Analiza skupa podataka Anime Recommendations

Seminarski rad u okviru kursa
Istraživanje podataka
Matematički fakultet

Nikola Ajzenhamer

19. juni 2017.

Sadržaj

1 Uvod

„Anime Recommendations" skup podataka sadrži informacije o korisničkim rejtinzima anime serijala. Preciznije, skup sadrži 73516 korisničkih rejtinga o 12294 anime serijala. Ovaj skup podataka je nastao tako što je svaki korisnik mogao da doda anime u svoju listu pogledanih anime serijala, a zatim da mu dâ rejting. Skup podataka se može pronaći na adresi https://www.kaggle.com/CooperUnion/anime-recommendations-database.

2 Analiza i pretprocesiranje podataka

U ovoj sekciji biće izvršena analiza podataka, zatim njihovo pretprocesiranje i vizuelizacija. Za ovaj zadatak će biti korišćen programski jezik Python.

2.1 Opis skupa podataka

Skup podataka „Anime Recommendations" čine dve datoteke: anime.csv i rating.csv. Njihove kolone i opisi dati su tabelama 1 i 2, redom.

Tabela 1: Opis datoteke anime.csv. Nazivi kolona su dati levo, a njihovi opisi desno.

anime_id	jedinstveni identifikator svakog anime serijala
name	pun naziv anime serijala
genre	lista žanrova (odvojenih zapetom) kojima anime serijal pripada
type	tip serijala
episodes	broj epizoda
rating	prosečni rejting (od 10)
members	broj članova koji pripadaju „grupi" datog anime serijala

Obratimo sada pažnju na sadržaj ovih datoteka. Obe datoteke ćemo analizirati na isti način: prvo ćemo pogledati prvih nekoliko redova da bismo stekli ideju o tome kako

Tabela 2: Opis datoteke rating.csv. Nazivi kolona su dati levo, a njihovi opisi desno.

user_id	nasumično generisan identifikator korisnika
anime_id	identifikator anime serijala za koji je dati korisnik dao rejting
rating	rejting (od 10) koji je dati korisnik dao (-1 ako je korisnik gledao serijal, ali nije dao rejting)

podaci izgledaju, zatim ćemo videti neke statističke mere poput kvartila, a zatim ćemo za svaku kolonu izdvojiti broj različitih vrednosti.

Pozabavimo se datotekom anime.csv. U nastavku sledi programski kôd opisane analize, a zatim i rezultat izvršavanja uz dodatne komentare.

Programski kôd

```
print('***********', '*Anime.csv*', '***********', '\n', sep='\n')
df_anime = pandas.read_csv('../podaci/anime.csv')
print(df_anime.head(), '\n')
print(df_anime.count(), '\n')
print(df_anime.describe(), '\n')
for column in df_anime.columns:
    print('Count values in column ' + column, '\n')
    print(df_anime[column].value_counts(dropna=False))
```

Rezultat izvršavanja

```
***********
*Anime.csv*
***********


   anime_id                              name  \
0     32281                     Kimi no Na wa.
1      5114  Fullmetal Alchemist: Brotherhood
2     28977                          Gintama°
3      9253                       Steins;Gate
4      9969                          Gintama'

                                              genre   type episodes  rating  \
0              Drama, Romance, School, Supernatural  Movie        1    9.37
1  Action, Adventure, Drama, Fantasy, Magic, Mili...     TV       64    9.26
2  Action, Comedy, Historical, Parody, Samurai, S...     TV       51    9.25
3                                  Sci-Fi, Thriller     TV       24    9.17
4  Action, Comedy, Historical, Parody, Samurai, S...     TV       51    9.16

   members
0   200630
1   793665
2   114262
3   673572
4   151266

anime_id    12294
name        12294
genre       12232
type        12269
episodes    12294
```

```
rating      12064
members     12294
dtype: int64
```

```
           anime_id        rating       members
count  12294.000000  12064.000000  1.229400e+04
mean   14058.221653      6.473902  1.807134e+04
std    11455.294701      1.026746  5.482068e+04
min        1.000000      1.670000  5.000000e+00
25%     3484.250000      5.880000  2.250000e+02
50%    10260.500000      6.570000  1.550000e+03
75%    24794.500000      7.180000  9.437000e+03
max    34527.000000     10.000000  1.013917e+06
```

