Predmet: "Vještačka inteligencija"
Laboratorijska vježba 2: Uvod u Python (2/2)
Odgovorna nastavnica: Vanr. prof. dr Amila Akagić

Sadržaj vježbe:

1 Cilj vježbe

Cilj vježbe je upoznavanje sa korištenjem popularnih biblioteka za vještačku inteligenciju i mašinsko učenje u sklopu jezika Python. Na vježbi će se studenti upoznati sa analizom i pripremom podataka koristeći Pandas bibliteku, kao i sa scikit-learn paketom za prediktivnu analizu podataka i mašinsko učenje. Na kraju vježbe, studenti dobijaju uvid u Tensorflow, popularan razvojni okvir za duboko učenje (eng. *deep learning*), koristeći Keras front-end.

2 Python biblioteke za vještačku inteligenciju

Na prethodnoj laboratorijskoj vježbi, dat je kratki uvod u Python, kao i u popularne biblioteke NumPy i Matplotlib. Na današnjoj vježbi će se opisati dodatne biblioteke koje su neizostavan dio svakog rada u oblasti vještačke inteligencije i mašinskog učenja, kao i data science-a, uzimajući u obzir usku povezanost ovih polja.

Svaka od biblioteka predstavljenih u nastavku svoju primjenu nalazi u određenoj fazi procesa dizajniranja sistema vještačke inteligencije, te su većinom građene na osnovu već opisane NumPy biblioteke (to jeste, ove biblioteke u pozadini rade sa NumPy nizovima). Najčešće korištene Python biblioteke za vještačku inteligenciju su prikazane na slici 1.

Biblioteke sa kojima će se raditi na ovoj laboratorijskoj vježbi su:

- Pandas (https://pandas.pydata.org/);
- Scikit-Learn (https://scikit-learn.org/stable/);
- TensorFlow (https://www.tensorflow.org/);
- Keras (https://keras.io/);
- Pickle (https://docs.python.org/3/library/pickle.html).

Pri tome, biblioteke Pandas i Scikit-Learn (skupa sa ranije obrađenim NumPy i Matplotlib) su dio SciPy ekosistema (https://www.scipy.org/), koji predstavlja skup Python biblioteka za naprednu matematiku i inžinjering.

Slika 1: Najpopularnije Python biblioteke za vještačku inteligenciju i mašinsko učenje. Biblioteke NumPy i Matplotlib su obrađene na prethodnoj laboratorijskoj vježbi, dok će ostale biblioteke biti opisane na ovoj vježbi.

2.1 Pandas

Pandas predstavlja biblioteku za jezik Python, čija je primarna svrha manipulacija i analiza velikih skupova podataka. Pandas pruža strukture podataka i operacije za manipulaciju numeričkih tabela (koje se u kontekstu ove biblioteke nazivaju *dataframes*).

Ova biblioteka objedinjuje ideju n-dimenzionalnih nizova visokih performansi iz NumPy sa fleksibilnošću i manipulacijom podataka koje pružaju relacione baze podataka (kao što je SQL). Pruža sofisticirano indeksiranje, koje olakšava operacije nad skupovima podataka kao što su preoblikovanje, particionisanje, agregacija, itd. Neke ključne funkcionalnosti koje Pandas pruža su:

- Brz i efikasan `DataFrame` objekat za manipulaciju podataka;
- Alati za čitanje i zapisivanje podataka u različitim formatima (CSV, obični tekst, MS Excel datoteke, SQL pbaze podataka, te HDF5 format);
- Fleksibilno preoblikovanje skupova podataka;
- Agregacija i transformisanje podataka pomoću group by funkcija, inspirisanih analognim funkcijama iz SQL-a;
- Efikasno spajanje različitih skupova podataka;
- Optimizovano za visoke performanse (u pozadini napisano koristeći C i Cython).

Učitavanje Pandas biblioteke se radi na isti način kao i za biblioteke opisane na prethodnoj vježbi, s tim što se po konvenciji uzima alias `pd`:

```
import pandas as pd
```

Također, konvencija nalaže i da se prije importovanja Pandas biblioteke uradi importovanje NumPy biblioteke.

2.1.1 Pandas DataFrame objekat

`DataFrame` je jedan od dva osnovna objekta Pandas biblioteke (pri čemu je drugi objekat `Series`, koji nije u fokusu ove laboratorijske vježbe). Ovaj objekat zapravo predstavlja tabelarni skup podataka, sličan tabelama koje se mogu naći u relacionim bazama podataka, i koje se koriste u sklopu jezika SQL. Kolone `DataFrame` objekta obično predstavljaju atribute, dok redovi predstavljaju instance. Na primjer, ukoliko se podaci o studentima predstave kao `DataFrame` , svaki red bi bio ekvivalentan jednom studentu, dok bi svaka kolona označavala jedan atribut studenta (npr. ostvarene bodove na prisustvo). Primjer ovakvog skupa podataka predstavljenog kao `DataFrame` je dat na slici 2.

[13]:

	id	Indeks	Prisustvo	Ispit1	Ispit2	UKUPNO	Ocjena
0	1	94-ST	10	9.4	7.5	35.81	7
1	2	77-ST	10	8.5	/	35.50	7
2	3	69-ST	10	17	/	39.50	9
3	4	79-ST	0	7.6	4.8	12.40	/
4	5	89-ST	10	10.6	12.5	35.08	6
5	6	78-ST	0	/	/	25.91	6
6	7	93-ST	10	/	12.5	35.17	8
7	8	68-ST	10	/	/	50.00	9
8	9	83-ST	0	/	/	0.00	/
9	10	92-ST	10	/	/	42.06	9

Slika 2: Prikaz Pandas `DataFrame` objekta.

2.1.2 Čitanje i pisanje podataka

Najčešći način učitavanja skupova podataka u Pandas DataFrame jeste kroz CSV (eng. *comma separated values*) datoteke. Ove datoteke se koriste zbog svoje jednostavnosti, i zbog činjenice da su kompatibilni sa svim popularnim operativnim sistemima. Skup podataka iz CSV datoteke se učitava koristeći funkciju `pd.read_csv`. Ova funkcija obavezno prima naziv CSV datoteke (to jeste putanju do iste) koja se namjerava učitati. Pored toga, sadrži jako puno opcionalnih parametara, od kojih je najbitniji parametar `sep`, kojim se može naznačiti separatorski znak (podrazumijevani znak je zarez odnosno `,`). Primjer poziva ove funkcije sa drugačijim separatorom je dat u nastavku:

```
data = pd.read_csv(putanja_do_datoteke, sep=';')
```

Ovdje je za separator uzet znak tačka-zarez. Kompletan kod koji daje rezultat sa slike 2 je kako slijedi:

```
import numpy as np  # ucitaj NumPy
import pandas as pd # ucitaj Pandas

data = pd.read_csv('dataset.csv') # ucitaj podatke

data # ispisi podatke
```

Prikaz učitane CSV datotkee van Python-a (u običnom uređivaču teksta) je dat na slici 3.

```
id,Indeks,Prisustvo,Ispit1,Ispit2,Ispit1_popravni,Ispit2_popravni,UKUPNO,Ocjena
1,94-ST,10,9.4,7.5,12.08,13.73,35.81,7
2,77-ST,10,8.5,/,/,/,35.5,7
3,69-ST,10,17,/,/,/,39.5,9
4,79-ST,0,7.6,4.8,/,/,12.4,/
5,89-ST,10,10.6,12.5,12.58,/,35.08,6
6,78-ST,0,/,/,13.08,12.83,25.91,6
7,93-ST,10,/,12.5,12.67,/,35.17,8
8,68-ST,10,/,/,/,/,50,9
9,83-ST,0,/,/,/,/,0,/
10,92-ST,10,/,/,/,/,42.06,9
```

