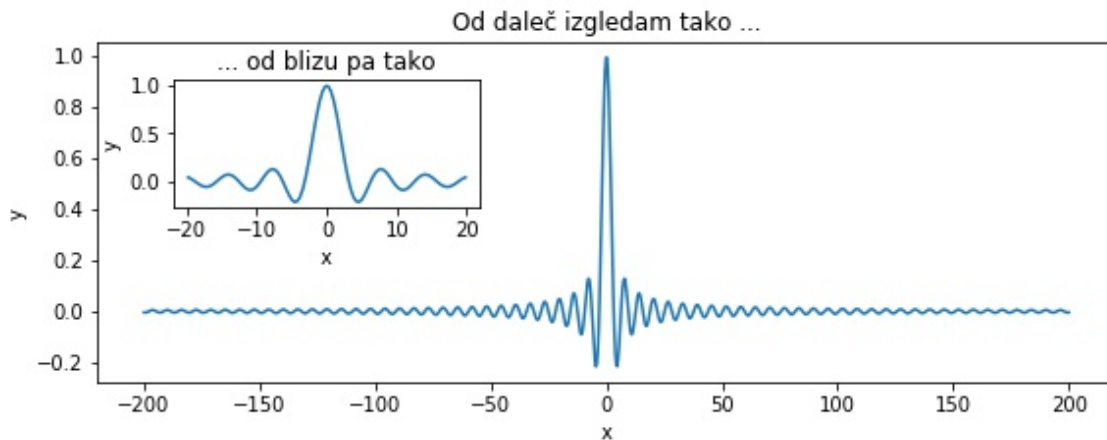
```
In [7]: fig = plt.figure(figsize=(9, 3))

x = np.linspace(-200, 200, 1000)
y = np.sin(x)/x

axes1 = fig.add_axes([0.1, 0.1, 0.8, 0.8]) # glavna os
axes2 = fig.add_axes([0.16, 0.51, 0.24, 0.3], facecolor='white') # os znotraj glavne osi.
# pomemben je tudi vrstni red ustvarjanja!

# Prikaži večji interval
axes1.plot(x, y)
axes1.set_xlabel('x')
axes1.set_ylabel('y')
axes1.set_title('Od daleč izgledam tako ...')

# Prikaži okolico ničle
axes2.plot(x[450:550], y[450:550])
axes2.set_xlabel('x')
axes2.set_ylabel('y')
axes2.set_title('... od blizu pa tako');
```



Za osi razporejene v lepo urejeno mrežo lahko uporabljamo upravljalnik subplots.

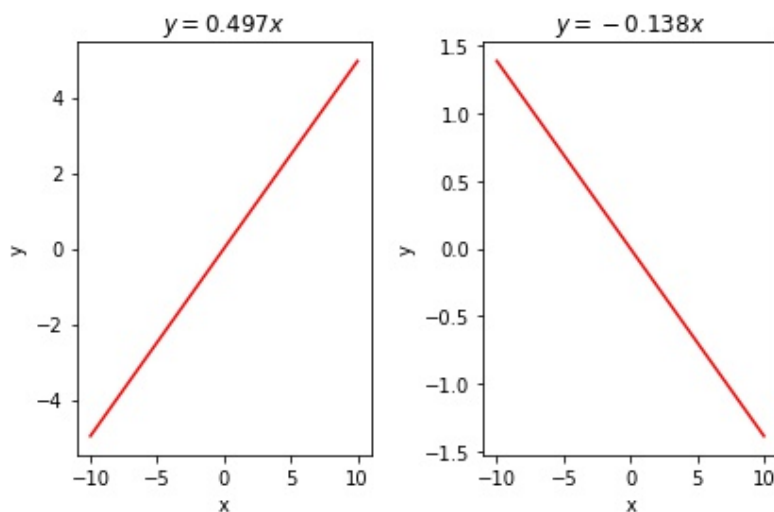
In [8]: `fig, axes = plt.subplots(nrows=1, ncols=2)`

```
x = np.linspace(-10, 10, 100)

for ax in axes:
    k = np.random.randn(1, 1)[0]  # sprehodimo se po oseh
    y = k * x                     # narišimo naključno premico

    ax.plot(x, y, 'r')
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_title('$y = %.3f x$' % k)

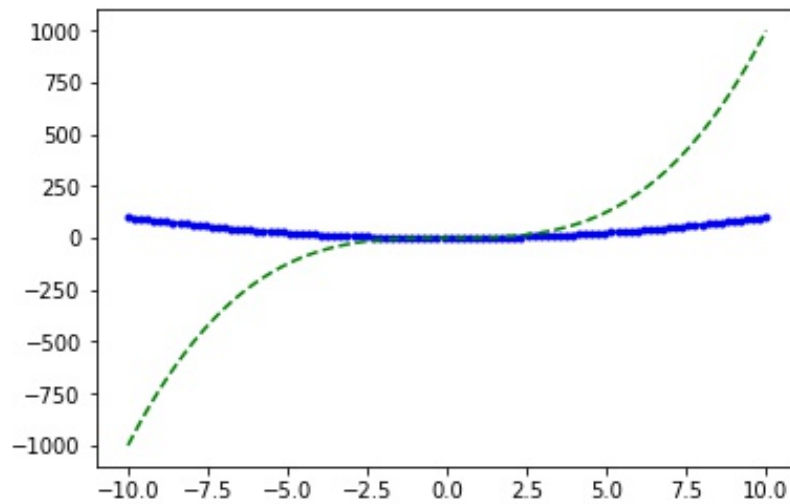
fig.tight_layout()  # poskrbi da se osi ne prekrivajo
```



2.1.2 Barvanje

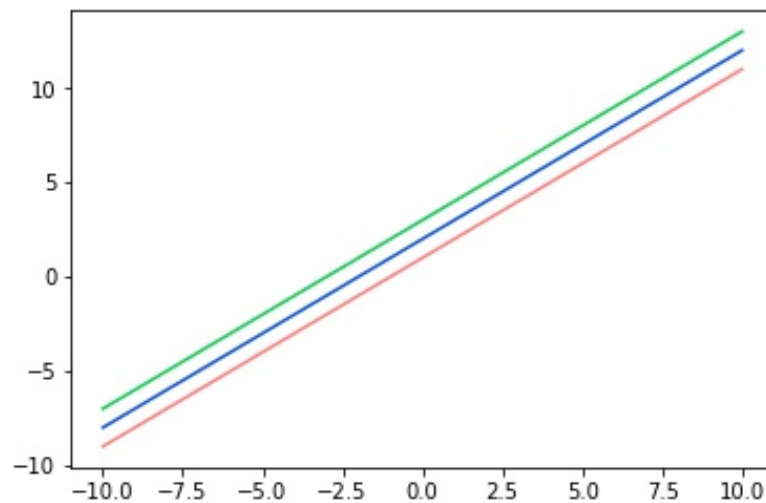
Najenostavnejši načina za nastavljanje barv je slog, podoben okolju MATLAB; g predstavlja zeleno barvo, b modro, r rdečo, itd.


```
In [9]: plt.figure()
plt.plot(x, x**2, 'b.-') # modra črta s označenimi točkami
plt.plot(x, x**3, 'g--'); # green dashed line
```



Lahko pa uporabimo argument `color=...`, kjer barvo podamo z njenim imenom oz. RGB kodo.

```
In [10]: plt.figure()
plt.plot(x, x+1, color="red", alpha=0.5) # Parameter alpha določa transparentnost; preizkusi!
plt.plot(x, x+2, color="#1155dd")
plt.plot(x, x+3, color="#15cc55");
```



2.1.3 Stili

Poizkusimo spreminjati še ostale lastnosti: debelino črt in različne oznake za točke.

```
In [11]: fig, ax = plt.subplots()

ax.plot(x, x+1, color="blue", linewidth=0.25)
ax.plot(x, x+2, color="blue", linewidth=0.50)
ax.plot(x, x+3, color="blue", linewidth=1.00)
```

```

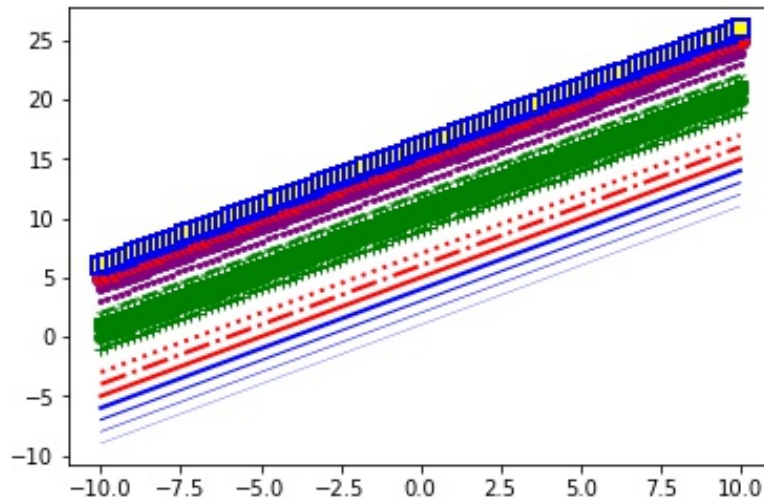
ax.plot(x, x+4, color="blue", linewidth=2.00)

# možnosti za izgled črte so '-', '--', '-.', ':', 'steps'
ax.plot(x, x+5, color="red", lw=2, linestyle='--')
ax.plot(x, x+6, color="red", lw=2, ls='-.')
ax.plot(x, x+7, color="red", lw=2, ls=':')

# oznake za točke: marker = '+', 'o', '*', 's', ',', '.', '1', '2', '3', '4', ...
ax.plot(x, x+ 9, color="green", lw=2, ls='--', marker='+')
ax.plot(x, x+10, color="green", lw=2, ls='--', marker='o')
ax.plot(x, x+11, color="green", lw=2, ls='--', marker='s')
ax.plot(x, x+12, color="green", lw=2, ls='--', marker='1')

# velikost in barva označb
ax.plot(x, x+13, color="purple", lw=1, ls='-', marker='o', markersize=2)
ax.plot(x, x+14, color="purple", lw=1, ls='-', marker='o', markersize=4)
ax.plot(x, x+15, color="purple", lw=1, ls='-', marker='o', markersize=8,
        markerfacecolor="red")
ax.plot(x, x+16, color="purple", lw=1, ls='-', marker='s', markersize=8,
        markerfacecolor="yellow", markeredgewidth=2, markeredgcolor="blue");

```



2.1.4 Vizualizacije različnih tipov podatkov

Oglejmo si še druge metode, ki so primerne za risanje različnih tipov podatkov. Seveda je način prikaza odvisen od vrste in lastnosti podatkov, ki jo z vizualizacijo želimo poudariti.

```
In [12]: n = np.array([0,1,2,3,4,5])
```

```
In [13]: fig, axes = plt.subplots(1, 3, figsize=(9, 3))
```

```

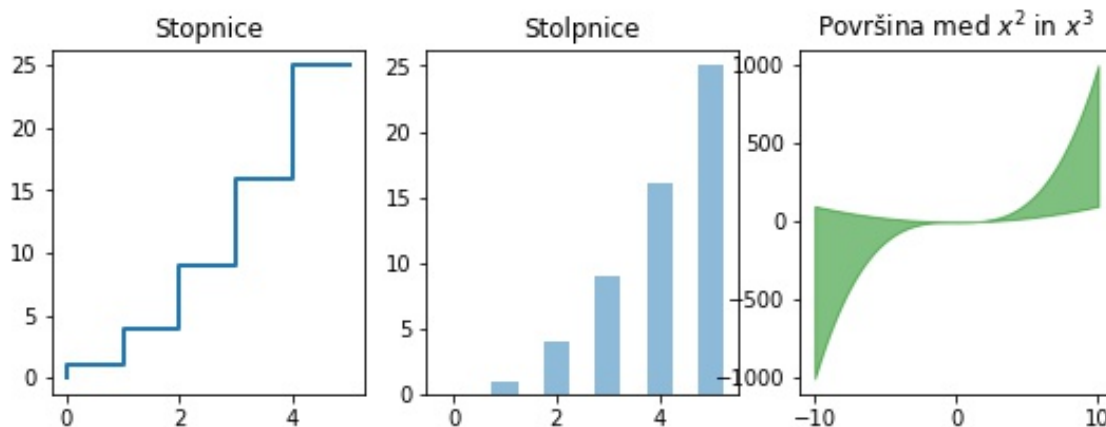
# Stopnice
axes[0].step(n, n**2, lw=2)
axes[0].set_title("Stopnice")

# Stolpični diagram
axes[1].bar(n, n**2, align="center", width=0.5, alpha=0.5)

```

```
axes[1].set_title("Stolpnice")

# Površina med krivuljama kvadratne in kubične funkcije
axes[2].fill_between(x, x**2, x**3, color="green", alpha=0.5);
axes[2].set_title("Površina med  $x^2$  in  $x^3$ ");
```



2.1.5 Verjetnostne porazdelitve

Verjetnostno porazdelitev končnega števila vzorcev pogosto predstavimo s histogramom - stolpičnim diagramom ki predstavlja število oz. verjetnost vrednosti spremenljivke.

Naj bo x naključna spremenljivka, porazdeljena po normalni (Gaussovi) porazdelitvi s sredino $\mu = 0$ in standardnim odklonom $\sigma = 1$.

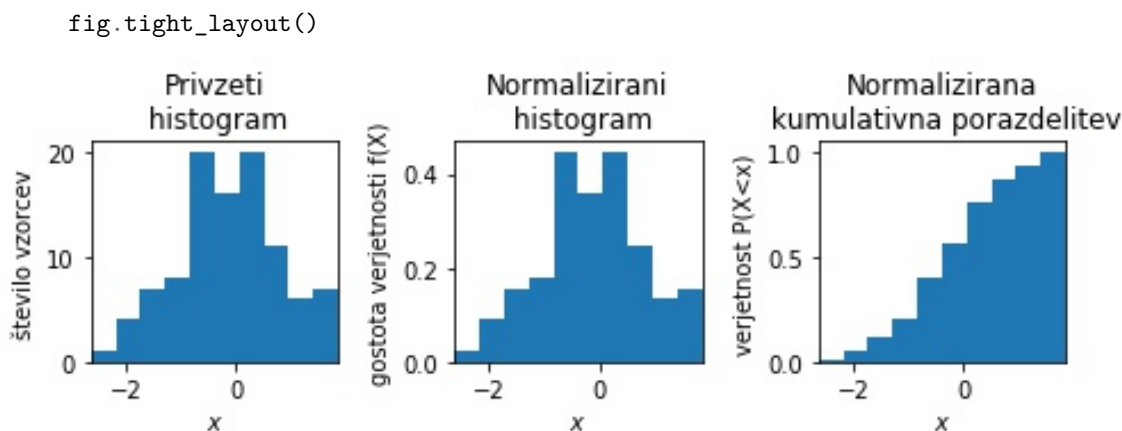
Vzamimo N naključnih vzorcev spremenljivke x . Funkcija `hist` izriše stolpični diagram verjetnostne porazdelitve glede na izide vzorčenj.

```
In [14]: # Histogram verjetnostne porazdelitve števil
N = 100
data = np.random.randn(N) # vzorčimo N točk
fig, axes = plt.subplots(1, 3, figsize=(7, 2.5))

axes[0].hist(data, bins=10)
axes[0].set_title("Privzeti\n histogram")
axes[0].set_xlim((min(data), max(data)));
axes[0].set_ylabel("število vzorcev")
axes[0].set_xlabel("$x$")

axes[1].hist(data, density=True, bins=10)
axes[1].set_title("Normalizirani\n histogram")
axes[1].set_xlim((min(data), max(data)));
axes[1].set_ylabel("gostota verjetnosti f(X)")
axes[1].set_xlabel("$x$")

axes[2].hist(data, cumulative=True, bins=10, density=True)
axes[2].set_title("Normalizirana\n kumulativna porazdelitev")
axes[2].set_ylabel("verjetnost P(X<x)")
axes[2].set_xlim((min(data), max(data)));
axes[2].set_xlabel("$x$")
```



Vprašanje 2-1-2 Poizkusi spreminjati število vzorcev N in predalčkov `bins`. Ali so katere nastavitve primernejše od drugih v odvisnosti od števila vzorcev?

In [15]:

Odgovor

Vprašanje 2-1-3 Funkcija `randn` predpostavlja sredino $\mu = 0$ in standardni odklon $\sigma = 1$. Kako bi modelirali poljubno sredino in standardni odklon, npr. $\mu = 5$ in $\sigma = 0.5$?

In [15]:

Odgovor

2.1.6 Toplotne karte in konture

Toplotne karte (ang. `heatmap`) uporabljamo za prikazovanje funkcij dveh spremenljivk. Narišimo funkcijo dveh spremenljivk:

$$z = \sin(x)\cos(y)$$

```
In [15]: def func(x, y):
         return np.sin(x) * np.cos(y)
```

```
In [16]: x = np.linspace(-np.pi, np.pi, 100)
         y = np.linspace(-np.pi, np.pi, 100)
         X,Y = np.meshgrid(x, y)
         Z = func(X, Y)
```

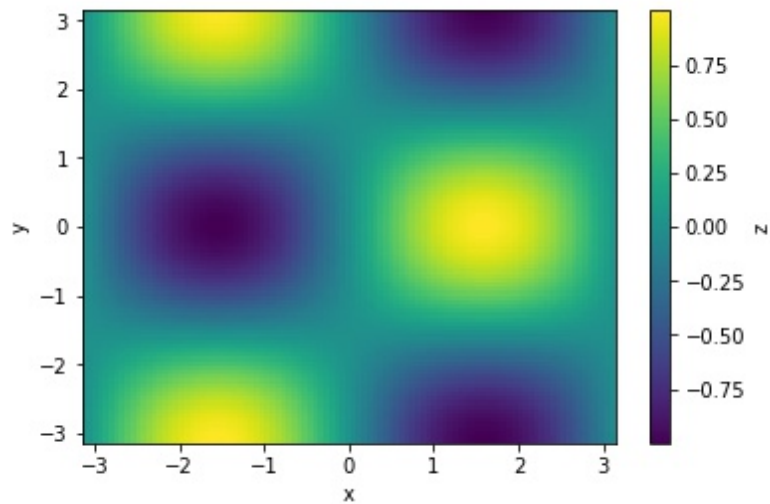
V `matplotlib` lahko izbiramo med več možnostmi.

2.1.6.1 Funkcija `pcolor`

```
In [17]: fig, ax = plt.subplots()

         p = ax.pcolor(X, Y, Z,)
         cb = fig.colorbar(p, ax=ax, label="z")
         ax.set_xlabel("x")
```

```
ax.set_ylabel("y")
ax.set_xlim(-np.pi, np.pi)
ax.set_ylim(-np.pi, np.pi);
```



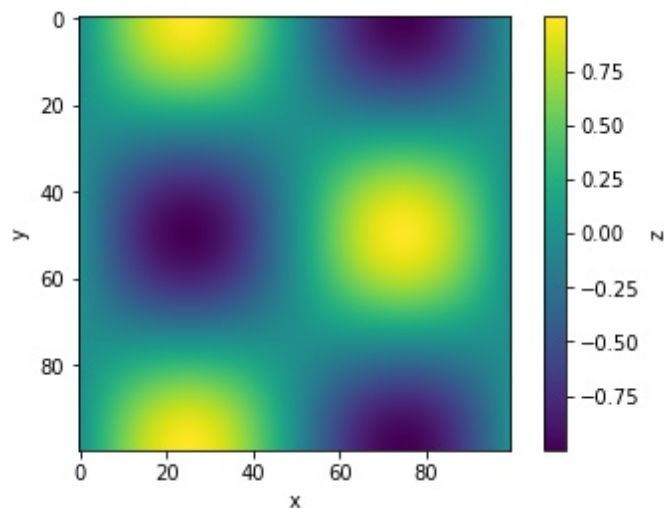
Modre vrednosti so vbočene v ekran, rumene pa izbočene.

2.1.6.2 Funkcija imshow

Dobimo čistejšo sliko tako, da uporabimo algoritem za interpolacijo.

```
In [18]: fig, ax = plt.subplots()

im = ax.imshow(Z)
im.set_interpolation('bilinear')
cb = fig.colorbar(im, ax=ax, label="z")
ax.set_xlabel("x")
ax.set_ylabel("y");
```



2.1.6.3 Funkcija contour

Konture uporabimo, da prikažemo točke z enako vrednostjo funkcije - podobno kot z izohipsami povežemo točke z isto višino na zemljevidu.

Narišimo naključno kvadratno funkcijo

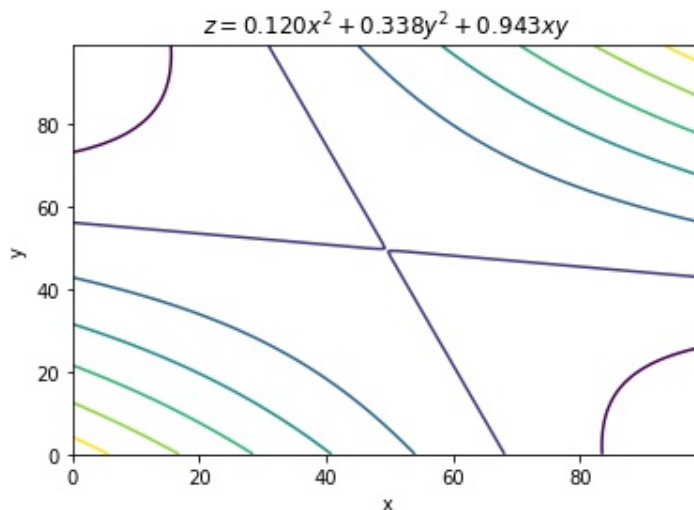
$$z = a_1x^2 + a_2y^2 + a_3xy$$

kjer koeficiente a_1, a_2, a_3 določimo naključno.

```
In [19]: def random_square_function_2D(x, y, a):
          return a[0] * x**2 + a[1] * y**2 + a[2] * x * y

a = np.random.rand(3, 1)
x1 = np.linspace(-np.pi, np.pi, 100)
y1 = np.linspace(-np.pi, np.pi, 100)
X,Y = np.meshgrid(x1, y1)
Z = random_square_function_2D(X, Y, a)

fig, ax = plt.subplots()
im = ax.contour(Z)
ax.set_title("$z = %.3f x^2 + %.3f y^2 + %.3f x y$" % (a[0], a[1], a[2]))
ax.set_xlabel("x")
ax.set_ylabel("y");
```



2.1.7 Nadzor nad velikostjo osi

V tem sklopu bomo spremenili velikost slike in nastavili razpon (interval) podatkov, ki bodo prikazani.

2.1.7.1 Domet

Za boljšo preglednost slike omejimo zgolj na domeno podatkov: ročno z uporabo `set_ylim` in `set_xlim` ali pa samodejno z `axis('tight')`.

```
In [20]: x = np.linspace(-10, 10, 100)
```

```

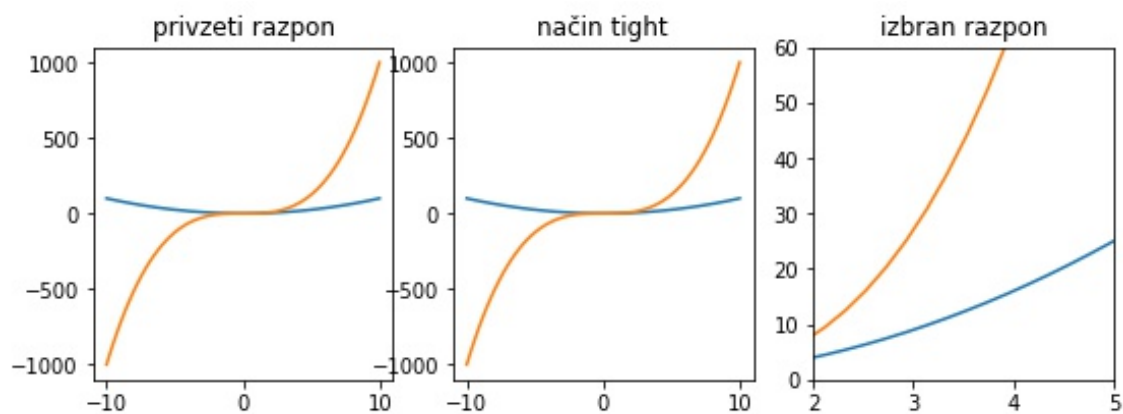
fig, axes = plt.subplots(1, 3, figsize=(9, 3))

axes[0].plot(x, x**2, x, x**3)
axes[0].set_title('privzeti razpon')

axes[1].plot(x, x**2, x, x**3)
axes[1].axis('tight')
axes[1].set_title('način tight')

axes[2].plot(x, x**2, x, x**3)
axes[2].set_ylim([0, 60])
axes[2].set_xlim([2, 5])
axes[2].set_title('izbran razpon');

```



2.1.7.2 Logaritemska lestvica

Enostavno nastavimo tudi logaritemsko naraščanje intervalov na posameznih oseh.

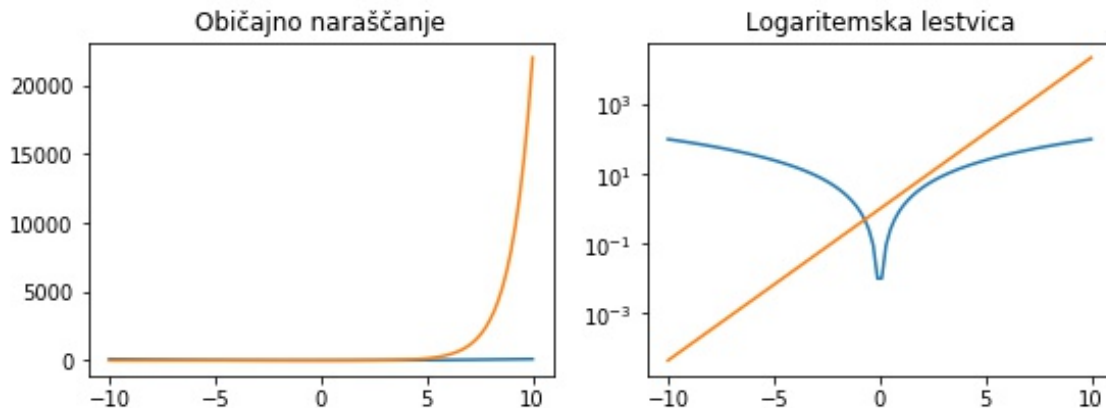
```

In [21]: fig, axes = plt.subplots(1, 2, figsize=(9, 3))

axes[0].plot(x, x**2, x, np.exp(x))
axes[0].set_title("Običajno naraščanje")

axes[1].plot(x, x**2, x, np.exp(x))
axes[1].set_yscale("log")
axes[1].set_title("Logaritemska lestvica");

```



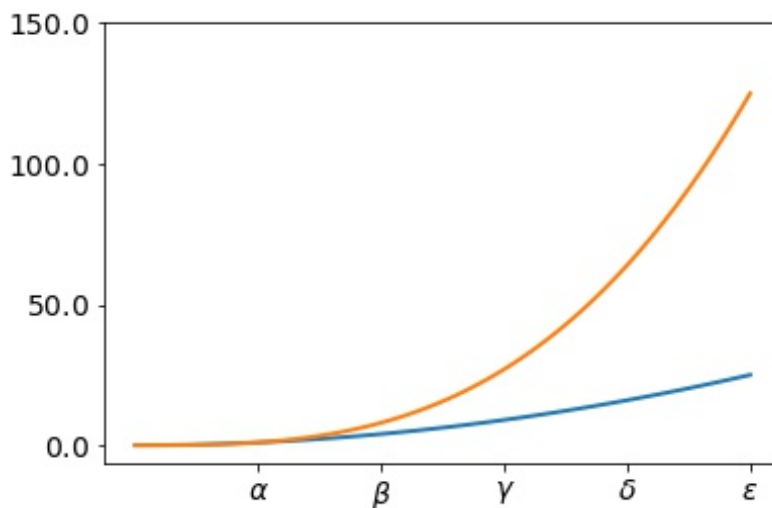
2.1.8 Nastavitev oznak na oseh

Z metodama `set_xticks` in `set_yticks` nastavimo lokacije oznak, nato pa z `set_xticklabels` in `set_yticklabels` eksplicitno določimo oznake.

```
In [22]: fig, ax = plt.subplots()
          x = np.linspace(0, 5, 100)
          ax.plot(x, x**2, x, x**3, lw=2)

          ax.set_xticks([1, 2, 3, 4, 5])
          ax.set_xticklabels(
              [r'$\alpha$', r'$\beta$', r'$\gamma$', r'$\delta$', r'$\epsilon$'],
              fontsize=14
          )

          yticks = [0, 50, 100, 150]
          ax.set_yticks(yticks)
          ax.set_yticklabels(["%.1f" % y for y in yticks], fontsize=14);
```



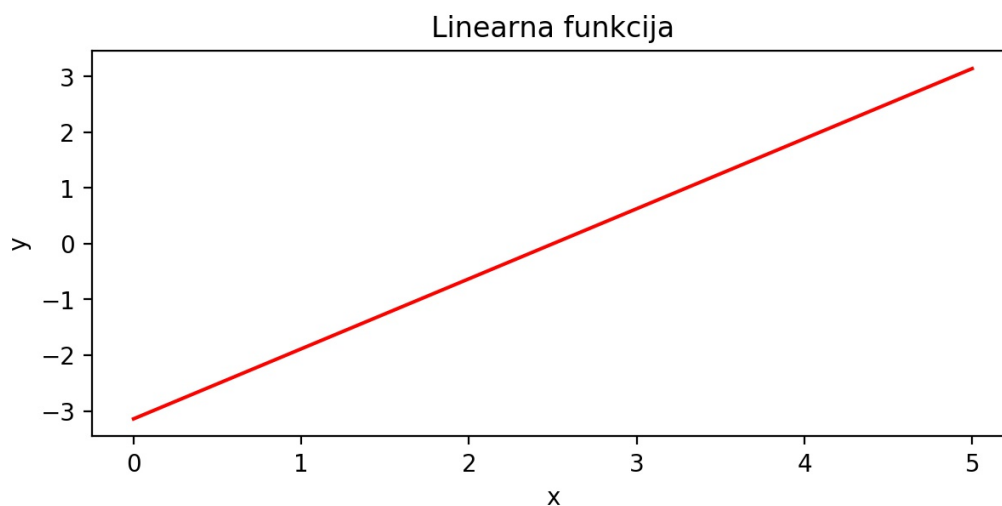
2.1.9 Velikost, razmerje in ločljivost

Velikost slike določamo s `figsize` v palcih (inčah, 1 in = 2.4 cm) ločljivost pa s parametrom `dpi` - število pik (pikselov) na palec. inch). Slednji ukaz ustvari sliko velikosti 1400x600 pik.

In [23]: `fig, axes = plt.subplots(figsize=(7, 3), dpi=200)`

```
x = np.linspace(0, 5, 100)
y = np.linspace(-np.pi, np.pi, 100)

axes.plot(x, y, 'r')
axes.set_xlabel('x')
axes.set_ylabel('y')
axes.set_title('Linearna funkcija');
```

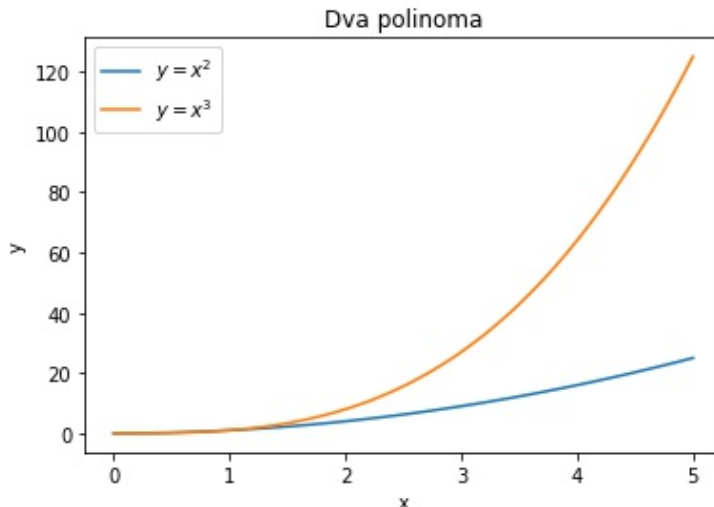


2.1.10 Legenda, oznake in naslovi

Za boljšo berljivost slike pogosto dodamo naslov, oznake osi in legendo. Na vseh mestih lahko uporabljamo \LaTeX sintakso. Kvalitetna slika vsebuje večino omenjenih elementov.

In [24]: `fig, ax = plt.subplots()`

```
ax.plot(x, x**2, label="$y = x^2$")
ax.plot(x, x**3, label="$y = x^3$")
ax.legend(loc=2) # upper left corner
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_title('Dva polinoma');
```



2.1.11 Shranjevanje slike

Za shranjevanje uporabimo metodo `savefig`, kjer lahko izbiramo med formati PNG, JPG, EPS, SVG, PGF in PDF.

```
In [25]: fig.savefig('slika.png')
```

Ločljivost nastavimo v enotah DPI.

```
In [26]: fig.savefig('slika.png', dpi=200)
```

2.2 Primer: zimske olimpijske igre, Soči 2014

Na primeru podatkov o olimpijskih igrah bomo spoznali tabelarično predstavitev podatkov (atribut-vrednost) v paketu Orange. Preizkusili bomo nekatere pogoste načine grafičnega prikaza podatkov.

```
In [1]: %matplotlib inline
%config InlineBackend.figure_formats = ['jpg']
import matplotlib
matplotlib.figure.Figure.__repr__ = lambda self: (
    f"<{self.__class__.__name__} size {self.bbox.size[0]:g}"
    f"x{self.bbox.size[1]:g} with {len(self.axes)} Axes>")

import matplotlib.image as mpimg
import matplotlib.pyplot as plt
plt.style.use('PR.mplstyle')
```

```
FileNotFoundError                                Traceback (most recent call last)

~/anaconda3/lib/python3.6/site-packages/matplotlib/style/core.py in use(style)
    112         try:
--> 113             rc = rc_params_from_file(style, use_default_template=False)
    114             _apply_style(rc)
```

```

~/anaconda3/lib/python3.6/site-packages/matplotlib/__init__.py in rc_params_from_file(fname, fa
1028     """
-> 1029     config_from_file = _rc_params_in_file(fname, fail_on_error)
1030

~/anaconda3/lib/python3.6/site-packages/matplotlib/__init__.py in _rc_params_in_file(fname, fai
944     rc_temp = {}
--> 945     with _open_file_or_url(fname) as fd:
946         try:

~/anaconda3/lib/python3.6/contextlib.py in __enter__(self)
80         try:
---> 81             return next(self.gen)
82         except StopIteration:

~/anaconda3/lib/python3.6/site-packages/matplotlib/__init__.py in _open_file_or_url(fname)
929         encoding = "utf-8"
--> 930         with io.open(fname, encoding=encoding) as f:
931             yield f

```

FileNotFoundError: [Errno 2] No such file or directory: 'PR.mplstyle'

During handling of the above exception, another exception occurred:

```

OSError                                Traceback (most recent call last)

<ipython-input-1-f11bad626864> in <module>
      8 import matplotlib.image as mpimg
      9 import matplotlib.pyplot as plt
--> 10 plt.style.use('PR.mplstyle')

~/anaconda3/lib/python3.6/site-packages/matplotlib/style/core.py in use(style)
117         "not a valid URL or path. See `style.available` for "
118         "list of available styles.")
--> 119         raise IOError(msg % style)
120
121

```

OSError: 'PR.mplstyle' not found in the style library and input is not a valid URL or path. See

2.2.1 Predstavitev podatkov

Tokrat imamo opravka s športniki, ki so nastopali na zimskih olimpijskih igrah v ruskem letovišču Soči ob Črnem morju leta 2014.

Za vsakega nastopajočega športnika so na voljo naslednji podatki (atributi):

- ime in priimek,
- starost v letih,
- datum rojstva,
- spol,
- telesna višina,
- telesna teža,
- št. osvojenih zlatih medalj,
- št. osvojenih srebrnih medalj,
- št. osvojenih bronastih medalj,
- št. vseh osvojenih medalj,
- športna panoga,
- država, katero zastopa.

Vprašanje 2-2-1 S kakšnim podatkovnim tipom bi predstavil/a vsakega od atributov?