Redova u ovoj datoteci ima 12294. Iz ovih rezultata vidimo da svaki anime serijal ima identifikator, naziv, broj epizoda i broj članova. Ipak, nemaju svi serijali žanr, tip i rejting. Dalje, vidimo da je raspon ocena u intervalu $(1.67, 10.00)$, kao i da je prosek ocena (6.47) blizak medijani ocena (6.57). Raspon korisnika koji pripadaju „grupama" je između 5 i nešto više od milion. Ovde je nešto iskrivljena slika s obzirom da je prosek članova oko 18 hiljada, a medijana oko 15.5 hiljada. Analizirajmo sada broj jedinstvenih vrednosti za kolone[1].

```
Count values in column anime_id

2047     1
8153     1
3403     1
34122    1
7497     1
        ..
597      1
2644     1
12883    1
21509    1
30177    1
Name: anime_id, dtype: int64


Count values in column name

Saru Kani Gassen                                                    2
Shi Wan Ge Leng Xiaohua                                             2
Naruto: Shippuuden                                                  1
Sekai Douwa Anime Zenshuu                                           1
Kaba Enchou no Doubutsuen Nikki                                     1
                                                                   ..
Mirai Arise                                                         1
Mahouka Koukou no Rettousei: Yoku Wakaru Mahouka! Special           1
Onigamiden                                                          1
2x1                                                                 1
Sagaken wo Meguru Animation                                         1
Name: name, dtype: int64
```

[1]Izlaz je skraćen u nekim slučajevima radi smanjenja iskorišćenog prostora u dokumentu.

```
Count values in column genre

Hentai                                                                823
Comedy                                                                523
Music                                                                 301
Kids                                                                  199
Comedy, Slice of Life                                                 179
                                                                     ...
Action, Comedy, Martial Arts, Shounen, Supernatural                     1
Hentai, Historical, Mystery                                             1
Action, Comedy, Kids, Magic, Mecha, Sci-Fi, Shounen, Super Power        1
Drama, Fantasy, Mystery                                                 1
Adventure, Drama, Supernatural                                          1
Name: genre, dtype: int64
Count values in column type

TV         3787
OVA        3311
Movie      2348
Special    1676
ONA         659
Music       488
NaN          25
Name: type, dtype: int64


Count values in column episodes

1          5677
2          1076
12          816
13          572
26          514
           ...
1274          1
694           1
141           1
191           1
147           1
Name: episodes, dtype: int64


Count values in column rating

NaN      230
 6.00    141
 7.00     99
 6.50     90
 6.25     84
        ...
 3.35      1
 3.14      1
 3.28      1
 8.66      1
 9.15      1
```

```
Name: rating, dtype: int64


Count values in column members

60        36
72        36
74        33
62        32
67        31
          ..
12153      1
1684       1
16251      1
1916       1
83438      1
Name: members, dtype: int64
```

S obzirom da su brojevi pojavljivanja sortirani opadajuće, možemo zaključiti sledeće: svaki anime ima svoj jedinstveni identifikacioni broj i naziv (izuzetak u nazivu imaju dva anime serijala, „Saru Kani Gassen" i „Shi Wan Ge Leng Xiaohua", ali pretpostavljamo da ovo neće imati uticaja na dalju analizu), te ove kolone (anime_id i name) možemo eliminisati iz dalje analize. Kolonu genre je teže analizirati na ovaj način, te ćemo se njom detaljnije pozabaviti kasnije.

Ponovimo ovu analizu nad datotekom rating.csv.