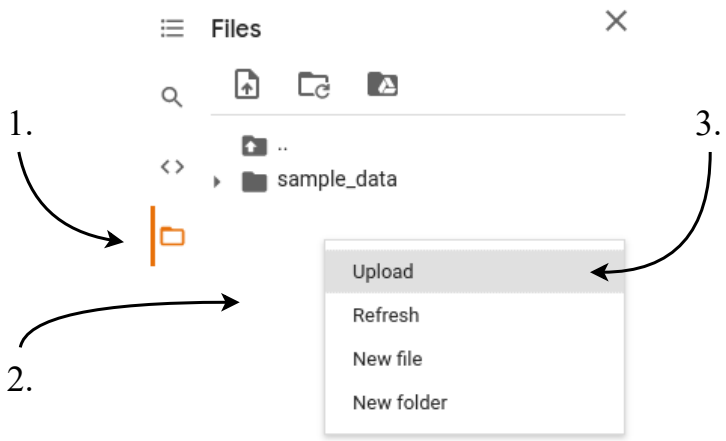
Slika 3: CSV datoteka. Svaka vrijednost je odvojena odgovarajućim znakom (separatorom) - u ovom slučaju, taj znak je zarez. Prvi red (po konvenciji) imenuje kolone.

Na učitanom skupu podataka se može primijetiti sljedeće: Pandas je, prilikom učitavanja, automatski uzeo prvi red CSV datoteke, te njega iskoristio za nazive kolona. Ovo ponašanje se može onemogućiti dodatnim parametrima koji se pošalju funkciji `pd.read_csv`[1].

Na ovom mjestu treba napomenuti da, prilikom rada sa Google Colab, eksterne datoteke koje se namjeravaju koristiti (kao što je u primjeru iznad datoteka *dataset.csv*) potrebno prvo učitati u Google Colab okruženje. To se može uraditi tako što se prije svega klikne na ikonicu datoteke sa lijeve strane prozora. Nakon toga, u novootvorenom prostoru

[1] Pogledati https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html za više informacija.

se desnim klikom otvori padajući meni, te se odabere opcija *Upload*. Zatim je potrebno odabrati datoteku koja se učitava u Google Colab okruženje. Treba napomenuti da se sve učitane datoteke gube nakon završetka Google Colab sesije, te se poslije opet mora vršiti njihovo učitavanje. Proces dodavanja datoteke u Google Colab je ilustrovan na slici 4.



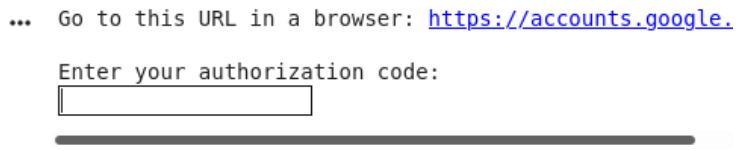
Slika 4: Dodavanje datoteke u sklopu Google Colab okruženja. Prvi korak jeste kliknuti na ikonicu direktorija. Drugi korak je desni klik na bilo kojem mjestu u novom prostoru. Treći korak je kliknuti na opciju *Upload*.

Alternativni način učitavanja vanjskih datoteka na Google Colab jeste direktno kroz Google Drive. Ovo je posebno korisno ukoliko se radi sa velikim skupovima podataka, jer se ne mora vršiti *upload* za svaku novu sesiju u Google Colab. Kako bi se povezao Google Drive na Google Colab, potrebno je prije svega u jednu kodnu ćeliju unijeti sljedeći isječak koda:

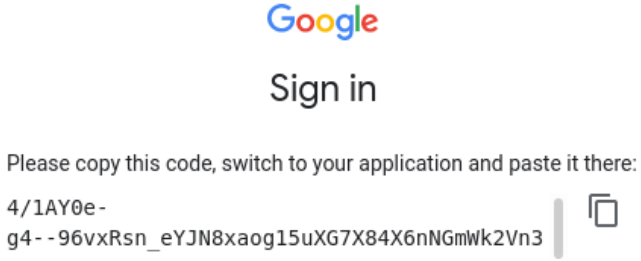
```
from google.colab import drive

drive.mount('/content/drive')
```

Nakon što se izvrši ova kodna ćelija, otvara se polje za unos autorizacijskog koda, kao na slici 5. Potrebno je kliknuti na URL koji je prikazan, te odabrati Google račun koji se želi spojiti na Colab. Nakon toga, dodbija se autorizacijski kod, kao što je prikazano na slici 6.

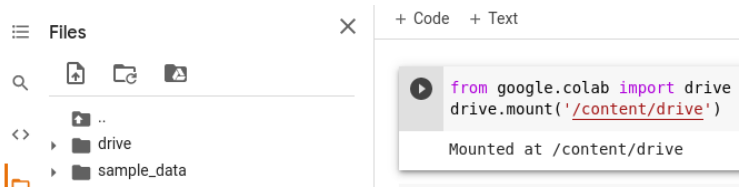


Slika 5: Polje za unos autorizacijskog koda.



Slika 6: Prikaz prozora nakon što se odabere odgovarajući Google račun.

Nakon što se autorizacijski kod kopira i unese u polje unutar Google Colab okruženja, Google Drive je uspješno spojen, te se može pristupiti svim podacima koji se nalaze na istom. Ovo je prikazano na slici 7.



Slika 7: Nakon što se unese autorizacijski kod, može se pristupiti sadržaju Google Drive-a (unutar novonastalog direktorija `drive`).

Ukoliko se podaci žele spasiti u eksternu datoteku, može se koristiti metoda `DataFrame.to_csv`. Primjer korištenja ove funkcije je dat u nastavku:

```
data.to_csv('izlaz.csv', sep=';')
```

U ovom primjeru, varijabla `data` (koja je tipa `DataFrame`) se sačuva u eksternu CSV datoteku `izlaz.csv`, uz separatorski znak tačka-zarez (`;`).

Pored čitanja i pisanja CSV datoteka, Pandas podržava i mnoštvo drugih formata. Za studente koji su zainteresovani za rad sa drugim tipovima datoteka, preporučuje se da pogledaju dokumentaciju za rad sa U/I u Pandas dostupnu na linku:

https://pandas.pydata.org/pandas-docs/stable/reference/io.html

2.1.3 Manipulacija nad skupovima podataka

Nakon što se podaci učitaju u Pandas `DataFrame` objekat, moguće je vršiti razne manipulacije nad njima. Obzirom da Pandas podržava jako puno različitih metoda za manipulaciju podacima, ovdje će biti navedene samo osnovne tehnike:

- Pregled podataka se može vršiti metodama `DataFrame.head` i `DataFrame.tail`. Ove metode se koriste kako bi se pregledao početak i kraj skupa podataka respektivno. Ovo je posebno korisno ukoliko se radi o velikim skupovima podataka. Primjer korištenja ovih metoda u sklopu Jupyter Notebook okruženja je dat na slici 8;

```
[9]: data.head()

[9]:    id Indeks  Prisustvo Ispit1  ... Ispit1_popravni Ispit2_popravni UKUPNO
     Ocjena
     0   1  94-ST         10    9.4  ...           12.08           13.73  35.81
     7
     1   2  77-ST         10    8.5  ...               /               /  35.50
     7
     2   3  69-ST         10     17  ...               /               /  39.50
     9
     3   4  79-ST          0    7.6  ...               /               /  12.40
     /
     4   5  89-ST         10   10.6  ...           12.58               /  35.08
     6

     [5 rows x 9 columns]

[10]: data.tail()

[10]:    id Indeks  Prisustvo Ispit1  ... Ispit1_popravni Ispit2_popravni UKUPNO
      Ocjena
      42  43   1-ST         10      /  ...               /               /  10.00
      /
      43  44  88-ST         10      /  ...               /               /  37.00
      7
      44  45  75-ST         10      /  ...            9.83            13.5  33.33
      6
      45  46  67-ST         10   16.5  ...               /               /  39.50
      7
      46  47  81-ST         10   15.7  ...               /               /  43.00
      8

      [5 rows x 9 columns]
```

Slika 8: Metode `head` i `tail` - prikazuju prvih 5 i posljednjih 5 redova u skupu podataka, respektivno. Ukoliko se želi prikazati drugi broj redova, isti se može proslijediti kao parametar (npr. `data.head(3)`).