In [2]:

Odgovor

Do sedaj smo spoznali načine za shranjevanje numeričnih podatkov, kot so cela in decimalna števila. Nenumerične podatke, kot so država ter naziv tekmovalca, ne moremo enostavno predstaviti v numerični obliki. Pomagali si bomo s knjižnjico **Orange**, ki skupaj s števili hrani naslednje tipe podatkov:

- **[c]ontinuous** ali zvezni atributi, s katerimi predstavimo številske podatke (tudi cela števila),
- **[d]iscrete** ali diskretni atributi imajo zalogo vrednosti iz končne množice. Npr. spol je element množice {moški, ženska} ali okusi sladoleda {čokolada, vanilija, jagoda}. Pomni, da za razliko od števil med elementi takih množic ne obstaja urejen vrstni red.
- **[s]tring** ali niz znakov, hrani nize znakov poljubne (končne) dolžine.

Vprašanje 2-2-2 Katerega od treh naštetih tipov podatkov bi uporabil za vsakega od atributov športnikov? Rešitev najdeš, če si ogledaš prvih nekaj vrstic datoteke **athletes.tab**.

In [2]:

Odgovor

2.2.2 Programski paket Orange

Podatke naložimo v objekt tabela, **Table**. Podatkovni tipi atributov so določeni v datoteki.

```
In [2]: from Orange.data.filter import SameValue
        from Orange.data import Table
        data = Table('podatki/athletes.tab')
```

Domena je množica imen stolpcev.

```
In [3]: data.domain
```

```
Out[3]: [age, gender, height, weight, gold_medals, silver_medals, bronze_medals, total_medals, sport, c
```

Preverimo tipe posameznih atributov.

```
In [4]: for column in data.domain.variables:
        print(column, type(column))
```

```
age ContinuousVariable
gender DiscreteVariable
height ContinuousVariable
weight ContinuousVariable
gold_medals ContinuousVariable
silver_medals ContinuousVariable
bronze_medals ContinuousVariable
total_medals ContinuousVariable
sport DiscreteVariable
country DiscreteVariable
```

Za diskretne attribute lahko dostopamo do zaloge vrednosti.

```
In [5]: data.domain['sport'].values
```

```
Out[5]: ['Alpine Skiing',
        'Biathlon',
        'Bobsleigh',
        'Cross-Country',
        'Curling',
        'Freestyle Skiing',
        'Ice Hockey',
        'Luge',
        'Nordic Combined',
        'Short Track',
        'Skeleton',
        'Ski Jumping',
        'Snowboard',
        'Speed Skating']
```

Dostopamo lahko do posameznih vrstic:

```
In [6]: print(data[0])
        print()
        print(data[1:3])
```

```
[17, Male, 1.72, 68, 0, 0, 0, 0, Freestyle Skiing, United States] {1996-04-12, Aaron Blunck}
```

```
[[27, Male, 1.85, 85, 0, 0, 0, 0, Snowboard, Italy] {1986-05-14, Aaron March},
```

```
 [21, Male, 1.78, 68, 0, 0, 0, 0, Short Track, Kazakhstan] {1992-06-30, Abzal Azhgaliyev}]
```

Dostopamo lahko do atributov posamezne vrstice. Navedeni načini so ekvivalentni za dostop do športa športnika v prvi vrstici:

```
In [7]: print(data[0, 8])
        print(data[0, data.domain['sport']])
        print(data[0, data.domain.index('sport')])
        print(data[0, 'sport']) # neposredno po imenu atributa
```

```
Freestyle Skiing
Freestyle Skiing
Freestyle Skiing
Freestyle Skiing
```

Dostopamo tudi do več stolpcev hkrati:

```
In [8]: print(data[0, ['sport', 'name', 'age']])
        print(data[0, [8, 0, -2]])
```

```
[[Freestyle Skiing, 17] {Aaron Blunck}]
[[Freestyle Skiing, 17] {Aaron Blunck}]
```

Številski podatki so shranjeni v numpyjevi tabeli znotraj objekta `Table`. Imena, datuma rojstva in države v tej matriki ne bomo našli. Zakaj?

```
In [9]: data.X
```

```
Out[9]: array([[ 17. ,  1. ,  1.72, ...,  0. ,  5. ,  79. ],
               [ 27. ,  1. ,  1.85, ...,  0. , 12. ,  36. ],
               [ 21. ,  1. ,  1.78, ...,  0. ,  9. ,  39. ],
               ...,
               [ 28. ,  0. ,  1.68, ...,  0. , 12. ,  28. ],
               [ 22. ,  1. ,  1.76, ...,  1. ,  5. ,  16. ],
               [ 19. ,  0. ,  1.58, ...,  0. ,  9. ,  30. ]])
```

2.2.3 Izbira podmnožice vrstic

Za izbiro podmnožice vrstic uporabimo filter. Naredimo objekt filter, ki vključuje pogoj ter ga pokličemo na podmnožici podatkov.

```
In [10]: # ustvarimo filter, SameValue(spremenljivka, vrednost)
        filt = SameValue(data.domain['sport'], 'Alpine Skiing')

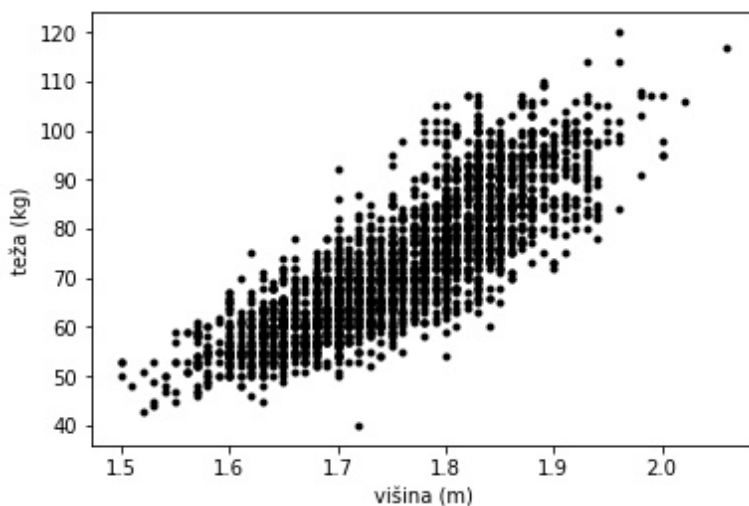
        # izberi vse alpske smučarje
        data_subset = filt(data)
        data_subset

Out[10]: [[21, Male, 1.86, 82, 0, ...] {1992-07-30, Adam Barwood},
          [18, Male, 1.70, 76, 0, ...] {1995-04-22, Adam Lamhamedi},
          [23, Male, 1.78, 80, 0, ...] {1990-09-13, Adam Zampa},
          [21, Female, 1.62, 56, 0, ...] {1992-09-28, Adeline Baud},
          [29, Male, 1.82, 80, 0, ...] {1984-09-18, Adrien Theaux},
          ...
          ]
```

2.2.4 Prikaz točk v prostoru

Poglejmo, ali sta višina in teža športnikov povezani. Za vsakega športnika narišimo točko v prostoru dveh spremenljivk - razsewni diagram (ang. *scatter plot*).

```
In [11]: plt.figure()
        x = data.X[:, 2]    # višina
        y = data.X[:, 3]    # teža
        plt.plot(x, y, "k.")
        plt.xlabel('višina (m)')
        plt.ylabel('teža (kg)');
```



Vprašanje 2-2-3 Videti je, da sta spremenljivki povezani. Ali sta višina in teža res povezani? Odgovor na to vprašanje lahko dobimo z merami korelacije. S pomočjo slednjih izmerimo, ali sta dve naključni spremenljivki povezani.

Pearsonova korelacija med spremenljivkama X in Y je definirana z naslednjim izrazom:

$$\rho = \frac{(x - \bar{x})(y - \bar{y})}{\sigma_x \sigma_y}$$

kjer sta x in y vektorja vzorcev naključnih spremenljivk X in Y , \bar{x} in \bar{y} povprečni vrednosti, σ_x , σ_y standardna odklona. Mera ρ zavzame vrednosti v intervalu $[-1, 1]$, kjer vrednost -1 pomeni, da med spremenljivki velja negativna korelacija - sta obratno sorazmerni, vrednost 1 pa da sta premo sorazmerni. Vrednost 0 nakazuje, da sta spremenljivki neodvisni.

```
In [12]: # implementiraj funkcijo, ki vrne Pearsonovo mero korelacije za vektorja x, y
def pearson(x, y):
    pass
```

```
In [13]: # preveri ali sta višina in teža povezani
pearson(x, y)
```

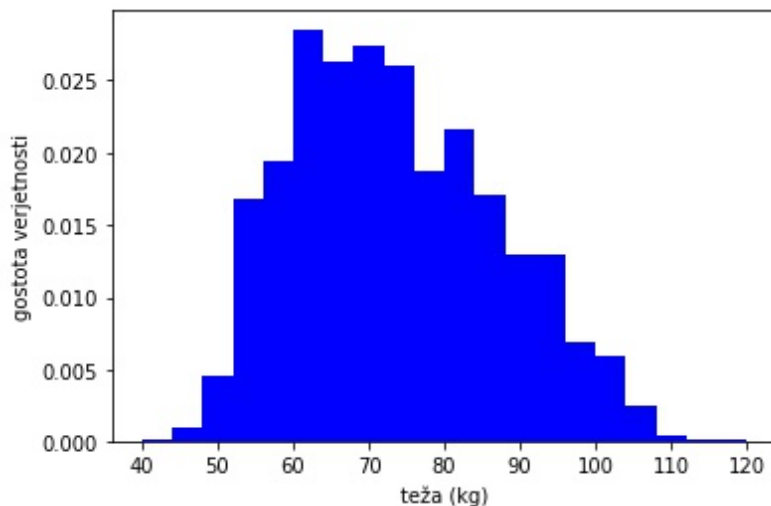
Odgovor

2.2.5 Prikaz porazdelitev

Negotovost pri opazovanju naključne spremenljivke predstavimo s funkcijo porazdelitve. Pogost način, kako dobimo oceno za porazdelitev iz podatkov je uporaba histograma - preštejemo, koliko primerov spada v interval vrednosti spremenljivke. Poglejmo primer za telesno težo.

```
In [14]: # porazdelitev tež
weights = data.X[:, 3]

plt.figure()
plt.hist(weights, density=True, bins=20, color='blue')
plt.xlabel('teža (kg)')
plt.ylabel('gostota verjetnosti');
```



Vprašanje 2-2-4 Ali so porazdelitev teže med posameznimi športi razlikuje? Kaj pa višine? Izberi športnike nekaterih športov in med njimi primerjaj porazdelitve.

```
In [15]: # Primerjaj športe po porazdelitvi tež
         # Primerjaj športe po porazdelitvi višin
```

Odgovor

2.2.6 Nagrade za doseg najvišjih mest

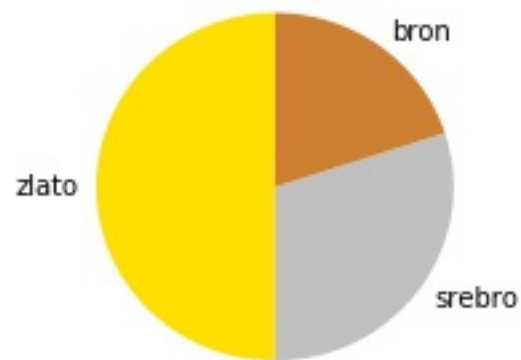
Še en način prikaza porazdelitev je tortni diagram. Prikažimo, kakšen kos pogače prinese vsaka od medalj (zlato \$25.000, srebro \$15.000, bron \$10.000).

```
In [16]: # prikaži primer slike in reprodukcija ; št medalj glede na državo
         # Nariši tortni diagram za vsako državo posebej

         # Denarni sklad; $25,000 za zlato, $15,000 za srebrno, $10,000 za bronasto medaljo
total      = 25 + 15 + 10
gold_ratio = 25 / total
silv_ratio = 15 / total
bron_ratio = 10 / total

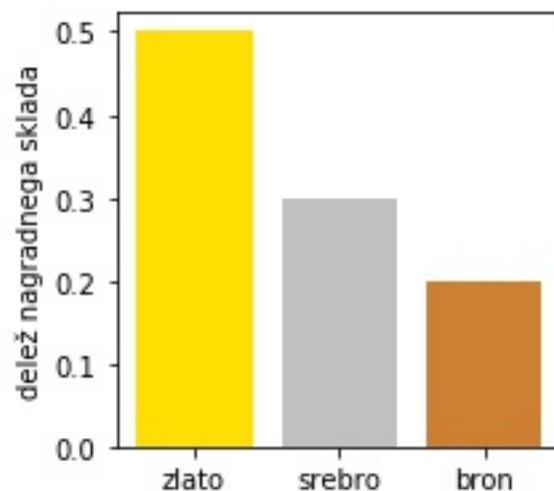
         # barve medalj
gold_color = '#FFDF00'
silv_color = '#C0C0C0'
bron_color = '#CD7F32'

plt.figure(figsize=(3, 3))
plt.pie((gold_ratio, silv_ratio, bron_ratio),
        labels=('zlato', 'srebro', 'bron', ),
        colors=(gold_color, silv_color, bron_color, ),
        startangle=90);
```

Lažje berljiv stolpični diagram:

```
In [17]: # lažje berljivi stolpični diagram
plt.figure(figsize=(3, 3))
plt.bar(range(3), height=(gold_ratio, silv_ratio, bron_ratio),
        tick_label=('zlato', 'srebro', 'bron'),
        color=(gold_color, silv_color, bron_color))
plt.ylabel('delež nagradnega sklada');
```



2.2.7 Spol udeležencev

Prikažimo še bolj informativno porazdelitev, ki pokaže število moških in ženskih udeležencev iger za posamezno državo. Najprej izračunamo porazdelitev.

```
In [18]: countries = data.domain['country'].values
gender_by_country = dict()

for country in countries:
    # Filter by countries
    filt = SameValue(data.domain['country'], country)
    data_subset = filt(data)
```

```

# Filter males
filt = SameValue(data.domain['gender'], 'Male')
data_subset_male = filt(data_subset)

# Filter females
filt = SameValue(data.domain['gender'], 'Female')
data_subset_female = filt(data_subset)

# Store gender counts
gender_by_country[country] = {
    'Male': len(data_subset_male),
    'Female': len(data_subset_female),
}

```

Nato narišemo sliko z uporabo funkcije bar:

```

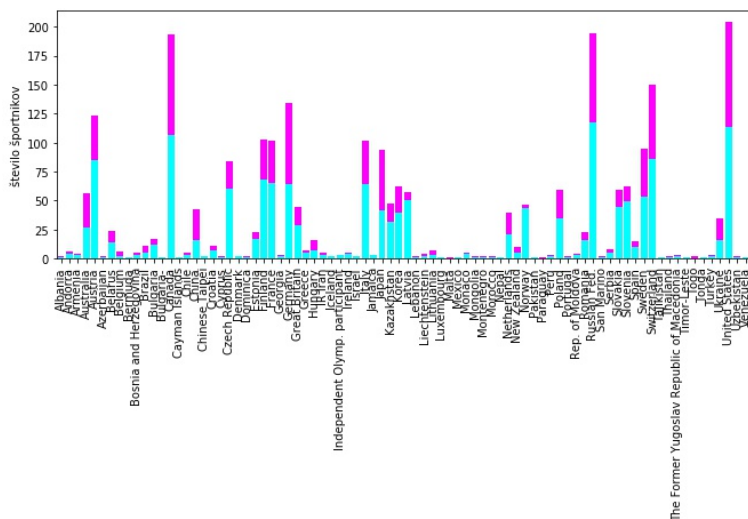
In [19]: m = [gender_by_country[country]['Male'] for country in countries]
         f = [gender_by_country[country]['Female'] for country in countries]
         x = range(len(countries))

```

```

plt.figure(figsize=(11, 4))
plt.bar(x, m, color='cyan', align='center')
plt.bar(x, f, bottom=m, color='magenta', align='center')
plt.xlim(-0.5, len(countries)-0.5)
plt.xticks(x)
plt.gca().set_xticklabels(countries, rotation=90)
plt.ylabel('število športnikov');

```



Vprašanje 2-2-5 Grafu dodaj legendo.

In [20]:

Odgovor

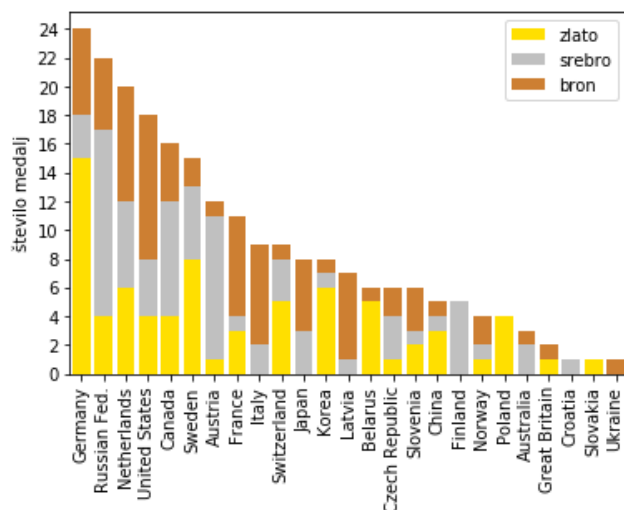
Vprašanje 2-2-6 Zgornji graf uredi tako, da bodo države urejene po številu udeležencev in dodaj legendo.

In [20]:

Odgovor

2.2.8 Najuspešnejše države

Vprašanje 2-2-7 Nariši sliko, podobno spodnji. Diagram prikazuje porazdelitev posameznih medalj po državah. Namig: najprej predpripravi podatke, nato pa nariši diagram. Zgleduj se po prejšnjih primerih.



In [20]: *# izračunaj distribucijo medalj*

In [21]: *# izriši distribucijo*

Odgovor

2.2.9 Sestavljene vizualizacije

Namen dobre vizualizacije je prava mera podatkov na danem prostoru. Ta naj ne bo prevelika, vseeno pa želimo čimbolje izkoristiti prostor. Oglejmo si primer risanja porazdelitev podatkov o višini in teži glede na posamezno športno panogo.

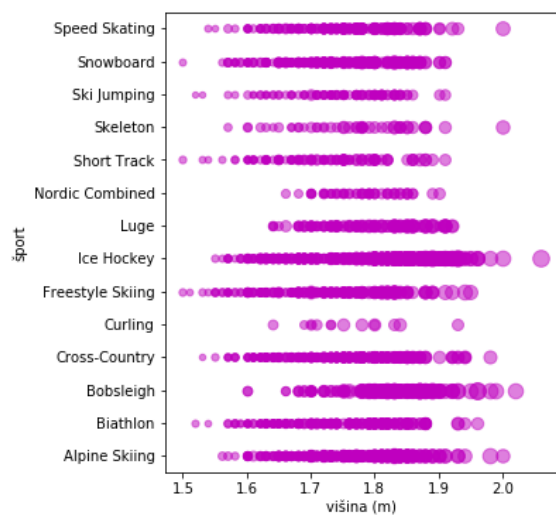
```
In [22]: # priprava podatkov
# teža in višina glede na sport; sport se nahaja v 8 stolpcu
sports = data.domain['sport'].values
weights_by_sport = dict()
heights_by_sport = dict()
ages_by_sport = dict()

for sport in sports:
    filt = SameValue(data.domain['sport'], sport)
    data_subset = filt(data)

    w = data_subset[:, data.domain.index('weight')].X.ravel()
    h = data_subset[:, data.domain.index('height')].X.ravel()
    a = data_subset[:, data.domain.index('age')].X.ravel()

    weights_by_sport[sport] = w
    heights_by_sport[sport] = h
    ages_by_sport[sport] = a
```

Vprašanje 2-2-8 Nariši sliko, podobno spodnji. Diagram prikazuje porazdelitev višine po športih. Za vsakega igralca narišimo točko, kjer bo velikost točke premo sorazmerna s težo športnika. Osi x in y bomo izkoristili tako, da na osi x narišemo višino, na osi y pa bo posamezna športna panoga.



In [23]: # napiši kodo za izris slike

Odgovor

Vprašanje 2-2-9 Uredi zgornji graf tako, da bodo športi urejeni po povprečni višini. Poizkusi tudi spreminjati količine na posameznih oseh (x, y, velikost pike).

In [24]:

Odgovor

Poglavje 3

Porazdelitve in osamelci

Verjetnostna porazdelitev P je funkcija nad naključno spremenljivko X , ki vsaki možni vrednosti spremenljivke priredi verjetnost - vrednost v intervalu $[0,1]$. Spremenljivka X je lahko zvezna, diskretna, eno- ali več dimenzionalna.

Vrednost $P(X)$ je za vsako možno vrednost spremenljivke X (celotno definicijsko območje), vsota preko definicijskega območja pa mora biti enaka 1.

Za vsako verjetnostno porazdelitev, ki jo bomo spoznali v nadaljevanju, navedemo: * definicijsko območje (t.j. kakšna je spremenljivka X), * obliko (formulo, ki vsaki vrednosti X priredi verjetnost), * parametre (konstante, ki določajo vrednosti in/ali obliko funkcije)

Vodilo: *Izbira porazdelitve za modeliranje je odvisna od narave podatkov.*

```
In [1]: %matplotlib inline
%config InlineBackend.figure_formats = ['jpg']
import matplotlib
matplotlib.figure.Figure.__repr__ = lambda self: (
    f"<{self.__class__.__name__} size {self.bbox.size[0]:g}"
    f"x{self.bbox.size[1]:g} with {len(self.axes)} Axes>")

import matplotlib.pyplot as plt
plt.style.use('PR.mplstyle')
import numpy as np
np.random.seed(42)

-----

FileNotFoundError                                Traceback (most recent call last)

~/anaconda3/lib/python3.6/site-packages/matplotlib/style/core.py in use(style)
    112         try:
--> 113             rc = rc_params_from_file(style, use_default_template=False)
    114             _apply_style(rc)

~/anaconda3/lib/python3.6/site-packages/matplotlib/__init__.py in rc_params_from_file(fname, fail_on_error)
    1028     """
-> 1029     config_from_file = _rc_params_in_file(fname, fail_on_error)
    1030
```

```

~/anaconda3/lib/python3.6/site-packages/matplotlib/__init__.py in _rc_params_in_file(fname, fail
944     rc_temp = {}
--> 945     with _open_file_or_url(fname) as fd:
946         try:

~/anaconda3/lib/python3.6/contextlib.py in __enter__(self)
80         try:
--> 81             return next(self.gen)
82         except StopIteration:

~/anaconda3/lib/python3.6/site-packages/matplotlib/__init__.py in _open_file_or_url(fname)
929         encoding = "utf-8"
--> 930         with io.open(fname, encoding=encoding) as f:
931             yield f

```

FileNotFoundError: [Errno 2] No such file or directory: 'PR.mplstyle'

During handling of the above exception, another exception occurred:

```

OSError                                Traceback (most recent call last)

<ipython-input-1-9e9777efd08e> in <module>
7
8 import matplotlib.pyplot as plt
----> 9 plt.style.use('PR.mplstyle')
10 import numpy as np
11 np.random.seed(42)

~/anaconda3/lib/python3.6/site-packages/matplotlib/style/core.py in use(style)
117         "not a valid URL or path. See `style.available` for "
118         "list of available styles.")
--> 119         raise IOError(msg % style)
120
121

```

OSError: 'PR.mplstyle' not found in the style library and input is not a valid URL or path. See

3.1 Gaussova (normalna) porazdelitev

Normalna (ali Gaussova) porazdelitev je porazdelitev na celotnem območju realnih števil. Je ena od najpogostejših porazdelitev, ki se uporablja v praksi, saj ima veliko podatkov znano, zvonasto obliko. Funkcija je *simetrična* in podana z dvema parametroma, sredino in varianco.

Tip spremenljivke: eno- ali več dimenzionalna, zvezna.

Definicijsko območje: $(-\infty, +\infty)$

Oblika:

$$P(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Parametri: * μ sredina/upanje * σ^2 varianca

In [2]: `from scipy.stats import multivariate_normal as mvn`

```
# Parametri določajo obliko funkcije
mu      = 0      # sredina
sigma2  = 1      # varianca

n = 500 # velikost vzorca
sample = mvn.rvs(mu, sigma2, size=n) # naključen vzorec n primerov

xr = np.linspace(-5, 5, 100) # interval X
P = [mvn.pdf(x, mu, sigma2) for x in xr] # porazdelitvena funkcija

# Histogram - porazdelitev naključnih VZORCEV x glede na P(x)
plt.figure(figsize=(9, 3))
plt.subplot(1, 2, 1)
plt.title("vzorec")
plt.hist(sample) #
plt.xlabel("X")
plt.ylabel("število primerov")

# Graf porazdelitvene funkcije
plt.subplot(1, 2, 2)
plt.title("graf porazdelitve")
plt.plot(xr, P) # nariši P(x)
plt.ylabel("P(x)")
plt.xlabel("X");
```

NameError

Traceback (most recent call last)

```
<ipython-input-2-9a8f12c221c3> in <module>
      8 sample = mvn.rvs(mu, sigma2, size=n) # naključen vzorec n primerov
      9
----> 10 xr = np.linspace(-5, 5, 100) # interval X
      11 P = [mvn.pdf(x, mu, sigma2) for x in xr] # porazdelitvena funkcija
      12
```

NameError: name 'np' is not defined

3.1.1 Učenje parametrov

V praksi resničnih vrednosti parametrov ne poznamo. *Parametrov se naučimo iz vzorca*. Prednosti postopka so, da nato lahko sklepamo o novih vzorcih, t.j., vsaki možni vrednosti spremenljivke priredimo verjetnost.

Imamo vzorec naključne spremenljivke X velikosti n .

$$X_1, X_2, \dots, X_n$$

Za normalno porazdelitev dobimo *oceno* za parametre na naslednji način:

$$\mu = E[X_i] = \bar{X}$$

$$\sigma^2 = \frac{n-1}{n} E[(X_i - \bar{X})^2] = \frac{n-1}{n} \text{var}[x]$$

Vrednost μ je povprečje vzorca. Vrednost σ^2 je popravljena varianca vzorca.

Ocenimo parametre iz vzorca:

```
In [3]: mu_fit = np.mean(sample)
        sigma2_fit = (n-1)/n * np.var(sample)

        mu_fit, sigma2_fit
```

NameError

Traceback (most recent call last)

```
<ipython-input-3-2546e0e7037a> in <module>
----> 1 mu_fit = np.mean(sample)
      2 sigma2_fit = (n-1)/n * np.var(sample)
      3
      4 mu_fit, sigma2_fit
```

NameError: name 'np' is not defined

Ocenjeni vrednosti parametrov sta podobni resničnim vrednostim ($\mu = 0$, $\sigma^2 = 1$).

Na eni sliki primerjamo porazdelitev z naučenimi parametri s pravo porazdelitvijo:

```
In [4]: P_fit = [mvn.pdf(x, mu_fit, sigma2_fit) for x in xr]

        plt.figure()
        plt.hist(sample, label="vzorec", normed=True)
        plt.plot(xr, P, label="P(X) resnična", linewidth=2.0)
        plt.plot(xr, P_fit, label="P(X) ocenjena", linewidth=2.0)
        plt.legend();
```

NameError

Traceback (most recent call last)

```
<ipython-input-4-444fba6df4f3> in <module>
----> 1 P_fit = [mvn.pdf(x, mu_fit, sigma2_fit) for x in xr]
      2
```



```

3 plt.figure()
4 plt.hist(sample,      label="vzorec", normed=True)
5 plt.plot(xr, P,       label="P(X) resnična", linewidth=2.0)

```

```
NameError: name 'xr' is not defined
```

Vprašanje 3-1-1 Preveri, kako se natančnost ocene parametrov spreminja z velikostjo vzorca n .

In [5]:

Odgovor

3.2 Studentova porazdelitev

Studentova porazdelitev (ali t-porazdelitev) je porazdelitev na celotnem območju realnih števil. Njena oblika je simetrična in podobna normalni porazdelitvi. Je manj občutljiva na *osamelce v majhnih vzorcih*.

Tip spremenljivke: eno-dimenzionalna, zvezna.

Definicijsko območje: $x \in (-\infty, +\infty)$

Oblika:

$$P(x) = \frac{\Gamma[(\nu+1)/2]}{\sqrt{\nu\pi}\Gamma(\nu/2)} \left(1 + \frac{x^2}{\nu}\right)^{-(\nu+1)/2}$$

Parametri: * ν število prostostnih stopenj

In [5]: `from scipy.stats import t as student`

```

# Parametri določajo obliko funkcije
nu = 2 # prostostne stopnje

n = 8 # velikost vzorca
sample = student.rvs(nu, size=n) # naključen vzorec n primerov spremenljivke

xr = np.linspace(-5, 5, 100) # interval X
P = [student.pdf(x, nu) for x in xr] # porazdelitvena funkcija

# Histogram - porazdelitev naključnih VZORCEV x glede na P(x)
plt.figure(figsize=(9, 3))
plt.subplot(1, 2, 1)
plt.title("Vzorec")
plt.hist(sample) #
plt.xlabel("X")
plt.ylabel("Število primerov")

# Graf porazdelitvene funkcije
plt.subplot(1, 2, 2)
plt.title("Graf porazdelitve")
plt.plot(xr, P) # nariši P(x)

```

```
plt.ylabel("P(x)")
plt.xlabel("X");

-----

NameError                                Traceback (most recent call last)

<ipython-input-5-5b4ee6bcbbcb> in <module>
      7 sample = student.rvs(nu, size=n) # naključen vzorec n primerov spremenljivke
      8
----> 9 xr = np.linspace(-5, 5, 100) # interval X
     10 P = [student.pdf(x, nu) for x in xr] # porazdelitvena funkcija
     11

NameError: name 'np' is not defined
```

3.2.1 Učenje parametrov iz vzorca

Večina porazdelitve v knjižnici `scipy` vsebuje funkcijo `fit`, ki izračuna najverjetnejše vrednosti parametrov porazdelitve glede na vzorec.

Na eni sliki primerjamo porazdelitev z naučenimi parametri s pravo porazdelitvijo

```
In [6]: pars = student.fit(sample)
        P_fit = [student.pdf(x, *pars) for x in xr ]

plt.figure()
plt.hist(sample, label="vzorec", normed=True)
plt.plot(xr, P, label="P(X) resnična", linewidth=2.0)
plt.plot(xr, P_fit, label="P(X) ocenjena", linewidth=2.0)
plt.legend();

-----

NameError                                Traceback (most recent call last)

<ipython-input-6-ff3cd866462a> in <module>
      1 pars = student.fit(sample)
----> 2 P_fit = [student.pdf(x, *pars) for x in xr ]
      3
      4 plt.figure()
      5 plt.hist(sample, label="vzorec", normed=True)

NameError: name 'xr' is not defined
```

Vprašanje 3-1-2 Generiraj vzorec z manjšim številom (do 20) vzorcev iz normalne porazdelitve. Primerjaj ocene porazdelitve s pomočjo normalne in Studentove porazdelitve. Katera porazdelitev bolje oceni resnično porazdelitev?

```
In [7]: # Primerjaj Normalno in Studentovo porazdelitev pri majhnem vzorcu
# ...
```

Odgovor

3.3 Porazdelitev Beta

Beta porazdelitev je porazdelitev spremenljivke na *omejenem intervalu* $[0, 1]$. Njena oblika je zelo prilagodljiva, lahko ima namreč en ali dva *maksimuma*. Porazdelitev lahko prevedemo na poljuben interval $[a, b]$ s seštevanjem (translacija) in množenjem (širjenje/ožanje) intervala.

Tip spremenljivke: x , enodimenzionalna, zvezna, na omejenem intervalu.

Definicijsko območje: $x \in [0, 1]$

Oblika:

$$P(x) = \frac{1}{B(\alpha, \beta)} x^{\alpha-1} (1-x)^{\beta-1}$$

Parametri: a, b

```
In [8]: from scipy.stats import beta

# Parametri določajo obliko funkcije
a, b = (3, 2) # parametra a, b

n = 100 # velikost vzorca
sample = beta.rvs(a, b, size=n) # naključen vzorec n primerov spremenljivke

xr = np.linspace(0, 1, 100) # interval X
P = [beta.pdf(x, a, b) for x in xr] # porazdelitvena funkcija

# Histogram - porazdelitev naključnih VZORCEV x glede na P(x)
plt.figure(figsize=(9, 3))
plt.subplot(1, 2, 1)
plt.title("Vzorec")
plt.hist(sample) #
plt.xlabel("X")
plt.ylabel("Število primerov")

# Graf porazdelitvene funkcije
plt.subplot(1, 2, 2)
plt.title("Graf porazdelitve")
plt.plot(xr, P) # nariši P(x)
plt.ylabel("P(x)")
plt.xlabel("X");

-----

NameError                                Traceback (most recent call last)

<ipython-input-8-0e4cc5111124> in <module>
      7 sample = beta.rvs(a, b, size=n) # naključen vzorec n primerov spremenljivke
```

```

      8
----> 9 xr = np.linspace(0, 1, 100)          # interval X
     10 P  = [beta.pdf(x, a, b) for x in xr]  # porazdelitvena funkcija
     11

```

```
NameError: name 'np' is not defined
```

Vprašanje 3-1-3 Spreminjaj parametra a in b . Kako se oblika funkcije spreminja?

In [9]:

Odgovor

3.3.1 Učenje parametrov iz vzorca

Tudi za učenje parametrov porazdelitve Beta uporabimo funkcijo `fit`.

Na eni sliki primerjamo porazdelitev z naučenimi parametri s pravo porazdelitvijo.

```

In [9]: parameters = beta.fit(sample)
       P_fit = [beta.pdf(x, *parameters) for x in xr ]

       plt.figure()
       plt.hist(sample,      label="Vzorec", normed=True)
       plt.plot(xr, P,      label="P(X) resnična", linewidth=2.0)
       plt.plot(xr, P_fit, label="P(X) ocenjena", linewidth=2.0)  # ocenjena porazdelitev je model
       plt.legend();

```

```
-----
NameError                                Traceback (most recent call last)
```

```

<ipython-input-9-1dd1df6971b7> in <module>
      1 parameters = beta.fit(sample)
----> 2 P_fit = [beta.pdf(x, *parameters) for x in xr ]
      3
      4 plt.figure()
      5 plt.hist(sample,      label="Vzorec", normed=True)

```

```
NameError: name 'xr' is not defined
```

Vprašanje 3-1-4 Spreminjaj parametra a in b ter velikost vzorca n . Kako se spreminja kakovost prilaganja?

In [10]:

Odgovor

3.4 Primer: iskanje neslanih šal

Tokrat si bomo ogledali zbirko podatkov Jester, ki je dokaj podobna tisti pri domači nalogi. Gre za zbirko 100 šal (vicev), ki jih je ocenilo 23500 uporabnikov z oceno -10 (porazno) do 10 (odlično). Ocena je torej zvezna spremenljivka.

Naš glavni cilj bo modeliranje statistik v podatkovni zbirki z uporabo znanih porazdelitev. To nam bo omogočalo, da **med šalami poiščemo osamelce** in ocenimo njihovo statistično značilnost - verjetnost, da gre za osamelca ali ne.