Programski kôd

```
print('************', '*Rating.csv*', '************', sep='\n')
df_rating = pandas.read_csv('../podaci/rating.csv')
print(df_rating.head(), '\n')
print(df_rating.count(), '\n')
print(df_rating.describe(), '\n')
for column in df_rating.columns:
    print('Count values in column ' + column, '\n')
    print(df_rating[column].value_counts(dropna=False))
```

Rezultat izvršavanja

```
************
*Rating.csv*
************
   user_id  anime_id  rating
0        1        20      -1
1        1        24      -1
2        1        79      -1
3        1       226      -1
4        1       241      -1

user_id     7813737
anime_id    7813737
rating      7813737
dtype: int64

            user_id      anime_id        rating
```

```
count  7.813737e+06  7.813737e+06  7.813737e+06
mean   3.672796e+04  8.909072e+03  6.144030e+00
std    2.099795e+04  8.883950e+03  3.727800e+00
min    1.000000e+00  1.000000e+00 -1.000000e+00
25%    1.897400e+04  1.240000e+03  6.000000e+00
50%    3.679100e+04  6.213000e+03  7.000000e+00
75%    5.475700e+04  1.409300e+04  9.000000e+00
max    7.351600e+04  3.451900e+04  1.000000e+01
```

U opisu datoteke (tabela 2) videli smo da vrednost -1 za rejting znači da korisnik nije dao rejting nakon gledanja serijala. Ipak, ono što vidimo ovde jeste da je datoteka konzistentna, odnosno da nema nedostajućih vrednosti. Pogledajmo broj vrednosti za kolone.

```
Count values in column user_id

48766    10227
42635     3747
53698     2905
57620     2702
59643     2633
          ...
60103        1
72442        1
15070        1
33749        1
5846         1
Name: user_id, dtype: int64


Count values in column anime_id

1535     39340
11757    30583
16498    29584
1575     27718
         ...
29716        1
20145        1
26229        1
21019        1
21509        1
Name: anime_id, dtype: int64


Count values in column rating

 8     1646019
-1     1476496
 7     1375287
 9     1254096
 10     955715
 6      637775
 5      282806
 4      104291
 3       41453
```

```
 2       23150
 1       16649
Name: rating, dtype: int64
```

Odavde vidimo da postoje korisnici koji su glasali samo za jedan anime. Zanimljivo je videti da je korisnik sa najvećim brojem rejtinga ocenio oko tri puta više anime serijala u odnosu na sledećeg korisnika u sortiranom prikazu. Takođe možemo videti da broj parova (korisnik, serijal) koji nemaju rejting (tj. čiji je rejting jednak -1) čini malo više od petine ove datoteke, kao i da su više ocene češće od nižih.

2.2 Pretprocesiranje podataka

Da bismo pripremili podatke za dalju analizu, trebalo bi izvršiti određene transformacije nad datim podacima. U ovom delu ćemo opisati neke funkcije za pretprocesiranje podataka.

2.2.1 Pretprocesiranje podataka o pojedinačnim rejtinzima

Počnimo od datoteke rating.csv. Već smo videli da su ocene od 1 do 10, pri čemu broj -1 označava da korisnik nije dao ocenu. Da bismo ovo vizualno prikazali, neophodno je da pretvorimo sve vrednosti -1 u *null* (odnosno, `None`, u programskom jeziku Python), kako nam ne bi uticale na prikaz. Definišemo narednu funkciju:

```
def get_rating_csv_with_nulls(sampled=False):
    start_before = time.time()
    df_rating = pandas.read_csv('../podaci/rating.csv')
    start_after = time.time()

    if sampled:
        df_rating = df_rating.sample(frac=0.6)

    invalid_values = 0
    for i, row in df_rating.iterrows():
        rating = row['rating']
        if rating == -1:
            df_rating.set_value(i, 'rating', None)
        elif rating < 0 or rating > 10:
            df_rating.set_value(i, 'rating', None)
            invalid_values += 1

    filename = '../podaci/rating_nulls'
    if sampled:
        filename += '_sampled.csv'
    else:
        filename += '.csv'

    with open(filename, 'w', encoding='utf-8') as csv_file:
        csv = df_rating.to_csv(index=False)
        csv_file.write(csv)

    end = time.time()
    print('[get_rating_csv_with_nulls] Execution time of function (in
    ↪ seconds): %.2f' % (end - start_before))
    print('[get_rating_csv_with_nulls] Execution time of data loading (
    ↪ in seconds): %.2f' % (start_after - start_before))

    return df_rating, invalid_values
```