- Transponovanje podataka se radi pomoću atributa `T`, na primjer:

```
data_transponovano = data.T
```

- Svaki Pandas `DataFrame` se može preoblikovati u NumPy niz, a važi i obrnuto. Pri konverziji u NumPy niz, podaci o navizima kolona se gube. Iz tog razloga je potrebno eksplicitno navesti nazive kolona kada se NumPy niz vraća u Pandas DataFrame. Metode koje se koriste su:
  - `data.to_numpy()`
  - `pd.DataFrame(neki_NumPy_niz, columns=['Kolona1, 'Kolona2', ...])`
- Ukoliko se želi odabrati jedna kolona, njen naziv se može navesti kao indeks. Primjer ovoga je dat na slici 9;

```
[11]: data["Indeks"]

[11]: 0     94-ST
      1     77-ST
      2     69-ST
      3     79-ST
      4     89-ST
      5     78-ST
      6     93-ST
      7     68-ST
      8     83-ST
      9     92-ST
      10    65-ST
      (ostatak izlaza izrezan radi preglednosti)
      Name: Indeks, dtype: object
```

Slika 9: Indeksacija kolone.

- Također je moguće odabrati određenu podsekvencu redova, koristeći operator dvotačka, kao što je prikazano na slici 10;

```
[12]: data[10:15]

[12]:    id Indeks  Prisustvo Ispit1  ... Ispit1_popravni Ispit2_popravni UKUPNO
      Ocjena
      10  11  65-ST         10      /  ...              /              /  43.19
      8
      11  12  70-ST         10     13  ...              /           16.3  39.30
      7
      12  13  90-ST         10   15.6  ...              /           14.3  39.90
      8
      13  14  97-ST         10      /  ...              /              /  41.05
      9
      14  15  84-ST         10      /  ...              /              /  43.56
      9

      [5 rows x 9 columns]
```

Slika 10: Indeksacija kolone.

- Atributom `loc` je moguće selektirati grupu redova i kolona po njihovim labelama, ili po nekom logičkom uslovu. Primjer korištenja `loc` je dat na slici 11.

```
[13]: # nadji indeks studenta u 11. redu
      data.loc[11, 'Indeks']

[13]: '70-ST'

[17]: # svi studenti koji imaju ukupno preko 43 boda
      data.loc[data['UKUPNO'] > 43]

[17]:     id Indeks  Prisustvo Ispit1  ... Ispit1_popravni Ispit2_popravni UKUPNO
      Ocjena
      7    8  68-ST         10      /  ...              /              /  50.00
      9
      10  11  65-ST         10      /  ...              /              /  43.19
      8
      14  15  84-ST         10      /  ...              /              /  43.56
      9
      16  17  91-ST         10      /  ...              /              /  43.48
      9
      26  27  63-ST         10   16.6  ...              /             20  46.60
      8

      [5 rows x 9 columns]

[21]: # izlistaj indeks i prisustvo za prvih 5 studenata
      data.loc[:4, ['Indeks', 'Prisustvo']]

[21]:   Indeks  Prisustvo
      0  94-ST         10
      1  77-ST         10
      2  69-ST         10
      3  79-ST          0
      4  89-ST         10
```

Slika 11: Korištenje `loc` za filtriranje podataka.

- Atribut `iloc` se također može koristiti za selekciju podskupa `DataFrame` objekta, uz razliku što `iloc` radi sa integer lokacijama podataka, dok `loc` radi sa labelama. Razlika između `loc` i `iloc` je demonstrirana na slici 12.

```
[27]: df = pd.DataFrame(np.arange(25).reshape(5,5),
                        index=[5,4,3,2,1],
                        columns=['x', 'y', 'z', 'v', 'i'])
      df

[27]:    x   y   z   v   i
      5  0   1   2   3   4
      4  5   6   7   8   9
      3  10  11  12  13  14
      2  15  16  17  18  19
      1  20  21  22  23  24

[29]: # loc radi sa kljucem (index) reda,
      # i sa nazivima kolona
      df.loc[1, 'x']

[29]: 20

[32]: # iloc radi isključivo sa stvarnim (integer) indeksima
      # (kao standardne matrice)
      df.iloc[1, 0]

[32]: 5
```

Slika 12: Razlika između `loc` i `iloc`.

- Često su skupovi podataka nepotpuni, te je potrebno raditi sa nedostajućim podacima (tzv. *NaN* vrijednosti). Pandas primarno koristi vrijednost `np.nan` kako bi označavao nedostatak nekog podatka. Međutim, u skupovima podataka se nedostajući podaci često prikazuju kao znak `/`. Prvi korak prilikom rada sa nedostajućim vrijednostima jeste iste zamijeniti sa `np.nan`. To se može postići naredbom `replace`, kao što je prikazano na slici 13.

```
[3]: data.replace('/', np.nan, inplace=True)
     data.head() # svaka pojava / zamijenjena sa NaN
```

```
[3]:    id Indeks  Prisustvo Ispit1  ... Ispit1_popravni Ispit2_popravni UKUPNO
     Ocjena
     0   1  94-ST         10    9.4  ...           12.08           13.73  35.81
     7
     1   2  77-ST         10    8.5  ...             NaN             NaN  35.50
     7
     2   3  69-ST         10     17  ...             NaN             NaN  39.50
     9
     3   4  79-ST          0    7.6  ...             NaN             NaN  12.40
     NaN
     4   5  89-ST         10   10.6  ...           12.58             NaN  35.08
     6

     [5 rows x 9 columns]
```

Slika 13: Zamjena podataka u skupu.

- Moguće je nedostajuće podatke popuniti i sa odgovarajućom vrijednosti. Sintaksa je

  `data.fillna(value=vrijednost)`

- Ukoliko se nedostajući podaci žele odbaciti, može se koristiti naredba `dropna`. Ova metoda prima dva parametra:
  - `axis` - određuje da li se odbacuju redovi ili kolone (0 za redove, 1 za kolone - podrazumijevana vrijednost je nula);
  - `method` - može biti *'any'* ili *'all'*: u prvom slučaju, odbacuje se red (kolona) ako ima barem jednu *NaN* vrijednost, dok se u drugom slučaju odbacuje red (kolona) ako je svaka vrijednost u tom redu (koloni) *NaN*.

