

Uvod v odkrivanje znanj iz podatkov

2. september 2026

Pozdravljeni na izpitu.

Izpit rešujate samostojno. Med izpitom ni dovoljena komunikacija z drugimi osebami, uporaba nedovoljenih pripomočkov ali kakršenkoli zunanji dostop tretje osebe do vašega okolja na spletni učilnici.

Seznam nalog

- Verjetnost iz logit vrednosti
Mapa: c-logit-prob
Št. točk: 3
- Preverjanje odvodov s strojnimi in numeričnimi odvaji
Mapa: c-agrad-check
Št. točk: 3
- Linearna regresija L1
Mapa: c-linreg-l1
Št. točk: 4
- Dodajanje novega primera v obstoječo MDS vizualizacijo
Mapa: c-mds-fit
Št. točk: 4
- Projekcije baznih vektorjev v ravnino PCA
Mapa: c-pca-base
Št. točk: 5

Naloge se točkujejo samo s polnimi točkami (implementacija pravilno deluje in je koda, s katero je dosežena točna rešitev, pravilna). Točke so nalogam dodeljene glede na ocenjeno težavnost naloge.

Verjetnost iz logit vrednosti

Število točk za pravilno rešitev: 3

Opis

Pri binarnem razvrščanju lahko linearno kombinacijo vhodnih značilnic zapišemo kot logit vrednost z . Ta vrednost meri, kako daleč je primer od odločitvene meje, vendar sama ni verjetnost. Za interpretacijo napovedi potrebujemo preslikavo logit vrednosti v verjetnost pozitivnega razreda. Ta preslikava mora vrniti število med 0 in 1.

Naloga

V datoteki `logit_prob.py` implementirajte funkcijo,

```
def prob(z):  
    ...
```

Argumenti

- z : logit vrednost (`float`), logaritem kvocienta verjetnosti $\log(p / (1 - p))$

Vrnjena vrednost

Funkcija vrne verjetnost pozitivnega razreda (`float`) za podano logit vrednost.

Primer

```
result = prob(0.6)  
print(round(result, 2))
```

Pričakovani rezultat:

0.65

Testiranje

Teste poženete z ukazom (iz mape naloge):

```
python -m unittest test_logit_prob.py
```

Preverjanje odvodov s strojnim in numeričnim odvajanjem

Število točk za pravilno rešitev: 3

Opis

Dana je funkcija $f(x)$, katere odvod želimo izračunati na dva načina: s strojnim odvajanjem (razred `Value` iz `agrad.py`) in numerično z metodo končnih diferenc.

Ideja seveda je, da sta na oba načina izračunana gradienta kar se da podobna. Tovrstno preverjanje nam lahko pride prav pri kakšnih bolj kompleksih funkcijah, da vidimo, če smo v knjižnici `agrad` pravilno implementirali račun gradientov.

Naloga

V datoteki `agrad_check.py` implementirajte funkciji:

```
def df_num(f, x, h):  
    ...
```

```
def df_agrad(f, x):  
    ...
```

Knjižnice `agrad.py` ne spreminjajte.

Argumenti

- `f`: funkcija, ki prejme en argument
- `x`: točka, v kateri računamo odvod funkcije `f`
- `h`: korak pri metodi končnih diferenc (samo pri `df_num`)

Vrnjena vrednost

- `df_num(f, x, h)` vrne številsko oceno odvoda (`float`)
- `df_agrad(f, x)` vrne številsko vrednost odvoda (`float`), izračunano s strojnim odvajanjem

Primer

```
print(round(df_agrad(f, 0.5), 4))  
print(round(df_num(f, 0.5, 1e-3), 4))
```

Pričakovani rezultat:

```
-0.04  
-0.04
```

Testiranje

Teste poženete z ukazom (iz mape naloge):

```
python -m unittest test_agrad_check.py
```

Linearna regresija L1

Število točk za pravilno rešitev: 4

Opis

Pri linearni regresiji model napove vrednost $\hat{y} = wx + b$. Običajna funkcija izgube kvadrira napake, zato imajo osamelci nesorazmerno velik vpliv na naučeno premico. Problem lahko rešimo s t. im. regresijo L1, kjer v funkciji izgube namesto kvadriranja napak uporabimo absolutne vrednosti napak in izračunamo njihovo povprečje.