Zadatak funkcije `get_rating_csv_with_nulls` jeste učitavanje skupa podataka iz datoteke rating.csv, a zatim zamenjivanje vrednosti -1 vrednošću `None`. Opcioni argument `sampled` određuje da li će se uzeti jednostavan nasumičan uzorak od 60%. Takođe, omogućen je prikaz vremena izvršavanja (u neizolovanom okruženju) celokupne funkcije, kao i vreme koje je utrošeno za učitavanje podataka iz datoteke. Jedno pozivanje ove funkcije je ispisalo naredne vrednosti:

```
[get_rating_csv_with_nulls] Execution time of function (in seconds): 1283.18
[get_rating_csv_with_nulls] Execution time of data loading (in seconds): 5.24
```

2.2.2 Pretprocesiranje podataka o anime serijalima

U analizi skupa podataka iz datoteke anime.csv primetili smo da postoje kolone koje imaju nedostajuće vrednosti. Pošto će nam te kolone biti ključne u daljem istraživanju, trebalo bi eliminisati sve redove koje imaju nedostajuće vrednosti. Definišemo narednu funkciju koja upravo to radi:

```
def remove_missing_values():
    print('[remove_missing_values] Function enter', '\n\n')
    df_anime = pandas.read_csv('../podaci/anime.csv')

    df_anime_tmp = df_anime[pandas.notnull(df_anime['genre'])]
    df_anime_tmp = df_anime_tmp[pandas.notnull(df_anime_tmp['type'])]
    df_anime_tmp = df_anime_tmp[pandas.notnull(df_anime_tmp['rating'])]

    print('[remove_missing_values] Function exit', '\n\n')
    return df_anime_tmp
```

Neki metodi koje ćemo koristiti zahtevaju da podaci budu numeričkog tipa. Stoga, kolona type nam može praviti problem. Zato definišemo narednu funkciju koja transformiše ovu kolonu u novi skup numeričkih kolona koristeći *dummy coding*:

```
def transform_types_into_numerical(df_anime):
    print('[transform_types_into_numerical] Function enter', '\n\n')

    all_types = ['TV', 'OVA', 'Movie', 'Special', 'ONA', 'Music']
    categories = ['type1', 'type2', 'type3', 'type4', 'type5']
    num_of_categories = 5

    # Auxiliary function which puts 1 in the column if the anime belongs
    ↪  to that genre, and 0 otherwise
    def create_column_data(row):
        data = []
        k = all_types.index(row['type'])
        for i in range(0, num_of_categories):
            if i == k:
                data.append(1)
            else:
                data.append(0)
        return pandas.Series(data)

    # Fill the columns with new data about genres
    df_anime[categories] = df_anime.apply(create_column_data, axis=1)

    print('[transform_types_into_numerical] Function exit', '\n\n')
    return df_anime
```

Radi udobnosti, definišemo funkciju koja podrazumevano obavlja obe akcije nad polaznim skupom podataka:

```
def initial_preprocessing(arg_remove_missing_values=True,
    ↪ arg_transform_types=True):
    print('[initial_preprocessing] Function enter', '\n\n')
    df_anime = pandas.read_csv('../podaci/anime.csv')
    if arg_remove_missing_values:
        df_anime = remove_missing_values(df_anime)
    if arg_transform_types:
        df_anime = transform_types_into_numerical(df_anime)

    print('[initial_preprocessing] Function exit', '\n\n')
    return df_anime
```