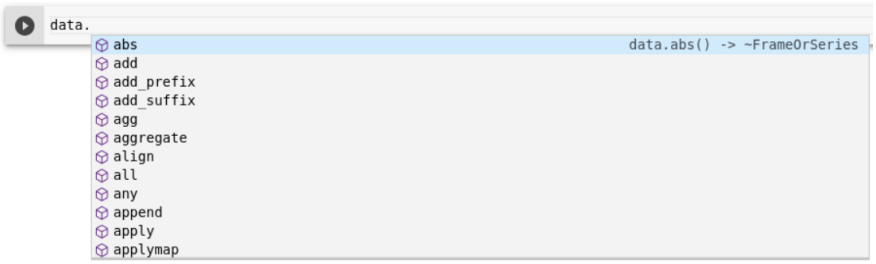
  Primjer korištenja ove metode je

  `data.dropna(0, 'all')`

- Nova kolona se u Pandas `DataFrame` može dodati jednostavnim indeksiranjem njenog imena. Na primjer, ukoliko u naš skup podataka želimo dodati novu kolonu *Kvizovi*, onda bismo pisali

  `data['Kvizovi'] = niz_vrijednosti`

- Za bilo kakve naprednije operacije nad svim poljima `DataFrame` objekta, može se koristiti automatsko izlistavanje mogućnosti na Google Colab okruženju, kao što je prikazano na slici 14.



Slika 14: Izlistavanje svih metoda klase `DataFrame`. Analogno se mogu izlistati i metode za druge klase.

- Konačno, ukoliko se želi ukloniti specifičan red ili kolona iz skupa podataka, to se može uraditi pomoću metode `drop`, kao što je demonstrirano na slici 15.

```
[44]: data.head()
```

```
[44]:    id Indeks  Prisustvo Ispit1  ... Ispit1_popravni Ispit2_popravni UKUPNO
      Ocjena
      0   1  94-ST         10    9.4  ...           12.08           13.73  35.81
      7
      1   2  77-ST         10    8.5  ...             NaN             NaN  35.50
      7
      2   3  69-ST         10     17  ...             NaN             NaN  39.50
      9
      3   4  79-ST          0    7.6  ...             NaN             NaN  12.40
      NaN
      4   5  89-ST         10   10.6  ...           12.58             NaN  35.08
      6

      [5 rows x 9 columns]
```

```
[45]: data.drop([2, 3, 4], inplace=True) # izbaci redove na indeksima 2, 3, i 4
      data.head()
```

```
[45]:    id Indeks  Prisustvo Ispit1  ... Ispit1_popravni Ispit2_popravni UKUPNO
      Ocjena
      0   1  94-ST         10    9.4  ...           12.08           13.73  35.81
      7
      1   2  77-ST         10    8.5  ...             NaN             NaN  35.50
      7
      5   6  78-ST          0    NaN  ...           13.08           12.83  25.91
      6
      6   7  93-ST         10    NaN  ...           12.67             NaN  35.17
      8
      7   8  68-ST         10    NaN  ...             NaN             NaN  50.00
      9

      [5 rows x 9 columns]
```

```
[47]: # izbaci kolone sa popravnim ispitima
      data.drop(columns=['Ispit1_popravni', 'Ispit2_popravni'], inplace=True)
      data.head()
```

```
[47]:    id Indeks  Prisustvo Ispit1 Ispit2  UKUPNO Ocjena
      0   1  94-ST         10    9.4    7.5   35.81      7
      1   2  77-ST         10    8.5    NaN   35.50      7
      5   6  78-ST          0    NaN    NaN   25.91      6
      6   7  93-ST         10    NaN   12.5   35.17      8
      7   8  68-ST         10    NaN    NaN   50.00      9
```

Slika 15: Izbacivanje redova i kolona. Obratiti pažnju na to kako se nakon izbacivanja redova indeksi ne ažuriraju (to jeste, nakon indeksa 1 ide indeks 5). Do ovoga dolazi iz razloga što se `DataFrame` u pozadini implementira kao heš-mapa. Kako bi se indeksi aktualizirali, potrebno je koristiti naredbu `reset_index`, npr. `data = data.reset_index(drop=True)`.

Pored opisanih naredbi, Pandas podržava puno drugih mogućnosti. Za detaljan uvid u ovu biblioteku, pogledati službenu dokumentaciju na

https://pandas.pydata.org/docs/user_guide/index.html

2.2 Scikit-Learn (sklearn)

Scikit-learn (često se piše u skraćenoj formi kao Sklearn) predstavlja najpoznatiju i najpopularniju *open-source* Python biblioteku za klasično mašinsko učenje. Pruža alaze za klasifikaciju, regeresiju, klastering, i obradu podataka. Biblioteka je bazirana na NumPy, SciPy, i Matplotlib. Za razliku od Pandasa, koji se više fokusira na učitavanje i manipulaciju podataka, Sklearn se više fokusira na modeliranje podataka. Neki najpopularnije grupe modela koje Sklearn podržava su:

- Algoritmi nadziranog učenja (eng. *supervised learning*) - uključuje algoritme kao što su linearna regresija, SVM, stabla odlučivanja, itd.;
- Algoritmi nenadziranog učenja (eng. *unsupervised learning*) - faktorska analiza, PCA (*principal component analysis*), nenadzirane neuralne mreže;
- Klastering (eng. *clustering*) - korisno za grupisanje nelabeliranih podataka;
- Kros-validacija (eng. *cross validation*) - koristi se za provjeru tačnosti nadziranih modela nad novim podacima;

- Ekstrackija značajki (eng. *feature extraction*) - definisanje atributa u slikovnim i tekstualnim podacima;

Na ovoj vježbi, biblioteka Sklearn će se koristiti za preprocesiranje podataka. Ono se obično sastoji od dva koraka:

1. Rukovanje *NaN* vrijednostima;

   - Za ovaj korak, Sklearn nudi mnoge predefinisane klase i funkcije, od kojih je najjednostavniji `SimpleImputer`. Jednostavan način za popunjavanje *NaN* vrijednosti je dat na slici 16.

```
[26]: from sklearn.impute import SimpleImputer

      si = SimpleImputer(strategy='mean')

      print('Prije zamjene: ')
      print(data.loc[:5, 'Ispit2'])

      data.loc[:, 'Ispit2'] = si.fit_transform(data.loc[:, 'Ispit2'].values.
       ↪reshape(-1,1))

      print('Nakon zamjene: ')
      print(data.loc[:5, 'Ispit2'])
```

```
Prije zamjene:
0     7.5
1     NaN
2     NaN
3     4.8
4    12.5
5     NaN
Name: Ispit2, dtype: object
Nakon zamjene:
0     7.500000
1     9.323529
2     9.323529
3     4.800000
4    12.500000
5     9.323529
Name: Ispit2, dtype: float64
```

Slika 16: Popunjavanje *NaN* vrijednosti.

U ovom primjeru, popunjavaju se sve *NaN* vrijednosti u koloni *Ispit2*, i to strategijom *mean* (srednja vrijednost). To znači da će se umjesto svake *NaN* vrijednosti upisati srednja vrijednost izračunata na osnovu postojećih podataka (u ovom slučaju, to su bodovi ostvareni na drugi parcijalni ispit). Obzirom da se radi samo mijenjanje *NaN* vrijednosti nad jednom kolonom, neophodno je koristiti funkciju `reshape`, jer `SimpleImputer` očekuje 2D niz. Ukoliko se radi zamjena nad više kolona (ili redova) istovremeno, tada se funkcija `reshape` treba izostaviti. Pored *mean* strategije, koja je korištena u ovom primjeru, postoji još nekoliko strategija, i to:

   - `median` - mijenja *NaN* vrijednosti sa vrijednošću mediana;
   - `most_frequent` - mijenja *NaN* vrijednosti sa najčešćom vrijednošću u skupu podataka;
   - `constant` - mijenja *NaN* vrijednosti nekom konstantom (analogno `fillna` iz Pandas biblioteke).