Začnimo z naključno šalo iz podatkovne zbirke:

A mechanical, electrical and a software engineer from Microsoft were driving through the desert when the car broke down. The mechanical engineer said "It seems to be a problem with the fuel injection system, why don't we pop the hood and I'll take a look at it." To which the electrical engineer replied, "No I think it's just a loose ground wire, I'll get out and take a look." Then, the Microsoft engineer jumps in. "No, no, no. If we just close up all the windows, get out, wait a few minutes, get back in, and then reopen the windows everything will work fine."

```
In [1]: %matplotlib inline
%config InlineBackend.figure_formats = ['jpg']
import matplotlib
matplotlib.figure.Figure.__repr__ = lambda self: (
    f"<{self.__class__.__name__} size {self.bbox.size[0]:g}"
    f"x{self.bbox.size[1]:g} with {len(self.axes)} Axes>")

import matplotlib.pyplot as plt
plt.style.use('PR.mplstyle')
import numpy as np
```

```
-----

FileNotFoundError                                Traceback (most recent call last)

~/anaconda3/lib/python3.6/site-packages/matplotlib/style/core.py in use(style)
    112         try:
--> 113             rc = rc_params_from_file(style, use_default_template=False)
    114             _apply_style(rc)

~/anaconda3/lib/python3.6/site-packages/matplotlib/__init__.py in rc_params_from_file(fname, fail_on_error)
   1028     """
-> 1029     config_from_file = _rc_params_in_file(fname, fail_on_error)
   1030

~/anaconda3/lib/python3.6/site-packages/matplotlib/__init__.py in _rc_params_in_file(fname, fail_on_error)
    944     rc_temp = {}
--> 945     with _open_file_or_url(fname) as fd:
    946         try:

~/anaconda3/lib/python3.6/contextlib.py in __enter__(self)
     80         try:
```

```

---> 81         return next(self.gen)
      82     except StopIteration:

~/anaconda3/lib/python3.6/site-packages/matplotlib/__init__.py in _open_file_or_url(fname)
929         encoding = "utf-8"
--> 930         with io.open(fname, encoding=encoding) as f:
931             yield f

```

FileNotFoundError: [Errno 2] No such file or directory: 'PR.mplstyle'

During handling of the above exception, another exception occurred:

```

OSError                                Traceback (most recent call last)

<ipython-input-1-779ea8d3610b> in <module>
      7
      8 import matplotlib.pyplot as plt
----> 9 plt.style.use('PR.mplstyle')
     10 import numpy as np

~/anaconda3/lib/python3.6/site-packages/matplotlib/style/core.py in use(style)
117         "not a valid URL or path. See `style.available` for "
118         "list of available styles.")
--> 119         raise IOError(msg % style)
     120
     121

```

OSError: 'PR.mplstyle' not found in the style library and input is not a valid URL or path. See

Podatki so matrika velikosti 23500×100 z zveznimi vrednostmi. Vrednost 99 predstavlja neznano vrednost, takih vrednosti zato ne smemo upoštevati.

```

In [2]: X = np.genfromtxt('podatki/jester-data.csv', delimiter=',')[:, 1:]
      X[np.where(X == 99)] = float("nan") # neznanih vrednosti ne smemo upoštevati

print("velikost:", X.shape)
print("skupno število ocen:", X.size - np.sum(np.isnan(X)))

```

```

-----
NameError                                Traceback (most recent call last)

```

```

<ipython-input-2-2286ff858560> in <module>
----> 1 X = np.genfromtxt('podatki/jester-data.csv', delimiter=',')[:, 1:]
      2 X[np.where(X == 99)] = float("nan") # neznanih vrednosti ne smemo upoštevati
      3
      4 print("velikost:", X.shape)

```

```
5 print("skupno število ocen:", X.size - np.sum(np.isnan(X)))
```

```
NameError: name 'np' is not defined
```

Poglejmo, kakšna je porazdelitev vseh veljavnih ocen.

```
In [3]: data = X[np.isnan(X) == False]
        plt.hist(data, bins=100)
        plt.xlabel("x - Ocena šale")
        plt.ylabel("P(x)");
```

```
-----
NameError                                Traceback (most recent call last)
```

```
<ipython-input-3-3a0ac0860547> in <module>
----> 1 data = X[np.isnan(X) == False]
      2 plt.hist(data, bins=100)
      3 plt.xlabel("x - Ocena šale")
      4 plt.ylabel("P(x)");
```

```
NameError: name 'X' is not defined
```

Vidimo, da je večina ocen nevtralnih (okoli 0), veliko pozitivnih (med 3 in 10) ter nekaj zelo slabih (-10). Najmanj je srednje slabih (-9 do -1). Navkljub temu ima ta porazdelitev naslednje težave: * Vzorec ni nepristranski. Vsak uporabnik je ocenil različno število šal. * Porazdelitev ne spominja na nobeno od znanih.

Kako bi primerjali šale glede na njihove ocene?

Poglejmo najprej, koliko veljavnih ocen je prejela vsaka od šala:

```
In [4]: (np.isnan(X) == False).sum(axis=0) # vsota po posameznih šalah
```

```
-----
NameError                                Traceback (most recent call last)
```

```
<ipython-input-4-12b9b805f100> in <module>
----> 1 (np.isnan(X) == False).sum(axis=0) # vsota po posameznih šalah
```

```
NameError: name 'np' is not defined
```

Vsaka šala je dobila nekaj tisoč ocen, kar zadostuje za statistično primerjavo.

Zamislimo si dve novi naključni spremenljivki:

- X povprečje ocen posamezne šale,
- Y varianca ocen posamezne šale.

Pomembno: spremenljivki sta izpeljani iz dveh izračunljivih količin. Spremenljivki X in Y nista parametra normalne porazdelitve!

Za vsako od navedenih spremenljivk X in Y imamo torej vzorec velikosti 100, po en primer za vsako šalo. Pri izračunu pazimo, da preskočimo neznane vrednosti:

```
In [5]: means      = []
        variances = []
        for i in range(X.shape[1]):
            s = np.mean(X[:, i][np.isnan(X[:, i]) == False])
            v = np.var(X[:, i][np.isnan(X[:, i]) == False])
            means.append(s)
            variances.append(v)
```

NameError

Traceback (most recent call last)

```
<ipython-input-5-f72e975a5add> in <module>
      1 means      = []
      2 variances = []
----> 3 for i in range(X.shape[1]):
      4     s = np.mean(X[:, i][np.isnan(X[:, i]) == False])
      5     v = np.var(X[:, i][np.isnan(X[:, i]) == False])
```

NameError: name 'X' is not defined

Vprašanje 3-2-1 Kakšna je interpretacija spremenljivk X in Y ? Kaj pomeni, če ima šala visoko varianco med vsemi ocenami? Kaj pomeni, če ima šala visoko povprečno oceno?

Odgovor

Izpišimo nekaj najboljše, najslabše ocenjenih šal ter nekaj takih z visoko oz. nizko varianco. Za zabavo jih lahko prebereš in primerjaš, n. pr., odpri datoteko podatki/jokes/init1.html:

A man visits the doctor. The doctor says "I have bad news for you. You have cancer and Alzheimer's disease". The man replies "Well, thank God I don't have cancer!"

```
In [6]: n = 3
        for data, name in [(means, "Povprečje (X)"), (variances, "Varianca (Y)")]:
            inxs = np.argsort(data)[:n]
            print("Kriterij: %s" % name)
            print("\tSpodnjih %d:" % n)
            for i in inxs:
                print("\t\tšala %d, povp.: %.2f, var.: %.2f" % (i+1, means[i], variances[i]))

            inxs = np.argsort(data)[::-1][:n]
            print("\tZgornjih %d:" % n)
            for i in inxs:
                print("\t\tšala %d, povp.: %.2f, var.: %.2f" % (i+1, means[i], variances[i]))
            print()
```

NameError

Traceback (most recent call last)


```

<ipython-input-6-755b10deed5b> in <module>
      1 n = 3
      2 for data, name in [(means, "Povprečje (X)", (variances, "Varianca (Y)"))]:
----> 3     inxs = np.argsort(data)[:n]
      4     print("Kriterij: %s" % name)
      5     print("\tSpodnjih %d:" % n)

```

```
NameError: name 'np' is not defined
```

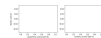
Narišimo še porazdelitvi vzorcev X in Y .

```

In [7]: plt.figure(figsize=(9, 3))
      plt.subplot(1, 2, 1)
      plt.hist(means, normed=False, bins=12)
      plt.xlabel("povprečna ocena šale (X)")
      plt.ylabel("število vzorcev")
      plt.text(-3, 2, "< osamelci?", rotation=90, verticalalignment="bottom", color="red")

      plt.subplot(1, 2, 2)
      plt.hist(variances, normed=False, bins=12)
      plt.xlabel("varianca ocene šale (Y)");

```



Tole izgleda že bolje. Večina šal je torej v povprečju ocenjenih pozitivno, zelo malo je negativnih. Porazdelitvi spominjata na znane porazdelitve, kjer je večina primerov (šal) porazdeljenih okoli srednje vrednosti, manj pa je ekstremnih vrednosti.

Poglejmo za trenutek porazdelitev povprečnih ocen. Izgleda, da imamo nekaj **osamelcev** - zelo slabih šal, ocenjenih z manj kot $X = -2$. Kako pomenljiv je padec od $X = -2$ navzdol? Da bi odgovorili na to vprašanje, spoznajmo osnove modeliranja podatkov s pomočjo verjetnostnih porazdelitev.

Povprečna ocena izgleda normalno porazdeljena. Kakšni so najbolj verjetni parametri porazdelitve?

```
In [8]: from scipy.stats import multivariate_normal as mvn
```

```
data = means
```

```

# Ocenimo parametre normalne (Gaussove) porazdelitve
n = len(data)

```

```

mu = np.mean(data)                # ocena sredine
sigma2 = (n-1)/n * np.var(data) # ocena variance

plt.figure()
counts, bins, _ = plt.hist(data, normed=True, label="vzorec", bins=10) # dobimo razpon
pdf = [mvn.pdf(x, mu, sigma2) for x in bins] # pdf: [p]robability
plt.plot(bins, pdf, "-", label="model", linewidth=2.0)
plt.xlabel("povprečna ocena (X)")
plt.ylabel("P(X)")

plt.legend(loc=2);

-----

NameError                                Traceback (most recent call last)

<ipython-input-8-917593c22610> in <module>
      5 # Ocenimo parametre normalne (Gaussove) porazdelitve
      6 n = len(data)
----> 7 mu = np.mean(data)                # ocena sredine
      8 sigma2 = (n-1)/n * np.var(data) # ocena variance
      9

NameError: name 'np' is not defined

```

Na oko lahko ocenimo, da se porazdelitev kar dobro ujema z vzorcem. Kako statistično značilne so šale, ki imajo vrednost meritve manjšo od $X = -2.0$? *Kako nenavadno slabe so v resnici te šale?* Za odgovor na to vprašanje bomo izračunali t.i. *p-vrednost*. S pomočjo *p-vrednosti* ocenimo *nenavadnost* meritve, v našem primeru povprečne ocene šale.

Definicija. *P-vrednost* je verjetnost, da pri vzorčenju ene vrednosti naključne spremenljivke dobimo dano ali manjšo (oz. večjo) vrednost.

Definicijo si najlažje ogledamo grafično. Oglejmo si funkcijo porazdelitve, dobljeno z ocenjenima parametroma μ in σ^2 .

```

In [9]: # Meritev, ki bi jo radi statistično ocenili
        qx = -2

        # Izračunamo P(x) za dovolj velik interval
        xr = np.linspace(-5, 5, 100)
        width = xr[1] - xr[0] # širina intervala
        Px = [mvn.pdf(x, mu, sigma2) * (xr[1]-xr[0]) for x in xr]

        # Vse vrednosti, ki so manjše ali enake od qx
        ltx = xr[xr <= qx]

        # Množimo s širino intervala, da dobimo ploščino pod krivuljo
        P_ltx = [mvn.pdf(x, mu, sigma2) * width for x in ltx]

        # p-vrednost: ploščina pod krivuljo P(x) za vse vrednosti, manjše od qx

```

```

p_value = np.sum(P_ltx)

# Graf funkcije
plt.figure()
plt.plot(xr, Px, linewidth=2.0)
plt.fill_between(ltx, 0, P_ltx, alpha=0.2)
plt.text(qx, mvn.pdf(qx, mu, sigma2) * width,
        "p=%f" % p_value,
        horizontalalignment="right",
        verticalalignment="center",
        )

plt.xlabel("X - povprečna ocena šale.")
plt.ylabel("P(X)")
plt.legend()

# Poglejmo, ali je meritev statistično značilna pri danem pragu alpha (0.05, 0.01, 0.001 ... )
alpha = 0.05
if p_value < alpha:
    sig = "JE"
else:
    sig = "NI"

# Rezultat statističnega testa
print("Verjetnost šale z oceno %.3f ali manj: " % qx + "%.3f" % (100 * p_value) + " %")
print("Nenavadnost šale %s statistično značilna (prag = %.3f" % (sig, 100*alpha), "%)")

-----

NameError                                Traceback (most recent call last)

<ipython-input-9-e30847b2d35c> in <module>
      4
      5 # Izračunamo P(x) za dovolj velik interval
----> 6 xr      = np.linspace(-5, 5, 100)
      7 width = xr[1] - xr[0]          # sirina intervala
      8 Px = [mvn.pdf(x, mu, sigma2) * (xr[1]-xr[0])   for x in xr]

NameError: name 'np' is not defined

```

Sedaj lahko za vsako ekstremno vrednost v podatkih (bodisi visoko ali nizko) statistično ocenimo vrednost njene nenavadnosti. Pri postavljenem pragu npr. $\alpha = 0.05$ lahko sprejmemo odločitev, ali je neka meritev osamelec ali ne.

Vprašanje 3-2-2 Izpiši vse šale osamelce, katerih povprečna ocena X je statistično značilna, pri pragu $\alpha = 0.05$. Poišči tudi osamelce med *dobro ocenjenimi* šalami.

In [10]:

Odgovor

Vprašanje 3-2-3 Poizkusi porazdelitev modelirati z drugimi porazdelitvami (Student, Beta). Je katera od teh porazdelitev bolj primerna?

In [10]:

Odgovor

Vprašanje 3-2-4 Ponovi analizo za spremenljivko Y - varianca ocen šale. Odgovori na vprašanja: * Katera od porazdelitev (normalna, Student, Beta) se najbolj prilega vzorcu? * Katere so statistično značilne šale (z visoko ali nizko varianco)? * Kaj pomeni, če ima šala visoko ali nizko vrednost Y ?

In [10]:

Odgovor

Poglavje 4

Odkrivanje skupin

Z metodami *nenadzorovanega modeliranja* odkrivamo strukturo v podatkih, ki jo ne poznamo. Osrednja predpostavka tovrstnih metod je, da v podatkih obstajajo podmnožice podobnih primerov.

```
In [1]: import numpy as np
        np.random.seed(42)
```

```
%matplotlib inline
%config InlineBackend.figure_formats = ['jpg']
import matplotlib
matplotlib.figure.Figure.__repr__ = lambda self: (
    f"<{self.__class__.__name__} size {self.bbox.size[0]:g}"
    f"x{x{self.bbox.size[1]:g} with {len(self.axes)} Axes>")

import matplotlib.pyplot as plt
plt.style.use('PR.mplstyle')
```

```
-----
FileNotFoundError                                Traceback (most recent call last)

~/anaconda3/lib/python3.6/site-packages/matplotlib/style/core.py in use(style)
    112         try:
--> 113             rc = rc_params_from_file(style, use_default_template=False)
    114             _apply_style(rc)

~/anaconda3/lib/python3.6/site-packages/matplotlib/__init__.py in rc_params_from_file(fname, fail_on_error)
    1028     """
-> 1029     config_from_file = _rc_params_in_file(fname, fail_on_error)
    1030

~/anaconda3/lib/python3.6/site-packages/matplotlib/__init__.py in _rc_params_in_file(fname, fail_on_error)
    944     rc_temp = {}
--> 945     with _open_file_or_url(fname) as fd:
    946         try:
```

```
~/anaconda3/lib/python3.6/contextlib.py in __enter__(self)
    80         try:
--> 81             return next(self.gen)
    82         except StopIteration:

~/anaconda3/lib/python3.6/site-packages/matplotlib/__init__.py in _open_file_or_url(fname)
    929         encoding = "utf-8"
--> 930         with io.open(fname, encoding=encoding) as f:
    931             yield f
```

FileNotFoundError: [Errno 2] No such file or directory: 'PR.mplstyle'

During handling of the above exception, another exception occurred:

```

OSError                                Traceback (most recent call last)

<ipython-input-1-4b7405d91cad> in <module>
     10
     11 import matplotlib.pyplot as plt
--> 12 plt.style.use('PR.mplstyle')

~/anaconda3/lib/python3.6/site-packages/matplotlib/style/core.py in use(style)
    117         "not a valid URL or path. See `style.available` for "
    118         "list of available styles.")
--> 119         raise IOError(msg % style)
    120
    121
```

OSError: 'PR.mplstyle' not found in the style library and input is not a valid URL or path. See

4.1 Metoda voditeljev

Metoda K voditeljev (ang. *K-means clustering*) sodi med enostavnejše in pogosto uporabljene metode nenadzorovanega algoritma. Pomembna prednost je tudi računska učinkovitost. Dodatne podrobnosti najdeš v literaturi.

Vprašanje 4-1-1 Implementiraj algoritem za odkrivanje skupin z metodo K voditeljev. Pomagaj si z naslednjo psevdokodo:

```

Naključno izberi *K* točk - centrov.
**ponavljaj**
    Vsaki točki določi najbližji center.
    Izračunaj nove centre - središča pripadajočih skupin.
**dokler** se centri ne spreminjajo več.
```

Razdaljo med točkama $\vec{x} = (x_1, x_2, \dots, x_p)$ in $\vec{y} = (y_1, y_2, \dots, y_p)$ izračunaj s pomočjo evklidske razdalje:

$$\|\vec{x} - \vec{y}\| = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \dots + (x_p - y_p)^2}$$

In [2]:

Odgovor

Vprašanje 4-1-2 Kakšna je časovna zahtevnost algoritma v odvisnosti od števila primerov in števila atributov?

prostor za odgovor

Odgovor

In [2]: `class KMeans:`

```
def __init__(self, k=10, max_iter=100):
    """
    Initialize KMeans clustering model.

    :param k
        Number of clusters.
    :param max_iter
        Maximum number of iterations.
    """
    self.k = k
    self.max_iter = max_iter

def fit(self, X):
    """
    Fit the Kmeans model to data.

    :param X
        Numpy array of shape (n, p)
        n: number of data examples
        p: number of features (attributes)

    :return
        labels: array of shape (n, ), cluster labels (0..k-1)
        centers: array of shape (k, p, )
    """

    n, p = X.shape
    labels = np.random.choice(range(self.k), size=n, replace=True)

    # Choose k random data points for initial centers
    centers = np.array([X[i] for i in np.random.choice(range(X.shape[0]),
                                                         size=self.k)])

    i = 0
    while i < self.max_iter:

        # Find nearest cluster
        for j, x in enumerate(X):
            ki = np.argmin(np.sum((centers - x) ** 2, axis=1))
            labels[j] = ki
```

```

        # Store previous centers
        previous_centers = centers.copy()

        # Move centroid
        for ki in range(self.k):
            centers[ki] = X[labels == ki].mean(axis=0)
        i += 1

    return labels, centers

```

Rešitev je dosegljiva v resitve/voditelji.py.

In [3]: %run 204-1.ipynb

ERROR:root:File `204-1.ipynb.py` not found.

4.1.1 Podatki

Metodo testiramo na podatkovni zbirki Iris, kjer za cvetlice v treh razredih merimo različne dimenzije cvetnih oz. venčnih listov. V podatkih najdemo tri gručice, približno takole. V rešitvi sta prikazana samo prva dva atributa, zato se gručice navidez prekrivajo.

```

In [4]: from sklearn.datasets import load_iris
        data = load_iris()

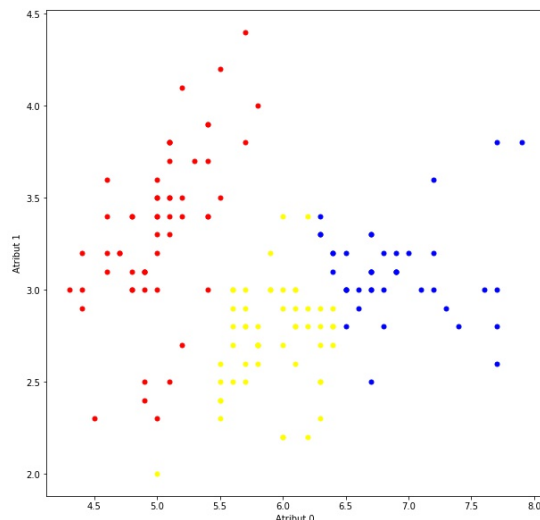
        X = data["data"]
        true_clusters = data["target"]

        # Testirajte razred KMeans

        # Trenutno so gručice dodeljene naključno
        model = KMeans(k=3, max_iter=10)
        labels, centers = model.fit(X[:, :2])

        plt.figure(figsize=(10, 10))
        color = {0:"red", 1:"blue", 2:"yellow"}
        for c, x in zip(labels, X):
            plt.plot(x[0], x[1], ".", color=color[c], markersize=10.0)
        plt.xlabel("Atribut 0")
        plt.ylabel("Atribut 1")
        plt.show()

```

Vprašanje 4-1-3 Nariši stanje oznak in sredine skupin v vsaki iteraciji algoritma.

In [5]: # ...

4.1.2 Ocenjevanje uspešnosti odkrivanja skupin

Ocenjevanje uspešnosti odkrivanja skupin je eden od nerešenih izzivov strojnega učenja. Poznamo mere kot sta silhueta ali metoda naključnih indeksov.

Druga možnost je, da poznamo resnične razrede, v katere spadajo podatki. To drži za podatkovno zbirko iris, kjer so cvetlice razporejene v tri razrede.

Vprašanje 4-1-4 Preveri, kako dobro se rezultat tvoje metode ujema z resničnimi razredi? Na kakšen način boš izmeril/a ujemanje? Ali naletiš pri tem na kakšno težavo?

```
In [6]: print(true_clusters)    # Resnični razredi primerov

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

Vprašanje 4-1-5 Preizkusi metodo na spodnjih, sintetičnih primerih podatkov. Kako se obnese metoda KMeans? Zakaj?

```
In [7]: from sklearn.datasets import make_circles, make_blobs, make_moons

n_samples=1000
noisy_circles, _ = make_circles(n_samples=n_samples, factor=.5, noise=.05)
noisy_moons, _ = make_moons(n_samples=n_samples, noise=.05)
blobs, _ = make_blobs(n_samples=n_samples, random_state=8)

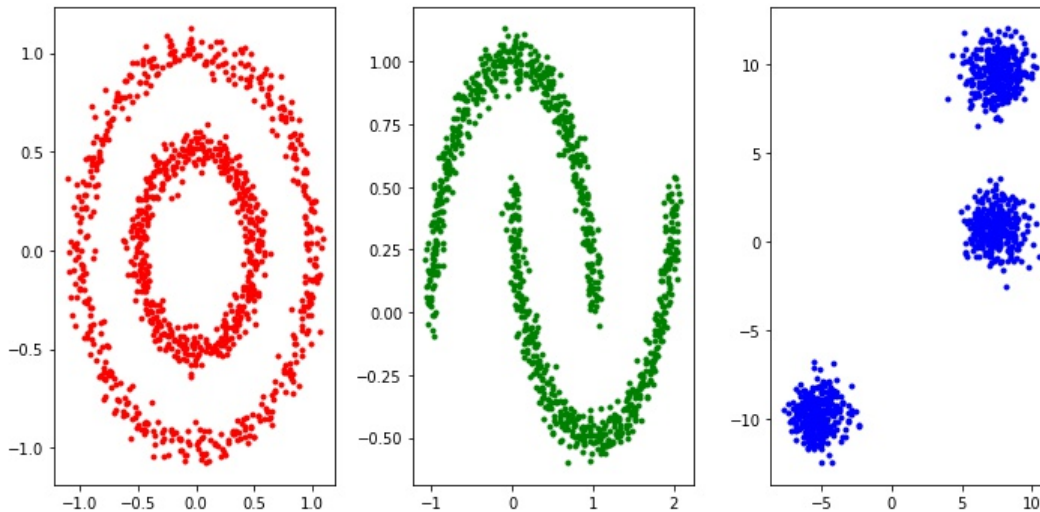
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(10, 5))
```

```

axes[0].plot(noisy_circles[:, 0], noisy_circles[:, 1], "r.")
axes[1].plot(noisy_moons[:, 0], noisy_moons[:, 1], "g.")
axes[2].plot(blobs[:, 0], blobs[:, 1], "b.")

fig.tight_layout()
plt.show()

```



4.1.3 Metoda DBSCAN

Vprašanje 4-1-6 Preizkusi metodo DBSCAN. Ali se ta metoda na istih podatkih obnese kaj bolje? Zakaj? Odgovor najdeš v opisu metode.

```
In [8]: from sklearn.cluster import DBSCAN
```

```

# model = DBSCAN(...)
# model.fit(X)
# labels = model.predict(X)

```

4.2 Hierarhično gručenje

Na predavanjih smo spoznali algoritem hierarhičnega gručenja. Njegova glavna značilnost je, da omogoča primerjavo objektov zgolj na podlagi poznavanja mere razdalje med njimi. Predstavitev podatkov torej ni nujno omejena na vektorske prostore.

Algoritem je determinističen in ne predpostavlja števila gruč. Rezultat gručenja bo izračunan naenkrat za vsa možna števila gruč v intervalu $[1, n]$, odločitev o številu pa bo sprejeta po izračunu.

Razmisli. Kakšna je časovna zahtevnost algoritma za hierarhično gručenje? Kako se primerja z metodo K-means?

```
In [1]: import numpy as np
```

```

%matplotlib inline
%config InlineBackend.figure_formats = ['jpg']
import matplotlib
matplotlib.figure.Figure.__repr__ = lambda self: (

```

```

f"<{self.__class__.__name__} size {self.bbox.size[0]:g}"
f"x{self.bbox.size[1]:g} with {len(self.axes)} Axes>")

import matplotlib.pyplot as plt
plt.style.use('PR.mplstyle')

import Orange
import scipy.cluster.hierarchy as sch
import scipy

-----

FileNotFoundError                                Traceback (most recent call last)

~/anaconda3/lib/python3.6/site-packages/matplotlib/style/core.py in use(style)
    112         try:
--> 113             rc = rc_params_from_file(style, use_default_template=False)
    114             _apply_style(rc)

~/anaconda3/lib/python3.6/site-packages/matplotlib/__init__.py in rc_params_from_file(fname, fail
1028     """
-> 1029     config_from_file = _rc_params_in_file(fname, fail_on_error)
    1030

~/anaconda3/lib/python3.6/site-packages/matplotlib/__init__.py in _rc_params_in_file(fname, fail
    944     rc_temp = {}
--> 945     with _open_file_or_url(fname) as fd:
    946         try:

~/anaconda3/lib/python3.6/contextlib.py in __enter__(self)
    80         try:
---> 81             return next(self.gen)
    82         except StopIteration:

~/anaconda3/lib/python3.6/site-packages/matplotlib/__init__.py in _open_file_or_url(fname)
    929         encoding = "utf-8"
--> 930         with io.open(fname, encoding=encoding) as f:
    931             yield f

```

FileNotFoundError: [Errno 2] No such file or directory: 'PR.mplstyle'

During handling of the above exception, another exception occurred:

```

OSError                                Traceback (most recent call last)

<ipython-input-1-a65c3d83fbbe> in <module>

```

```

9
10 import matplotlib.pyplot as plt
---> 11 plt.style.use('PR.mplstyle')
12
13 import Orange

~/anaconda3/lib/python3.6/site-packages/matplotlib/style/core.py in use(style)
117             "not a valid URL or path. See `style.available` for "
118             "list of available styles.")
--> 119         raise IOError(msg % style)
120
121

```

OSError: 'PR.mplstyle' not found in the style library and input is not a valid URL or path. See

4.2.1 Podatki

Današnji podatki spominjajo na (starejši generaciji dobro znani) album sličic z živalmi. Vsebuje 59 živalskih vrst ter 16 atributov, ki opisuje pripadajoče anatomske značilnosti. Živali so razdeljene v 7 razredov.

```

In [2]: data = Orange.data.Table('podatki/zoo.tab')
        print(data.domain)
        print(data[:2])

        print(np.linalg.norm(data.X[0]-data.X[1]))
        print(np.linalg.norm(data.X[0]-data.X[1], ord=1))

```

```

-----

NameError                                Traceback (most recent call last)

<ipython-input-2-09d556922ece> in <module>
----> 1 data = Orange.data.Table('podatki/zoo.tab')
      2 print(data.domain)
      3 print(data[:2])
      4
      5 print(np.linalg.norm(data.X[0]-data.X[1]))

NameError: name 'Orange' is not defined

```

Zanimala nas bo predvsem matrika X, ki podatke hrani v številski obliki.

```

In [3]: X = data.X
        Y = data.Y
        print(X.shape)
        print(X)

```

```

NameError                                Traceback (most recent call last)

<ipython-input-3-8fe55c01bd25> in <module>
----> 1 X = data.X
      2 Y = data.Y
      3 print(X.shape)
      4 print(X)

NameError: name 'data' is not defined

```

Rezultat gručenja dobimo z uporabo modula `scipy.cluster.hierarchy` in metode `linkage`. Slednja izračune povezave v drevesu (dendrogramu) glede na dano mero razdalje (`metric`) in načinom merjenja razdalj med gručami (`method`).

```
In [4]: L = sch.linkage(X, method="average", metric="cityblock")
```

```

-----

NameError                                Traceback (most recent call last)

<ipython-input-4-f9b3263f20b0> in <module>
----> 1 L = sch.linkage(X, method="average", metric="cityblock")

NameError: name 'sch' is not defined

```

Z uporabo funkcije `dendrogram` narišemo drevo in mu priredimo oznake. Funkcija deluje v navezi z že znano knjižnico `matplotlib`.

```
In [5]: plt.figure(figsize=(25, 6))
        labels = [row["name"].value for row in data]
        D = sch.dendrogram(L, labels=labels, leaf_font_size=15)
        plt.ylabel("Razdalja")
        plt.show()

-----

NameError                                Traceback (most recent call last)

<ipython-input-5-08bf720563cc> in <module>
      1 plt.figure(figsize=(25, 6))
----> 2 labels = [row["name"].value for row in data]
      3 D = sch.dendrogram(L, labels=labels, leaf_font_size=15)
      4 plt.ylabel("Razdalja")
      5 plt.show()

NameError: name 'data' is not defined

```

<Figure size 1800x432 with 0 Axes>

V redu, za prvi poizkus. Vseeno izgleda dendrogram nekoliko sploščen. Preveri, kako na graf vplivajo različne ...

4.2.2 Metode povezovanja

Metode povezovanja določajo način, kako izračunati razdaljo med dvema poljubno velikima gručama točk. * Posamično povezovanje (method="single"); Razdalja med gručama je razdalja med najbližjima točkama gruč. * Povprečna razdalja (method="average"); Povprečna razdalja med vsemi pari točk. * Razdalja med središčema (method="centroid"); Izračuna središči gruč v prostoru ter njuno medsebojno razdaljo. Mera razdalje je nujno evklidska.

Vprašanje 4-2-1 Preizkusi različne oblike dendrograma glede na izbrano mero razdalje.

In [6]: # Preizkusi različne načine merjenja razdalje med gručami

```
L = sch.linkage(X, method="single",)
```

```
-----

NameError                                Traceback (most recent call last)

<ipython-input-6-d282c08e9e82> in <module>
      1 # Preizkusi različne načine merjenja razdalje med gručami
----> 2 L = sch.linkage(X, method="single",)

NameError: name 'sch' is not defined
```

Odgovor

Ali je evklidska razdalja res najbolj primeren način primerjanja atributov, ki so diskretni? Ne vedno.

4.2.3 Mere razdalje

Način določanja interpretacije razdalje med točkama $\vec{x} = (x_1, x_2, \dots, x_p)$ in $\vec{y} = (y_1, y_2, \dots, y_p)$ vpliva na rezultat hierarhičnega gručenja. Izbira ustrezne mere je odvisna od narave podatkov in čimbolje odgovarja na vprašanje: kaj pomeni, da sta dva primera podobna?

Na izbiro ustrezne mere lahko vplivajo: * Prisotnost manjkajočih vrednosti * Predstavitev podatkov (vektorji, nizi znakov, slike, ...) * Tip atributov in interpretacija vrednosti

Nekaj pogostih mer razdalje: * Evklidska razdalja (metric="euclidean")

$$d(\vec{x}, \vec{y}) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \dots + (x_p - y_p)^2}$$

- Manhattanska razdalja (metric="cityblock")

$$d(\vec{x}, \vec{y}) = |x_1 - y_1| + |x_2 - y_2| + \dots + |x_p - y_p|$$

- Kosinusna razdalja (metric="cosine")

Predstavlja kosinus kota med vektorjema \vec{x} in \vec{y} - manjši kot pomeni večjo podobnost. Uporabna za primerjavo podobnosti med vektorji, neupoštevajoč absolutnih velikosti.

$$d(\vec{x}, \vec{y}) = 1 - \frac{\vec{x} \cdot \vec{y}}{\|\vec{x}\| \|\vec{y}\|}$$

- Jaccardov index (metric="jaccard")

Izmeri delež ujemanj med pari soležnih komponent (x_i, y_i) , kjer je vsaj ena izmed vrednosti x_i ali y_i večja od nič. Primerna za uporabo v primerih, ko imamo opravka z manjkajočimi vrednostmi ali diskretnimi atributi.

Popoln spisek razdalj najdeš v dokumentaciji.

Razmisli. Poizkusi se spomniti vrste podatkov, kjer bi bilo smiselno uporabiti vsako posamezno mero.

4.2.4 Določanje števila gruč

Koliko gruč je v podatkih? Na to vprašanje je težko odgovoriti in tudi sicer velja za odprto vprašanje na področju storjnega učenja. Vseeno poznamo nekaj kazalcev, ki jih v grobem delimo na * nadzorovane (znani so resnični razredi podatkov) * nenadzorovane (znane so samo značilke in/ali razdalje med primeri)

Za določitev pripadnosti primerov gručam uporabimo funkcijo `fcluster`. Slednja prejme parameter `t`, ki določa razdaljo pri kateri odsekamo dendrogram, t.j. odstranimo vse povezave, ki so daljše od dane dolžine. Preostale povezane komponente grafa dendrograma tako tvorijo skupine.

```
In [7]: L = sch.linkage(X, method="average", metric="cityblock")
        t = 3.5
        predictions = sch.fcluster(L, t=t, criterion="distance").ravel()
        classes      = data.Y.ravel()    # resnicni razredi

        print("Primer", "Resnični razred", "Gruča")
        for row, category, prediction in list(zip(data, classes, predictions))[2:10]:
            print("%s\t%d\t%d" % (row["name"], category, prediction))
```

```
-----

NameError                                Traceback (most recent call last)
```

```
<ipython-input-7-fcc7194cb2fb> in <module>
----> 1 L = sch.linkage(X, method="average", metric="cityblock")
      2 t = 3.5
      3 predictions = sch.fcluster(L, t=t, criterion="distance").ravel()
      4 classes      = data.Y.ravel()    # resnicni razredi
      5
```

```
NameError: name 'sch' is not defined
```

Ponovno narišemo dendrogram in ga odsekamo pri dani razdalji. V nadaljevanju si bomo ogledali številske ocene uspešnosti gručenja.

```
In [8]: D = sch.dendrogram(L, labels=labels)
        plt.plot([0, 1000], [t, t], "k--")
        plt.ylabel("Razdalje")
        plt.show()
```

```

NameError                                Traceback (most recent call last)

<ipython-input-8-1311667edc45> in <module>
----> 1 D = sch.dendrogram(L, labels=labels)
      2 plt.plot([0, 1000], [t, t], "k--")
      3 plt.ylabel("Razdalje")
      4 plt.show()

NameError: name 'sch' is not defined

```

4.2.4.1 Skupna deljena informacija

Mera skupne deljene informacije je uporabna, ko so na voljo informacije o resničnih razredih, v katere spadajo primeri. Pri tem ni odveč poudariti, da resnični razredi ne smejo biti uporabljeni pri deljenju primerov v skupine.

Naključne dodelitve oznak gruč imajo vrednost skupne deljene informacije blizu 0.0 za vsako vrednost števila skupin in števila primerov. Popolno ujemanje gruč z obstoječimi razredi ima vrednost 1. Mera ni odvisna od predstavitve podatkov, t.j. ni potrebno da so podatki v vektorskem prostoru, saj je odvisna samo od oznak.

```

In [9]: from sklearn.metrics import adjusted_mutual_info_score
        score = adjusted_mutual_info_score(classes, predictions)
        score

```

```

-----

NameError                                Traceback (most recent call last)

<ipython-input-9-12bf9055f1a0> in <module>
      1 from sklearn.metrics import adjusted_mutual_info_score
----> 2 score = adjusted_mutual_info_score(classes, predictions)
      3 score

NameError: name 'classes' is not defined

```

4.2.4.2 Koeficient silhuete

Koeficient silhuete je nenadzorovana mera v območju med -1 (napačno dodeljene skupine) in 1 (zelo goste, dobro ločene skupine). Večja notranja gostota znotraj skupin in večja razdalja sta prenosorazmeni s koeficientom. Tudi ta mera ne predpostavlja, da so podatki v vektorskem prostoru, je pa odvisna od izbrane mere razdalje.

Vprašanje 4-2-2 Preveri, kako se ocena spreminja glede na izbrano mero razdalje. Katera mera razdalje najboljše oceni gručenje? Ali je rezultat smiseln?