Namig: najbrž boste morali v razred `Value` dodati novo metodo. Če želite reimplementirati Pythonovo funkcijo `abs` (s tem imenom), to storite z implementacijo `__abs__`.

Naloga

V datoteki `linreg_l1.py` implementirajte metodo razreda `LinRegL1`.

Izris

Če poženete skripto `linreg_l1.py`, se model nauči na podatkih iz `data.csv`, izpiše MAE in shrani razsevni diagram z naučeno premico v datoteko `result.png`. Izris je namenjen samo vašemu preverjanju, če rešitev z regresijo L1 izboljša prilaganje podatkom (primerjajte z navadno linearno regresijo). Izris se ne preverja v testih.

Testiranje

Teste poženete z ukazom (iz mape naloge):

```
python -m unittest test_linreg_l1.py
```

Dodajanje novega primera v obstoječo MDS vizualizacijo

Število točk za pravilno rešitev: 4

Opis

Včasih želimo na že obstoječi karti – MDS vizualizaciji – dodati nov primer tako, da karte za primere, na katerih smo jo naučili, ne spreminjamo. V že obstoječo karto bi torej radi umestili samo nov primer.

Umestitev novega primera lahko izvedemo tako, da pri gradientnem sestopu posodabljam le njegove koordinate, vse ostale pa pustimo take, kot so. To umestitev lahko izvedemo tako, da je funkcija izgube odvisna samo o pozicije tega primera (in ne od pozicije ostalih točk v MDS vizualizaciji).

Preuči kodo v `mds-fit.py` in spiši manjkajoče vrstice na označenem mestu. Večino kode za zgornjo nalogo smo že spisali, malce smo tudi prilagodili zapis razreda MDS, ki sedaj (posredno) podpira tudi dodajanje novih primerov v že obstoječe karte (namig: parameter `self.cities`). Tudi v namene preverjanja, da je tvoja implementacija v redu, smo izrisali karto z mesti brez celja ([cities.pdf](#)) in to isto karto, kjer predpostavimo, da je dodatni primerno vključeno njegovo pozicijo tudi Celje ([cities-celje.pdf](#)).

Knjižnice `agrad.py` in razreda MDS ne spreminjajte.

Naloga

V datoteki `mds_fit.py` dopolnite kodo med označevalcema `# --- VAŠA KODA ---` in `# --- KONEC ---`.

Primer

Po pravilni implementaciji morajo razdalje od Celja približno ustrezati podanim vrednostim:

Celje-Ljubljana: 88.2 (cilj 77)

Celje-Maribor: 61.2 (cilj 55)

Celje-Koper: 190.9 (cilj 183)

Testiranje

Teste poženete z ukazom (iz mape naloge):

```
python -m unittest test_mds_fit.py
```

Projekcije baznih vektorjev v ravnino PCA

Število točk za pravilno rešitev: 5

Opis

Projekcije z glavnimi komponentami lahko dobro razložimo tudi tako, da v vizualizacijski prostor (npr. v dveh dimenzijah) proiciramo bazne vektorje oziroma značilke osnovnega (začetnega) prostora. Tako npr. ugotovimo, katere značilke so pomembne, kako nam te razpenjajo projekcijski prostor in kako so glavne komponente od njih odvisne.

V nalogi boste izračunali projekcije baznih vektorjev (značilk) osnovnega podatkovnega prostora v ravnini prvih dveh glavnih komponent. Projekcije baznih vektorjev boste uredili skladno z njihovo dolžino in v seznam `vectors` shranili 8 vektorjev, ki so v projekciji najbolj razpeti. V tem seznamu boste v terki ali pa seznamu zapisali ime značilke (ime baznega vektorja) in obe komponenti v PCA prostoru. Na primer:

```
vectors = [  
    ("hair=yes", 0.41, -0.32),  
    ("fins=yes", -0.33, 0.27),  
    # ... skupaj 8 teric (ime, x1, x2), urejenih po padajoči dolžini  
]
```

Terke naj bodo urejene po dolžini proiciranih vektorjev, tako da značilka z najdaljšo projekcijo prva.

Kodo, ki jo razvijete, vstavite v `pca.py` na označenem mestu. V pomoč pri preverjanju smo implementirali tudi `plot_projection`, ki te vektorje predstavi skupaj s primeri iz učne množice in shrani v `pca-2d.pdf`.

Testiranje

Teste poženete z ukazom (iz mape naloge):

```
python -m unittest test_pca.py
```