Već smo diskutovali o problemu vezanom za kolonu genre. Podaci u ovoj koloni su predstavljeni tipom string, pri čemu je svaka vrednost oblika

$$\texttt{'genre\_1, genre\_2, ..., genre\_n'}$$

Takav složeni podatak je prilično težak za istraživanje, pa ga je neophodno pretvoriti u nešto jednostavnije. Za početak, uradimo sledeću transformaciju: uzmimo opisani string i razbijmo ga u listu pojedinačnih žanrova, odnosno, definišimo funkciju

$$f : \texttt{'genre\_1, genre\_2, ..., genre\_n'} \mapsto [\texttt{'genre\_1'}, \texttt{'genre\_2'}, ..., \texttt{'genre\_n'}]$$

koju ćemo primeniti za svaki red datoteke anime.csv. Dodatno, možemo u hodu konstruisati listu svih žanrova koje postoje. Opisana procedura definisana je narednom funkcijom:

```
def get_all_genres_with_data():
    print('[get_all_genres] Function enter', '\n\n')
    df_anime = remove_missing_values()
    column_genre = df_anime['genre']

    all_genres = []

    for i, genres in column_genre.iteritems():
        if type(genres) is str:
            genres_list = genres.split(',')
            genres_list_trimmed = []
            for genre in genres_list:
                genres_list_trimmed.append(genre.strip())
            df_anime.set_value(i, 'genre', genres_list_trimmed)
            for genre in genres_list_trimmed:
                if genre not in all_genres:
                    all_genres.append(genre)

    print('[get_all_genres] Function exit', '\n\n')
    return all_genres, df_anime
```

Sada, kada smo pronašli sve moguće vrednosti `genre_i`, možemo da konstruišemo nove kolone, `genre_i` (odnosno, nazivi tih kolona odgovaraju nazivima žanrova), pri čemu će vrednost u koloni `genre_i` biti jednaka 1 ukoliko anime serijal u odgovarajućem redu pripada tom žanru, a 0 inače. Implementacija ove procedure data je narednom funkcijom:

```
def generate_anime_genres():
    print('[generate_anime_genres] Function enter', '\n\n')
    all_genres, df_anime = get_all_genres_with_data()

    # Auxiliary function which puts 1 in the column if the anime belongs
    ↪  to that genre, and 0 otherwise
    def create_column_data(row):
        data = []
        for genre in all_genres:
            if genre in row['genre']:
                data.append(1)
            else:
                data.append(0)
        return pandas.Series(data)

    # Fill the columns with new data about genres
    df_anime[all_genres] = df_anime.apply(create_column_data, axis=1)

    # Saving a new DataFrame to a new file
    with open('../podaci/anime_genres.csv', 'w', encoding='utf-8') as
    ↪ csv_file:
        csv = df_anime.to_csv(index=False)
        csv_file.write(csv)

    print('[generate_anime_genres] Function exit', '\n\n')
    return df_anime
```

Takođe, biće nam potrebno da znamo za svaki žanr njegove frekvencije. Naredna funkcija izračunava te vrednosti:

```
def generate_genre_frequencies():
    print('[generate_genre_frequencies] Function enter', '\n\n')
    all_genres = get_all_genres_with_data()[0]
    df_anime = generate_anime_genres()
    df_all_genres_count = pandas.DataFrame(columns=['Genre', 'Count'])

    for genre in all_genres:
        sum_plus = 0
        for value in df_anime[genre]:
            if value == 1:
                sum_plus += 1
        tmp = pandas.DataFrame([[genre, sum_plus]], columns=['Genre', '
    ↪ Count'])
        df_all_genres_count = df_all_genres_count.append(tmp,
    ↪ ignore_index=True)

    df_all_genres_count = df_all_genres_count.sort_values(by='Count',
    ↪ ascending=False)

    print('[generate_genre_frequencies] Function exit', '\n\n')
    return df_all_genres_count
```