2. Transformacija podataka (skaliranja/normalizacija).

   - Za transformaciju podataka, Sklearn nudi mnogo različitih metoda, od kojih će kratko biti opisane dvije:
     - Z-score normalizacija, koja se dobija po formuli

$$Z = \frac{x - \mu}{\sigma},$$

       gdje je μ prosječna vrijednost uzorka, a σ standardna devijacija. U sklopu Sklearn, Z-score normalizacija se može izvršiti koristeći funkciju `scale`;
     - MinMax normalizacija, koja se dobija po formuli

$$x_{novo} = \frac{x - min}{max - min}$$

   Transformacija podataka je izuzetno bitna u algoritmima mašinskog učenja i vještačke inteligencije. Mnogo algoritama pokušava pronaći trendove u podacima tako što ih međusobno poredi - problem nastaje kada

su različiti atributi predstavljeni drastično različitim opsezima. Na primjer, ukoliko se radi sa skupom podataka kuća, jedan od atributa može biti broj soba (koji se u većini slučajeva kreće između 1 i 15), dok drugi može biti starost kuće u godinama, gdje se mogu naći vrijednosti i veće od 100. Sa ovako različitim opsezima, algoritam ne bi bio u stanju da zaključi da postoji velika razlika između kuće sa dvije sobe, i kuće sa 15 soba. Primjer primjene Z-score normalizacije i MinMax normalizacije nad nekim atributom skupa podataka je dat na slici 17.

```
[32]: from sklearn.preprocessing import scale

      data['Ispit2'] = scale(data['Ispit2']) # Z score

      data.loc[:5, 'Ispit2']
```

```
[32]: 0   -7.156096e-01
      1    6.970976e-16
      2    6.970976e-16
      3   -1.775173e+00
      4    1.246546e+00
      5    6.970976e-16
      Name: Ispit2, dtype: float64
```

```
[40]: from sklearn.preprocessing import MinMaxScaler

      mm = MinMaxScaler()

      data['Ispit2'] = mm.fit_transform(data['Ispit2'].values.reshape(-1,1))

      data.loc[:5, 'Ispit2']
```

```
[40]: 0    0.379747
      1    0.495160
      2    0.495160
      3    0.208861
      4    0.696203
      5    0.495160
      Name: Ispit2, dtype: float64
```

Slika 17: Z-score normalizacija i MinMax normalizacija.

Još jedna često korištena funkcija u sklopu Sklearn biblioteke jeste funkcija `train_test_split`. Ova funkcija ulazne nizove ili matrice dijeli u dva podskupa: trening podskup i test podskup. Prvi ulazni niz (matrica) predstavlja niz značajki, dok drugi predstavlja niz labela - konvencija je da se označavaju sa x i y, respektivno. Posebno je korisna prilikom rada sa neuralnim mrežama, jer automatizira podjelu podataka. Primjer poziva ove funkcije je dat u isječku koda koji slijedi:

```
import numpy as np
from sklearn.model_selection import train_test_split

X, Y = np.arange(10).reshape((5, 2)), range(5)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
                                                    test_size=0.33,
                                                    random_state=42)
```

U ovom primjeru, kreira se matrica dimenzija 5×2 za varijablu `X`, kao i niz (lista) od 5 elemenata za varijablu `y`. Nakon toga, ova dva niza se šalju u funkciju `train_test_split`, kako bi se razdvojili na skupove za trening i testiranje. Pri tome, skup za testiranje će sadržavati 33% podataka iz originalnih skupova, dok je *seed* za generator slučajnih brojeva postavljen na vrijednost 42.

Pored prezentiranih funkcija, biblioteka Sklearn sadrži još jako puno ugrađenih algoritama, klasifikatora, pomoćnih klasa, skupova podataka, itd. Primjer korištenja mogućnosti Sklearn biblioteke za rješavanje jednostavnog problema mašinskog učenja će biti predstavljen u sekciji *Zadaci za rad u laboratoriji*.

Detaljniji opis biblioteke je dostupan na linku

https://scikit-learn.org/stable/user_guide.html

Za one studente koje zanimaju napredni koncepti biblioteka NumPy, Pandas, i Sklearn, preporučuje se knjiga *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*, koja je besplatno dostupna na linku

https://www.programmer-books.com/wp-content/uploads/2019/04/Python-for-Data-Analysis-2nd-Edition.pdf

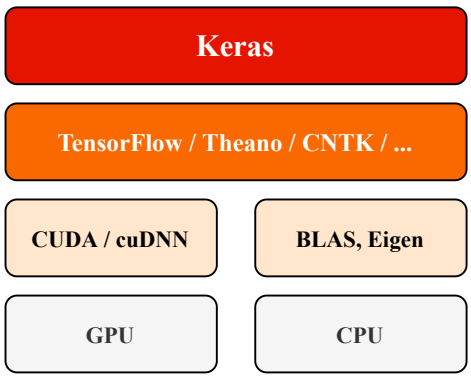
2.3 Keras i TensorFlow

U posljednje vrijeme, grana vještačke inteligencije koja dobija na sve većem značaju je tzv. duboko učenje (eng. *deep learning*). Duboko učenje je bazirano na vještačkim neuralnim mrežama (eng. *ANN - artificial neural networks*), te svoju primjenu nalazi u računarskoj i mašinskoj viziji, prepoznavanju govora, obradi prirodnog jezika, bioinformatici, famraciji, analizi slike u medicini, itd.

Uzimajući u obzir rast popularnosti ove oblasti, razvijeni su mnogi alati za lakši dizajn i evaluaciju modela dubokog učenja, od kojih su dva najpopularnija TensorFlow (napravljen od strane Google-a), i PyTorch (kojeg razvija Facebook istraživački laboratorij za vještačku inteligenciju FAIR). TensorFlow je našao svoju upotrebu u industriji, dok je PyTorch relativno nov razvojni okvir, koji se (barem za sada) primarno koristi u istraživačke svrhe. Zbog jednostavnosti korištenja, kao i većoj zajednici korisnika i dostupnosti dokumentacije, na ovom predmetu ćemo koristiti TensorFlow. Studentima koji su zainteresovani i za PyTorch se preporučuje da isti pogledaju detaljnije na službenoj web stranici: https://pytorch.org/.

Glavni razlog zbog kojeg je TensorFlow dobio na popularnosti jeste njegova jednostavnost. Nju zahvaljuje biblioteci Keras, koja predstavlja API na visokom nivou apstrakcije namijenjen za brzo i efikasno razvijanje i eksperimentisanje sa neuronskim mrežama. Keras je napisan u Python-u, i može se pokretati preko TensorFlow-a (kao i preko CNTK-a ili Theano-a). Trenutno aktuelna verzija Keras-a je 2.4.0 i podržava TensorFlow 2.0.

U pozadini, TensorFlow koristi specijalne biblioteke za brze matematičke operacije. Ukoliko se izvršava na CPU, koriste se biblioteke BLAS i Eigen, dok se za GPU izvršavanje (koje je znatno brže) koristi CUDA odnosno cuDNN. Pored CPU i GPU načina rada, TensorFlow podržava i tzv. TPU (eng. *tensor processing unit*), procesor specijaliziran za rad sa modelima dubokog učenja. Generalna struktura Keras-a i TensorFlow-a je data na slici 18.