```

In [10]: from sklearn.metrics import silhouette_score

```



```

score = silhouette_score(X, predictions, metric="cityblock")
score

-----

NameError                                Traceback (most recent call last)

<ipython-input-10-6cdd011a4efa> in <module>
      1 from sklearn.metrics import silhouette_score
      2
----> 3 score = silhouette_score(X, predictions, metric="cityblock")
      4 score

NameError: name 'X' is not defined

```

Odgovor

Vprašanje 4-2-3 Izvedi analizo gručenja na podatkih o živalih tako, da izbereš ustrezno metodo povezovanja, mero razdalje in število gruč. Uporabi eno od predstavljenih mer podobnosti ter poišči tako kombinacijo omenjenih nastavitev, da bo rezultat gručenja karseda visok.

In [11]: # ...

Odgovor

4.3 Primer: genomski podatki

Stopnja razvoja na področju biotehnologije omogoča pridobivanje bistveno več podatkov o organizmih. Eden pogostih podatkovnih tipov, s katerimi primerjamo vrste so genske zapisi. Ti so pripravljeni za predstavitev v računalništvu, saj jih lahko posplošimo na zaporedna štirih nukleotidov: A, C, G, T. Celoten genski zapis ki določa vse, od vaše barve oči do nagnjenosti do določenih bolezni je podano z nekaj več kot 3×10^{12} dolgim zaporednjem DNK.

Pri razmoževanju prihaja do prepisovanja in kombiniranja DNA zapisov staršev. Ta proces seveda ni popoln, zato prihaja do napak - mutacij. Dolgoročna posledica mutacij pa je natanek različnih živalskih vrst, kar pomeni, da imajo sorodnejše vrste bolj podobne genske zapise.

Iz baze genskih zapisov smo naložili zaporednja mitohondrijskega gena za 13 vrst: 'Gorilla gorilla', 'Homo sapiens', 'Carassius auratus auratus', 'Delphinus capensis', 'Chamaeleo calyptratus', 'Canis lupus familiaris', 'Homo sapiens neanderthalensis', 'Rattus norvegicus', 'Equus caballus', 'Daboia russellii', 'Pan troglodytes', 'Takifugu rubripes', 'Pongo abelii', 'Sus scrofa'.

Podatke najprej pridobimo iz spleta.

```

In [1]: from Bio import Entrez
        from Bio import SeqIO
        import json

        species = [
            ("Homo sapiens", "NC_012920.1"),
            ("Pan troglodytes", "NC_001643.1"),
            ("Equus caballus", "NC_001640.1"),

```

```

        ("Chamaeleo calyptratus", "NC_012420.1"),
        ("Delphinus capensis", "NC_012061.1"),
        ("Takifugu rubripes", "NC_004299.1"),
        ("Canis lupus familiaris", "NC_002008.4"),
        ("Gorilla gorilla", "NC_001645.1"),
        ("Pongo abelii", "NC_002083.1"),
        ("Sus scrofa", "NC_000845.1"),
        ("Daboia russellii", "NC_011391.1"),
        ("Carassius auratus auratus", "NC_006580.1"),
        ("Rattus norvegicus", "AC_000022.2"),
        ("Homo sapiens neanderthalensis", "NC_011137.1"),
    ]

    # Data loading
    infile = "podatki/seqs.json"
    seqs = dict()
    for name, sid in species:
        print("Loading ...", name)
        t = False
        while not t:
            try:
                handle = Entrez.efetch(db="nucleotide", rettype="gb", id=sid,
                                       email="a@gmail.com")
                rec = SeqIO.read(handle, "gb")
                handle.close()
                t = True
            except:
                continue
        seqs[name] = str(rec.seq)

    json.dump(seqs, open(infile, "w"))

```

```

Loading ... Homo sapiens
Loading ... Pan troglodytes
Loading ... Equus caballus
Loading ... Chamaeleo calyptratus
Loading ... Delphinus capensis
Loading ... Takifugu rubripes
Loading ... Canis lupus familiaris
Loading ... Gorilla gorilla
Loading ... Pongo abelii
Loading ... Sus scrofa
Loading ... Daboia russellii
Loading ... Carassius auratus auratus
Loading ... Rattus norvegicus
Loading ... Homo sapiens neanderthalensis

```

```

In [2]: import json
        sequences = json.load(open("podatki/seqs.json"))
        print(sequences["Homo sapiens"])
        print(len(sequences["Homo sapiens"]))

```

```

GATCACAGGTCTATCACCTATTAACCACTCACGGGAGCTCTCCATGCATTGGTATTTTCGTCTGGGGGTATGCACGCATAGCATTGCGAGACGCTGGAG
16569

```

Vprašanje 4-3-1 Kako bi lahko primerjali živalske vrste glede na zapise, ki so podani kot nizi znakov? Prva ideja je, da podatke pretvorimo v vektorski prostor, v katerem bomo računali razdalje. Namig: zaporedja lahko razbiješ na manjše dele in prešteješ število pojavitev posameznih znakov, parov, trojk, ... k-terk. ahko upoštevaš tudi položaj v zaporedju.

Dopolni in si pomagaj s funkcijo `seq_to_kmer_count`, ki pretvori niz znakov v vektor števila pojavitev vseh mogočih k-terk.

Prevedi podatke v ustrezno obliko, izvedi hierarhično gručenje in prikaži rezultate. Ali so vrste na dendrogramu postavljene smiselno? Dobiti morate sliko:

```
In [3]: from itertools import product
def seq_to_kmer_count(seq, k=4):
    """
    Pretvori zaporedje seq v vektor x.
    AAAA AAAC AAAG AAAT ... TTTG TTTT
    x = [ 1 1      2 10 ... 12 7]
    len(x) == len(seq) - k + 1
    """
    ktuples = list(zip(*[seq[i:] for i in range(k)])) # razbijemo trenutni niz seq na k-terke
    kmers = list(product(*[k*["A", "C", "T", "G"]])) # vse mozne k-terke

    x = np.zeros((len(kmers), ))
    ### Your code here ###
    # for i, kmer in enumerate(kmers)
    # ...

    return x

In [4]: # ...
# Za vsako zaporedje (organizem), izračunaj x
# X = np.array([x1, x2, ..., x13]) - matrika 13 x 256 (k=4)
# pozeni gručenje
```

Odgovor

Poglavje 5

Nadzorovano učenje

Scenarij pri metodah *nadzorovanega modeliranja* je pogosto naslednji. Podatki so predstavljeni s pari

$$(\vec{x}_1, y_1), (\vec{x}_2, y_2), \dots, (\vec{x}_n, y_n)$$

kjer \vec{x}_i imenujemo *neodvisne*, y_i pa *odvisne* spremenljivke. Zanima nas *preslikava* $h(\vec{x})$, ki vrednosti neodvisne spremenljivke slika v odvisne, z napako ϵ_i . Torej,

$$y_i = h(\vec{x}_i) + \epsilon_i$$

Spremenljivke \vec{x}_i , y so v splošnem lahko zvezne, diskretne in druge. Preslikava $h(\vec{x})$ predstavlja *model* podatkov. Preslikava je lahko poljubna matematična funkcija (ali tudi algoritem, program), ki je odvisna od enega ali več *parametrov*.

Strojno učenje pogosto pojmuje kot iskanje parametrov (ali kar funkcije same) tako, da bo napaka ϵ_i karseda majhna.

5.1 Linearna regresija

Linearna regresija je primer enostavnega modela, kjer predpostavljamo: * tako odvisne kot neodvisne spremenljivke so realna števila * odvisna spremenljivka je linearna kombinacija neodvisnih * napaka ϵ je normalno porazdeljena z upanjem $\mu_\epsilon = 0$ in neznano varianco

Odvisne spremenljivke so v splošnem vektorji v p -dimenzionalnem prostoru realnih števil, $\vec{x} = (x_1, x_2, \dots, x_p)$.

Model je oblike

$$h(\vec{x}) = \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p + \beta_0$$

kjer vektor $\vec{\beta} = (\beta_0, \beta_1, \dots, \beta_p)$ predstavlja neznane parametre oz. koeficiente. Model je torej premica (pri $p = 1$) oz. ravnina v p -dimenzionalnem prostoru.

Učenje predstavlja iskanje (optimizacijo) parametrov $\vec{\beta}$ s ciljem zmanjšanja povprečne napake v podatkih.

$$\min_{\beta} \frac{1}{n} \sum_1^n (y_i - h(\vec{x}_i))^2 = \frac{1}{n} \sum_1^n \epsilon^2$$

Vrednost zgornjega izraza se imenuje **srednja kvadratična napaka** (ang *mean squared error* ali MSE). Iz statističnega vidika pa predstavlja **nepojasnjeno varianco**.

Algoritmov za minimizacijo zgornjega izraza tokrat ne bomo izpeljevali, temveč se raje osredotočimo na praktično uporabo. Več napotkov je na voljo tukaj.

```
In [1]: %matplotlib inline
%config InlineBackend.figure_formats = ['jpg']
import matplotlib
matplotlib.figure.Figure.__repr__ = lambda self: (
    f"<{self.__class__.__name__} size {self.bbox.size[0]:g}"
    f"x{self.bbox.size[1]:g} with {len(self.axes)} Axes>")

import matplotlib.pyplot as plt
plt.style.use('PR.mplstyle')

import numpy as np
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.metrics import mean_squared_error

-----

FileNotFoundError                                Traceback (most recent call last)

~/anaconda3/lib/python3.6/site-packages/matplotlib/style/core.py in use(style)
    112         try:
--> 113             rc = rc_params_from_file(style, use_default_template=False)
    114             _apply_style(rc)

~/anaconda3/lib/python3.6/site-packages/matplotlib/__init__.py in rc_params_from_file(fname, fail
1028     """
-> 1029     config_from_file = _rc_params_in_file(fname, fail_on_error)
    1030

~/anaconda3/lib/python3.6/site-packages/matplotlib/__init__.py in _rc_params_in_file(fname, fail
    944     rc_temp = {}
--> 945     with _open_file_or_url(fname) as fd:
    946         try:

~/anaconda3/lib/python3.6/contextlib.py in __enter__(self)
    80         try:
----> 81             return next(self.gen)
    82         except StopIteration:

~/anaconda3/lib/python3.6/site-packages/matplotlib/__init__.py in _open_file_or_url(fname)
    929         encoding = "utf-8"
--> 930         with io.open(fname, encoding=encoding) as f:
    931             yield f
```

```
FileNotFoundError: [Errno 2] No such file or directory: 'PR.mplstyle'
```

During handling of the above exception, another exception occurred:

```

OSError                                Traceback (most recent call last)

<ipython-input-1-e5cae4941953> in <module>
      7
      8 import matplotlib.pyplot as plt
----> 9 plt.style.use('PR.mplstyle')
     10
     11 import numpy as np

~/anaconda3/lib/python3.6/site-packages/matplotlib/style/core.py in use(style)
    117         "not a valid URL or path. See `style.available` for "
    118         "list of available styles.")
--> 119         raise IOError(msg % style)
     120
     121

```

```
OSError: 'PR.mplstyle' not found in the style library and input is not a valid URL or path. See
```

Začnimo s preprostim primerom z eno neodvisno spremenljivko x ter odvisno spremenljivko y .

```
In [2]: data = np.loadtxt("podatki/sintetični/data_A.txt")
      x    = data[:, [0]]
      y    = -data[:, [1]]
```

```

print(x.shape)
print(y.shape)

plt.figure()
plt.plot(x, y, "k.")
plt.xlabel("x")
plt.ylabel("y")
plt.show()

```

```

-----
NameError                                Traceback (most recent call last)

<ipython-input-2-aed59bb829e1> in <module>
----> 1 data = np.loadtxt("podatki/sintetični/data_A.txt")
      2 x    = data[:, [0]]
      3 y    = -data[:, [1]]
      4
      5 print(x.shape)

```

```
NameError: name 'np' is not defined
```

Podatki kar dobro spominjajo na premico.

Poizkusimo poiskati linearni model, ki bo zmanjšal srednjo kvadratično napako.

Na levi sliki prikazujemo vrednosti modela za vse vrednosti x na danem intervalu.

Desna slika pa prikazuje vrednost ostankov $y_i - h(\vec{x}_i)$. Bolje, kot se model prilega podatkom, manj povezana bosta odvisna spremenljivka in ostanek.

```
In [3]: from scipy.stats import pearsonr
def plot_fit_residual(x, y, yp):

    # Model
    fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(15, 4))
    axes[0].plot(x.ravel(), y.ravel(), "k.", label="Podatki")
    axes[0].plot(x.ravel(), yp.ravel(), "g-", label="Model h(x)")
    axes[0].set_xlabel("x")
    axes[0].set_ylabel("y")
    axes[0].legend(loc=4)

    # Ostanke
    r = pearsonr(y.ravel(), y.ravel()-yp.ravel())[0]
    axes[1].plot(y.ravel(), y.ravel()-yp.ravel(), "k.", label="Ostanek")
    axes[1].set_xlabel("y")
    axes[1].set_ylabel("y-h(x)")
    axes[1].set_title("Graf ostankov, R=%.3f" % r)
    axes[1].legend(loc=4)
    plt.show()
```

```
In [4]: # Ucenje modela
model = LinearRegression()
model.fit(x, y)

print(model.intercept_, model.coef_)

# Napoved vrednosti za podatke
hx = model.predict(x)

plot_fit_residual(x, y, hx)
```

```
-----
NameError
```

```
Traceback (most recent call last)
```

```
<ipython-input-4-2f0316d70b18> in <module>
      1 # Ucenje modela
----> 2 model = LinearRegression()
      3 model.fit(x, y)
      4
      5 print(model.intercept_, model.coef_)
```



```
NameError: name 'LinearRegression' is not defined
```

Izmerimo srednjo kvadratično napako...

```
In [5]: mean_squared_error(hx, y)
```

```
-----

NameError                                Traceback (most recent call last)

<ipython-input-5-91031a8d16e9> in <module>
----> 1 mean_squared_error(hx, y)

NameError: name 'mean_squared_error' is not defined
```

... ki je enaka varianci razlike.

```
In [6]: np.var(hx-y)
```

```
-----

NameError                                Traceback (most recent call last)

<ipython-input-6-320614238b73> in <module>
----> 1 np.var(hx-y)

NameError: name 'np' is not defined
```

Tako lahko dobimo *delež pojasnene variance*. Delež v odstotkih si lažje intuitivno razlagamo.

```
In [7]: explained_var = 100.0 * ( np.var(y) - np.var(hx-y) ) / np.var(y)
        print("Explained variance: %.2f " % explained_var + "%" )
```

```
-----

NameError                                Traceback (most recent call last)

<ipython-input-7-d7f1e86fae41> in <module>
----> 1 explained_var = 100.0 * ( np.var(y) - np.var(hx-y) ) / np.var(y)
      2 print("Explained variance: %.2f " % explained_var + "%" )

NameError: name 'np' is not defined
```

5.2 Polinomska regresija

Oglejmo si naslednji motivacijski primer.


```
<ipython-input-10-d7f1e86fae41> in <module>
----> 1 explained_var = 100.0 * ( np.var(y) - np.var(hx-y) ) / np.var(y)
      2 print("Explained variance: %.2f " % explained_var + "%" )
```

```
NameError: name 'np' is not defined
```

5.3 Model polinomske regresije

Z pomočjo linearnih modelov lahko modeliramo tudi nelinearne odvisnosti, kar je glede na začetne predpostavke nekoliko presenetljivo. Vrednost x je v tem primeru enodimenzionalna spremenljivka ($p=1$).

Model polinomske regresije v eni dimenziji je polinom stopnje D :

$$h(\vec{x}) = \beta_1 x + \beta_2 x^2 + \dots + \beta_D x^D + \beta_0$$

Učinek dosežemo z ustrezno priredbo prostora. Spremenljivko x preslikamo v vektor tako, da izračunamo ustrezne potence:

$$x \rightarrow (x, x^2, x^3, \dots, x^D) = \vec{x}$$

V tako sestavljenem prostoru ni polinom nič drugega kot linearna preslikava!

```
In [11]: # Iz 1-D sestavimo nov 2-D prostor
```

```
        X = np.zeros((len(x), 2))
```

```
        X[:, 0] = x.ravel()
```

```
        X[:, 1] = x.ravel()**2
```

```
        # Učenje
```

```
        model = LinearRegression()
```

```
        model.fit(X, y)
```

```
        # Napoved
```

```
        hx = model.predict(X)
```

```
        plot_fit_residual(x, y, hx)
```

```
NameError
```

```
Traceback (most recent call last)
```

```
<ipython-input-11-18e49aed2024> in <module>
      1 # Iz 1-D sestavimo nov 2-D prostor
----> 2 X = np.zeros((len(x), 2))
      3 X[:, 0] = x.ravel()
      4 X[:, 1] = x.ravel()**2
      5
```

```
NameError: name 'np' is not defined
```

Vprašanje 5-1-1 Primerjaj pojasnjeno varianco linearnega in polinomskega modela.

```
In [12]: # ...
```

Odgovor

5.4 Pretirano prileganje

Optimalnega modela seveda pogosto ne poznamo. Uporaba pretirano kompleksnih modelov (kompleksnost si lahko predstavljamo kot velikost družine funkcij), lahko vodi v pretirano prileganje (ang. overfitting).

Oglejmo si primer polinoma stopnje 20:

```
In [13]: def plot_coefficients(coef):
          coef=coef.ravel()
          D = len(coef)
          plt.title("Parametri modela")
          plt.bar(np.arange(D), coef)
          plt.xticks(np.arange(D))
          plt.grid()
          plt.ylabel("beta")
          plt.xlabel("d")
          plt.show()
```

```
In [14]: D = 20 # stopnja polinoma
          X = np.zeros((len(x), D))
          for d in range(0, D):
              X[:, d] = x.ravel()**d

          model = LinearRegression()
          model.fit(X, y)

          hx = model.predict(X)

          plot_fit_residual(X[:, 1], y, hx)
          plot_coefficients(model.coef_)
```

NameError

Traceback (most recent call last)

```
<ipython-input-14-1e1e1a76f177> in <module>
      1 D = 20 # stopnja polinoma
----> 2 X = np.zeros((len(x), D))
      3 for d in range(0, D):
      4     X[:, d] = x.ravel()**d
      5
```

NameError: name 'np' is not defined

Model se na videz odlično prilega podatkom. Tudi graf ostankov kaže spodbudno sliko. Težava pretiranega prileganja se pojavi pri **napovedovanju novih podatkov**.

Vprašanje 5-1-2 Izmeri pojasnjeno varianco polinomskega modela.

In [15]: # ...

Odgovor

5.5 Rešitev: kaznovanje pretirano kompleksnih modelov

Poleg minimizacije srednje kvadratične napake lahko pri iskanju rešitve tudi *kaznujemo kompleksnost modelov*. Želimo torej, da so najdeni parametri v geometrijskem smislu čim manjši. Ta postopek je znan tudi kot regularizacija. Stopnjo regularizacije nadzoruje parameter α , ki ga določimo kot uporabniki. Dve najpogostejši različici modelov sta: * Regresija Lasso

“Kaznovanje manhattanske razdalje vektorja $\vec{\beta}$ od izhodišča”

$$\min_{\beta} \sum_1^n (y_i - h(\vec{x}_i))^2 + \alpha \|\vec{\beta}\|_1$$

Prednost: vrača **redke** vektorje parametrov $\vec{\beta}$. Večina komponent β_j bo enaka 0 - ZELO ZAŽELENO!

Slabost: zahtevno načrtovanje algoritmov za optimizacijo

- Regresija Ridge “Kaznovanje evklidske razdalje vektorja $\vec{\beta}$ od izhodišča”

$$\min_{\beta} \sum_1^n (y_i - h(\vec{x}_i))^2 + \alpha \|\vec{\beta}\|_2$$

Prednost: Enostaven izračun

Slabost: V splošnem ne vrača redkih vrednosti parametrov.

In [16]: D = 20 # stopnja polinoma

```
# Ustvarimo ustrezen prostor
X = np.zeros((len(x), D))
for d in range(0, D):
    X[:, d] = x.ravel()**d

model = Lasso(alpha=0.1)
model.fit(X, y)

hx = model.predict(X)

plot_fit_residual(X[:, 1], y, hx)
plot_coefficients(model.coef_)
model.coef_
```

NameError

Traceback (most recent call last)

<ipython-input-16-ac4ca0212785> in <module>

```

2
3 # Ustvarimo ustrezen prostor
----> 4 X = np.zeros((len(x), D))
5 for d in range(0, D):
6     X[:, d] = x.ravel()**d

```

NameError: name 'np' is not defined

Vprašanje 5-1-3 Kakšen je vpliv parametra `alpha` na a) kvaliteto prilaganja b) koeficiente modela ? Poizkusi podatke modelirati z regresijo Ridge.

In [17]: # ...

Odgovor

Funkcija izgleda “ravno pravi” model za podatke. Na grafu koeficientov (parametrov) vidimo, da so večino težje dobili koeficienti nižjih stopenj polinoma, kar predstavlja manj kompleksen model.

Vprašanje 5-1-4 Poišči modele polinomske regresije za spodnje tri nabore podatkov. Izberi stopnjo polinoma ter morda vrsto regularizacijskega modela. Nariši graf funkcije in diagram ostankov. Komentiraj rezultate.

Pravilne rešitve (koeficiente in stopnjo polinomov najdeš v `podatki/sintetični/coefficients_*.txt`)

In [18]: `fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(20, 6))`

```

for example, ax in zip(["C", "D", "E"], axes):
    data = np.loadtxt("podatki/sintetični/data_%s.txt" % example)
    x     = data[:, [0]]
    y     = data[:, [1]]

    ax.plot(x, y, "k.")
    ax.set_xlabel("x")
    ax.set_ylabel("y")
    ax.set_title("Primer %s" % example)

# ...

```

NameError

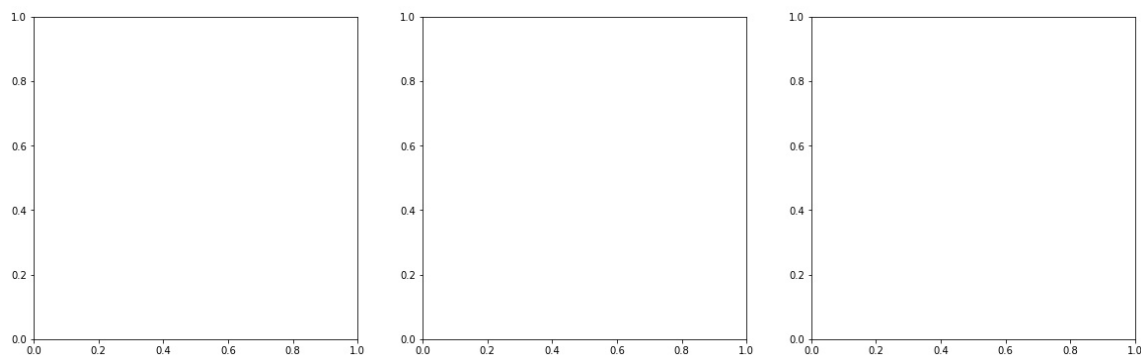
Traceback (most recent call last)

```

<ipython-input-18-28eb6e4a7e29> in <module>
2
3 for example, ax in zip(["C", "D", "E"], axes):
----> 4     data = np.loadtxt("podatki/sintetični/data_%s.txt" % example)
5     x     = data[:, [0]]
6     y     = data[:, [1]]

```

NameError: name 'np' is not defined



Odgovor

5.6 Uporaba v praksi: analiza sentimenta

Za konec si oglejmo povsem praktičen primer uporabe regresijskih modelov. V podatkovni zbirki imamo 1101 recenzij knjig. Vsaka recenzija je sestavljena iz besedila (niz znakov, besed) in ocene med 1 in 5 (1-porazno, 5-odlično). Izvirna podatkovna zbirka in članek sta na voljo tukaj.

Primer pozitivne recenzije ene izmed knjig (ocena = 5).

I'm a little late in reading this book. I am trying to pace myself between the movies and the books so
 I think Goblet of Fire is the best in the series, so naturally it would be pretty difficult for Phoenix
 I didn't mind the length of the book, but it did seem to drag in a couple of places. The gang spent
 My biggest problem with the book was Dumbledore's secrecy. Good stories have real roadblocks to keep
 Don't get me wrong. I love the Harry Potter series. And, perhaps my expectations have risen too high

Primer negativne recenzije ene izmed knjig (ocena = 2).

This book was horrible. If it was possible to rate it lower than one star i would have. I am an avid

I wish i had the time spent reading this book back so i could use it for better purposes. This book was

Vsako recenzijo predstavimo v prostoru 4000 najpogostejših besed oz. parov besed v podatkovni zbirki (predstavitev bag-of-words). Vsaka komponenta vrstice x (vektorja) šteje, kolikor se beseda/par besed pojavi v določeni recenziji.

```
In [19]: from pickle import load
         from os.path import join

         def load_data(dset):
             data = dict()

             indir = "podatki/%s/" % dset

             for name in "data", "target", "data_test", "target_test":
                 fname = join(indir, name + ".pkl")
                 data[name] = load(open(fname, "rb"))

                 fname = join(indir, "features.txt")
                 fp = open(fname, "rt")
                 data["features"] = list(map(lambda l: l.strip(), fp.readlines()))

             return data
```

```

books = load_data("books")
X = books["data"]
y = books["target"]

print(str(books['features'][:3]) + '...' + str(books['features'][-3:]))
print(X.todense())
print(y)
print(X.shape, y.shape)

['the', 'a', 'and']...['colors', 'and_most', 'introduced']
[[ 3  4  0 ...,  0  0  0]
 [ 1  1  1 ...,  0  0  0]
 [ 0  0  2 ...,  0  0  0]
 ...,
 [ 4  2  1 ...,  0  0  0]
 [10  2  5 ...,  0  0  0]
 [ 7  3  3 ...,  0  0  0]]
[1 2 2 ..., 4 5 2]
(1101, 4000) (1101,)

```

Vrstni red stolpcev v matriki X:

```
In [20]: features = books["features"]
         features
```

```
Out[20]: ['the',
          'a',
          'and',
          'to',
          'of',
          'this',
          'book',
          'is',
          'in',
          'i',
          'it',
          'for',
          'that',
          'this_book',
          'with',
          'but',
          'on',
          'not',
          'are',
          'have',
          'as',
          'of_the',
          'was',
          'be',
          'you',
          'in_the',
          'an',
          'all',
```


'read',
'from',
'if',
'about',
'one',
'or',
'by',
'at',
'is_a',
'more',
'the_book',
'what',
'very',
'my',
'who',
'so',
'has',
'like',
'some',
'good',
'would',
'his',
'there',
'<num>',
'to_the',
'how',
'they',
'he',
'it_is',
'out',
'just',
'other',
'book_is',
'will',
'much',
'can',
'great',
'this_is',
'do',
'no',
'your',
'when',
'up',
'only',
'which',
'and_the',
'than',
'on_the',
'to_be',
'even',
'many',
'had',
'me',
'get',

'time',
'also',
'well',
'their',
"it's",
'into',
'for_the',
'if_you',
'them',
'really',
'first',
'were',
'books',
'most',
'reading',
'then',
'any',
"don't",
'been',
'with_the',
'author',
'as_a',
'because',
'way',
'know',
'in_a',
'could',
'these',
'people',
'in_this',
'story',
'too',
'life',
'is_the',
'little',
'through',
'i_was',
'think',
'of_a',
'make',
'i_have',
'her',
'want',
'its',
'those',
'for_a',
'work',
'better',
'written',
'we',
'does',
'that_the',
'after',
'while',

'to_read',
'own',
'the_author',
'should',
'new',
'is_not',
'such',
'did',
'from_the',
'found',
'she',
'information',
'find',
'best',
'one_of',
'it_was',
'am',
'a_good',
'never',
'at_the',
'there_is',
'being',
'few',
'over',
'interesting',
'why',
'of_this',
'every',
'say',
'years',
'there_are',
'made',
'and_i',
'writing',
'two',
'that_i',
'see',
'real',
'with_a',
'use',
'another',
'a_book',
'a_great',
'i_am',
'have_been',
'however',
'same',
'want_to',
'i_would',
'each',
'all_the',
'world',
'now',
'still',

'before',
'love',
'go',
'without',
'book_i',
'recommend',
'about_the',
'where',
'the_same',
'may',
'by_the',
'looking',
'give',
'us',
'history',
'something',
'money',
'here',
'was_a',
'down',
'book_and',
'to_get',
'things',
'need',
'the_first',
'a_very',
"i'm",
'characters',
'thought',
'help',
'back',
'as_the',
'actually',
'some_of',
'excellent',
'book_for',
'though',
'our',
'must',
'take',
'ever',
'different',
'be_a',
'and_a',
'you_are',
"doesn't",
'nothing',
'both',
'book_to',
"didn't",
'thing',
'him',
'the_story',
'have_to',

'understand',
'to_make',
'a_few',
'off',
'pages',
'used',
'look',
'rather',
'lot',
'makes',
'buy',
'seems',
'reader',
'long',
'but_i',
'between',
'book_was',
'is_an',
'to_do',
'how_to',
'man',
'to_a',
'anyone',
'that_is',
'of_his',
'a_lot',
'they_are',
'but_the',
'times',
'someone',
'yet',
'old',
'would_be',
'might',
'far',
'stories',
'but_it',
'again',
'the_best',
'i_found',
'series',
'fact',
'easy',
'have_a',
'i've',
'chapter',
'quite',
'point',
'all_of',
'bought',
'example',
'you_can',
'others',
'out_of',

'the_most',
'i_think',
'least',
'around',
'last',
'true',
'a_little',
'bad',
'style',
'part',
'is_that',
'right',
'highly',
'end',
'believe',
'come',
'going',
'that_it',
'as_well',
'always',
'almost',
'short',
'since',
'i_had',
'can't',
'page',
'hard',
'would_have',
'enough',
'read_the',
'useful',
'into_the',
'said',
'put',
'the_way',
'authors',
'novel',
'to_say',
'the_other',
'making',
'readers',
'having',
'learn',
'instead',
'review',
'feel',
'anything',
'character',
'i_don't',
'read_this',
'bit',
'should_be',
'the_only',
'such_as',

'trying',
"that's",
'got',
'through_the',
'you_want',
'once',
'works',
'in_my',
'has_a',
'i_read',
'not_a',
'on_a',
'<year>',
'interested',
'when_i',
'given',
'the_reader',
'does_not',
'this_one',
'idea',
'worth',
'book_that',
'looking_for',
'lot_of',
'already',
'high',
'is_very',
'course',
'place',
'done',
'trying_to',
'person',
'along',
'more_than',
'reason',
'left',
'difficult',
'which_is',
'words',
'takes',
'mind',
'american',
'big',
'you_have',
'and_then',
'especially',
'often',
'in_his',
'ideas',
'that_are',
'experience',
'those_who',
'will_be',
'like_the',

'keep',
'comes',
'away',
'that_this',
'at_least',
'do_not',
'with_this',
'perhaps',
'sure',
'maybe',
'try',
'that_he',
'simply',
'case',
'past',
'plot',
'and_his',
'need_to',
'tell',
'wanted',
'important',
'start',
'although',
'school',
'sense',
'to_have',
'of_all',
'text',
'everything',
'reviews',
'probably',
'and_it',
'human',
'book_the',
'problem',
'children',
'a_bit',
'part_of',
'less',
'the_world',
'whole',
'goes',
'are_not',
'it_to',
'full',
'did_not',
'disappointed',
'knowledge',
'pretty',
'getting',
'and_how',
'wonderful',
'interested_in',
'next',

'subject',
'clear',
'are_a',
"you're",
'second',
'become',
'based',
'but_this',
'i_thought',
'young',
'most_of',
'home',
'using',
'title',
"it's_a",
'who_is',
'could_have',
'i_bought',
'to_see',
'came',
'interest',
'poor',
'was_the',
'year',
'word',
'personal',
'called',
'well_as',
'unfortunately',
'lives',
'simple',
'easy_to',
'enjoy',
'over_the',
'book_but',
'to_know',
'examples',
'like_a',
'day',
'sometimes',
'not_the',
'that_they',
'kind',
'today',
'the_end',
'as_i',
'seem',
'let',
"isn't",
"author's",
'question',
'family',
'he_is',
'small',

'chapters',
'write',
'lack',
'order',
'it_would',
'guide',
'points',
'to_find',
'as_it',
'the_last',
'can_be',
'going_to',
'this_was',
'is_one',
'general',
'i_can',
'of_them',
'i_did',
'what_i',
'shows',
'truly',
'able',
'it_i',
'issues',
'three',
'truth',
'business',
'doing',
'so_much',
'enjoyed',
'the_characters',
'to_learn',
'from_a',
'main',
'able_to',
'with_his',
'there's',
'god',
'had_to',
'etc',
'gives',
'is_to',
'bought_this',
'study',
'is_no',
'may_be',
'great_book',
'many_of',
'problems',
'that_you',
'various',
'are_the',
'reference',
'provides',

'change',
'class',
'has_been',
'seems_to',
'the_time',
'rather_than',
'an_excellent',
'historical',
'of_what',
'name',
'myself',
'ways',
'good_book',
'follow',
'against',
'disappointing',
'his_own',
'basic',
'i_could',
'loved',
'level',
'approach',
'several',
'beautiful',
'but_not',
'for_those',
'age',
'you_will',
'himself',
'view',
'living',
'number',
'of_their',
'else',
'book_in',
'john',
'of_how',
'job',
'say_that',
'of_her',
'ago',
'practical',
'get_a',
'material',
'possible',
'pictures',
'mr',
'started',
'insight',
'either',
'the_whole',
'stars',
'together',
'hand',

'students',
'cannot',
'large',
'complete',
'detail',
'themselves',
'says',
'recommended',
'set',
'i_will',
'of_course',
'to_go',
'throughout',
'gave',
'read_it',
'yourself',
'show',
'advice',
'to_use',
'he_has',
'parts',
'needs',
'lack_of',
'to_this',
'expect',
"i_didn't",
'writer',
'friends',
'fan',
'woman',
'liked',
'nice',
'for_you',
'<num>_years',
'present',
'because_i',
'within',
'waste',
'helpful',
'late',
'hope',
'seen',
'as_an',
'book_on',
'modern',
'to_help',
'it_has',
"the_author's",
'questions',
'boring',
'for_example',
'<num>_pages',
'introduction',
'fun',

'collection',
'hard_to',
'book_as',
'such_a',
'and_that',
'by_a',
'taking',
'extremely',
'because_of',
'much_better',
'to_me',
'tells',
'was_not',
'book_with',
'at_all',
'we_are',
'to_take',
'child',
'look_at',
'during',
'from_this',
'entire',
'people_who',
'for_my',
'way_to',
'reading_this',
'research',
'mostly',
'the_fact',
'kind_of',
'mean',
'understanding',
'covers',
'theory',
'so_many',
'for_me',
'full_of',
'war',
'perfect',
'when_the',
'form',
'lots',
'details',
'wanted_to',
'care',
'in_which',
'everyone',
'book_it',
'clearly',
'power',
'up_with',
'tried',
'based_on',
'save',

'beyond',
'lost',
'told',
'the_authors',
'up_to',
'and_this',
'history_of',
'social',
'finally',
'certainly',
'because_it',
'you_know',
'friend',
'he_was',
'what_the',
'kids',
'version',
'who_are',
'and_not',
'wrong',
'suggest',
'value',
'wants',
'took',
'number_of',
'quality',
'head',
'until',
'to_give',
'wish',
'gets',
'story_of',
'not_only',
'much_more',
'uses',
'later',
'talking',
'about_a',
'my_own',
'you_don't',
'fine',
'including',
'the_main',
'original',
'under',
'was_very',
'early',
'not_be',
'went',
'of_it',
'time_and',
'certain',
'upon',
'means',

'culture',
'writers',
'someone_who',
'book_has',
'college',
'science',
'what_you',
'complex',
'of_these',
'facts',
'lots_of',
'is_also',
'section',
'perspective',
'the_subject',
'resource',
'to_understand',
'for_an',
'working',
'too_much',
'matter',
'if_i',
'so_i',
'pick',
'very_good',
'rest',
'among',
'major',
'christian',
'fact_that',
'after_reading',
'it_and',
'piece',
'overall',
'side',
'attention',
'have_the',
'is_in',
'ones',
'it_does',
'stuff',
'fascinating',
'yes',
'couple',
'a_new',
'of_us',
'their_own',
'completely',
'century',
'won't',
'could_be',
'i_know',
'novels',
'writes',

'line',
'thinking',
'have_read',
'and_her',
'felt',
'of_my',
'topic',
'the_plot',
'development',
'wife',
'due',
'at_a',
'and_to',
'itself',
'about_this',
'for_all',
'society',
'the_writing',
'not_for',
'expected',
'there_were',
'half',
'white',
'system',
'and_was',
'what_they',
'practice',
'volume',
'analysis',
'know_what',
'i_really',
'for_this',
'due_to',
'end_of',
'they_were',
'and_he',
'death',
'events',
'strong',
'recommend_this',
'ending',
'reality',
'black',
'single',
'live',
'process',
'in_fact',
'known',
'huge',
'is_so',
'of_our',
'and_is',
'whose',
'reviewers',

'days',
'to_his',
'type',
'what_a',
'on_this',
'described',
'the_title',
'saying',
'country',
'as_much',
'what_is',
'the_<num>',
'language',
'and_other',
'cover',
'light',
'to_keep',
'had_a',
'in_order',
'serious',
'instead_of',
'imagine',
'art',
'on_how',
'the_rest',
'explain',
'be_the',
'which_i',
'giving',
'entertaining',
'opinion',
'finished',
'how_the',
'i'd",
'to_buy',
'women',
'much_of',
'the_truth',
'current',
'is_just',
'include',
'despite',
'agree',
'particular',
'read_and',
'than_the',
'four',
'deal',
'i_do',
'men',
'but_in',
'indeed',
'written_by',
'should_have',

'out_the',
'better_than',
'and_even',
'otherwise',
'guy',
'the_great',
'there_was',
'edition',
'knows',
'it_will',
'what_he',
'respect',
'taken',
'they_have',
'focus',
'fiction',
'needed',
'even_though',
"if_you're",
'sort_of',
'sense_of',
'like_this',
'the_real',
'and_in',
'a_real',
'likely',
'and_what',
'america',
"you'll",
'if_the',
'seemed',
'decided',
'up_the',
'particularly',
'starting',
'like_to',
'source',
'political',
'nor',
'successful',
'try_to',
'a_must',
'get_the',
'available',
'turn',
'library',
'call',
"wasn't",
'nature',
'design',
'the_point',
'worst',
'tale',
'reading_the',

```
...]
```

Priložena je tudi podatkovna zbirka testnih primerov, kjer lahko testiramo napovedno točnost modela na novih podatkih.