Konačno, biće nam neophodno da napravimo skup sličan onom koji se dobija pozivom funkcije `generate_anime_genres`, s tom razlikom da se umesto 1 ili 0 u kolonama `genre_i` nalazi rejting ili `None`, redom. Funkciju definišemo na sledeći način:

```
def generate_anime_genres_ratings():
    print('[generate_anime_genres_ratings] Function enter', '\n\n')
    df_anime = pandas.read_csv('../podaci/anime_genres.csv')

    all_genres = get_all_genres_with_data()[0]
    all_genres_ratings = all_genres
    all_genres_ratings.append('rating')
    df_all_genres_ratings = df_anime[all_genres_ratings].astype(float)

    for i, row in df_all_genres_ratings.iterrows():
        rating = row['rating']
        for genre in all_genres:
            if row[genre] == 1:
                df_all_genres_ratings.set_value(i, genre, rating)
            else:
                df_all_genres_ratings.set_value(i, genre, None)

    with open('../podaci/anime_genres_ratings.csv', 'w', encoding='utf-8
    ↪ ') as csv_file:
        csv = df_all_genres_ratings.to_csv(index=False)
        csv_file.write(csv)

    print('[generate_anime_genres_ratings] Function exit', '\n\n')
    return df_all_genres_ratings
```

Na samom kraju, želimo da eliminišemo sve kolone koje nam nisu neophodne. Na osnovu prethodnih analiza, to su kolone anime_id, name, genre i type. Takođe, ono što se nije videlo isprva jeste da kolona episodes sadrži vrednosti `'Unknown'` za neke redove, te je takve vrednosti potrebno ukloniti. Funkcija se definiše na sledeći način:

```
def additional_preprocessing(df_anime):
    print('[additional_preprocessing] Function enter', '\n\n')
    df_anime_main = df_anime.filter(regex=r'^(?!(anime_id)|(name)|(genre
    ↪ )|(type)).*')
    df_anime_types = df_anime.filter(items=['type1', 'type2', 'type3', '
    ↪ type4', 'type5'])
    df_anime_ret = pandas.concat([df_anime_main, df_anime_types], axis
    ↪ =1, join_axes=[df_anime_main.index])
    df_anime_ret.query('episodes != "Unknown"', inplace=True)
    print('[additional_preprocessing] Function exit', '\n\n')
    return df_anime_ret
```

2.3 Vizualizacija podataka

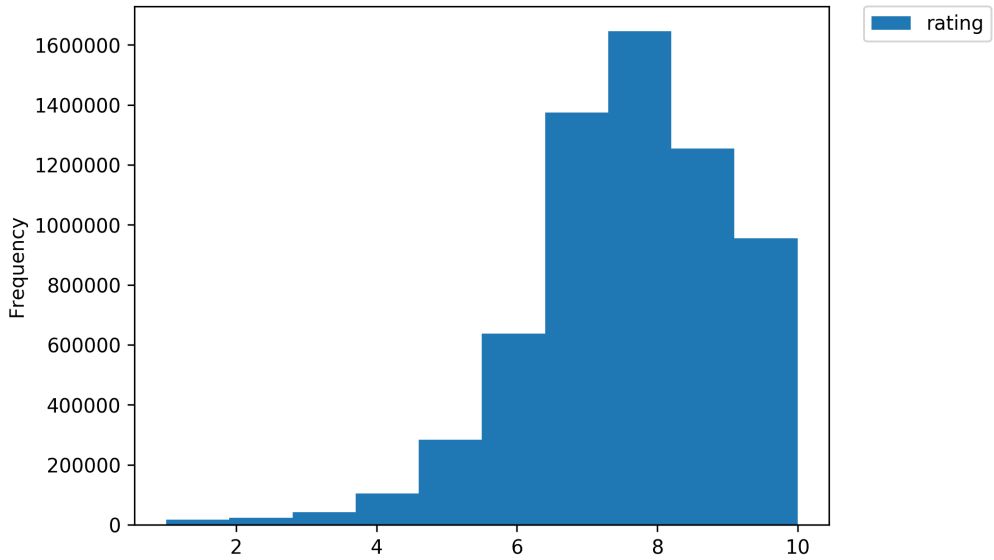
Nakon obavljenog pretprocesiranja podataka, možemo pristupiti vizualizaciji podataka, da bismo odgovorili na neka pitanja. Pitanja koja možemo postaviti su razna, ali mi ćemo se inicijalno fokusirati na:

1. Kako izgleda raspodela pojedinačnih rejtinga?
2. Koje su frekvencije žanrova?
3. Da li postoji veza između žanrova i rejtinga? Da li se svi žanrovi ponašaju isto u odnosu na raspodelu rejtinga (ili možda međusobno)?