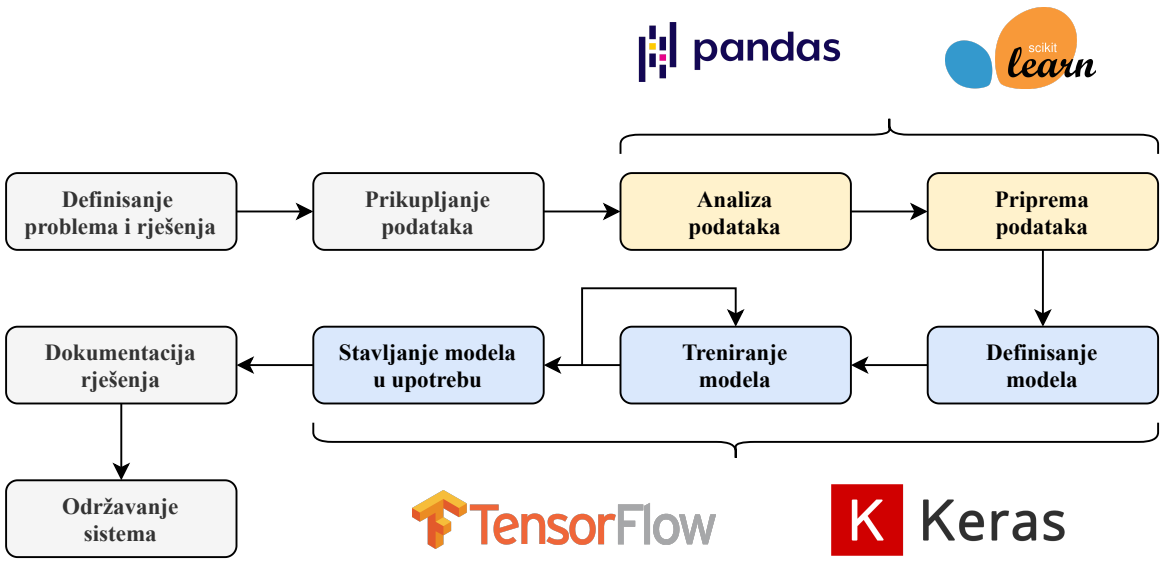
Slika 18: Odnos između Keras-a i TensorFlow-a.

Počevši sa TensorFlow verzijom 2.0, Keras je postao integrisani dio TensorFlow-a, te se obje biblioteke po konvenciji importuju pomoću sljedećeg koda:

```
import tensorflow as tf
from tensorflow import keras
```

U literaturi se često može pročitati pojam tenzor (eng. *tensor*), te na ovom mjestu treba napomenuti da tenzor nije ništa više nego n-dimenzionalni niz. TensorFlow sve svoje operacije interno realizuje koristeći tenzore (po kojima je i nazvan).

Razvijanje modela dubokog učenja je proces koji se sastoji od više faza. Na slici 19 su prikazane faze ovog procesa, kao i biblioteke koje se obično koriste prilikom izrade istog.



Slika 19: Razvojni tok sistema baziranog na vještačkoj inteligenciji (konkretno, dubokog učenja). Biblioteke sa kojima smo se upoznali na ovoj vježbi su neizostavan dio svakog kvalitetnog projekta iz vještačke inteligencije.

Sa bibliotekom TensorFlow (odnosno Keras) ćemo se detaljnije upoznati kroz jednostavan primjer u sekciji *Zadaci za rad u laboratoriji*, kao i u narednim laboratorijskim vježbama. Konkretno, pomoću TensorFlow možemo rješavati zadatke:

- Klasifikacije - pridruživanje određene klase odgovarajućoj instanci (na primjer, prepoznavanje cifara);
- Regresije - predikcija kontinualne izlazne varijable (na primjer estimacije cijene stana na osnovu njegovih karakteristika);
- Napredni zadaci, kao što su obrada slike[2], generisanje uvjerljivih novih podataka[3], autonomna vožnja[4], itd.

Ključna razlika između dubokog učenja i klasičnih metoda mašinskog učenja (kakve se mogu naći u biblioteci Sklearn) leži u tome da mašinsko učenje koristi tačno definisane algoritme (npr. k-NN, Naive Bayes, SVM, itd.) kako bi parsirao podatke, naučio odgovarajuće uzorke, te napravio odluku baziranu na onome što je naučio. Duboko učenje struktuira algoritam u slojeve duboke neuralne mreže, koja se trenira kako bi samostalno napravila odluku. Za takvu metodologiju obično su potrebne velike količine podataka, na osnovu kojih model može naučiti donošenje ispravnih odluka. Iz tog razloga, duboko učenje je hardverski veoma zahtjevno, te se za treniranje velikih modela dubokog učenja obično koriste superračunari.

Pored standardnog TensorFlow-a, koji će biti obrađen na ovom predmetu, postoje još i:

- TensorFlow Lite (https://www.tensorflow.org/lite) - specijalna verzija TensorFlow-a za IoT i mobilne uređaje;
- TensorFlow.js (https://www.tensorflow.org/js) - razvoj i konverzija jednostavnih TensorFlow modela u JavaScript, kako bi se omogućilo izvršavanje u web pregledniku;
- TensorFlow Extended (https://www.tensorflow.org/tfx) - ML *pipeline* namijenjen za produkciju.

Konačno, TensorFlow posjeduje i vlastiti softver za vizualizaciju i kolaboraciju - TensorBoard (https://tensorboard.dev/). Detaljne informacije o TensorFlow i Keras bibliotekama se mogu naći na linkovima:

https://www.tensorflow.org/overview
https://keras.io/guides/

[2] https://developer.nvidia.com/maxine
[3] https://thispersondoesnotexist.com/
[4] https://lexfridman.com/

2.4 Pickle

Iako se obično ne navodi u sklopu ekosistema biblioteka za vještačku inteligenciju, biblioteka Pickle predstavlja neizostavan dio razvojnog toka za svakog inžinjera. Kao što je već ranije naglašeno, rad na projektu iz oblasti vještačke inteligencije podrazumijeva manipulaciju nad odgovarajućim skupovima podataka. Ovo je pogotovo izraženo kada se radi o dubokom učenju. Nakon procesiranja tih podataka, često postoji potreba da se isti sačuvaju u neku datoteku, kako bi se mogli koristiti poslije (da se ne bi ponavljao korak preprocesiranja), ili poslati nekome drugom. Upravo to radi Pickle biblioteka - serijalizira objekte kako bi se mogli sačuvati u neku datoteku, te se poslije ponovo učitati.

Serijalizacija je proces konvertovanja objekta (npr. NumPy niza) koji se nalazi u memoriji u tzv. *bytestream*, kako bi se mogao sačuvati na glavnoj memoriji (HDD ili SSD). Tako sačuvan (serijalizirani) objekat se onda može, na primjer, slati kroz mrežu drugim korisnicima. Serijalizacija se ne treba miješati sa pojmom kompresije, obzirom da serijalizacija prvenstveno radi konverziju objekta u *bytestream*, dok smanjenje njegove veličine nije garantovano. Pickle biblioteka se može učitati naredbom

```
import pickle
```

Sama biblioteka sadrži dvije osnovne funkcije - `dump` i `load`. Prva funkcija uzima varijablu i istu serijalizira, kreirajući pri tome izlaznu datoteku sa imenom koje se preda kao parametar. Druga naredba radi suprotno - ona postojeću datoteku čita, te ju deserijalizira, te kao rezultat vraća odgovarajuću varijablu. Primjer korištenja ove dvije funckije je dat na slici 20.