```
In [21]: X_test = books["data_test"]
        y_test = books["target_test"]
```

Vprašanje 5-1-5 Uporabi omenjene linearne modele za modeliranje podatkov pri problemu analize sentimenta. Izmeri srednjo kvadratično napako in pojasnjeno varianco na testnih primerih.

```
In [22]: # ...
```

Odgovor

Vprašanje 5-1-6 Ali lahko ugotoviš, katere besedne zveze močno pozitivno in močno negativno vplivajo na končno oceno recenzije? Namig: pomagaj si z vrednostjo koeficientov za posamezni stolpec.

```
In [23]: # ...
```

Odgovor

5.7 Naivni Bayesov klasifikator

```
In [1]: import numpy as np

        from Orange.data import Table
        from Orange.data.filter import SameValue
```

5.7.1 Primer za ogrevanje

V letniku športne gimnazije imamo 20 učencev. Vsak od njih sodeluje pri enem od športov: **kosarka**, **nogomet**, **gimnastika**. Njihovo višino smo ocenili “na oko” in vsakemu učencu pripisali eno od možnih vrednosti: **nizek**, **srednji** ali **visok**.

Kako bi novemu učencu Marku, ki je **srednje** rasti predlagali najprimernejši šport?

```
In [2]: data = Table('podatki/sportniki.tab')

In [3]: data.domain

Out[3]: [visina | sport]

In [4]: data

Out[4]: [[visok | kosarka],
        [visok | kosarka],
        [visok | kosarka],
        [visok | kosarka],
        [srednji | kosarka],
        ...
        ]
```

Za začetek pogledjmo kako popularni so posamezni športi:

```
In [5]: for sport in data.domain["sport"].values:
        subset = SameValue(data.domain["sport"], sport)(data)

        print(sport)
        print(subset)
        print()

        py      = len(subset) / len(data)
        print("Sport (Y): %s, število: %d, verjetnost P(Y): %f" % (sport, len(subset), py))
```

```
gimnastika
[[nizek | gimnastika],
 [nizek | gimnastika],
 [nizek | gimnastika],
 [srednji | gimnastika],
 [srednji | gimnastika]]
```

Sport (Y): gimnastika, število: 5, verjetnost P(Y): 0.250000

```
kosarka
[[visok | kosarka],
 [visok | kosarka],
 [visok | kosarka],
 [visok | kosarka],
 [srednji | kosarka],
 [srednji | kosarka],
 [nizek | kosarka],
 [visok | kosarka]]
```

Sport (Y): kosarka, število: 8, verjetnost P(Y): 0.400000

```
nogomet
[[srednji | nogomet],
 [srednji | nogomet],
 [srednji | nogomet],
 [visok | nogomet],
 [visok | nogomet],
 [nizek | nogomet],
 [nizek | nogomet]]
```

Sport (Y): nogomet, število: 7, verjetnost P(Y): 0.350000

Najpopularnejši šport je košarka, s katerim se ukvarja 8 oz. 40 učencev. Naš prvi predlog je torej, naj se Marko ukvarja s košarko. S tem rezultatom nismo najbolj zadovoljni, saj vidimo da med košarkaši ni veliko športnikov *srednje* višine. Razlog? Pri izračunu nismo upoštevali verjetnosti lastnosti oz. *atributa* o Markovi višini.

Splošnim verjetnostmi razredov, ki smo jih izračunali pravimo *apriorne* verjetnosti.

Označimo jih s $P(Y)$, kjer je Y spremenljivka razreda.

V našem primeru Y zavzame vrednostmi {kosarka, nogomet, gimnastika}.

```
In [6]: for sport in data.domain["sport"].values:
        subset_y = SameValue(data.domain["sport"], sport)(data)
        subset_x = SameValue(data.domain["visina"], "srednji")(subset_y)
        p_xy = len(subset_x) / len(subset_y)
```

```
print("Sport (Y): %s, št. srednje visokih: %d, verjetnost P(X=srednji|Y=%s): %f" % (sport,
print(subset_x)
print()
```

```
Sport (Y): gimnastika, št. srednje visokih: 2, verjetnost P(X=srednji|Y=gimnastika): 0.400000
[[srednji | gimnastika],
 [srednji | gimnastika]]
```

```
Sport (Y): kosarka, št. srednje visokih: 2, verjetnost P(X=srednji|Y=kosarka): 0.250000
[[srednji | kosarka],
 [srednji | kosarka]]
```

```
Sport (Y): nogomet, št. srednje visokih: 3, verjetnost P(X=srednji|Y=nogomet): 0.428571
[[srednji | nogomet],
 [srednji | nogomet],
 [srednji | nogomet]]
```

Zanimivo! Verjetnost *srednje* višine je največja med nogometaši. Ali podatek zadošča za spremembo prvotne odločitve?

Verjetnosti $P(X|Y)$ pravimo pogojna verjetnost spremenljivke X pri znanem Y . Opredeljuje verjetnost, da je v primerih razreda Y atribut X zavzame določeno vrednost.

Katera verjetnost pa nas v resnici zanima? Želimo, da izračun upošteva Markovo višino in oceni verjetnost vsakega od športov. To je verjetnost

$$P(Y|X)$$

oz. v Markovem primeru

$$P(Y|X = srednji)$$

Za izračun te verjetnosti uporabimo

5.8 Bayesov obrazec

Da bi izračunali verjetno razreda pri danih atributih $P(Y|X)$, potrebujemo verjetnost za vse možne kombinacije razreda Y in atributov X , ki jo označimo z $P(X, Y)$. Iz pravil o pogojni verjetnosti sledi:

$$P(X, Y) = P(X|Y) \cdot P(Y) = P(Y|X) \cdot P(X)$$

Iz česar sledi Bayesov obrazec za izračun $P(Y|X)$:

$$P(Y|X) = \frac{P(X|Y) \cdot P(Y)}{P(X)}$$

Izračun verjetnosti razreda Y pri znanih atributih X je torej odvisen od apriorne verjetnosti razreda $P(Y)$, pogojne verjetnosti $P(X|Y)$ in apriorne verjetnosti atributov $P(X)$. V Markovem primeru torej:

$$P(Y|X = srednji) = \frac{P(X = srednji|Y) \cdot P(Y)}{P(X = srednji)}$$

Če verjetnost ocenimo za vsako možno vrednost razreda Y , torej $\{\text{kosarka, nogomet, gimnastika}\}$, dobimo odgovor na prvotno vprašanje.

In [7]: `for sport in data.domain["sport"].values:`

```
subset_y = SameValue(data.domain["sport"], sport)(data)      # vsi sportniki danega sp
subset_x = SameValue(data.domain["visina"], "srednji")(data)   # vsi srednje visoki učen

subset_xy = SameValue(data.domain["visina"], "srednji")(subset_y) # vsi srednje visoki učen

# Izračunamo verjetnosti
p_y = len(subset_y) / len(data)
p_x = len(subset_x) / len(data)
p_xy = len(subset_xy) / len(subset_y)

p_yx = (p_xy * p_y) / p_x

print("Sport (Y): %s, napoved P(Y=%s | X=srednji): %f" % (sport, sport, p_yx))
```

Sport (Y): gimnastika, napoved P(Y=gimnastika | X=srednji): 0.285714

Sport (Y): kosarka, napoved P(Y=kosarka | X=srednji): 0.285714

Sport (Y): nogomet, napoved P(Y=nogomet | X=srednji): 0.428571

5.9 Implementacija Naivnega Bayesovega klasifikatorja

Naivni Bayesov klasifikator predpostavlja, da so atributi neodvisni med seboj, pri znanem razredu.

$$P(Y|X_1, X_2, \dots, X_p) = \frac{P(Y) \cdot P(X_1|Y) \cdot P(X_2|Y) \cdots P(X_p|Y)}{P(X)}$$

Vprašanje 5-2-1 Dopolni implementacijo naivnega Bayesovega klasifikatorja, ki je definiran v spodnjem odseku. Dopolniti je potrebno del kode, kjer izračunamo * verjetnostne porazdelitev razredov $P(Y)$ * verjetnostne porazdelitve atributov pri znanem razredu $P(X|Y)$

5.9.1 Sklepanje o podatkih

V primeru diskretnih atributov lahko obe porazdelitvi dobimo s *preštevanjem*. * $P(Y)$ Kolikokrat se v podatkih pojavi razred Y ? * $P(X|Y)$ Kolikokrat se v podatkih, ki spadajo v razred Y , pojavi atribut X ?

Kaj pa $P(X)$? Ta verjetnost je včasih težko izračunljiva, posebej pri visoko dimenzionalnih podatkih, saj ni nujno, da bodo v podatki prisotne vse kombinacije atributov. Na srečo ta vrednost ne vpliva na izbiro najverjetnejšega razreda za posamezen primer!

5.9.2 Napovedovanje

Za nov primer $X^* = (X_1^*, X_2^*, \dots, X_p^*)$ med vsemi vrednostmi razreda $Y = y$, izberi tisto, ki maksimizira naslednji izraz:

$$\arg \max_y P(Y = y) \cdot P(X_1^* | Y = y) \cdot P(X_2^* | Y = y) \cdots P(X_p^* | Y = y)$$

5.9.3 Log-transformacija

Težava pri zgornjem pristopu je praktične narave; množenje velikega števila verjetnosti hitro privede do zelo majhnih števil, ki lahko presežejo strojno natančnost. Najenostavnejša rešitev, ki privede do enake izbire razreda je naslednja

$$\arg \max_y \log P(Y = y) + \log P(X_1 | Y = y) + \log P(X_2 | Y = y) + \dots + \log P(X_p | Y = y)$$

Pri implementaciji si pomagaj s podatki potnikov ladje Titanic.

Podatke najprej razdelimo na učno in testno množico.

```
In [8]: from Orange.data import Table
        from numpy import random
        random.seed(42) # zagotovi ponovljivost naključnih rezultatov

        data = Table('titanic')
        inxs = list(range(len(data)))
        n = len(inxs)

        random.shuffle(inxs)

        data_training = data[inxs[:n//2]]
        data_test      = data[inxs[n//2:]]

        data_training.save('podatki/titanic-training.tab')
        data_test.save('podatki/titanic-test.tab')
```

Naložimo učne podatke in izračunamo verjetnosti.

```
In [9]: data = Table('podatki/titanic-training.tab')
        print(data.domain.class_var)
        print(data.domain.class_var.values)

        # P(X=child | Y = yes)
        filt_child = SameValue(data.domain["age"], "child")
        filt_survived = SameValue(data.domain["survived"], "yes")

        p_xy = len(filt_survived(filt_child(data))) / len(filt_survived(data))
        p_xy

survived
['no', 'yes']
```

```
Out[9]: 0.08379888268156424
```

```

In [10]: class NaiveBayes:
    """
    Naive Bayes classifier.

    :attribute self.probabilities
        Dictionary that stores
        - prior class probabilities  $P(Y)$ 
        - attribute probabilities conditional on class  $P(X/Y)$ 

    :attribute self.class_values
        All possible values of the class.

    :attribute self.variables
        Variables in the data.

    :attribute self.trained
        Set to True after fit is called.
    """

    def __init__(self):
        self.trained = False
        self.probabilities = dict()

    def fit(self, data):
        """
        Fit a NaiveBayes classifier.

        :param data
            Orange data Table.
        """
        class_variable = data.domain.class_var # class variable (Y)
        self.class_values = class_variable.values # possible class values
        self.variables = data.domain.attributes # all other variables (X)

        n = len(data) # number of all data points

        # Compute  $P(Y)$ 
        for y in self.class_values:

            # A not too smart guess (INCORRECT)
            self.probabilities[y] = 1/len(self.class_values)

            # <your code here>
            # Compute class probabilities and correctly fill
            # probabilities[y] = ...
            # Select all examples (rows) with class = y

            # </your code here>

        # Compute  $P(X/Y)$ 
        for y in self.class_values:

            # Select all examples (rows) with class = y

```



```

        filty = SameValue(class_variable, y)

        for variable in self.variables:
            for x in variable.values:

                # A not too smart guess (INCORRECT)
                p = 1 / (len(self.variables) * len(variable.values) * len(self.class_values))

                #  $P(\text{variable}=x|Y=y)$ 
                self.probabilities[variable, x, y] = p

                # <your code here>
                # Compute correct conditional class probability
                # probabilities[x, value, c] = ...
                #
                # Select all examples with class == y AND
                # variable x == value
                # Hint: use SameValue filter twice

                # </your code here>

        self.trained = True

def predict_instance(self, row):
    """
    Predict a class value for one row.

    :param row
        Orange data Instance.
    :return
        Class prediction.
    """
    curr_p = float("-inf") # Current highest "probability" (unnormalized)
    curr_c = None          # Current most probable class

    for y in self.class_values:
        p = np.log(self.probabilities[y])
        for x in self.variables:
            p = p + np.log(self.probabilities[x, row[x].value, y])

        if p > curr_p:
            curr_p = p
            curr_c = y

    return curr_c, curr_p

def predict(self, data):
    """
    Predict class labels for all rows in data.

```

```

        :param data
            Orange data Table.
        :return y
            NumPy vector with predicted classes.
        """

        n = len(data)
        predictions = list()
        confidences = np.zeros((n, ))

        for i, row in enumerate(data):
            pred, cf = self.predict_instance(row)
            predictions.append(pred)
            confidences[i] = cf

        return predictions, confidences

```

Rešitev je dostopna na: 205-2.ipynb

In [11]: %run '205-2.ipynb'

ERROR:root:File `205-2.ipynb.py` not found.

5.10 Uporaba klasifikatorja

Primer uporabe na podatkih potnikov ladje Titanic.

```

In [12]: model = NaiveBayes()
         model.fit(data)
         model.probabilities

```

```

Out[12]: {'no': 0.5,
          'yes': 0.5,
          (DiscreteVariable(name='status', values=['crew', 'first', 'second', 'third']),
           'crew',
           'no'): 0.041666666666666664,
          (DiscreteVariable(name='status', values=['crew', 'first', 'second', 'third']),
           'first',
           'no'): 0.041666666666666664,
          (DiscreteVariable(name='status', values=['crew', 'first', 'second', 'third']),
           'second',
           'no'): 0.041666666666666664,
          (DiscreteVariable(name='status', values=['crew', 'first', 'second', 'third']),
           'third',
           'no'): 0.041666666666666664,
          (DiscreteVariable(name='age', values=['adult', 'child']),
           'adult',
           'no'): 0.08333333333333333,
          (DiscreteVariable(name='age', values=['adult', 'child']),
           'child',
           'no'): 0.08333333333333333,
          (DiscreteVariable(name='sex', values=['female', 'male']),
           'female',

```

```

    'no'): 0.08333333333333333,
(DiscreteVariable(name='sex', values=['female', 'male']),
 'male',
 'no'): 0.08333333333333333,
(DiscreteVariable(name='status', values=['crew', 'first', 'second', 'third']),
 'crew',
 'yes'): 0.041666666666666664,
(DiscreteVariable(name='status', values=['crew', 'first', 'second', 'third']),
 'first',
 'yes'): 0.041666666666666664,
(DiscreteVariable(name='status', values=['crew', 'first', 'second', 'third']),
 'second',
 'yes'): 0.041666666666666664,
(DiscreteVariable(name='status', values=['crew', 'first', 'second', 'third']),
 'third',
 'yes'): 0.041666666666666664,
(DiscreteVariable(name='age', values=['adult', 'child']),
 'adult',
 'yes'): 0.08333333333333333,
(DiscreteVariable(name='age', values=['adult', 'child']),
 'child',
 'yes'): 0.08333333333333333,
(DiscreteVariable(name='sex', values=['female', 'male']),
 'female',
 'yes'): 0.08333333333333333,
(DiscreteVariable(name='sex', values=['female', 'male']),
 'male',
 'yes'): 0.08333333333333333}

```

```
In [13]: predictions, confidences = model.predict(data)
```

```

for row, p, c in zip(data, predictions, confidences):
    print("Row=%s, predicted class=%s confidence=%.5f" % (row, p, c))

```

```

Row=[third, adult, male | no], predicted class=no confidence=-8.84101
Row=[second, adult, female | no], predicted class=no confidence=-8.84101
Row=[crew, adult, male | no], predicted class=no confidence=-8.84101
Row=[crew, adult, male | no], predicted class=no confidence=-8.84101
Row=[third, adult, male | no], predicted class=no confidence=-8.84101
Row=[second, adult, male | no], predicted class=no confidence=-8.84101
Row=[crew, adult, male | no], predicted class=no confidence=-8.84101
Row=[second, adult, male | no], predicted class=no confidence=-8.84101
Row=[third, adult, male | yes], predicted class=no confidence=-8.84101
Row=[third, adult, male | no], predicted class=no confidence=-8.84101
Row=[third, adult, male | no], predicted class=no confidence=-8.84101
Row=[crew, adult, male | no], predicted class=no confidence=-8.84101
Row=[second, adult, male | no], predicted class=no confidence=-8.84101
Row=[crew, adult, male | no], predicted class=no confidence=-8.84101
Row=[third, adult, male | no], predicted class=no confidence=-8.84101
Row=[third, adult, male | no], predicted class=no confidence=-8.84101
Row=[crew, adult, female | yes], predicted class=no confidence=-8.84101
Row=[crew, adult, male | yes], predicted class=no confidence=-8.84101
Row=[first, adult, male | no], predicted class=no confidence=-8.84101
Row=[crew, adult, male | no], predicted class=no confidence=-8.84101
Row=[crew, adult, male | yes], predicted class=no confidence=-8.84101

```

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

5.11 Ocenjevanje uspešnosti klasifikacije

Za ocenjevanje uspešnosti klasifikacije vsak napovedani primer primerjamo s pripadajočim resničnim razredom. Štirje možni izidi primerjave so naslednji:

TP: True positives (pravilno napovedani pozitivni primeri)

FP: False positives (napačno napovedani negativni primeri)

TN: True negatives (pravilno napovedani negativni primeri)

FN: False negatives (napačno napovedani pozitivni primeri)

5.11.1 Delež pravilno razvrščenih razredov (ang. classification accuracy)

$$ca = \frac{TP + TN}{TP + TN + FP + FN}$$

Prednosti: * Enostaven izračun, jasna interpretacija * Uporabna mera za poljubno število razredov

Slabosti: * Lahko zavaja pri neuravnoteženih porazdelitvah razredov

5.11.2 Natančnost, priklic (ang. precision, recall)

$$p = \frac{TP}{TP + FP}$$

$$r = \frac{TP}{TP + FN}$$

Prednosti: * Enostaven izračun, jasna interpretacija * Ločitev obeh tipov napak (napačno pozitivni in napačno negativni primeri) * Uporabna tudi pri neuravnoteženih porazdelitvah razredov

Slabosti: * Uporabno pretežno za klasifikacijo v dva razreda * Težko povzeti obe meri ; približek je F1-vrednost (ang. F1-score)

$$F1 = 2 \frac{p \cdot r}{p + r}$$

Naredi sam/a. Napovej razrede na testni množici. Napovedane razrede primerjaj z resničnimi in izmeri klasifikacijsko točnost, natančnost, priklic in F1-vrednost.

```
In [14]: from sklearn.metrics import accuracy_score
         from sklearn.metrics import precision_score
         from sklearn.metrics import recall_score
         from sklearn.metrics import f1_score

         # uporaba metod:
         test_data      = Table('podatki/titanic-test.tab')
         predictions, _ = model.predict(test_data)
         truth          = [row["survived"].value for row in test_data]
         accuracy_score(truth, predictions)
```

```
Out[14]: 0.67938237965485926
```

Izziv. Nekateri atributi imajo verjetnost 0 pri posameznem razredu. Kako bi popravili klasifikator?

In [15]:

Razmisli. Kako bi dopolnili klasifikator, če bi bili nekateri atributi lahko tudi zvezni? Namig: spomni se vaj, ko smo spoznali *verjetnostne porazdelitve* zveznih spremenljivk.

Poglavje 6

Nenegativna matrična faktorizacija in priporočilni sistemi

Do sedaj smo obravnavali modele, ki so iz *več neodvisnih* napovedovali *eno* odvisno spremenljivko. V scenariju priporočilnega sistema smo tako za vsakega uporabnika zgradili svoj model.

Glavna motivacija metod za priporočilne sisteme je, da modeli uporabnikov med sabo *niso neodvisni*. Želimo enoten model, ki bo ovrednotil poljubno kombinacijo uporabnika in izdelka, ter implicitno izkoriščal medsebojno informacijo med različnimi modeli uporabnikov.

Eden od modelov, ki se zelo pogosto uporabljajo v praksi je model matrične faktorizacije. Ta predpostavlja matriko uporabnikov in izdelkov, ki ji predstavimo kot produkt dveh matrik *nižjega ranga*. Slednja lastnost omogoča stiskanje informacije in sklepanje o novih (ne-videnih, manjkajočih vrednostih) v izvorni matriki.

6.1 Uvodne definicije

Matriko podatkov \mathbf{X} , ki vsebuje manjkajoče vrednosti, z modelom matrične faktorizacije predstavimo na naslednji način:

$$\mathbf{X} = \mathbf{W}\mathbf{H}^T + \mathbf{E}$$

,

torej kot produkt matrike \mathbf{W} , ki predstavlja prostor vrstic, \mathbf{H} predstavlja prostor stolpcev, \mathbf{E} pa ostanek oz. napako. Matriki \mathbf{W}, \mathbf{H} si včasih predstavljamo kot hkratno gručenje stolpcev in vrstic. Matrike so naslednjih velikosti:

$$\mathbf{X} \in \mathbb{R}^{m \times n}, \mathbf{W} \in \mathbb{R}^{m \times r}, \mathbf{H} \in \mathbb{R}^{n \times r}, \mathbf{E} \in \mathbb{R}^{m \times n}$$

Predpostavljamo, da sta matriki \mathbf{W}, \mathbf{H} *nizkega ranga*, kar v praksi pomeni da celotno informacijo iz \mathbf{X} predstavljamo v stisnjeni obliki, torej

$$r < m, r < n$$

.

Predpostavljamo tudi, da so matrike \mathbf{X}, \mathbf{W} in \mathbf{H} nenegativne. Tedaj govorimo o **nenegativni matrični faktorizaciji** (NMF).

$$x_{i,j} > 0, w_{i,k} > 0, h_{j,k} > 0, \forall i, j, k$$

Matrika napake \mathbf{E} te omejitve nima (razmisli: zakaj?).

6.2 Definicija problema

Želimo torej poiskati matriki \mathbf{W} in \mathbf{H} , tako da vrednost napake karseda nizka. To lahko zapišemo kot naslednji optimizacijski problem:

$$\min_{\mathbf{W}, \mathbf{H}} \|\mathbf{X} - \mathbf{W}\mathbf{H}^T\|_F^2 = \min_{\mathbf{W}, \mathbf{H}} J$$

Oznaka $\|\mathbf{A}\|_F = \sqrt{\sum_{i,j} a_{i,j}^2}$ predstavlja *Frobeniusovo normo* matrike \mathbf{A} . (razmisli: Opaziš podobnost s srednjo kvadratično napako, ki smo jo spoznali v kontekstu linearne regresije?)

Vrednost J imenujemo *kriterijska funkcija*, problem iskanja minimuma pa *optimizacijski oz. minimizacijski problem*. **Posebnost** priporočilnih sistemov je ta, da napako računamo samo na vrednostih v \mathbf{X} , ki so znane. Kriterijska funkcija je torej:

$$J = \sum_{i,j | x_{i,j} \neq 0} (x_{i,j} - \sum_{l=1}^r w_{i,l} h_{j,l})^2$$

Za ta konkreten problem velja, da nima globalno optimalne rešitve za spremenljivke \mathbf{W}, \mathbf{H} . Vseeno ga lahko rešimo npr. z odvajanjem kriterijske funkcije in premikanjem v negativni smeri gradienta. Dobimo *pravila za posodabljanje* vrednosti v \mathbf{W}, \mathbf{H} :

Vse vrednosti $w_{i,k}$ in $h_{j,k}$ popravimo tako, da vrednost v prejšnji iteraciji *popravimo* v negativni smeri gradienta, s *korakom* η :

$$w_{i,k}^{(t+1)} = w_{i,k}^{(t)} - \eta \frac{\delta J}{\delta w_{i,k}} = w_{i,k}^{(t)} + \eta \sum_{j | x_{i,j} \neq 0} (x_{i,j} - \sum_{l=1}^r w_{i,l} h_{j,l}) (w_{i,k}^{(t)})$$

$$h_{j,k}^{(t+1)} = h_{j,k}^{(t)} - \eta \frac{\delta J}{\delta h_{j,k}} = h_{j,k}^{(t)} + \eta \sum_{i | x_{i,j} \neq 0} (x_{i,j} - \sum_{l=1}^r w_{i,l} h_{j,l}) (h_{j,k}^{(t)})$$

6.3 Stohastični gradientni sestop

Stohastični gradientni sestop (SGD) je postopek za reševanje optimizacijskih problemov, ki niso globalno rešljivi, za vse nastopajoče spremenljivke (v našem primeru vse $w_{i,k}$ in $h_{j,k}$) pa znamo izračunati odvod glede na kriterijsko funkcijo. To smo storili v prešnjem delu. Postopek za iskanje *lokalnega minimuma* je naslednji.

1. Naključno nastavi vrednosti vseh spremenljivk $w_{i,k}$ in $h_{j,k}$. V našem primeru velja $w_{i,k} > 0$ in $h_{j,k} > 0$.
2. V iteraciji $t = 1 \dots T$:
 - 2.1 V naključnem vrstnem redu posodablja $\forall i, k, j$

$$w_{i,k}^{(t+1)} = w_{i,k}^{(t)} - \eta \frac{\delta J}{\delta w_{i,k}}$$

$$h_{j,k}^{(t+1)} = h_{j,k}^{(t)} - \eta \frac{\delta J}{\delta h_{j,k}}$$

Shematski prikaz gradientnega sestopa za hipotetični spremenljivki w , h in kriterijsko funkcijo $J(w, h)$.

Vprašanje 6-1-1 Dopolni spodnjo implementacijo algoritma NMF, tako da uporabiš posodobitvena pravila v več iteracijah stohastičnega gradientnega sestopa. **Namig.** Pri računanju gradienta upoštevaj samo vrednosti $x_{i,j}$, ki so znane (različne od 0). Za učinkovito implementacijo izračuna vsot $\sum_i | x_{i,j} \neq 0$ in $\sum_j | x_{i,j} \neq 0$ najprej (pred začetkom iteracij): * za vsako vrstico i shranimo neničelne stolpce * za vsak stolpec j shranimo neničelne vrstice

```
In [1]: import numpy as np
import itertools
np.random.seed(42)

class NMF:

    """
    Fit a matrix factorization model for a matrix X with missing values.
    such that
        X = W H.T + E
    where
        X is of shape (m, n)      - data matrix
        W is of shape (m, rank) - approximated row space
        H is of shape (n, rank) - approximated column space
        E is of shape (m, n)      - residual (error) matrix
    """

    def __init__(self, rank=10, max_iter=100, eta=0.01):
        """
        :param rank: Rank of the matrices of the model.
        :param max_iter: Maximum number of SGD iterations.
        :param eta: SGD learning rate.
        """
        self.rank = rank
        self.max_iter = max_iter
        self.eta = eta

    def fit(self, X, verbose=False):
        """
        Fit model parameters W, H.
        :param X:
            Non-negative data matrix of shape (m, n)
            Unknown values are assumed to take the value of zero (0).
        """
        m, n = X.shape

        W = np.random.rand(m, self.rank)
        H = np.random.rand(n, self.rank)

        # Indices to model variables
        w_vars = list(itertools.product(range(m), range(self.rank)))
```

```

h_vars = list(itertools.product(range(n), range(self.rank)))

# Indices to nonzero rows/columns
nzcols = dict([(j, X[:, j].nonzero()[0]) for j in range(n)])
nzrows = dict([(i, X[i, :].nonzero()[0]) for i in range(m)])

# nzrows[i] <- vrni stolpce j, tako da x_ij > 0

# Errors
self.error = np.zeros((self.max_iter,))

for t in range(self.max_iter):
    np.random.shuffle(w_vars)
    np.random.shuffle(h_vars)

    for i, k in w_vars:
        # TODO: your code here
        # Calculate gradient and update W[i, k]
        pass

    for j, k in h_vars:
        # TODO: your code here
        # Calculate gradient and update H[j, k]
        pass

    self.error[t] = np.linalg.norm((X - W.dot(H.T))[X > 0])**2
    if verbose: print(t, self.error[t])

self.W = W
self.H = H

def predict(self, i, j):
    """
    Predict score for row i and column j
    :param i: Row index.
    :param j: Column index.
    """
    return self.W[i, :].dot(self.H[j, :])

def predict_all(self):
    """
    Return approximated matrix for all
    columns and rows.
    """
    return self.W.dot(self.H.T)

```

Rešitev najdete v 206-1.ipynb.

In [2]: %run 206-1.ipynb

ERROR:root:File `206-1.ipynb.py` not found.

Testirajmo metodo na matriki naključnih podatkov.

```
In [3]: m = 100      # St. vrstic
        n = 80       # St. stolpcev
        rank = 5     # Rang model
        error = 0.1  # Naključni šum
        A = np.random.rand(m, rank*2)
        B = np.random.rand(n, rank*2)
        X = A.dot(B.T) + error * np.random.rand(m, n) # generiramo podatke
```

Poženemo iskanje parametrov **W**, **H**.

```
In [4]: model = NMF(rank=rank, max_iter=20, eta=0.001)
        model.fit(X, verbose=True)
```

```
0 17520.902048
1 17520.902048
2 17520.902048
3 17520.902048
4 17520.902048
5 17520.902048
6 17520.902048
7 17520.902048
8 17520.902048
9 17520.902048
10 17520.902048
11 17520.902048
12 17520.902048
13 17520.902048
14 17520.902048
15 17520.902048
16 17520.902048
17 17520.902048
18 17520.902048
19 17520.902048
```

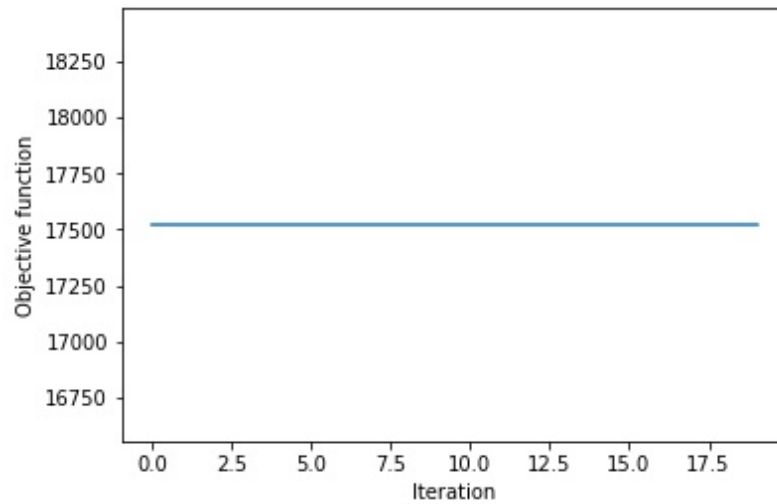
Napaka modela pada s številom iteracij.

```
In [5]: %matplotlib inline
        %config InlineBackend.figure_formats = ['jpg']
        import matplotlib
        matplotlib.figure.Figure.__repr__ = lambda self: (
            f"<{self.__class__.__name__} size {self.bbox.size[0]:g}"
            f"x{self.bbox.size[1]:g} with {len(self.axes)} Axes>")

        import matplotlib.pyplot as plt

        plt.figure()
        plt.plot(model.error)
        plt.xlabel("Iteration")
        plt.ylabel("Objective function")
```

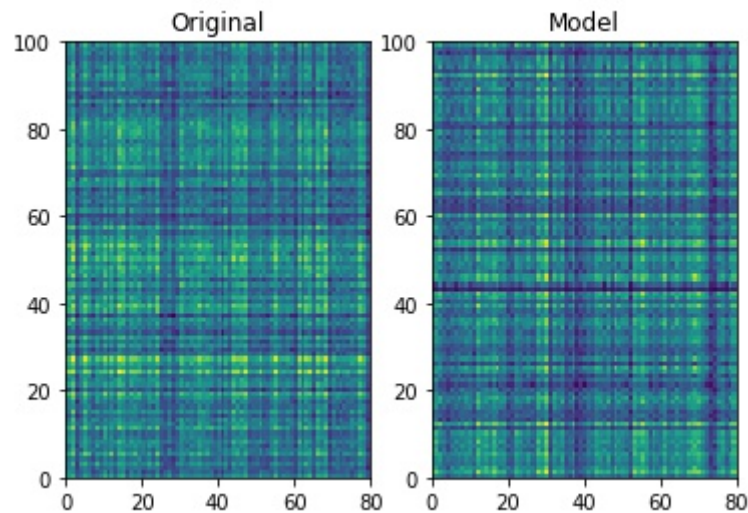
```
Out[5]: Text(0,0.5,'Objective function')
```



Primerjajmo model in izvirne podatke.

```
In [6]: fig, ax = plt.subplots(nrows=1, ncols=2)
        ax[0].pcolor(X)
        ax[0].set_title("Original")

        ax[1].pcolor(model.predict_all())
        ax[1].set_title("Model")
        plt.show()
```



Izračunamo pojasnjeno varianco.

```
In [7]: Xp = model.predict_all()
        expl_var = (np.var(X) - np.var(X-Xp))/np.var(X)
        expl_var
```

```
Out[7]: -0.38198039813109452
```

Vprašanje 6-1-2 Kako se pojasnjena varianca spreminja z rangom modela, št. iteracij?


```
In [8]: for rank in range(3, 10):
        model = NMF(rank=rank, max_iter=20, eta=0.001)
        model.fit(X)
        Xp = model.predict_all()
        expl_var = (np.var(X) - np.var(X-Xp))/np.var(X)
        print(rank, expl_var)

3 -0.30690999008
4 -0.433325233176
5 -0.446298698026
6 -0.577543124238
7 -0.630121526449
8 -0.753749766289
9 -1.036617355
```

Vprašanje 6-1-3 Preizkusi metodo NMF na podatkovni zbirki Jester. Podatki so razdeljeni na učno in testno množico, kjer je v učni množici prisoten delež p ocen. Poženi model na učni množici in izračunaj testno napako (RMSE, pojasnjeno varianco) na ocenah, ki niso bile uporabljene za učenje. Izračunaj, kako se testna napaka spreminja v odvisnosti od: * delež učnih ocen p , * ranga matrik modela (število r , parameter rank)

```
In [9]: # Naložimo podatkovno zbirko Jester z 1% upoštevanih ocen
def load_jester(p=0.05):
    """
    :param p: Probability of rating appearing in the training set.
    :return
        X training grades (retining with probability p)
        Y test grades (whole dataset)
    """

    Y = np.genfromtxt("podatki/jester-data.csv", delimiter=",", dtype=float, )
    Y = Y[:, 1:]
    Y[Y == 99] = 0
    Y[Y != 0] = Y[Y!=0] + abs(Y[Y!=0].min())

    # Separate data in test/train with probability p
    M = np.random.rand(*Y.shape)
    M_tr = M < p
    M_te = M > p
    X = Y * M_tr
    Y = Y * M_te

    return X, Y

# X: 1% podatkov, Y ostalih 99%
X, Y = load_jester(p=0.5)

X = X[:1000, :]
Y = Y[:1000, :]
print("X shape:", X.shape)
print("Y shape:", Y.shape)

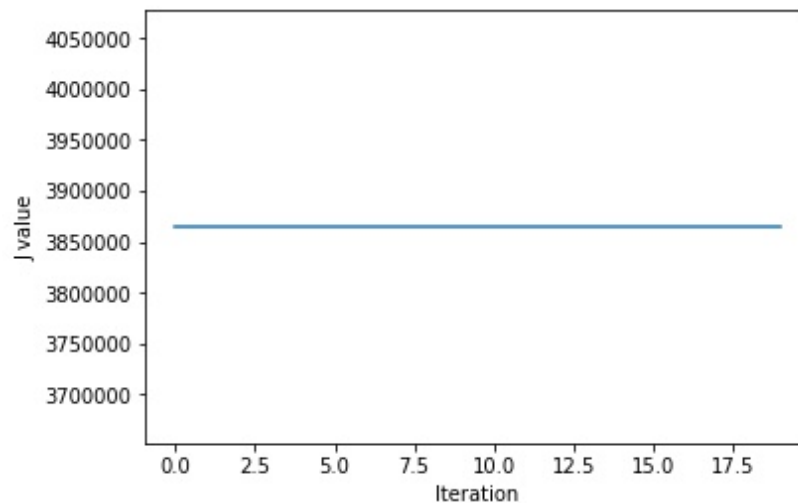
print("X, Nonzeros:", np.sum(X>0), "Total:", X.shape[0]*X.shape[1])
```

```
print("Y, Nonzeros:", np.sum(Y>0), "Total:", Y.shape[0]*Y.shape[1])
```

```
X shape: (1000, 100)
Y shape: (1000, 100)
X, Nonzeros: 35573 Total: 100000
Y, Nonzeros: 35772 Total: 100000
```

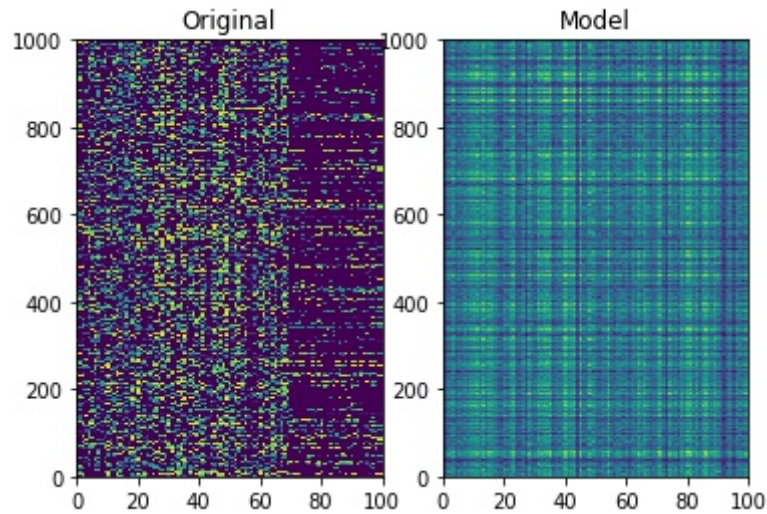
```
In [10]: model = NMF(rank=7, max_iter=20, eta=0.001)
         model.fit(X)
         Yp = model.predict_all()
```

```
In [11]: plt.figure()
         plt.plot(model.error)
         plt.xlabel("Iteration")
         plt.ylabel("J value")
         plt.show()
```



```
In [12]: fig, ax = plt.subplots(nrows=1, ncols=2)
         ax[0].pcolor(Y)
         ax[0].set_title("Original")

         ax[1].pcolor(Yp)
         ax[1].set_title("Model")
         plt.show()
```



```
In [13]: expl_var = (np.var(Y[Y>0]) - np.var(Y[Y>0] - Yp[Y>0])) / np.var(Y[Y>0])
          print(expl_var)

-0.00695500063601
```

Vprašanje 6-1-4 Na podatkovni zbirki Jester izberite eno celico z vrednostjo različno od 0 in jo nastavite na 0. Matriko faktorizirajte in napovedajte vrednost te celice.

Vprašanje 6-1-5 Poiščite celice, kjer je razlika med aproksimirano in originalno matriko največja.

Vprašanje 6-1-6 Ustvarite priporočilni sistem. Izberite nekaj uporabnikov in za vsakega izpišite pet še neocenjenih šal, ki mu bodo najbolj všeč, glede na napoved.

Poglavje 7

Omrežja

7.1 Knjižnica networkx

Enostavno opravljanje z omrežnimi podatki v Pythonu.

```
In [1]: import networkx as nx
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
%config InlineBackend.figure_formats = ['jpg']
import matplotlib
matplotlib.figure.Figure.__repr__ = lambda self: (
    f"<{self.__class__.__name__} size {self.bbox.size[0]:g}"
    f"x{self.bbox.size[1]:g} with {len(self.axes)} Axes>")
```

7.1.1 Gradnja grafa

Ustvarimo enostaven graf.

```
In [2]: G = nx.Graph()      # Undirected
        # G = nx.DiGraph()  # Directed

G.add_node("Ana")
G.add_nodes_from(["Bojan", "Cene", "Danica"])

G.add_edge("Ana", "Bojan")
G.add_edge("Ana", "Cene")
G.add_edge("Ana", "Danica")
G.add_edge("Bojan", "Danica")

In [3]: G.nodes
Out[3]: NodeView(('Ana', 'Bojan', 'Cene', 'Danica'))

In [4]: G.edges
Out[4]: EdgeView([('Ana', 'Bojan'), ('Ana', 'Cene'), ('Ana', 'Danica'), ('Bojan', 'Danica')])
```

Graf zapišemo v datoteko.

```
In [5]: nx.write_pajek(G, 'podatki/mreza-primer.net')
```

Preberemo .net datoteko v Graph strukturo.

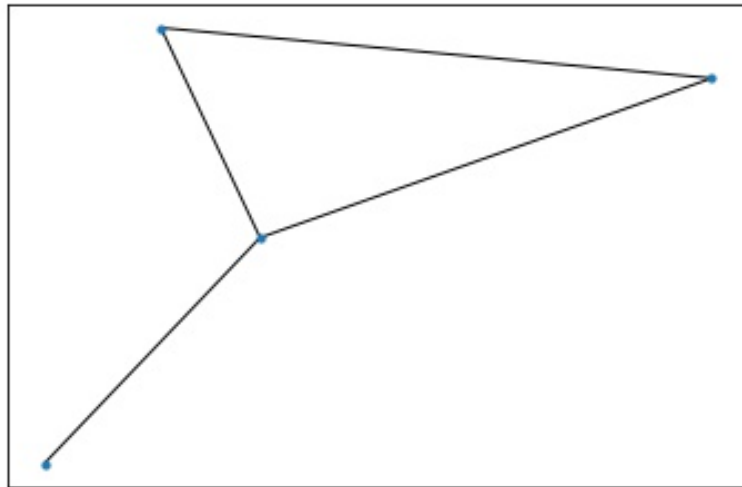
```
In [6]: G = nx.read_pajek('podatki/mreza-primer.net')
```

7.1.2 Prikaz grafa

Narišite strukturo grafov z uporabo matplotlib.

Za več možnosti glejte the documentation.

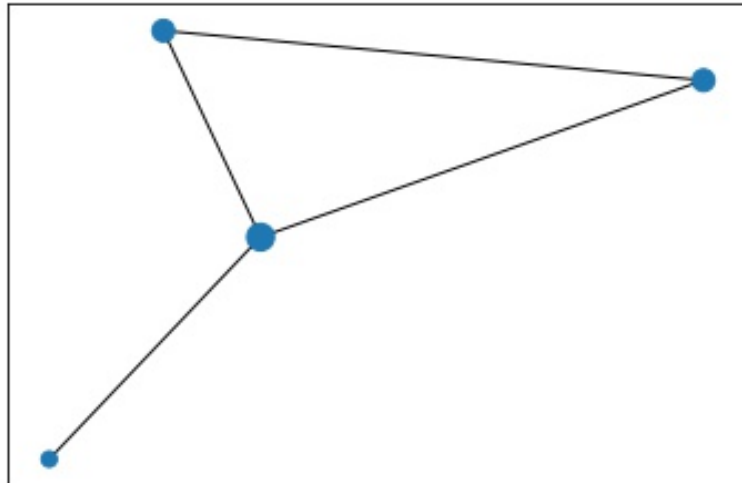
```
In [7]: plt.figure()
        np.random.seed(42)
        nx.draw_networkx(G, with_labels=False, node_size=10)
        plt.show()
```



Izračunamo velikosti vozlišč sorazmerno s številom povezav vozlišča. Rišite z uporabo `draw.networkx(..., node_size=node_size)`

```
In [8]: node_size = [50 * G.degree(ky) for ky in G.node]
        node_size

        plt.figure()
        np.random.seed(42)
        nx.draw_networkx(G, with_labels=False, node_size=node_size)
        plt.show()
```



7.1.3 Segmentacija omrežja

Iskanje močno povezanih komponent v omrežju.

Najprej, naložimo podatke. Ker je to mreža e-poštnih dopisnikov za določen naslov, odstranimo osrednje vozlišče (zakaj?).

```
In [9]: H = nx.read_pajek("podatki/email.net")
        H = nx.Graph(H)
```

```
# Remove central node
myself = "rok0"
H.remove_node(myself)
```

FileNotFoundError

Traceback (most recent call last)

```
<ipython-input-9-fe2cbc88c35d> in <module>
----> 1 H = nx.read_pajek("podatki/email.net")
      2 H = nx.Graph(H)
      3
      4 # Remove central node
      5 myself = "rok0"
```

```
</Users/tomazc/anaconda3/lib/python3.6/site-packages/decorator.py:decorator-gen-717> in read_pajek
```

```
~/anaconda3/lib/python3.6/site-packages/networkx/utils/decorators.py in _open_file(func_to_be_d
212     if is_string_like(path):
213         ext = splitext(path)[1]
--> 214         fobj = _dispatch_dict[ext](path, mode=mode)
215         close_fobj = True
216         elif hasattr(path, 'read'):
```

```
FileNotFoundError: [Errno 2] No such file or directory: 'podatki/email.net'
```

Nato najdemo k-povezanih komponent. k-komponent je povezani podgraf, za katerega moramo odstraniti vsaj k vozlišč, da jih razbijemo v več komponent. Intuitivno, podgrafi z veliko vrednostjo k težje razbijemo in so posledično močnejše povezani.

```
In [10]: from networkx.algorithms import approximation as apxa
         k_components = apxa.k_components(H)
```

```
k_components
```

```
-----
NameError                                Traceback (most recent call last)
```

```
<ipython-input-10-593d9d9e0cd7> in <module>
      1 from networkx.algorithms import approximation as apxa
----> 2 k_components = apxa.k_components(H)
      3
      4 k_components
```

```
NameError: name 'H' is not defined
```

Oglejmo si rešitve za določen k in pogledjmo število vozlišč na vsaki povezani komponenti.

```
In [11]: k = 2                                # Subgraphs of connectivity k
         sol = k_components[k]                 # Multiple solutions of k_components
         list(map(len, sol))                   # Each component breaks a graph
```

```
-----
NameError                                Traceback (most recent call last)
```

```
<ipython-input-11-eec503e45b4e> in <module>
      1 k = 2                                # Subgraphs of connectivity k
----> 2 sol = k_components[k]                 # Multiple solutions of k_components
      3 list(map(len, sol))                   # Each component breaks a graph
```

```
NameError: name 'k_components' is not defined
```

Za vsako povezano komponento dodelite črno barvo ustreznim vozliščem in belo za vsa druga vozlišča.

```
In [12]: colors_groups = list()
         for gi, group in enumerate(sol):
             colors_arr = ["red" if (n in group) else "gray" for n in H.node]
             colors_groups.append(colors_arr)
```

```
-----
```



```

NameError                                Traceback (most recent call last)

<ipython-input-12-8dab3a9a22b1> in <module>
      1 colors_groups = list()
----> 2 for gi, group in enumerate(sol):
      3     colors_arr = ["red" if (n in group) else "gray" for n in H.node]
      4     colors_groups.append(colors_arr)

NameError: name 'sol' is not defined

```

Narišite izbrano komponento.

```

In [13]: comp_index = 1
         plt.figure()
         nx.draw_networkx(H, with_labels=False,
                        node_color=colors_groups[comp_index],)
         plt.show()

-----

NameError                                Traceback (most recent call last)

<ipython-input-13-a60994b2ce76> in <module>
      1 comp_index = 1
      2 plt.figure()
----> 3 nx.draw_networkx(H, with_labels=False,
      4                 node_color=colors_groups[comp_index],)
      5 plt.show()

NameError: name 'H' is not defined

<Figure size 432x288 with 0 Axes>

```

7.2 Primer: analiza in vizualizacija omrežja elektronskih sporočil

V tej kratki vaji bomo spoznali osnove analize omrežij, format `.net` in funkcionalnosti modula Orange - Networks.

Pripravili smo funkcijo, ki iz email računa (protokol IMAP) prebere vse naslovnike sporočil. Tako zgradimo omrežje so-naslovnikov danega računa.

```

In [1]: from get_email import generate_addressee_network
         help(generate_addressee_network)

-----

ModuleNotFoundError                      Traceback (most recent call last)

<ipython-input-1-92545730b825> in <module>

```

```
----> 1 from get_email import generate_addressee_network
      2 help(generate_addressee_network)
```

```
ModuleNotFoundError: No module named 'get_email'
```

S spodnjo funkcijo zgradimo podatkovne datoteke .txt (seznam sonaslovnikov), .tab (atributi vozlišč), .net (graf omrežja). Oglej si kodo in format datotek.

```
In [2]: generate_addressee_network("someone@gmail.com", imap='imap.gmail.com', max_tuples=10000,
                                   email_folder="INBOX", file_prefix="ds",
                                   min_tuple_length=2, max_tuple_length=15)
```

```
-----
NameError
```

```
Traceback (most recent call last)
```

```
<ipython-input-2-fa8231f156e0> in <module>
----> 1 generate_addressee_network("someone@gmail.com", imap='imap.gmail.com', max_tuples=10000,
      2                               email_folder="INBOX", file_prefix="ds",
      3                               min_tuple_length=2, max_tuple_length=15)
```

```
NameError: name 'generate_addressee_network' is not defined
```

Uporabimo programski paket Orange oz. dodatek “Networks”.

Podatke naložimo z vtičnikom Network File, ki prebere podatke o vozliščih in povezavah.

Z uporabo algoritmov v vtičniku Network Clustering poiščemo močno povezane komponente v omrežju.

Naredi sam/a. Ponovi zgornjo analizo za podatke iz svojega e-poštnega računa.

```
In [3]: # ...
```

Naredi sam/a. Zgradi omrežje igralcev v podatkovni zbirki MovieLens.

```
In [4]: # ...
```

Naredi sam/a. Zgradi omrežje uporabnikov v podatkovni zbirki MovieLens.

```
In [5]: # ...
```

Poglavje 8

Zaporedja

8.1 Skriti Markovi modeli

Skriti markov model (ang. Hidden Markov model - HMM) je generativni model, ki ponazarja zaporedje diskretnih podatkov. Je razširitev Markovih verig (ang. Markov chain), na način da so opazovane spremenljivke odvisne od trenutnega skritega stanja.

Denimo, da opazujemo mete kovanca, ki jih izvaja druga oseba. Na voljo ima dva kovanca: pošten (F - fair) in utežen (L - loaded). Pri vsakem metu lahko opazujemo le izid (o ali -), ne pa tudi kovanca. Skriti Markov model je zapis tovrstnega problema, s poljubnim končnim številom tako skritih stanj in kot tudi opazovanih spremenljivk (abecede).

Primer zaporedja skritih stanj in opazovanih spremenljivk:

S: FFFFFLLLLLFFFFLLLLFFFFLL...

X: -o-o-ooooo-o--o-ooo-oo-ooo...

Celoten model je podan z naborom verjetnosti. Te predstavljajo parametre modela.

Verjenosto opazovanih spremenljivk X v koraku i glede trenutno stanje S:

$$P(X_i = o \mid S_i = F) = \frac{1}{2}, \quad P(X_i = - \mid S_i = F) = \frac{1}{2}$$

$$P(X_i = o \mid S_i = L) = \frac{19}{20}, \quad P(X_i = - \mid S_i = L) = \frac{1}{20}$$

Za vsako skrito stanje je torej definirana verjetnostna porazdelitev opazovanih spremenljivk.

V praktičnih primerih uporabe HMM se stanja ohranjajo. Verjetnost ohranitve stanja je torej navadno večja od zamenjave stanja. Verjetnosti prehodov podajajo drugo skupino parametrov.

$$P(S_{i+1} = F \mid S_i = F) = \frac{19}{20}, \quad P(S_{i+1} = L \mid S_i = F) = \frac{1}{20}$$

$$P(S_{i+1} = L \mid S_i = L) = \frac{19}{20}, \quad P(S_{i+1} = F \mid S_i = L) = \frac{1}{20}$$

Navadno definiramo tudi začetne verjetnosti skritih stanj (verjetnost v koraku $i = 0$):

$$P(S_0 = F) = \frac{1}{2}, \quad P(S_0 = L) = \frac{1}{2}$$

Tako definiran model uporabljamo za praktične naloge, kot so: * generiranje zaporedij iz danega modela,

- učenje parametrov modela iz danih podatkov:
 - podana so skrita stanja in opazovane spremenljivke (štetje pojavitev)
 - podane so samo opazovanje spremenljivke in število skritih stanj (algoritem Baum-Welch)
- napoved skritih stanj za dano zaporedje opazovanih spremenljivk pri danem modelu (algoritma Viterbi ter Posterior-decoding)

Primeri praktičnih problemov, ki jih rešujemo z uporabo Skritih Markovih modelov: * prepoznavanje in generiranje govora, * strojno prevajanje, * prepoznavanje pisave, * segmentacija besedil (prepoznavanje besednih vrst), * analiza biološki zaporedij (iskanje genov, poravnava zaporedij), * kriptanaliza, * ...

Model lahko zapišemo s slovarjem slovarjev. Na primer, za metanje kovancev:

```
In [1]: # Transition matrix
T = {"F": {"F": 0.95, "L": 0.05},
      "L": {"F": 0.05, "L": 0.95}}

# Emission matrix
E = {"F": {"o": 0.5, "-" : 0.5},
      "L": {"o": 0.95, "-" : 0.05}}
start = "F"
```

8.1.1 Generiranje zaporedij

Naredi sam/a. Zapiši funkcijo `generate_hmm_sequence`, ki sprejme skriti Markov model in vrne zaporedje dolžine `n` (skrito in vidno zaporedje).

Še prej zapišite funkcijo `weighted_choice`, ki na podlagi uteži (v vrednosti) naključno izbere vrednost (v ključu slovarja).

```
In [2]: import random
random.seed(42)

def weighted_choice(weighted_items):
    """Random choice given the list of elements and their weights
       example weighted_items: {"F": 0.95, "L": 0.05}
    """

    pass
```

Zdaj pa funkcijo `generate_hmm_sequence`:

```
In [3]: def generate_hmm_sequence(h, T, E, n):
    """
    h: given start state,
    T: transition probabilities
    E: emission probabilities
    n: sequence length

    return:
        hidden_sequence
        observable_sequence
```

```
"""
pass
```

Rešitev lahko pogledate v 208-1.ipynb.

```
In [4]: %run '208-1.ipynb'
```

```
ERROR:root:File `208-1.ipynb.py` not found.
```

Generiraj nekaj zaporedij različnih dolžin.

```
In [5]: list(generate_hmm_sequence('F', T, E, 5))
```

```
-----

TypeError                                Traceback (most recent call last)

<ipython-input-5-b7ac19a359e0> in <module>
----> 1 list(generate_hmm_sequence('F', T, E, 5))

TypeError: 'NoneType' object is not iterable
```

```
In [6]: list(generate_hmm_sequence('F', T, E, 20))
```

```
-----

TypeError                                Traceback (most recent call last)

<ipython-input-6-d419da9b7f30> in <module>
----> 1 list(generate_hmm_sequence('F', T, E, 20))

TypeError: 'NoneType' object is not iterable
```

Model poskusite uporabiti tudi na primeru goljufive igralnice. Kovanec smo zamenjali z igralno kocko, ki vrača vrednosti 1-6.

```
In [7]: # Alphabet
```

```
A = ["1", "2", "3", "4", "5", "6"]
```

```
# Emission probabilities
```

```
E = {"F": {a: 1/6. for a in A},
      "L": {a: 1/10. if a != "6" else 0.5 for a in A}}
```

```
# Transition probabilities
```

```
T = {0: {0: 0, "F": 0.5, "L": 0.5},
      "F": {0: 0, "F": 0.95, "L": 0.05},
      "L": {0: 0, "F": 0.1, "L": 0.9}}
start = "F"
```

```
In [8]: list(generate_hmm_sequence('F', T, E, 5))
```

```
-----
```

```

TypeError                                Traceback (most recent call last)

<ipython-input-8-b7ac19a359e0> in <module>
----> 1 list(generate_hmm_sequence('F', T, E, 5))

TypeError: 'NoneType' object is not iterable

```

```
In [9]: list(generate_hmm_sequence('F', T, E, 20))
```

```

-----

TypeError                                Traceback (most recent call last)

<ipython-input-9-d419da9b7f30> in <module>
----> 1 list(generate_hmm_sequence('F', T, E, 20))

TypeError: 'NoneType' object is not iterable

```

8.1.2 Učenje parametrov modela iz podatkov

Napišite funkcijo `learn_hmm`, ki bo sprejela vidno in skrito zaporedje, ter vrnila parametre skritega Markovega modela (slovarja `T` in `E`).

```
In [10]: from collections import Counter
```

```

def normalize(dic, eps=1e-8):
    """
    Normalize probabilities of items in a dictionary `dic`.
    Correct probabilities with a small constant to prevent probability 0.

    dic = {"o": 90, "-": 10}

    return
        dic = {"o": 0.9, "-": 0.1}
    """
    pass

def learn_hmm(h, x):
    """
    h: hidden sequence
    x: observable sequence
    """

    return T, E

```

Rešitev lahko pogledate v 208-1.ipynb.

```
In [11]: %run '208-1.ipynb'
```

ERROR:root:File `208-1.ipynb.py` not found.

```
In [12]: n = 40
         h, x = zip(*list(generate_hmm_sequence('F', T, E, n)))
```

```
-----

TypeError                                Traceback (most recent call last)

<ipython-input-12-731639424f0c> in <module>
      1 n = 40
----> 2 h, x = zip(*list(generate_hmm_sequence('F', T, E, n)))

TypeError: 'NoneType' object is not iterable
```

```
In [13]: # Estimated parameters from data
         T_est, E_est = learn_hmm(h, x)
```

```
-----

NameError                                Traceback (most recent call last)

<ipython-input-13-88a310a8e988> in <module>
      1 # Estimated parameters from data
----> 2 T_est, E_est = learn_hmm(h, x)

NameError: name 'h' is not defined
```

```
In [14]: T_est
```

```
-----

NameError                                Traceback (most recent call last)

<ipython-input-14-86299011f15e> in <module>
----> 1 T_est

NameError: name 'T_est' is not defined
```

```
In [15]: E_est
```

```
-----

NameError                                Traceback (most recent call last)

<ipython-input-15-865fe85fe9c1> in <module>
----> 1 E_est
```

```
NameError: name 'E_est' is not defined
```

Primerjamo z dejanskimi:

```
In [16]: T
```

```
Out[16]: {0: {0: 0, 'F': 0.5, 'L': 0.5},
          'F': {0: 0, 'F': 0.95, 'L': 0.05},
          'L': {0: 0, 'F': 0.1, 'L': 0.9}}
```

```
In [17]: E
```

```
Out[17]: {'F': {'1': 0.16666666666666666,
                '2': 0.16666666666666666,
                '3': 0.16666666666666666,
                '4': 0.16666666666666666,
                '5': 0.16666666666666666,
                '6': 0.16666666666666666},
          'L': {'1': 0.1, '2': 0.1, '3': 0.1, '4': 0.1, '5': 0.1, '6': 0.5}}
```

8.1.3 Viterbijev algoritem

Algoritem za iskanje najverjetnejšega zaporedja skritih stanj (Viterbi).

Zaporedja, s katerimi delamo, so lahko zelo dolga. Množenje (majhnih) verjetnosti nas lahko hitro privede do napake *underflow*. Težavi se izognemo tako, da namesto množenja verjetnosti, seštevamo logaritme verjetnosti.

```
In [18]: import math
def logmv(a):
    min_val = 0.0000000001
    return math.log(max(a, min_val))

def viterbi_log(s, hmm):
    t, e = hmm

    # seznam skritih stanj
    zh = set()
    for h, tmpd in e.items():
        zh.add(h)
    zh = [0] + list(zh)

    # Create table V
    V = [{ } for i in range(len(s)+1)]
    ptr = [{ } for i in range(len(s)+1)]

    # Initialize i = 0; V(0, 0) = 1; V(k, 0) = 0 for k > 0
    for k in zh:
        V[0][k] = logmv(0.0) #t[0][k]*e[k][s[0]]
    V[0][0] = logmv(1.0)

    # for l = 1 : n, compute
    for i in range(1, len(s)+1):
        for l in zh:
```



```

vals = [(V[i-1][k] + logmv(t[k].get(l, 0.0)), k) for k in zh]
max_val, max_k = max(vals)
V[i][l] = logmv(e.get(l, {}), get(s[i-1], 0.0)) + max_val
ptr[i][l] = max_k

# trace back
pi = []
pi_L = max([(V[-1][k], k) for k in zh])[1]
pi.append(pi_L)

for p in ptr[-1:1:-1]:
    pi.append(p[pi[-1]])

pi.reverse()
return V, zh, ptr, "".join(pi)

```

Pokliči funkcijo, ki za dano zaporedje x in za dani model $(T \text{ in } E)$ vrne najbolj verjetno skrito pot (h_najv).

Primerjaj jo z dejansko skrito potjo.

```
In [19]: # Alphabet
A = ["1", "2", "3", "4", "5", "6"]

# Emission probabilities
E = {"F": {a: 1/6. for a in A},
     "L": {a: 1/10. if a != "6" else 0.5 for a in A}}

# Transition probabilities
T = {0: {0: 0, "F": 0.5, "L": 0.5},
     "F": {0: 0, "F": 0.95, "L": 0.05},
     "L": {0: 0, "F": 0.1, "L": 0.9}}

hmm = (T, E)
#s = "1233516266666666666666663561253612365611213231524112666666666611666666666612"

random.seed(442)
skrito, vidno = zip(*generate_hmm_sequence('L', T, E, 71))
skrito = "".join(skrito)
vidno = "".join(vidno)

_, _, _, napoved = viterbi_log(vidno, hmm)

print(vidno)
print(skrito)
print(napoved)
```

[illegible]

```
<ipython-input-19-7ee266dc12ce> in <module>
    16
    17 random.seed(442)
--> 18 skrito, vidno = zip(*generate_hmm_sequence('L', T, E, 71))
```

```
19 skrito = "".join(skrito)
20 vidno = "".join(vidno)
```

TypeError: type object argument after * must be an iterable, not NoneType

Izračunaj delež ujemanja:

```
In [20]: sum(pi == pj for pi, pj in zip(skrito, napoved))/len(skrito)
```

```
-----

NameError                                Traceback (most recent call last)

<ipython-input-20-f814a227f9fd> in <module>
----> 1 sum(pi == pj for pi, pj in zip(skrito, napoved))/len(skrito)

NameError: name 'skrito' is not defined
```

8.2 Časovne vrste

Modeliranje časovnih vrst je pomemben del ekonomskih modelov, borznega posredništva, analize časovnih meritev v fiziki, biologiji, kemiji, ipd.

Podatki v obliki časovnih vrst se od dosedanjih scenarijev razlikujejo po pomembni lastnosti: vzorci niso neodvisni med seboj. Podatke predstavlja vektor časovnih točk (ki niso nujno enako oddaljene):

$$\mathbf{t} = (t_1, t_2, \dots, t_n)$$

Običajno nas zanima funkcija oz. signal v vsaki časovni točki.

$$x(\mathbf{t}) = (x(t_1), x(t_2), \dots, x(t_n))$$

```
In [1]: import os
import sys
import mlp

%matplotlib inline
%config InlineBackend.figure_formats = ['jpg']
import matplotlib
matplotlib.figure.Figure.__repr__ = lambda self: (
    f"<{self.__class__.__name__} size {self.bbox.size[0]:g}"
    f"x{self.bbox.size[1]:g} with {len(self.axes)} Axes>")

import matplotlib.pyplot as plt
plt.style.use('PR.mplstyle')
import matplotlib.cm as cm
import numpy as np
import scipy
import os
```

```
import GPY

np.random.seed(42)

-----

ModuleNotFoundError                                Traceback (most recent call last)

<ipython-input-1-86e0c314098b> in <module>
      1 import os
      2 import sys
----> 3 import mlp
      4
      5 get_ipython().run_line_magic('matplotlib', 'inline')

ModuleNotFoundError: No module named 'mlp'
```

8.2.1 Primerjava časovnih vrst

Oglejmo si preprost primer dveh podobnih signalov $x(t)$ in $y(t)$.

```
In [2]: resolution = 100
        t = np.linspace(-5, 5, resolution)
        x = np.sin(t) * np.cos(2*t) + 0.2*np.random.rand(1, resolution).ravel()
        y = -np.sin(t-2) * np.cos(2*t + 4) + 0.2*np.random.rand(1, resolution).ravel()

        from sklearn.linear_model import LinearRegression
        model = LinearRegression()
        model.fit(t.reshape((len(t), 1)), x)
        z = model.predict(t.reshape((len(t), 1)))

        plt.figure()
        plt.plot(x, "b.-", label="$x(t)$")
        # plt.plot(z, "r-", label="$x(t)$")
        plt.plot(y, "r.-", label="$y(t)$")
        plt.gca().set_xticklabels(np.linspace(-5, 5, 6))
        plt.xlabel("t")
        plt.legend()
        plt.show()

-----

NameError                                Traceback (most recent call last)

<ipython-input-2-14cb7c2528c3> in <module>
      1 resolution = 100
----> 2 t = np.linspace(-5, 5, resolution)
      3 x = np.sin(t) * np.cos(2*t) + 0.2*np.random.rand(1, resolution).ravel()
      4 y = -np.sin(t-2) * np.cos(2*t + 4) + 0.2*np.random.rand(1, resolution).ravel()
      5
```

```
NameError: name 'np' is not defined
```

Opazimo, da sta si signala zelo podobna, vendar so vrhovi zelo oddaljeni med seboj. Kako izmeriti to razdaljo (npr. za potrebe hierarhičnega gručenja)? Evklidska razdalja vrne navidez zelo visoko vrednost.

```
In [3]: np.linalg.norm(x[:-2]-x[2:], ord=2)
```

```
-----

NameError                                Traceback (most recent call last)

<ipython-input-3-a6d5c6a3198d> in <module>
----> 1 np.linalg.norm(x[:-2]-x[2:], ord=2)

NameError: name 'np' is not defined
```

Korelacija med signaloma je nizka oz. celo obratna.

```
In [4]: scipy.stats.pearsonr(x, y)[0]
```

```
-----

NameError                                Traceback (most recent call last)

<ipython-input-4-f38797f4b1fb> in <module>
----> 1 scipy.stats.pearsonr(x, y)[0]

NameError: name 'scipy' is not defined
```

8.2.2 Dinamična poravnava signalov

V splošnem imamo dva različno dolga signala:

$$x(\mathbf{t}) = (x(t_1), x(t_2), \dots, x(t_n))$$

$$y(\mathbf{t}) = (y(t_1), y(t_2), \dots, y(t_m))$$

Dinamična poravnava signalov (ang. Dynamic time warping, DTW) je algoritem dinamičnega programiranja, ki poišče signala $x_w(\mathbf{t})$ in $y_w(\mathbf{t})$, tako, da je razdalja med vrednostmi signalov $|x_w(t_k) - y_w(t_k)|$ čim manjša. Dovoljeno je lokalno raztezanje in krčenje obeh signalov.

Algoritem DTW sestavi matriko \mathbf{W} velikosti $m \times n$ ki hrani razdalje, tako da

$$w_{ij} = |x(t_i) - y(t_j)|$$

Cilj algoritma DTW je iskanje poti dolžine $\max(m, n)$, ki gre iz levega spodnjega v desni zgornji kot matrike \mathbf{W} , tako da zmanjšamo skupno razdaljo

$$\min \sum_k \sqrt{w_k}$$

Rezultat algoritma je matrika poravnave, optimalna pot in skupna razdalja med signaloma.

```
In [5]: dist, cost, path = mlp.dtw_std(x, y, dist_only=False)
        print("Oddaljenost med x in y", dist)
```

```
-----

NameError                                Traceback (most recent call last)

<ipython-input-5-68b036f3cefd> in <module>
----> 1 dist, cost, path = mlp.dtw_std(x, y, dist_only=False)
      2 print("Oddaljenost med x in y", dist)

NameError: name 'mlp' is not defined
```

```
In [6]: fig = plt.figure(2)
        ax = fig.add_subplot(111)
        plot1 = plt.imshow(cost.T, origin='lower', cmap=cm.gray, interpolation='nearest')
        plot2 = plt.plot(path[0], path[1], 'y', label="Pot w_k")
        xlim = ax.set_xlim((-0.5, cost.shape[0]-0.5))
        ylim = ax.set_ylim((-0.5, cost.shape[1]-0.5))
        plt.xlabel("Indeks $y$")
        plt.ylabel("Indeks $x$")
        plt.title("Optimalna matrika poravnave",)
        plt.legend( loc=4)
        plt.show()
```

```
-----

NameError                                Traceback (most recent call last)

<ipython-input-6-6b4a84ec98cc> in <module>
----> 1 fig = plt.figure(2)
      2 ax = fig.add_subplot(111)
      3 plot1 = plt.imshow(cost.T, origin='lower', cmap=cm.gray, interpolation='nearest')
      4 plot2 = plt.plot(path[0], path[1], 'y', label="Pot w_k")
      5 xlim = ax.set_xlim((-0.5, cost.shape[0]-0.5))

NameError: name 'plt' is not defined
```

Poravnana signala dobimo s poravnavo vrednosti na ustreznih mestih v zaporedju.

```
In [7]: xw = x[path[0]]
        yw = y[path[1]]
```

```

NameError                                Traceback (most recent call last)

<ipython-input-7-8f2a52d1f936> in <module>
----> 1 xw = x[path[0]]
      2 yw = y[path[1]]

NameError: name 'x' is not defined

```

Korelacija med signaloma je bistveno večja!

```
In [8]: scipy.stats.pearsonr(xw, yw)[0]
```

```

-----

NameError                                Traceback (most recent call last)

<ipython-input-8-edc467517e3d> in <module>
----> 1 scipy.stats.pearsonr(xw, yw)[0]

NameError: name 'scipy' is not defined

```

Oba signala sta lokalno deformirana.

```
In [9]: plt.figure()
        plt.plot(xw, label="$x_w(t)$")
        plt.plot(yw, label="$y_w(t)$")
        plt.legend()
        plt.show()
```

```

-----

NameError                                Traceback (most recent call last)

<ipython-input-9-d80ac1b5a14f> in <module>
----> 1 plt.figure()
      2 plt.plot(xw, label="$x_w(t)$")
      3 plt.plot(yw, label="$y_w(t)$")
      4 plt.legend()
      5 plt.show()

NameError: name 'plt' is not defined

```

Naredi sam/a. Poišči slovenske občine s podobnimi trendi spreminjanja gostote prebivalstva. Oglej si trende nekaj najpodobnejših krajev.