[11]:
```
import pickle
import os
import numpy as np

niz = np.arange(1000000)

print("Zauzima {0} kB".format(niz.nbytes/1000))

pickle.dump(niz, open( "datoteka.p", "wb" ), protocol=pickle.HIGHEST_PROTOCOL)

velicina = os.path.getsize('datoteka.p')
print("Datoteka zauzima {0} kB".format(velicina/1000))

niz2 = pickle.load(open("datoteka.p", 'rb'))

if (niz2 == niz).all():
  print("Nizovi su identicni!")
```

```
Zauzima 8000.0 kB
Datoteka zauzima 8000.153 kB
Nizovi su identicni!
```

Slika 20: Primjer korištenja Pickle biblioteke za serijalizaciju i deserijalizaciju podataka.

Treba napomenuti da je biblioteka Pickle izuzetno brza - preporučuje se da se veliki skupovi podataka čuvaju kao Pickle datoteke (.p), čak i kada su dostupni na HDD, jer će čitanje Pickle datoteke u kojoj se nalazi serijaliziran skup podataka (npr. slike) biti znatno brže od čitanja skupa podataka element po element (npr. sliku po sliku). Detaljna dokumentacija Pickle biblioteke se može naći na linku

https://wiki.python.org/moin/UsingPickle

3 Zadaci za rad u laboratoriji

Predviđeno je da se svi zadaci u nastavku rade u sklopu Jupyter Notebook okruženja. Svaki podzadatak treba biti zasebna Jupyter ćelija.

Zadatak 1 - Obrada podataka kroz Pandas

Uz vježbu ste dobili datoteku *izvjestaj.csv*. Potrebno je, koristeći Pandas:

a) Učitati datoteku kao Pandas `DataFrame`, te prikazati prvih 5 i posljednjih 10 unosa u istoj;

b) Ispisati samo podatke vezane za redovne parcijalne ispite (kolone *Ispit1* i *Ispit2*);

c) Ispisati sve studente koji su izgubili prisustvo;

d) Ispisati indeks, ukupne bodove, i ocjenu, za sve studente koji su upisali ocjenu 8 ili više;

e) Sve nedostajuće vrijednosti (označene sa simbolom /) zamijeniti sa vrijednošću `np.nan`;

f) Iz skupa podataka odbaciti sve studente koji nemaju upisanu ocjenu;

g) Kreirati novu kolonu *Ispit1_final* u koju ćete za svakog studenta upisati onaj rezultat ispita koji je bolji (na primjer, ako je student bolje uradio popravni ispit, onda tu pišete bodove sa popravnog, a u protivnom sa redovnog);

h) Ponoviti postupak za *Ispit2*;

i) Odbaciti četiri stare kolone za ispite;

j) Skup podataka sačuvati kao CSV datoteku pod nazivom *izvjestaj_modificirano.csv*. Koristiti znak tačka-zarez kao separator;

k) Skup podataka sačuvati kao Pickle datoteku pod nazivom *izvjestaj_modificirano_pickle.p*.

Zadatak 2 - Normalizacija podataka pomoću Sklearn

a) Učitati datoteku *izvjestaj.csv* kao Pandas `DataFrame`;

b) Za kolone koje predstavljaju redovne ispite, izvršiti zamjenu nedostajućih vrijednosti strategijom *median*;

c) Za kolone koje predstavljaju popravne ispite, izvršiti zamjenu nedostajućih vrijednosti pomoću strategije *mean*;

d) Izvršiti Z-score normalizaciju vrijednosti za redovne parcijalne ispite;

e) Izvršiti MinMax normalizaciju vrijednosti za popravne parcijalne ispite;

f) Sve ostale nedostajuće vrijednosti u skupu podataka zamijeniti sa nulama;

g) Izdvojiti kolonu *Ocjena* u posebnu varijablu, na način da sada u originalnom skupu podataka ta kolona više ne postoji;

h) Konvertovati obje varijable (originalni skup podataka bez kolone *Ocjena*, kao i posebnu kolonu *Ocjena*) u NumPy nizove;

i) Ukoliko pretpostavimo da NumPy niz sa ocjenama predstavlja labele, a niz sa ostalim kolonama atribute (značajke), izvršiti podjelu na trening i testni skup podataka, pri čemu za testni skup treba uzeti 20% podataka.

Zadatak 3 - Klasični algoritam mašinskog učenja

U ovom zadatku, upoznat ćemo se sa primjenom Sklearn biblioteke u klasičnim algoritimma mašinskog učenja. Konkretno, radit će se klasifikacija cvijeta iris na tri različite vrste (prikazane na slici 21), i to:

- Setosa;
- Versicolour;
- Virginica.

Iris setosa | Iris versicolor | Iris virginica

Slika 21: Tri različite vrste cvijeta iris.

Algoritam će prvo na osnovu postojećeg skupa podataka, koji sadrži značajke i labele, naučiti kako koja značajka djeluje samu vrstu cvijeta, te će biti u stanju da za novi (neviđeni) podatak pretpostavi tačnu klasu kojoj cvijet pripada.

a) Sam skup podataka dolazi spreman uz Sklearn, te ga je potrebno učitati:

```
from sklearn.datasets import load_iris

iris = load_iris()
X = iris.data
y = iris.target

feature_names = iris.feature_names
target_names = iris.target_names

print("Nazivi znacajki:", feature_names)
print("Nazivi labela:", target_names)
print("\nPrvih 5 redova X:\n", X[:5])
```

Pokrenite ovaj isječak koda. Analizirajte kod, kao i izlaz koji isti proizvodi.

b) Sljedeći korak jeste dijeljenje skupa podataka na dva dijela - trening i testni skup. Trening skup se koristi kako bi algoritam naučio uzorke ponašanja, dok testni skup predstavlja nove podatke, na osnovu kojih se vrši evaluacija algoritma (to jeste evaluacija naučenog). Dijeljenje skupa podataka se vrši na ranije opisani način:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size = 0.3, random_state = 1
)

print(X_train.shape)
print(X_test.shape)
```

```
9
10 print(y_train.shape)
11 print(y_test.shape)
```

Analizirajte ovaj isječak koda. Koja funkcija je korištena za podjelu skupa podataka? Koje parametre prima ova funkcija (pomoć: pogledati dokumentaciju funkcije). Koji je oblik novonastalih skupova? Da li je ovo veliki skup podataka? Zašto jeste/nije?

c) Sada je vrijeme da se primijeni odgovarajući algoritam. U ovom primjeru, koristit ćemo k-NN (eng. *k-nearest neighbours*). Ovaj algoritam je izuzetno jednostavan, te klasifikaciju vrši na način da svaki podatak mapira kao n-dimenzionalnu tačku. Klasa novog podatka se zatim određuje na osnovu k najbližih tačaka u tom prostoru. Uzima se ona klasa koja se pojavljuje najčešće u k najbližih susjeda. Ovaj algoritam je već implementiran u sklopu Sklearn, pa se može jednostavno pozvati:

```
1 from sklearn.neighbors import KNeighborsClassifier
2 from sklearn import metrics
3
4 classifier_knn = KNeighborsClassifier(n_neighbors = 3)
5
6 classifier_knn.fit(X_train, y_train)
```

Analizirajte isječak koda iznad. Na koji način se učitava k-NN? Koja je vrijednost k?

d) Konačno, potrebno je algoritam primijeniti nad testnim skupom, kako bi se odredila uspješnost. U tu svrhu, koristi se isječak koda kao što slijedi:

```
1 y_pred = classifier_knn.predict(X_test)
2
3 print("Tacnost:", metrics.accuracy_score(y_test, y_pred))
```

Koja je tačnost ovog algoritma? Da li je ovo zadovoljavajuća tačnost? Da li je ovakav algoritam primjenjiv u stvarnom svijetu?

e) Ponovite ovaj proces za nekoliko različitih vrijednosti k. Da li se rezultat mijenja? Ako da, da li su drastične promjene, i koja je 'optimalna' vrijednost k? Šta bi se desilo ukoliko bismo postavili da je vrijednost k jednaka broju uzoraka u testnom skupu podataka?

Zadatak 4 - Duboko učenje

Duboko učenje ćemo demonstrirati na primjeru klasifikacije rukom pisanih cifara. Koristit ćemo MNIST skup podataka[5], koji sadrži 70000 slika u nijansama sive boje (eng. *grayscale*) dimenzija 28×28 podijeljenih u 10 različitih klasa (za svaku cifru po jedna). Nekoliko instanci iz ovog skupa podataka je prikazano na slici 22.

Slika 22: MNIST skup podataka.

a) Učitavanje podataka je prvi korak. Keras ovaj skup podataka ima spreman, te je on unaprijed podijeljen na trening i test skup. Učitavanje se može uraditi pomoću sljedećeg isječka koda:

```
from keras.datasets import mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

Koliko slika se nalazi u testnom skupu podataka, a koliko u trening skupu? Koji je oblik (*shape*) ovih varijabli? Koliko bajta zauzimaju u memoriji?

b) Kako bi se ove slike mogle poslati u neuralnu mrežu, potrebno je iste prvo pretvoriti u brojeve u pokretnom zarezu, te ih svesti na opseg između 0 i 1. Obzirom da pikseli *grayscale* slika uzimaju vrijednosti između 0 i 255, svođenje na traženi opseg se može uraditi prostim dijeljenjm sa 255. Sljedeći isječak koda demonstrira ovaj korak:

```
train_images = x_train.reshape((x_train.shape[0], 28 * 28))
train_images = train_images.astype('float32') / 255

test_images = x_test.reshape((x_test.shape[0], 28*28))
test_images = test_images.astype('floate32') / 255
```

Šta radi naredba `reshape`, a šta `astype`? Koliko bajta u memoriji sada zauzimaju slike za testiranje i trening?

c) Nakon što su podaci spremni, može se napraviti jednostavan model duboke neuralne mreže. U tu svrhu, koristi se Keras. Prije svega je potrebno kreirati sam model, nakon čega se na isti dodaju slojevi različitog tipa:

```
from keras import models
from keras import layers

network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
network.add(layers.Dense(10, activation='softmax'))
```

O detaljima kreiranja modela, različitim vrstama slojeva, aktivacijskih funkcija, i dr. će biti govora u narednim laboratorijskim vježbama, kao i na predavanjima. Današnji primjer služi za upoznavanje sa Keras API-jem kao i sa generalnim tokom kreiranja modela dubokog učenja. Model koji smo upravo kreirali ima dva sloja - to je potpuno dovoljno za ovaj jednostavan zadatak, međutim u praksi modeli znaju imati dosta više slojeva - tada se kaže da su modeli duboki[6]. Sloj se može zamisliti kao filter za podatke. Podaci uđu u sloj, te se na izlazu

[5] http://yann.lecun.com/exdb/mnist/

[6] Duboko učenje je naziv dobilo upravo zbog ove činjenice.

dobijaju podaci u nešto korisnijoj formi. Slojevi vrše ekstrakciju reprezentacije iz podataka koji se dovedu na njihov ulaz - pri tome se uvijek nadamo da se tu radi o korisnim reprezentacijama. Većina dubokog učenja se sastoji od nadovezivanja različitih jednostavnih slojeva koji će progresivno raditi filtriranje podataka (*data distilation*). Model se može zamisliti kao sito za podatke - sito napravljeno od više filtera/slojeva. Slojevi koji se koriste u ovom modelu su *dense* slojevi, koji se još nazivaju i potpuno povezani slojevi. Pri tome, drugi (posljednji) sloj je *softmax* sloj sa 10 neurona, što znači da će on vratiti niz od 10 vjerovatnoća (čija će suma biti 1). Svaka vrijednost u tom nizu označava vjerovatnoću kojom trenutna slika pripada odgovarajućoj cifri. Na primjer, ukoliko je na prvom mjestu ovog izlaznog niza broj 0.35, onda je šansa da se radi o cifri nula jednaka 35%. Pokušajte istražiti Keras dokumentaciju, te se upoznati sa *dense* slojem. Koje sve parametre prima ovaj sloj? Na koji način je on povezan sa narednim slojem, te zašto za ove slojeve kažemo da su potpuno povezani?

d) Kada je model definisan, on se mora pripremiti za treniranje. Kako bi to bilo moguće, model je prvo potrebno kompajlirati. Kompajliranje modela se sastoji od odabira tri parametra:

- Funkcije gubitka (eng. *loss function*) - Označava kako neuralna mreža može mjeriti svoje performanse nad trening skupom, te se na osnovu nje prilagođava prilikom procesa treniranja;
- Optimizator (eng. *optimizer*) - Mehanizam na osnovu kojeg se neuralna mreža aktualizira;
- Metrike koje treba pratiti prilikom treniranja i testiranja - U ovom slučaju nas zanima samo tačnost (eng. *accuracy*). Za različite zadatke dubokog učenja su razvijene i različite metrike uspjeha.

Kompajliranje modela se vrši pomoću sljedećeg isječka koda:

```
network.compile(optimizer='rmsprop',
                loss='categorical_crossentropy',
                metrics=['accuracy'])
```

e) U prethodnom koraku smo rekli da je izlaz iz neuralne mreže zapravo niz od 10 elemenata. Shodno tome, moramo labele prilagoditi izlazu, na način da izvršimo *one-hot* kodiranje istih. Kodiranje podrazumijeva pretvaranje labele iz konkretne vrijednosti u niz od n elemenata gdje su svi elementi nula, osim elementa na indeksu koji odgovara tačnoj klasi. Primjer *one-hot* kodiranja je dat u nastavku:

$$3 \rightarrow [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]$$

Kako bi se ovo postiglo u Python-u, može se koristiti gotova funkcija koju pruža Keras, kao što je prikazano u nastavku:

```
from keras.utils import to_categorical

train_labels = to_categorical(y_train)
test_labels = to_categorical(y_test)
```

Pomoću Keras dokumentacije analizirajte funkciju `to_categorical`.

f) Sada je sve spremno da se mreža trenira. Treniranje je izuzetno jednostavno, te se obavlja pomoću jedne linije koda:

```
network.fit(train_images, train_labels, epochs=5, batch_size=128)
```

Pomoću Keras dokumentacije pronađite značenje parametara `epochs` i `batch_size`.

g) Nakon što se treniranje završi, može se izvršiti evaluacija nad testnim skupom podataka:

```
test_loss, test_acc = network.evaluate(test_images, test_labels)
print('Tacnost za testni skup:', test_acc)
```

Kolika je tačnost nad testnim skupom podataka? Da li se drastično razlikuje od tačnosti nad trening skupom? Da li je ovo zadovoljavajući rezultat? Zašto jeste/nije?