```
In [10]: x = np.loadtxt('podatki/ages/Maribor_starost-20-24_let.txt')
        y = np.loadtxt('podatki/ages/Ljubljana_starost-20-24_let.txt')
```

```

NameError                                Traceback (most recent call last)

<ipython-input-10-875ce3b25d70> in <module>
----> 1 x = np.loadtxt('podatki/ages/Maribor_starost-20-24_let.txt')
      2 y = np.loadtxt('podatki/ages/Ljubljana_starost-20-24_let.txt')

NameError: name 'np' is not defined

In [11]: # ... load all cities and store them into a matrix
import glob
labels = []
X = []

for f in glob.glob('podatki/ages/*_starost-20-24_let.txt'):
    city = os.path.basename(f).split("_")[0]
    labels.append(city)
    data = np.loadtxt(f)
    d = scipy.stats.zscore(data)
    X.append(d)
X = np.array(X)
X.shape

-----

NameError                                Traceback (most recent call last)

<ipython-input-11-c3983a4792d4> in <module>
      7 city = os.path.basename(f).split("_")[0]
      8 labels.append(city)
----> 9 data = np.loadtxt(f)
     10 d = scipy.stats.zscore(data)
     11 X.append(d)

NameError: name 'np' is not defined

```

Namig. Uporabi hierarhično razzvržanje, kjer namesto funkcije za razdaljo (`sch.linkage(metric=...)`) po-
daš razdaljo izmerjeno po DTW.

```

In [12]: import scipy.cluster.hierarchy as sch
        # TODO: your code here

```

8.3 Napovedovanje trendov

8.3.1 Gaussovi procesi

Gaussovi procesi so paradni konj družine modelov, ki ji pravimo neparametrična regresija. Napovedni model tokrat ne bo predstavljen kot vektor uteži, temveč bo vsa informacija za napovedovanje vsebovana v učnem

vzorcu. Prednosti pristopa sta: * predpostavka, da so primeri neodvisni med seboj ne drži več, * model se posodobi, ko se pojavijo novi primeri.

Glana predpostavka je naslednja. Funkcija $x(\mathbf{t})$ je porazdeljena po multivariatni normalni porazdelitvi. To ne pomeni, da je vsaka vrednost ($x(t_i)$) porazdeljena normalno, temveč da celoten vektor $x(\mathbf{t})$ prihaja iz skupne normalne porazdelitve, kjer so posamezne vrednosti ($x(t_i)$) lahko odvisne med sabo!

Torej:

$$x(\mathbf{t}) \sim \mathcal{N}(m(\mathbf{t}), k(\mathbf{t}, \mathbf{t}))$$

Funkcija $m(\mathbf{t})$ je funkcija povprečja, funkcija $k(\mathbf{t}, \mathbf{t})$ pa funkcija kovariance. Večinoma funkcijo povprečja nastavimo na 0, na obliko modela pa bistveno vpliva struktura kovariance. Zapišemo

$$x(\mathbf{t}) \sim \mathcal{N}(\mathbf{0}, k(\mathbf{t}, \mathbf{t}))$$

V praksi to pomeni, da za vsak končen učni vzorec ($x(t_1), x(t_2), \dots, x(t_n)$) lahko statistično sklepamo o vsaki drugi časovni točki. Ob predpostavki normalne porazdelitve tako lahko analitično izračunamo naslednjo pogojno verjetnost

$$p(x(t_*) | x(t_1), x(t_2), \dots, x(t_n))$$

Za vsako časovno točko t_* . Kje je torej skrita informacija o podobnosti med primeri? V kovariančni funkciji!

8.3.2 Primer

Oglejmo si spreminjanje bruto državnega proizvoda v Združenih državah amerike med leti 1970 in 2012.

In [1]: `data = np.genfromtxt("podatki/GDP-USD-countries.csv", delimiter=",")`

```
i = 205
x = data[0, 1:]
y = data[i, 1:]
n = len(x)
```

```
plt.figure()
plt.plot(x, y)
plt.show()
```

NameError

Traceback (most recent call last)

```
<ipython-input-1-e6615157aac7> in <module>
----> 1 data = np.genfromtxt("podatki/GDP-USD-countries.csv", delimiter=",")
      2
      3 i = 205
      4 x = data[0, 1:]
      5 y = data[i, 1:]
```

NameError: name 'np' is not defined

Uporabili bomo tipično funkcijo kovariance, eksponentno-kvadratno funkcijo (ang. “Exponentiated-quadratic” oz. “RBF”), dano z izrazom

$$k(t, t') = \exp\left\{-\frac{\|t - t'\|^2}{2\ell^2}\right\}$$

kjer parametru ℓ pravimo dolžina vpliva (ang. lengthscale).

In [2]: resolution = 10

```
t = np.linspace(-5, 5, resolution)
x = np.sin(t) * np.cos(2*t) + 0.2*np.random.rand(1, resolution).ravel()

t = t.reshape((len(t), 1))[0::1]
x = x.reshape((len(x), 1))[0::1]

x = x - x.mean()

# Gaussian kernel, RBF, sq exp, exponentiated quadratic, stationary kernel
kernel = GPy.kern.RBF(input_dim=1, lengthscale=1)
model = GPy.models.GPRegression(t, x, kernel, noise_var=0.1) # noise_var=10.0

# model.optimize(messages=True)
model.plot(lower=5, upper=95)
plt.gca().set_xlabel("t")
plt.gca().set_ylabel("GDP($)")
plt.xlim(-5, 10)
plt.show()
```

```
-----

NameError                                Traceback (most recent call last)

<ipython-input-2-0a9721c61fe7> in <module>
      1 resolution = 10
      2
----> 3 t = np.linspace(-5, 5, resolution)
      4 x = np.sin(t) * np.cos(2*t) + 0.2*np.random.rand(1, resolution).ravel()
      5

NameError: name 'np' is not defined
```

Dobimo model neparametrične regresije, ki nam omogoča ekstrapolacijo v naslednja leta. Opazimo, da se negotovost (varianca) napovedi povečuje, s tem ko se oddaljujemo od podatkov.

8.3.3 Kovariančne funkcije

Kovariančne funkcije bistveno vplivajo na obliko družine funkcij, ki jih vzorčimo iz Gaussovega Procesa. Oglejmo si nekaj tipičnih primerov kovariačnih funkcij. Bodite pozorni na lastnosti družine funkcij.

In [3]: kernels = [GPy.kern.Linear, GPy.kern.RBF, GPy.kern.Brownian, GPy.kern.PeriodicExponential, GPy.kern.
names = ["linear", "exp", "brownian", "per_exp", "poly"]

```

fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(14, 4))
for i, k, name in zip(range(5), kernels, names):

    # Narisi funkcijo kovariance
    knl = k(input_dim=1)
    knl.plot(x=1, ax=axes[0][i])
    axes[0][i].set_xlabel("t, t'")
    axes[0][i].set_ylabel("k(t, t')")
    axes[0][i].set_title(name)

    # Narisi vzorce iz družine funkcij
    X = np.linspace(0, 10, 100).reshape((100, 1))
    mu = np.zeros((100, ))
    C = knl.K(X,X)
    Z = np.random.multivariate_normal(mu,C,5)
    for z in Z:
        axes[1][i].plot(z)

fig.tight_layout()
plt.show()

```

NameError

Traceback (most recent call last)

```

<ipython-input-3-0549116d34f6> in <module>
----> 1 kernels = [ GPy.kern.Linear, GPy.kern.RBF, GPy.kern.Brownian, GPy.kern.PeriodicExponential,
      2 names = ["linear", "exp", "brownian", "per_exp", "poly"]
      3
      4 fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(14, 4))
      5 for i, k, name in zip(range(5), kernels, names):

```

NameError: name 'GPy' is not defined

Oglejmo si nekoliko bolj zanimiv signal. Spodnji podatki prikazujejo koncentracijo ogljikovega dioksida (CO₂) v ozračju od leta 1960.

In [4]: `co2 = np.genfromtxt("podatki/co2.csv", delimiter=",", skip_header=1)`

```

n = len(co2)

t = co2[:, 0].reshape((n, 1))
x = co2[:, 2].reshape((n, 1))
x = x - x.mean()
co2

```

NameError

Traceback (most recent call last)

```
<ipython-input-4-0a9320767844> in <module>
----> 1 co2 = np.genfromtxt("podatki/co2.csv", delimiter=",", skip_header=1)
      2
      3
      4 n = len(co2)
      5
```

```
NameError: name 'np' is not defined
```

Opazimo sezonsko periodično spreminjanje signala, v kombinaciji z naraščajočim trendom.

```
In [5]: plt.figure(figsize=(10, 4))
        plt.plot(t, x, ".")
        plt.ylabel("$CO_2$")
        plt.xlabel("t (mesec)")
        plt.show()
```

```
-----
NameError                                Traceback (most recent call last)
```

```
<ipython-input-5-f3205e50f927> in <module>
----> 1 plt.figure(figsize=(10, 4))
      2 plt.plot(t, x, ".")
      3 plt.ylabel("$CO_2$")
      4 plt.xlabel("t (mesec)")
      5 plt.show()
```

```
NameError: name 'plt' is not defined
```

Naredi sam/a. S seštevanjem kovariančnih funkcij poizkušaj modelirati podatke o koncentraciji CO_2 . Poizkusi najto kombinacijo funkcij, ki najboljše ekstrapolirajo koncentracijo CO_2 v prihodnja leta.

```
In [6]: kernel = GPY.kern.RBF(1, lengthscale=1)
        model = GPY.models.GPRegression(t, x, kernel, noise_var=0.1)
        model.optimize(messages=True)
        model.plot()
        plt.gca().set_xlim(200, 600)
```

```
-----
NameError                                Traceback (most recent call last)
```

```
<ipython-input-6-be96130dff9b> in <module>
----> 1 kernel = GPY.kern.RBF(1, lengthscale=1)
      2 model = GPY.models.GPRegression(t, x, kernel, noise_var=0.1)
      3 model.optimize(messages=True)
      4 model.plot()
      5 plt.gca().set_xlim(200, 600)
```

```
NameError: name 'GPY' is not defined
```

Odgovori

1.1 Knjižnica numpy

Odgovor 1-1-1

```
In [1]: import numpy as np
```

```
X = np.array([
    [[1, 2, 3, 4], [2, 3, 4, 5]],
    [[3, 4, 5, 6], [4, 5, 6, 7]],
    [[5, 6, 7, 8], [6, 7, 8, 9]]
])
X
```

```
Out[1]: array([[1, 2, 3, 4],
               [2, 3, 4, 5]],

              [[3, 4, 5, 6],
               [4, 5, 6, 7]],

              [[5, 6, 7, 8],
               [6, 7, 8, 9]]])
```

```
In [2]: X.shape
```

```
Out[2]: (3, 2, 4)
```

```
In [3]: X.size
```

```
Out[3]: 24
```

Odgovor 1-1-2

```
In [4]: A = np.array([[n+m*10 for n in range(5)] for m in range(5)])
A
```

```
Out[4]: array([[ 0,  1,  2,  3,  4],
               [10, 11, 12, 13, 14],
               [20, 21, 22, 23, 24],
               [30, 31, 32, 33, 34],
               [40, 41, 42, 43, 44]])
```

```
In [5]: A[A[:, 0]>10, 0:2]
```

```
Out[5]: array([[20, 21],  
              [30, 31],  
              [40, 41]])
```

```
In [6]: A[:, A[0, :]>2]
```

```
Out[6]: array([[ 3,  4],  
              [13, 14],  
              [23, 24],  
              [33, 34],  
              [43, 44]])
```

1.2 Primer: statistika temperatur na severu

Naložimo podatke o dnevni temperaturah v Stockholmu.

```
In [1]: import numpy as np
```

```
data = np.loadtxt('podatki/stockholm.csv', delimiter=",", skiprows=1)
```

Odgovor 1-2-1

```
In [2]: data[(data[:, 0] == 1817) * (data[:, 1] == 12) * (data[:, 2] == 5), :]
```

```
Out[2]: array([[ 1817. ,    12. ,     5. ,   -5.8]])
```

Odgovor 1-2-2

```
In [3]: np.mean(data[(data[:, 1] == 1), 3])
```

```
Out[3]: -3.0447656725502132
```

Odgovor 1-2-3

```
In [4]: odkloni = [(np.std(data[(data[:, 1] == mesec), 3]), mesec) for mesec in range(1, 13)]
odkloni
```

```
Out[4]: [(4.9892658658329561, 1),
(5.0903907687662713, 2),
(4.2923064618199263, 3),
(3.76783651629394, 4),
(4.029747854809286, 5),
(3.5320797349808082, 6),
(2.995916472129954, 7),
(2.8473127640895139, 8),
(3.0389674027350599, 9),
(3.4875394481813999, 10),
(3.8200293557907226, 11),
(4.5026210415550008, 12)]
```

```
In [5]: max(odkloni)
```

```
Out[5]: (5.0903907687662713, 2)
```

Največji odklon znaša 5.1 C, pojavi se v februarju.

Odgovor 1-2-4

```
In [6]: povprečja = []
for leto in range(1800, 2012):
    for mesec in range(1, 13):
        t = np.mean(data[(data[:, 1] == mesec) * (data[:, 0] == leto), 3])
        povprečja.append((t, (leto, mesec)))
max(povprečja)
```

```
Out[6]: (21.532258064516132, (1994, 7))
```

Odgovor 1-2-5

```
In [7]: # Izračunajmo povprečno temperaturo za vsako leto posebej
        letna_povprečja = dict()

        for leto in range(1800, 2012):
            # Uporabimo pogojno naslavljanje polja
            letna_povprečja[leto] = data[data[:, 0] == leto, 3].mean()

In [8]: # Izpiši vsako leto, ki ima večjo povprečno temperaturo od prejšnjega
        leto_t = sorted(letna_povprečja.items())
        večji_od_lani = [leto_t[i][0] for i in range(1, len(leto_t)) if leto_t[i-1][1] < leto_t[i][1]]
        večji_od_lani[-10:] # izpišimo le nekaj letnic

Out[8]: [1992, 1994, 1997, 1999, 2000, 2002, 2005, 2006, 2008, 2011]

In [9]: # Poišči 10 najtoplejših let
        t_leto = sorted(((t, leto) for leto, t in leto_t))
        t_leto[-10:]

Out[9]: [(8.2189041095890421, 1934),
          (8.2657534246575342, 1999),
          (8.3997260273972589, 1990),
          (8.4134246575342466, 1822),
          (8.4797260273972608, 1975),
          (8.4808219178082194, 1989),
          (8.4882191780821916, 2006),
          (8.4978142076502738, 2000),
          (8.5330601092896181, 2008),
          (8.5394520547945199, 2011)]
```

Odgovor 1-2-6

```
In [10]: %matplotlib inline
          %config InlineBackend.figure_formats = ['jpg']
          import matplotlib
          matplotlib.figure.Figure.__repr__ = lambda self: (
              f"<{self.__class__.__name__} size {self.bbox.size[0]:g}"
              f"x{self.bbox.size[1]:g} with {len(self.axes)} Axes>")

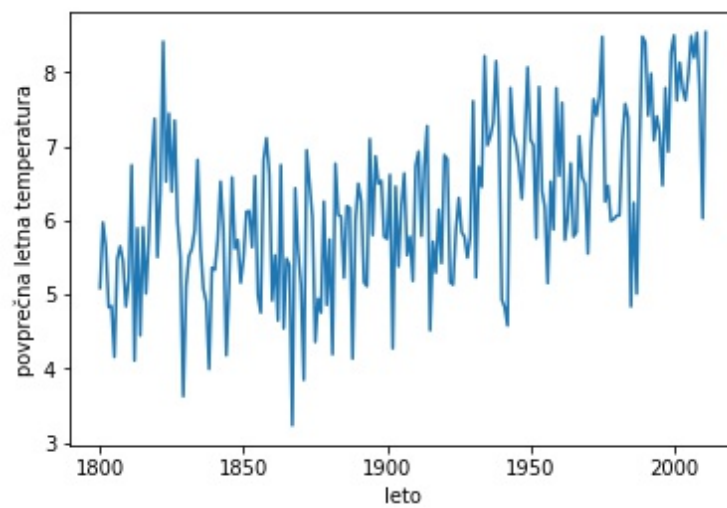
          import matplotlib.pyplot as plt

In [11]: # Pomagaj si s letna_povprečja.
          # Os x: leto
          # Os y: povprečna letna temperatura

          plt.figure()

          # Narišimo izvirne podatke
          leta, temperature = zip(*sorted(letna_povprečja.items()))
          plt.plot(lleta, temperature)

          # Vedno označimo osi.
          plt.xlabel("leto")
          plt.ylabel("povprečna letna temperatura")
          plt.show()
```

2.1 Knjižnica matplotlib

In [1]: %matplotlib inline

```
import matplotlib
%config InlineBackend.figure_format = 'jpg'
matplotlib.figure.Figure.__repr__ = lambda self: (
    f"<{self.__class__.__name__} size {self.bbox.size[0]:g}"
    f"x{self.bbox.size[1]:g} with {len(self.axes)} Axes>")
import matplotlib.pyplot as plt
plt.style.use('PR.mplstyle')

import numpy as np
np.random.seed(42)
```

```
-----

FileNotFoundError                                Traceback (most recent call last)

~/anaconda3/lib/python3.6/site-packages/matplotlib/style/core.py in use(style)
    112         try:
--> 113             rc = rc_params_from_file(style, use_default_template=False)
    114             _apply_style(rc)

~/anaconda3/lib/python3.6/site-packages/matplotlib/__init__.py in rc_params_from_file(fname, fail_on_error)
   1028     """
-> 1029     config_from_file = _rc_params_in_file(fname, fail_on_error)
   1030

~/anaconda3/lib/python3.6/site-packages/matplotlib/__init__.py in _rc_params_in_file(fname, fail_on_error)
    944     rc_temp = {}
--> 945     with _open_file_or_url(fname) as fd:
    946         try:

~/anaconda3/lib/python3.6/contextlib.py in __enter__(self)
    80         try:
---> 81             return next(self.gen)
    82         except StopIteration:

~/anaconda3/lib/python3.6/site-packages/matplotlib/__init__.py in _open_file_or_url(fname)
    929         encoding = "utf-8"
--> 930         with io.open(fname, encoding=encoding) as f:
    931             yield f
```

FileNotFoundError: [Errno 2] No such file or directory: 'PR.mplstyle'

During handling of the above exception, another exception occurred:

```

OSError                                Traceback (most recent call last)

<ipython-input-1-4709b06cc9e5> in <module>
      6      f"x{self.bbox.size[1]:g} with {len(self.axes)} Axes>")
      7  import matplotlib.pyplot as plt
----> 8  plt.style.use('PR.mplstyle')
      9
     10  import numpy as np

~/anaconda3/lib/python3.6/site-packages/matplotlib/style/core.py in use(style)
    117             "not a valid URL or path. See `style.available` for "
    118             "list of available styles.")
--> 119         raise IOError(msg % style)
    120
    121

```

OSError: 'PR.mplstyle' not found in the style library and input is not a valid URL or path. See

Odgovor 2-1-1

```

In [2]: plt.figure()
        x = np.linspace(1, 5, 10)

        for exp in range(-3, 4):
            y = x ** exp
            plt.plot(x, y)

        plt.xlabel('x')
        plt.ylabel('y');

```

```

-----

NameError                                Traceback (most recent call last)

<ipython-input-2-ad75e5174fb6> in <module>
      1  plt.figure()
----> 2  x = np.linspace(1, 5, 10)
      3
      4  for exp in range(-3, 4):
      5      y = x ** exp

```

NameError: name 'np' is not defined

<Figure size 432x288 with 0 Axes>

Odgovor 2-1-2

```
In [3]: N = 10000
        data = np.random.randn(N)

        plt.hist(data, bins=25);
```

```
-----

NameError                                Traceback (most recent call last)

<ipython-input-3-203c78b08417> in <module>
      1 N = 10000
----> 2 data = np.random.randn(N)
      3
      4 plt.hist(data, bins=25);

NameError: name 'np' is not defined
```

Odgovor 2-1-3

```
In [4]: data = np.random.randn(1000) / 2 + 5
        plt.hist(data, bins=10);
```

```
-----

NameError                                Traceback (most recent call last)

<ipython-input-4-4c0984270b17> in <module>
----> 1 data = np.random.randn(1000) / 2 + 5
      2 plt.hist(data, bins=10);

NameError: name 'np' is not defined
```

2.2 Primer: zimske olimpijske igre, Soči 2014

```
In [1]: %matplotlib inline
%config InlineBackend.figure_formats = ['jpg']
import matplotlib
matplotlib.figure.Figure.__repr__ = lambda self: (
    f"<{self.__class__.__name__} size {self.bbox.size[0]:g}"
    f"x{self.bbox.size[1]:g} with {len(self.axes)} Axes>")

import matplotlib.pyplot as plt

import Orange
from Orange.data.filter import SameValue
from Orange.data import Table
data = Table('podatki/athletes.tab')

# barve medalj
gold_color = "#FFDF00"
silv_color = "#C0C0C0"
bron_color = "#CD7F32"

sports = data.domain["sport"].values
```

Odgovor 2-2-1

```
In [2]:
```

Odgovor 2-2-2

```
In [2]:
```

Odgovor 2-2-3

```
In [2]: import numpy as np
def pearson(x, y):
    return np.mean(((x - np.mean(x))*(y-np.mean(y)))/(np.std(x)*np.std(y)))

In [3]: x = data.X[:, 2]      # višina
        y = data.X[:, 3]      # teža
        pearson(x, y)

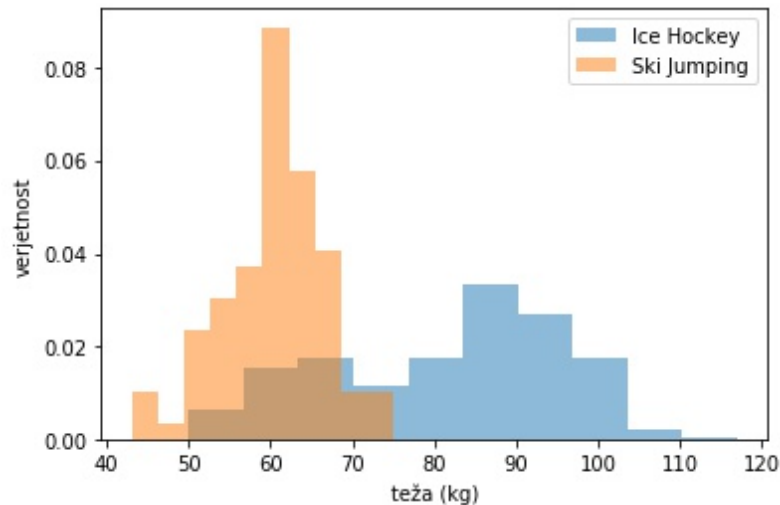
Out[3]: 0.83074658486272401
```

Odgovor 2-2-4

```
In [4]: filt = SameValue(data.domain['sport'], 'Ice Hockey')
        data_subset = filt(data)
        weights = data_subset.X[:, 3]
        plt.hist(weights, normed=True, bins=10, label="Ice Hockey", alpha=0.5)

        filt = SameValue(data.domain['sport'], 'Ski Jumping')
        data_subset = filt(data)
        weights = data_subset.X[:, 3]
        plt.hist(weights, normed=True, bins=10, label="Ski Jumping", alpha=0.5)
```

```
plt.xlabel('teža (kg)')
plt.ylabel('verjetnost')
plt.legend();
```



Odgovor 2-2-5

```
In [5]: countries = data.domain['country'].values
gender_by_country = dict()

for country in countries:
    # Filter by countries
    filt = SameValue(data.domain['country'], country)
    data_subset = filt(data)

    # Filter males
    filt = SameValue(data.domain['gender'], 'Male')
    data_subset_male = filt(data_subset)

    # Filter females
    filt = SameValue(data.domain['gender'], 'Female')
    data_subset_female = filt(data_subset)

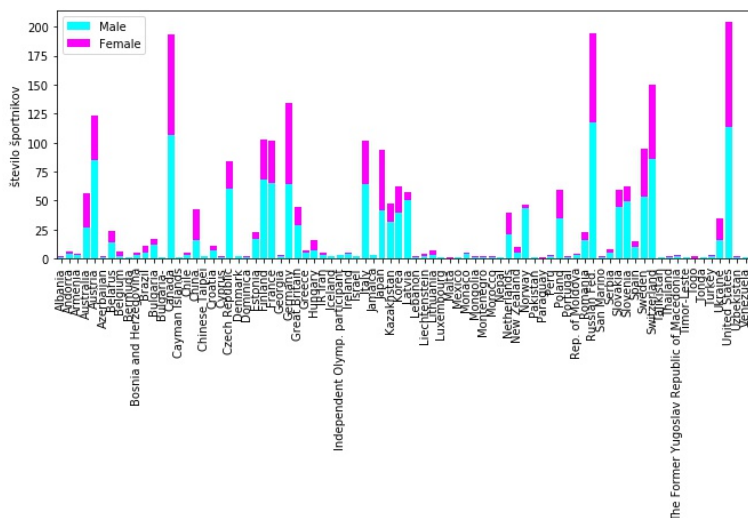
    # Store gender counts
    gender_by_country[country] = {
        'Male': len(data_subset_male),
        'Female': len(data_subset_female),
    }

In [6]: m = [gender_by_country[country]['Male'] for country in countries]
f = [gender_by_country[country]['Female'] for country in countries]
x = range(len(countries))

plt.figure(figsize=(11, 4))
plt.bar(x, m, color='cyan', align='center', label="Male")
plt.bar(x, f, bottom=m, color='magenta', align='center', label="Female")
plt.xlim(-0.5, len(countries)-0.5)
```

```
plt.xticks(x)
plt.gca().set_xticklabels(countries, rotation=90)
plt.ylabel('število športnikov')

plt.legend();
```



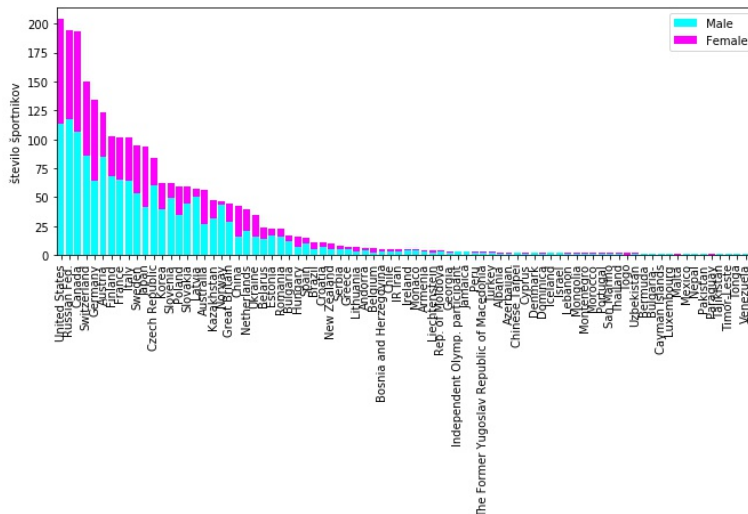
Odgovor 2-2-6

```
In [7]: countries = filter(lambda c: sum([gender_by_country[c][m] for m in gender_by_country[c].keys()]),
countries = sorted(countries, key=lambda c: -sum([gender_by_country[c][m] for m in gender_by_country[c].keys()]))
```

```
m = [gender_by_country[country]['Male'] for country in countries]
f = [gender_by_country[country]['Female'] for country in countries]
x = range(len(countries))

plt.figure(figsize=(11, 4))
plt.bar(x, m, color='cyan', align='center', label="Male")
plt.bar(x, f, bottom=m, color='magenta', align='center', label="Female")
plt.xlim(-0.5, len(countries)-0.5)
plt.xticks(x)
plt.gca().set_xticklabels(countries, rotation=90)
plt.ylabel('število športnikov')

plt.legend();
```



Odgovor 2-2-7 Najprej izračunamo distribucijo vrednosti.

```
In [8]: # poišči indekse
gold_inx = data.domain.index("gold_medals")
silv_inx = data.domain.index("silver_medals")
bron_inx = data.domain.index("bronze_medals")

# pripravi podatke ; shrani št. medalj za vsako državo in šport
countries = data.domain["country"].values

# preštej medalje
medals_by_country = dict()
for country in countries:
    medals_by_country[country] = dict()
    filt = SameValue(data.domain["country"], country)
    data_subset = filt(data)
    medals_by_country[country] = {
        "gold": data_subset.X[:, gold_inx].sum(),
        "silver": data_subset.X[:, silv_inx].sum(),
        "bronze": data_subset.X[:, bron_inx].sum(),
    }
```

Nato distribucijo narišemo.

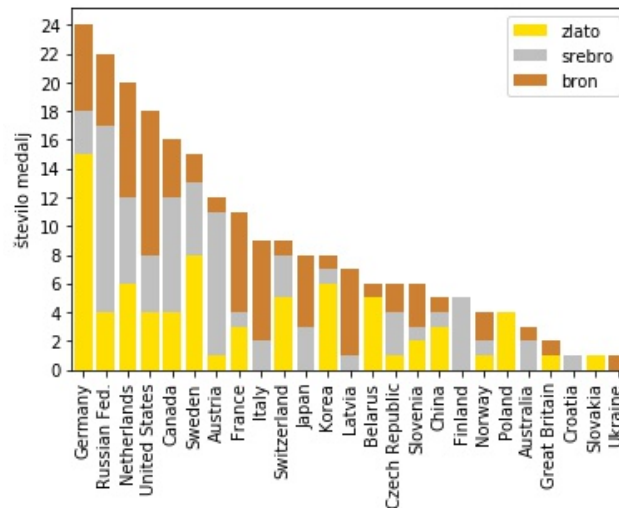
```
In [9]: import numpy

countries = filter(lambda c: sum([medals_by_country[c][m] for m in medals_by_country[c].keys()]),
countries = sorted(countries, key=lambda c: -sum([medals_by_country[c][m] for m in medals_by_country[c].keys()]))
gx = numpy.array([medals_by_country[c]["gold"] for c in countries])
sx = numpy.array([medals_by_country[c]["silver"] for c in countries])
bx = numpy.array([medals_by_country[c]["bronze"] for c in countries])
x = range(len(countries))

plt.bar(x, gx, align="center", color=gold_color, label="zlato")
plt.bar(x, sx, align="center", bottom=gx, color=silv_color, label="srebro")
plt.bar(x, bx, align="center", bottom=gx+sx, color=bron_color, label="bron")
```



```
plt.xlim(-0.5, len(x)-0.5)
plt.legend()
plt.xticks(x)
plt.yticks(range(0, 25, 2))
plt.gca().set_xticklabels(countries, rotation=90)
plt.ylabel("število medalj")
plt.savefig('slike/odgovori/2-2-7.png', bbox_inches='tight')
```



Odgovor 2-2-8 Priprava podatkov - teža in višina glede na sport (sport se nahaja v 8 stolpcu)

```
In [10]: sports = data.domain["sport"].values
weights_by_sport = dict()
heights_by_sport = dict()
ages_by_sport = dict()

for sport in sports:
    filt = SameValue(data.domain["sport"], sport)
    data_subset = filt(data)

    w = data_subset[:, data.domain.index("weight")].X.ravel()
    h = data_subset[:, data.domain.index("height")].X.ravel()
    a = data_subset[:, data.domain.index("age")].X.ravel()

    weights_by_sport[sport] = w
    heights_by_sport[sport] = h
    ages_by_sport[sport] = a

In [11]: plt.figure(figsize=(5, 6))

for si, sport in enumerate(sports):
    xs = heights_by_sport[sport] # x os
    ys = [si for x in xs]        # y os je v visini sporta
    zs = weights_by_sport[sport] # velikost točke je prenosorazmerna s tezo

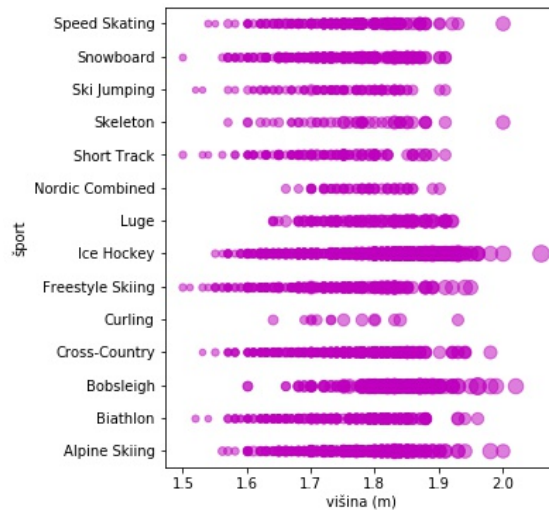
    for x, y, z in zip(xs, ys, zs): # rišemo točko po točko
        plt.plot(x, y, "m.", alpha=0.5, markersize=z/5)
```

```

plt.yticks(range(len(sports)))
plt.ylim(-0.5, len(sports)-0.5)
plt.gca().set_yticklabels(sports)

plt.xlabel("višina (m)")
plt.ylabel("šport");
plt.savefig('slike/odgovori/2-2-8.png', bbox_inches='tight')

```



Odgovor 2-2-9

```
In [12]: plt.figure(figsize=(5, 6))
```

```

sport_order = []
for si, sport in enumerate(sports):
    xs = heights_by_sport[sport]      # x os
    sport_order.append((numpy.average(xs), si))
sport_order.sort()

sport_label = []
for nsi, (avg_xs, si) in enumerate(sport_order):
    sport = sports[si]
    sport_label.append(sport)

xs = heights_by_sport[sport]      # x os
ys = [nsi for x in xs]            # y os je v visini sporta
zs = weights_by_sport[sport]      # velikost točke je premosorazmerna s tezo

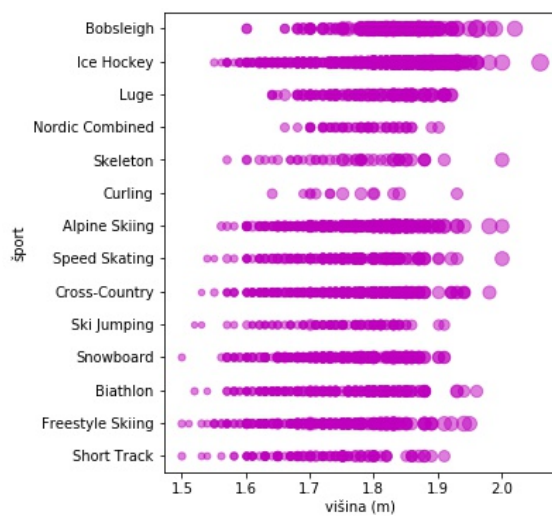
for x, y, z in zip(xs, ys, zs): # rišemo točko po točko
    plt.plot(x, y, "m.", alpha=0.5, markersize=z/5)

plt.plot(avg_xs, nsi, 'k', markersize=1)

plt.yticks(range(len(sports)))
plt.ylim(-0.5, len(sports)-0.5)
plt.gca().set_yticklabels(sport_label)

```

```
plt.xlabel("višina (m)")
plt.ylabel("šport");
```



3.1 Pogoste verjetnostne porazdelitve

Odgovor 3-1-1

In [1]:

Odgovor 3-1-2

In [1]:

Odgovor 3-1-3

In [1]:

Odgovor 3-1-4

In [1]:

3.2 Primer: iskanje neslanih šal

Odgovor 3-2-1

In [1]:

Odgovor 3-2-2

In [1]:

Odgovor 3-2-3

In [1]:

Odgovor 3-2-4

In [1]:

4.1 Odkrivanje skupin

Odgovor 4-1-1

```
In [1]: import numpy as np
        np.random.seed(42)

        class KMeans:

            def __init__(self, k=10, max_iter=100):
                """
                Initialize KMeans clustering model.

                :param k
                    Number of clusters.
                :param max_iter
                    Maximum number of iterations.
                """
                self.k = k
                self.max_iter = max_iter

            def fit(self, X):
                """
                Fit the Kmeans model to data.

                :param X
                    Numpy array of shape (n, p)
                    n: number of data examples
                    p: number of features (attributes)

                :return
                    labels: array of shape (n, ), cluster labels (0..k-1)
                    centers: array of shape (p, )
                """

                n, p = X.shape
                labels = np.random.choice(range(self.k), size=n, replace=True)
                centers = np.random.rand(self.k, p)

                ### Your code here ###
                centers = np.min(X, axis=0) + centers * (np.max(X, axis=0) - np.min(X, axis=0))

                i = 0
                while i < self.max_iter:

                    # Find nearest cluster
                    for j, x in enumerate(X):
                        ki = np.argmin(np.sum((centers - x) ** 2, axis=1))
                        labels[j] = ki

                    # Move centroid
                    for ki in range(self.k):
                        centers[ki] = X[labels == ki].mean(axis=0)

                    i += 1
```

```
### Your code here ###  
return labels, centers
```

Odgovor 4-1-2

In [2]:

Odgovor 4-1-3

In [2]:

Odgovor 4-1-4

In [2]:

Odgovor 4-1-5

In [2]:

Odgovor 4-1-6

In [2]:

4.2 Hierarhično gručenje

Odgovor 4-2-1

In [1]:

Odgovor 4-2-2

In [1]:

Odgovor 4-2-3

In [1]:

4.3 Primer: genomske podatke v obliki nizov znakov

```
In [1]: import json
        sequences = json.load(open("podatki/seqs.json"))
```

Odgovor 4-3-1

```
In [2]: from itertools import product
        import numpy as np
        import scipy.cluster.hierarchy as sch
        %matplotlib inline
        %config InlineBackend.figure_formats = ['jpg']
        import matplotlib
        matplotlib.figure.Figure.__repr__ = lambda self: (
            f"<{self.__class__.__name__} size {self.bbox.size[0]:g}"
            f"x{self.bbox.size[1]:g} with {len(self.axes)} Axes>")

        import matplotlib.pyplot as plt

        def seq_to_kmer_count(seq, k=4):
            ktuples = list(zip(*[seq[i:] for i in range(k)]))    # razbijemo niz na k-terke
            kmers = list(product(*(k*["A", "C", "T", "G"])))    # vse mozne k-terke

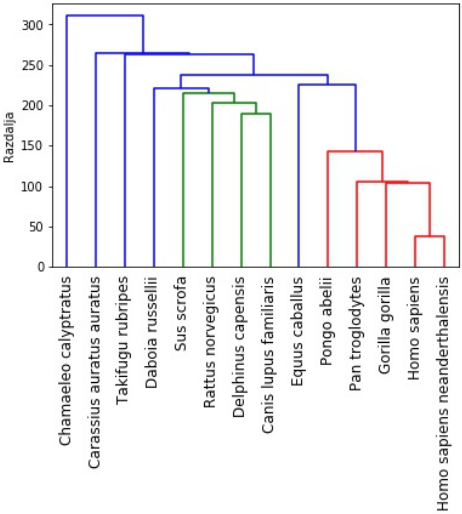
            x = np.zeros((len(kmers), ))

            for ki, kmer in enumerate(kmers):
                x[ki] = ktuples.count(kmer)
            return x

In [3]: # ...k = 4
        k = 4
        keys = sequences.keys()
        X = np.zeros((len(keys), 4**k))
        for ki, ky in enumerate(keys):
            seq = sequences[ky]
            X[ki] = seq_to_kmer_count(seq, k=k)

        print(X)
        print(X.shape)
        H = sch.linkage(X)
        D = sch.dendrogram(H, labels=list(sequences.keys()), leaf_rotation=90)
        plt.ylabel("Razdalja")
        plt.show()

[[ 182.  157.  110. ...,  22.  18.  15.]
 [ 187.  149.  120. ...,  14.  13.  12.]
 [ 174.  159.  124. ...,  18.  13.  14.]
 ...,
 [ 158.  125.  120. ...,  22.  31.  27.]
 [ 238.  160.  158. ...,  12.  18.  14.]
 [ 184.  156.  110. ...,  25.  18.  19.]]
(14, 256)
```



5.1 Linearna regresija

Odgovor 5-1-1

```
In [1]: explained_var = 100.0 * ( np.var(y) - np.var(hx-y) ) / np.var(y)
        print("Explained variance: %.2f " % explained_var + "%" )
```

```
-----

NameError                                Traceback (most recent call last)

<ipython-input-1-d7f1e86fae41> in <module>
----> 1 explained_var = 100.0 * ( np.var(y) - np.var(hx-y) ) / np.var(y)
      2 print("Explained variance: %.2f " % explained_var + "%" )

NameError: name 'np' is not defined
```

Odgovor 5-1-2

```
In [2]: explained_var = 100.0 * ( np.var(y) - np.var(hx-y) ) / np.var(y)
        print("Explained variance: %.2f " % explained_var + "%" )
```

```
-----

NameError                                Traceback (most recent call last)

<ipython-input-2-d7f1e86fae41> in <module>
----> 1 explained_var = 100.0 * ( np.var(y) - np.var(hx-y) ) / np.var(y)
      2 print("Explained variance: %.2f " % explained_var + "%" )

NameError: name 'np' is not defined
```

Odgovor 5-1-3

```
In [3]: D = 20 # stopnja polinoma

        # Ustvarimo ustrezen prostor
        X = np.zeros((len(x), D))
        for d in range(0, D):
            X[:, d] = x.ravel()**d

        model = Ridge(alpha=0.1)
        model.fit(X, y)

        hx = model.predict(X)

        plot_fit_residual(X[:, 1], y, hx)
        plot_coefficients(model.coef_)
        model.coef_
```

```

-----

NameError                                Traceback (most recent call last)

<ipython-input-3-8e0a57f74d20> in <module>
      2
      3 # Ustvarimo ustrezen prostor
----> 4 X = np.zeros((len(x), D))
      5 for d in range(0, D):
      6     X[:, d] = x.ravel()*d

NameError: name 'np' is not defined

```

Odgovor 5-1-4

```

In [4]: explained_var = 100.0 * ( np.var(y) - np.var(hx-y) ) / np.var(y)
        print("Explained variance: %.2f " % explained_var + "%" )

```

```

-----

NameError                                Traceback (most recent call last)

<ipython-input-4-d7f1e86fae41> in <module>
----> 1 explained_var = 100.0 * ( np.var(y) - np.var(hx-y) ) / np.var(y)
      2 print("Explained variance: %.2f " % explained_var + "%" )

NameError: name 'np' is not defined

```

Odgovor 5-1-5

```

In [5]: model = Lasso(alpha=0.1)
        model.fit(X, y)

        hx = model.predict(X_test)

        print("MSE: %.2f " %mean_squared_error(hx, y_test))
        explained_var = 100.0 * ( np.var(y_test) - np.var(hx-y_test) ) / np.var(y_test)
        print("Explained variance: %.2f" % explained_var + "%" )

```

```

-----

NameError                                Traceback (most recent call last)

<ipython-input-5-748a51cac842> in <module>
----> 1 model = Lasso(alpha=0.1)
      2 model.fit(X, y)
      3
      4 hx = model.predict(X_test)

```

5

```
NameError: name 'Lasso' is not defined
```

Odgovor 5-1-6

In [6]:

5.2 Naivni Bayesov klasifikator

```
In [1]: class NaiveBayes:
        """
        Naive Bayes classifier.

        :attribute self.probabilities
            Dictionary that stores
            - prior class probabilities  $P(Y)$ 
            - attribute probabilities conditional on class  $P(X|Y)$ 

        :attribute self.class_values
            All possible values of the class.

        :attribute self.variables
            Variables in the data.

        :attribute self.trained
            Set to True after fit is called.
        """

    def __init__(self):
        self.trained = False
        self.probabilities = dict()

    def fit(self, data):
        """
        Fit a NaiveBayes classifier.

        :param data
            Orange data Table.
        """
        class_variable = data.domain.class_var # class variable (Y)
        self.class_values = class_variable.values # possible class values
        self.variables = data.domain.attributes # all other variables (X)

        n = len(data) # number of all data points

        # Compute P(Y)
        for y in self.class_values:

            # A not too smart guess (INCORRECT)
            self.probabilities[y] = 1/len(self.class_values)

            # <your code here>
            # Compute class probabilities and correctly fill
            # probabilities[y] = ...
            # Select all examples (rows) with class = y
            filt = SameValue(data.domain.class_var, y)
            data_subset = filt(data)
            m = len(data_subset)

            self.probabilities[y] = m/n
```

```

        # </your code here>

    # Compute  $P(X|Y)$ 
    for y in self.class_values:

        # Select all examples (rows) with class = y
        filty = SameValue(class_variable, y)

        for variable in self.variables:
            for x in variable.values:

                # A not too smart guess (INCORRECT)
                p = 1 / (len(self.variables) * len(variable.values) * len(self.class_values))

                #  $P(\text{variable}=x|Y=y)$ 
                self.probabilities[variable, x, y] = p

                # <your code here>
                # Compute correct conditional class probability
                # probabilities[x, value, c] = ...
                #
                # Select all examples with class == y AND
                # variable x == value
                # Hint: use SameValue filter twice
                filtx = SameValue(variable, x)
                data_subset = filtx(filty(data))
                m = len(data_subset)

                data_subset = filty(data)
                p = len(data_subset)

                self.probabilities[variable, x, y] = m/p
                # </your code here>

    self.trained = True

def predict_instance(self, row):
    """
    Predict a class value for one row.

    :param row
        Orange data Instance.
    :return
        Class prediction.
    """
    curr_p = float("-inf") # Current highest "probability" (unnormalized)
    curr_c = None          # Current most probable class

    for y in self.class_values:
        p = np.log(self.probabilities[y])
        for x in self.variables:
            p = p + np.log(self.probabilities[x, row[x].value, y])

        if p > curr_p:

```

```
        curr_p = p
        curr_c = y

    return curr_c, curr_p

def predict(self, data):
    """
    Predict class labels for all rows in data.

    :param data
        Orange data Table.
    :return y
        NumPy vector with predicted classes.
    """

    n = len(data)
    predictions = list()
    confidences = np.zeros((n, ))

    for i, row in enumerate(data):
        pred, cf = self.predict_instance(row)
        predictions.append(pred)
        confidences[i] = cf

    return predictions, confidences
```


6 Nenegativna matrična faktorizacija in priporočilni sistemi

```
In [1]: import numpy as np
import itertools
import time

np.random.seed(42)

class NMF:

    """
    Fit a matrix factorization model for a matrix X with missing values.
    such that
         $X = W H.T + E$ 
    where
        X is of shape (m, n) - data matrix
        W is of shape (m, rank) - approximated row space
        H is of shape (n, rank) - approximated column space
        E is of shape (m, n) - residual (error) matrix
    """

    def __init__(self, rank=10, max_iter=100, eta=0.01):
        """
        :param rank: Rank of the matrices of the model.
        :param max_iter: Maximum number of SGD iterations.
        :param eta: SGD learning rate.
        """
        self.rank = rank
        self.max_iter = max_iter
        self.eta = eta

    def fit(self, X, verbose=False):
        """
        Fit model parameters W, H.
        :param X:
            Non-negative data matrix of shape (m, n)
            Unknown values are assumed to take the value of zero (0).
        """
        m, n = X.shape

        W = np.random.rand(m, self.rank)
        H = np.random.rand(n, self.rank)

        # Indices to model variables
        w_vars = list(itertools.product(range(m), range(self.rank)))
        h_vars = list(itertools.product(range(n), range(self.rank)))

        # Indices to nonzero rows/columns
        nzcols = dict([(j, X[:, j].nonzero()[0]) for j in range(n)])
        nzrows = dict([(i, X[i, :].nonzero()[0]) for i in range(m)])

        # Errors
        self.error = np.zeros((self.max_iter,))
```

```

for t in range(self.max_iter):
    t1 = time.time()
    np.random.shuffle(w_vars)
    np.random.shuffle(h_vars)

    for i, k in w_vars:
        wgrad = sum([(X[i, j] - W[i, :].dot(H[j, :]))*W[i, k] for j in nzrows[i]])
        W[i, k] = max(0, W[i, k] + self.eta * wgrad)

    for j, k in h_vars:
        hgrad = sum([(X[i, j] - W[i, :].dot(H[j, :]))*H[j, k] for i in nzcols[j]])
        H[j, k] = max(0, H[j, k] + self.eta * hgrad)

    self.error[t] = sum([sum([(X[i, j] - W[i, :].dot(H[j, :]))**2 for j in nzrows[i]])
                        for i in range(X.shape[0])])

    if verbose: print(t, self.error[t])

self.W = W
self.H = H

def predict(self, i, j):
    """
    Predict score for row i and column j
    :param i: Row index.
    :param j: Column index.
    """
    return self.W[i, :].dot(self.H[:, j])

def predict_all(self):
    """
    Return approximated matrix for all
    columns and rows.
    """
    return self.W.dot(self.H.T)

```

7.1 Knjižica networkx

7.2 Primer: analiza in vizualizacija omrežja elektronskih sporočil

8.1 Skriti Markovi modeli

```
In [1]: import random
        random.seed(42)

        def weighted_choice(weighted_items):
            """Random choice given the list of elements and their weights"""
            rnd = random.random() * sum(weighted_items.values())
            for i, w in weighted_items.items():
                rnd -= w
                if rnd < 0:
                    return i

        def generate_hmm_sequence(h, T, E, n):
            """
            HMM sequence given start state,
            transition, emission matrix and sequence length

            return zip(hidden_path, visible_sequence)
            """

            s = weighted_choice(E[h])
            yield h, s
            for _ in range(n-1):
                h = weighted_choice(T[h])
                yield h, weighted_choice(E[h])

        from collections import Counter

        def normalize(dic):
            s = sum(dic.values())
            return {k: dic[k]/s for k in dic}

        def learn_hmm(h, x):
            t = {}
            for (i, j), cn in Counter(zip(h, h[1:])).items():
                t.setdefault(i, {}).setdefault(j, cn)
            T = {}
            for i, d in t.items():
                T[i] = normalize(d)

            c = Counter(zip(h, x))
            E = {}
            for h in T.keys():
                E[h] = normalize({xi: c[(pi, xi)] for pi, xi in c if pi == h})
            return T, E
```

8.2 Modeliranje časovnih vrst

8.3 Nelinearna regresija ali napovedovanje trendov

Naloge

Priprava podatkov, osnovne statistike in vizualizacija

Podatkovno rudarjenje, naloga, <VPIŠI DATUM ODDAJE>
<VPIŠI Ime in priimek>

Neizogiben del vsakega projekta na področju podatkovnega rudarjenja je iskanje, urejanje in priprava podatkov. V tej nalogi boste spoznali primer podatkovne zbirke in uporabili postopke za pretvorbo podatkov v ustrezno obliko ter pregled in prikaz osnovnih statistik.

Podatki

V nalogi boste pregledali in pripravili podatke gledanosti Hollywoodskih filmov zbirke [MovieLens](#) v obdobju **1995-2016**.

Iste podatke boste uporabili v vseh nalogah, zato jih dodobra spoznajte. Gre za podatkovno zbirko za vrednotenje priporočilnih sistemov, ki vsebuje gledalce ter njihove ocene za posamezni film na lestvici 1 do 5. Poleg osnovne matrike uporabnikov in ocen vsebuje še dodatne podatke o filmih (npr. žanr, datum, oznake, igralci).

Podatki so v mapi `./podatki/ml-latest-small`. Podatkovna zbirka vsebuje naslednje datoteke:

- `ratings.csv`: podatki o uporabnikih in ocenah,
- `movies.csv`: podatki o žanrih filmov,
- `cast.csv`: podatki o igralcih,
- `tags.csv`: podatki o oznakah (ang. *tags*),
- `links.csv`: povezave na sorodne podatkovne zbirke.

Pred pričetkom reševanja naloge si dobro oglejte podatke in datoteko **README.txt**. Podrobnosti o zbirki lahko preberete na [spletni strani](#).

Pripravite metode za nalaganje podatkov v ustrezne podatkovne strukture. Te vam bodo prišle prav tudi pri nadaljnjih nalogah. Bodite pozorni na velikost podatkov.

Zapišite kodo za branje datotek in pripravo ustreznih matrik (in drugih struktur) podatkov, ki jih boste uporabi pri odgovarjanju na spodnja vprašanja.

Kodo lahko razdelite v več celic.

In []:

Vprašanja

Glavni namen podatkovnega rudarjenja je *odkrivanje znanj iz podatkov*, torej odgovarjanje na vprašanja z uporabo računskih postopkov.

Z uporabo principov, ki ste jih spoznali na vajah in predavanjih, odgovorite na spodnja vprašanja. Pri vsakem vprašanju dobro premislite, na kakšen način boste najbolje podali, prikazali oz. utemeljili odgovor. Bistven del so odgovori na vprašanja in ne implementacija vaše rešitve.

1. vprašanje (15Kateri filmi so v povprečju najboljše ocenjeni? Pripravite seznam

filmov ter njihovih povprečnih ocen in izpišite 10 filmov z vrha seznama. Opazite pri takem ocenjevanju kakšno težavo? Kako bi jo lahko rešili? Kakšni so rezultati tedaj?

Kodo lahko razdelite v več celic.

In []:

Odgovor: **zapišite utemeljitev odgovora**

2. vprašanje (15) Posamezni film pripada enemu ali več žanrom.

Koliko je vseh žanrov? Prikaži porazdelitev žanrov z uporabo ustrezne vizualizacije.

Kodo lahko razdelite v več celic.

In []:

Odgovor: **zapišite utemeljitev odgovora**

3. vprašanje (20) Število ocen (ogledov) se za posamezni film razlikuje. Ali

obstaja povezava med gledanostjo in povprečno oceno filma? Opišite postopek, ki ste ga uporabili pri odgovarjanju na vprašanje.

Kodo lahko razdelite v več celic.

In []:

Odgovor: **zapišite utemeljitev odgovora**

4. vprašanje (30) Vsaka ocena je bila vnešena na določen datum (stolpec

timestamp). Ali se popularnost posameznih filmov s časom spreminja? Problem reši tako, da za dani film ocene razporediš po času ter v vsaku časovni točki izračunaš povprečje za zadnjih 30, 50, ali 100 ocen. Nariši graf, kako se ocena spreminja in ga prikaži za dva zanimiva primera filmov.

Kodo lahko razdelite v več celic.

In []:

Odgovor: **zapišite utemeljitev odgovora**

5. vprašanje (20) Kako bi ocenili popularnost posameznih igralcev? Opišite postopek

ocenitve ter izpišite 10 najbolj popularnih igralcev.

Kodo lahko razdelite v več celic.

In []:

Odgovor: **zapišite utemeljitev odgovora**

Bonus vprašanje (5)

Kateri je tvoj najljubši film? Zakaj?

Odgovor: **zapišite utemeljitev odgovora**

Zapiski

Za nalaganje podatkov lahko uporabite modul `csv`.

```
In [1]: from csv import DictReader

        reader = DictReader(open('podatki/ml-latest-small/ratings.csv', 'rt', encoding='utf-8'))
        for row in reader:
            user = row["userId"]
            movie = row["movieId"]
            rating = row["rating"]
            timestamp = row["timestamp"]
```

Podatki v zadnji vrstici datoteke:

```
In [2]: user, movie, rating, timestamp
Out[2]: ('671', '6565', '3.5', '1074784724')
```

Pretvorba časovnega formata (*Unix time*). Kode za oblikovanje so navedene v dokumentaciji modula `datetime`.

```
In [3]: from datetime import datetime

        t = 1217897793 # Unix-time
        ts = datetime.fromtimestamp(t).strftime('%Y-%m-%d %H:%M')
        ts
Out[3]: '2008-08-05 02:56'
```

Iskanje strukture v podatkih

Podatkovno rudarjenje, naloga, <VPIŠI DATUM ODDAJE>
<VPIŠI Ime in priimek>

Z modeliranjem skušamo poiskati strukturo v podatkih. Z metodami nenadzorovanga modeliranja skušamo poiskati skupine podobnih podatkov oz. skupine primerov.

V nalogi boste uporabili modeliranje verjetnostnih porazdelitev za iskanje osamelcev ter metode za iskanje skupin podobnih primerov (gručenje).

Podatki

Opis podatkovne zbirke MovieLens ostaja enak prvi nalogi.

Vprašanja

Z uporabo principov, ki ste jih spoznali na vajah in predavanjih, odgovorite na spodnja vprašanja. Pri vsakem vprašanju dobro premislite, na kakšen način boste najboljše podali, prikazali oz. utemeljili odgovor. Bistven del so odgovori na vprašanja in ne toliko implementacija vaše rešitve.

1. Iskanje osamelcev (500 ocenah katerih filmov so si uporabniki najmanj enotni? Povedano drugače, za katere filme so pripadajoče ocene najbolj razpršene?

Formuliraj problem kot modeliranje verjetnostne porazdelitve. Premisli o naslednjih vprašanjih, naredi ustrezne poizkuse in odgovori.

In [1]: *# kodo lahko razdelite v več celic*

Odgovor: **odgovor lahko zapišete v več celic**

1.1. vprašanje:

Katera je ustrezna naključna spremenljivka (količina) v podatkih, ki odgovarja na vprašanje?

In [2]: *# kodo lahko razdelite v več celic*

Odgovor: **odgovor lahko zapišete v več celic**

1.2. vprašanje:

Nariši njeno porazdelitev, npr., s pomočjo histograma.

In [3]: *# kodo lahko razdelite v več celic*

Odgovor: **odgovor lahko zapišete v več celic**

1.3. vprašanje:

Ali porazdelitev spominja na kakšno znano porazdelitev? Ali je porazdelitev morda normalna ali katera druga?

In [4]: *# kodo lahko razdelite v več celic*

Odgovor: **odgovor lahko zapišete v več celic**

1.4. vprašanje:

Oceni parametre te porazdelitve s pomočjo postopkov, ki smo jih spoznali na vajah. Izmed porazdelitev, ki smo jih spoznali na vajah, izberi tisto, ki se podatkom najbolj prilega.

In [5]: # kodo lahko razdelite v več celic

Odgovor: **odgovor lahko zapišete v več celic**

1.5. vprašanje:

Izpiši filme z vrednostjo naključne spremenljivke, ki spada v zgornjih 5% statistično značilnih primerov.

In [6]: # kodo lahko razdelite v več celic

Odgovor: **odgovor lahko zapišete v več celic**

2. Gručenje filmov (50)

Priporočilni sistemi pogosto odkrivajo skupine predmetov (v našem primeru filme), za katere velja visoka medsebojna podobnost.

Poiščite 100 najbolj gledanih filmov. Ali med njimi obstajajo skupine? Uporabite ustrezen algoritem za gručenje. Na film lahko gledamo kot vektor, kjer je število komponent enako številu uporabnikov.

Vektorji vsebujejo tudi *neznane vrednosti*. Primer vektorjev za deset filmov prikazuje spodnja tabela.

Algoritme gručenja lahko izvajamo v izvornem prostoru (koordinatni sistem filmi-uporabniki) ali pa filme primerjamo z merami podobnosti, ki smo jih spoznali na vajah. Premisli, kateri način je primernejši glede na obliko podatkov.

x	Movie	u_0	u_1	u_2	...
\vec{x}_0	Fight Club (1999)	?	?	?	...
\vec{x}_1	Twelve Monkeys (a.k.a. 12 Monkeys) (1995)	?	?	2.5	...
\vec{x}_2	Independence Day (a.k.a. ID4) (1996)	?	?	?	...
\vec{x}_3	Dances with Wolves (1990)	4.0	?	?	...
\vec{x}_4	Fargo (1996)	?	?	?	...
\vec{x}_5	Speed (1994)	?	?	?	...
\vec{x}_6	Apollo 13 (1995)	?	2.0	?	...
\vec{x}_7	Seven (a.k.a. Se7en) (1995)	?	?	?	...
\vec{x}_8	Sixth Sense, The (1999)	3.0	?	4.0	...
\vec{x}_9	Aladdin (1992)	?	?	?	...
...

Pri tem odgovori na naslednja vprašanja.

2.1. vprašanje:

Utemelji izbiro algoritma in mere podobnosti.

In [7]: # kodo lahko razdelite v več celic

Odgovor: **odgovor lahko zapišete v več celic**

2.2. vprašanje:

Koliko skupin filmov je med izbranimi? Ali poznamo kvantitativne ocene za različne možnosti razvrščanja v skupine?

In [8]: *# kodo lahko razdelite v več celic*

Odgovor: **odgovor lahko zapišete v več celic**

2.3. vprašanje:

Prikaži rezultate z uporabo ustrezne vizualizacije.

In [9]: *# kodo lahko razdelite v več celic*

Odgovor: **odgovor lahko zapišete v več celic**

2.4. vprašanje:

Komentiraj smiselnost dobljenih rezultatov.

In [10]: *# kodo lahko razdelite v več celic*

Odgovor: **odgovor lahko zapišete v več celic**

Napovedovanje vrednosti

Podatkovno rudarjenje, naloga, <VPIŠI DATUM ODDAJE>
<VPIŠI Ime in priimek>

Spoznali bomo praktično uporabo enostavnih metod nadzorovanega modeliranja oz. napovedovanja. Skupna lastnost vseh omenjenih metod je, da s pomočjo naključnih spremenljivk (atributov) modelirajo vrednosti posebne spremenljivke, ki ji pravimo *razred* (v kontekstu uvrščanja v razrede, klasifikacije) ali *odziv* (v kontekstu regresije). Osnovne razlike med kontekstoma smo spoznali na predavanjih in vajah.

Praktična cilja, ki ju bomo zasledovali sta: * modeliranje ocen posameznega uporabnika (odziva) s pomočjo vseh ostalih uporabnikov, * primerjava metod nadzorovanega modeliranja.

Podatki

Opis podatkovne zbirke MovieLens ostaja enak prvi nalogi.

Predpriprava podatkov

Za potrebe te naloge bomo podatke pripravili na naslednji način: 1. Izberi m filmov z vsaj 100 ogledi. 2. Izberi n uporabnikov, ki si je ogledalo vsaj 100 filmov. 3. Pripravi matriko X velikosti $m \times n$, kjer vrstice predstavljajo filme, stolpci pa uporabnike. Neznane vrednosti zamenjaj z 0.

Za vsakega od izbranih n uporabnikov bo zgrajen regresijski model, katerega cilj bo napoved ocen za filme.

$y^{(0)}$

$X^{(0)}$

Film/uporabnik

u_0

u_1

u_2

...

f_1

Twelve Monkeys (a.k.a. 12 Monkeys) (1995)

0

0

2.5

...

f_2

Dances with Wolves (1990)

4

0

0

...

f_3

Apollo 13 (1995)

0

2

0

...

f_4

Sixth Sense, The (1999)

3

0

4

...

...

...

...

...

...

...

$y^{(1)}$

$X^{(1)}$

Film/uporabnik

u_1

u_0

u_2

...

f_1

Twelve Monkeys (a.k.a. 12 Monkeys) (1995)

0

0

2.5

...

f_2

Dances with Wolves (1990)

0

4

0

...

 f_3

Apollo 13 (1995)

2

0

0

...

 f_4

Sixth Sense, The (1999)

0

3

4

...

...

...

...

...

...

...

Razdelitev podatkov za model uporabnika u_0 (zgorja matrika) in uporabnika u_1 (spodaj matrika).

Vprašanja

1. Regresija (100) Za vsakega uporabnika postavite regresijski model. Uporabite eno ali več metod za učenje regresijskih modelov (linearna regresija, Ridge, Lasso, itd.).

Za vsakega od n uporabnikov izberite ustrezni stolpec v matriki podatkov. Za uporabnika i imamo torej:

- Vektor odziva $y^{(i)}$,
- Matriko podatkov $X^{(i)}$, ki vsebuje vse stolpce *razen* i .

Za lažjo predstavo si oglej zgornji tabeli. Nekajkrat (npr., trikrat) ponovite postopek preverjanja s pomočjo učne in testne množice:

- Množico filmov, ki si jih je uporabnik ogledal, *naključno* razdelite v razmerju 75množica). * Naučite regresijski model na učni množici (izberite ustrezne vrstice v X in y).
- Ovrednotite model na testni množici (ponovno izberite ustrezne vrstice v X in y).

Oceno vrednotenja nato delite s številom poizkusov, da dobite končno oceno.

Poročajte o uspešnosti vašega modela. Pri tem se osredotočite na naslednja vprašanja: * Utemeljite ustrezno mero vrednotenja. Ali model dobro napoveduje ocene? * Z izbrano mero ocenite modele za vseh n uporabnikov.

Kodo za odgovore lahko razdelite v več celic.

In []:

Bonus vprašanje (15)Ustvarite novega uporabnika, ki predstavlja vaše ocene

filmov. Ocenite nekaj filmov po lastnem okusu in preverite, kako modeli ocenijo neizbrane filme. Ali se vam zdijo napovedi primerne?

Kodo za odgovore lahko razdelite v več celic.

In []:

Zapiski

Implementacijo, opis in vrednotenje metod za nadzorovanje učenje vsebujejo knjižnice `sklearn` ali `Orange`.

Uporaba matrične faktorizacije za napovedovanje

Podatkovno rudarjenje, naloga, <VPIŠI DATUM ODDAJE>
<VPIŠI Ime in priimek>

V prejšnji domači nalogi smo uporabili metode nadzorovanega modeliranja na problemu napovedovanja ocen neocenjenih filmov. Ker smo za vsakega od m uporabnikov zgradili svoj model, dobimo m modelov, ki si med seboj ne delijo nobene informacije.

Metode matrične faktorizacije so pomemben gradnik sodobnih priporočilnih sistemov. Omogočajo nam, da vsakega uporabnika in vsak izdelek (film) modeliramo s pomočjo r regresijskih modelov, kar vodi v enoten model, ki omogoča napoved ocene za poljubno kombinacijo uporabnika in filma.

Model *matrične faktorizacije* matriko podatkov $X \in \mathbb{R}^{m \times n}$ oceni s produktom dveh matrik nižjega ranga $W \in \mathbb{R}^{m \times r}$ in $H \in \mathbb{R}^{n \times r}$, tako da

$$X = WH^T + E \quad (8.1)$$

kjer je $E \in \mathbb{R}^{m \times n}$ matrika napak oz. ostankov. Matriki modela W in H lahko poiščemo tudi, če nekatere vrednosti v X niso znane, kar velja za priporočilne sisteme. Model omogoča *napoved* vseh omenjenih neznanih vrednosti.

Vrednotenje priporočilnih sistemov se razlikuje od običajnih regresijskih modelov, saj na napovedne vrednosti gledamo kot na *seznam priporočil*, kjer nas zanima samo nekaj vrhnjih elementov tega seznama oz. ali se med njimi nahajajo relevantna priporočila.

Podatki

Opis podatkovne zbirke MovieLens 1996-2016 ostaja enak [prvi nalogi](#).

Predpriprava podatkov

Za potrebe te naloge podatke pripravite na naslednji način:

1. Izberite n filmov, ki imajo vsaj 20 ocen.
2. Izberite m uporabnikov, ki je ocenilo vsaj 20 filmov. Upoštevajte samo filme, izbrane v prejšnjem koraku.
3. Sestavite matriko X velikosti $m \times n$ (v vsaki vrstici vsebuje vsaj 20 ocen).

Nato sestavite učno in testno množico, kot je prikazano na sliki. Za vsakega uporabnika (vrstico v X) izberite k (npr. $k = 5$) visoko ocenjenih filmov (z ocenami 5 ali 4). Učno matriko X_U sestavite tako, da izbrane filme odstranite, in jih shranite v testno matriko X_T .

Vprašanja

1. (30 NMF, predstavljenega na laboratorijskih vajah. Pri izračunu gradienta (odvoda) za vsako spremenljivko upoštevajte samo znane ocene. Na kratko opišite, kateri parametri vplivajo na učenje modela in kako? Kakšne kompromise predstavljajo?
2. (50 testno množico v skladu z opisom na Sliki~??a. Za vsakega uporabnika naključno odstranite $k = 5$ visoko ocenjenih filmov (z ocenami 4 ali 5). Omenjeni filmi predstavljajo *testno množico*.

S pomočjo algoritma poiščite matriki W in H , ki modelirata učno matriko X_U , kot je prikazano na Sliki~\ref{f:nmf-shema}b.

Za vsakega uporabnika i nato napovedajte ocene za vse neocenjene filme. Vektor

ocen pretvorite v seznam priporočil tako, da ocene uredite po padajočem vrstnem redu (višje napovedane ocene se nahajajo v vrhu seznama). Postopek je prikazan na Sliki~\ref{f:nmf-shema}c.

Ocenite, ali se filmi, ki ste jih odstranili za uporabnika i v povprečju pojavljajo bližje vrhu seznama, kot bi to pričakovali po naključju. Na ta način ugotovite, ali model smiselno priporoča filme. Opišite, kako ste izvedli postopek vrednotenja in komentirajte rezultate.

\item (20 \%) Kako parametri modela NMF vplivajo na uspešnost napovedi? Preizkusite npr. nekaj različnih vrednosti za rang (r) matrik W in H in preverite, kako različne nastavitve vplivajo na napoved.

\item (Bonus 10 \%) Ustvarite novega uporabnika, ki predstavlja vaše ocene filmov. Ocenite nekaj filmov po lastnem okusu in ponovite analizo.

Komentirajte ustreznost predlogov.

Zapiski

Pri implementaciji, uporabi in opisu algoritma za reševanje matrične faktorizacije si lahko pomagate z zapiski laboratorijskih vaj, ki jih najdete [na spletni učilnici](#).

Viri

1. Y. Koren, R. Bell, and C. Volinsky, “Matrix factorization techniques for recommender systems,” *Computer* (Long Beach, Calif.), no. 8, pp. 30–37, 2009. [\[Povezava\]](#).

Implementacija priporočilnega sistema

Podatkovno rudarjenje, naloga, <VPIŠI DATUM ODDAJE>
<VPIŠI Ime in priimek>

Aplikacija naj združuje vse pridobljeno znanje. Implementirana naj bo s poljubno tehnologijo in omogočala prijavo novega uporabnika, ocenjevanje filmov ter priporočanje še ne ocenjenih filmov.

Podatki

Opis podatkovne zbirke MovieLens ostaja enak prvi nalogi.

Vprašanja

1. Prototip (100)

Implementirajte prototip aplikacije priporočilnega sistema. Pri tem lahko uporabite poljubne metode podatkovnega rudarjenja. Povsod, kjer implementacija funkcijskih zahteve ni natančno določena, sami sprejmite odločitve, potrebne za implementacijo.

Aplikacija naj omogoča naslednje funkcionalnosti:

- Uporabniški vmesnik. Po lastni izbiri načrtujte preprost uporabniški vmesnik. Vmesnik je lahko ukazna vrstica, grafični vmesnik ali spletni vmesnik. Sistem naj bo možno enostavno zagnati oz. naj bo dostopen za uporabo. Vmesnik omogoča komunikacijo med uporabnikom in priporočilnim sistemom.
- Prijava novega uporabnika. Sistem naj omogoča dodajanje novih uporabnikov in vnos ocen preko uporabniškega vmesnika. Uporabnike lahko identificirate npr. z uporabniškimi imeni, številkami, ipd. Vnesene ocene se ob izhodu iz sistema shranijo in so upoštevane pri ponovnem zagonu. Uporabnik filme, ki so v podatkih, oceni z ocenami med 1 (nezadostno) in 5 (odlično).
- Na zahtevo uporabnika generirajte spisek petih uporabniku najbolj ustreznih filmov, ki jih uporabnik še ni ocenil. Metoda za priporočanje filmov je lahko izbrana iz množice metod, ki smo jih spoznali v okviru predmeta (metode nadzorovanega modeliranja, matrična faktorizacija, ...) ali drugih. V poročilu opišite, katero metodo uporabljate.

Bonus vprašanje (20V aplikacijo dodajte različne vizualizacije trenutnih podatkov, ki jih je mogoče zagnati iz uporabniškega vmesnika. To lahko vključuje gruč uporabnikov, trende različnih žanrov skozi čas, porazdelitve (povprečnih ocen), trenutno najpopularnejše filme, ipd.

V poročilu opišite primere za uporabo vaše aplikacije - slike zaslona s spremnim besedilom oz. ukaze in rezultate, če je vmesnik ukazna vrstica. Pri tem prikažite najmanj en primer za vsako funkcionalnost, ki ste jo implementirali.

Rezultati

Delovanje aplikacije na kratko predstavite v poročilu.

Rezultati naloge so: * poročilo, ki ga sestavljajo kratka navodila za uporabo aplikacije. Oddajte datoteko `.tex` in `.pdf` poročila, * izvorna koda programov (datoteke `.ipynb`, `.py`, ...).

Literatura

- [1] S. van der Walt, S. C. Colbert, and G. Varoquaux. The numpy array: A structure for efficient numerical computation. *Computing in Science Engineering*, 13(2):22–30, March 2011.
- [2]
- [3]
- [4]