Raspodela rejtinga Da bismo stekli vizualnu predstavu o raspodeli ocena, koristićemo histogram. Definisaćemo funkciju koja konstruiše histogram, a zatim ćemo ga prikazati.

```
def generate_histogram_of_rating_values():
    print('[generate_histogram_of_rating_values] Function enter', '\n\n'
    ↪ )

    # df_rating, invalid_values = preprocessing_anime.
    ↪ get_rating_csv_with_nulls()
    df_rating = pandas.read_csv('../podaci/rating_nulls.csv')
    # if invalid_values > 0:
    #    print('[Rating.csv][WARN] There were %d invalid values in data'
    ↪  % invalid_values)

    df_rating['rating'].plot.hist(bins=10)
    lgd = plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
    plt.savefig('../podaci/slike/histogram_anime_ratings.png', dpi=300,
    ↪ bbox_extra_artists=(lgd,), bbox_inches='tight')
    plt.close()

    print('[generate_histogram_of_rating_values] Function exit', '\n\n')
```

Primetimo da ovaj kôd poziva funkciju `get_rating_csv_with_nulls`. Ukoliko smo je prethodno pozvali, onda imamo generisanu datoteku rating_nulls.csv koju možemo direktno da učitamo i time smanjimo vreme izvršavanja.

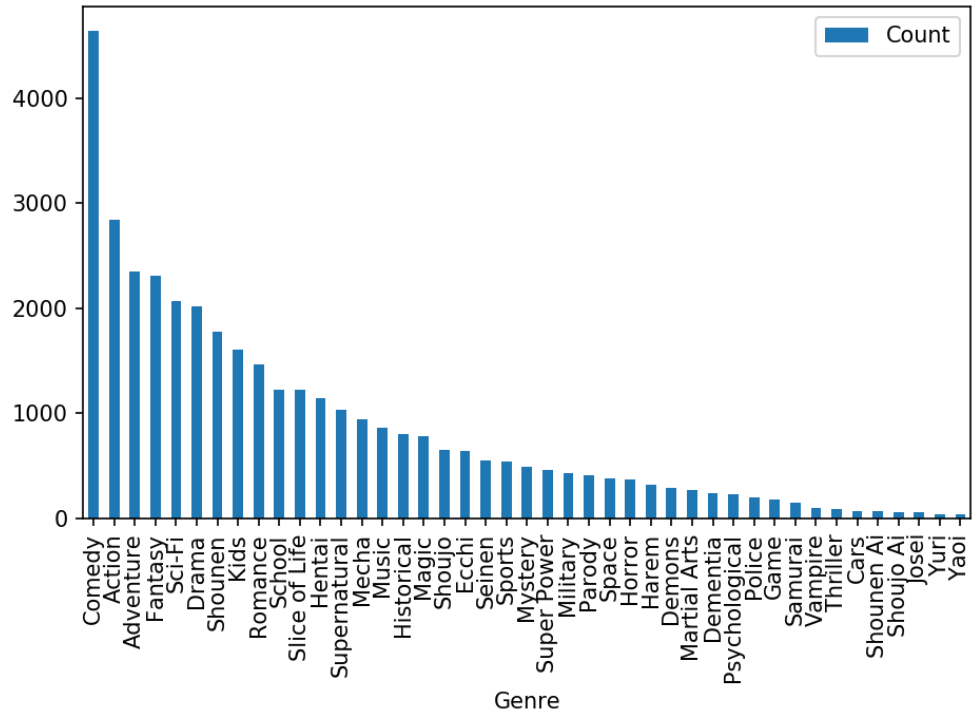


Slika 1: Histogram pojedinačnih rejtinga svih anime serijala.

Sa slike 1 možemo se uveriti u prethodnu analizu – korisnici češće daju više ocene nego niže ocene.

Frekvencije žanrova Može nam biti korisno da vidimo koji žanrovi dominiraju u odnosu na neke druge. Naredna funkcija nam generiše stubičasti dijagram koji nam daje tu informaciju.

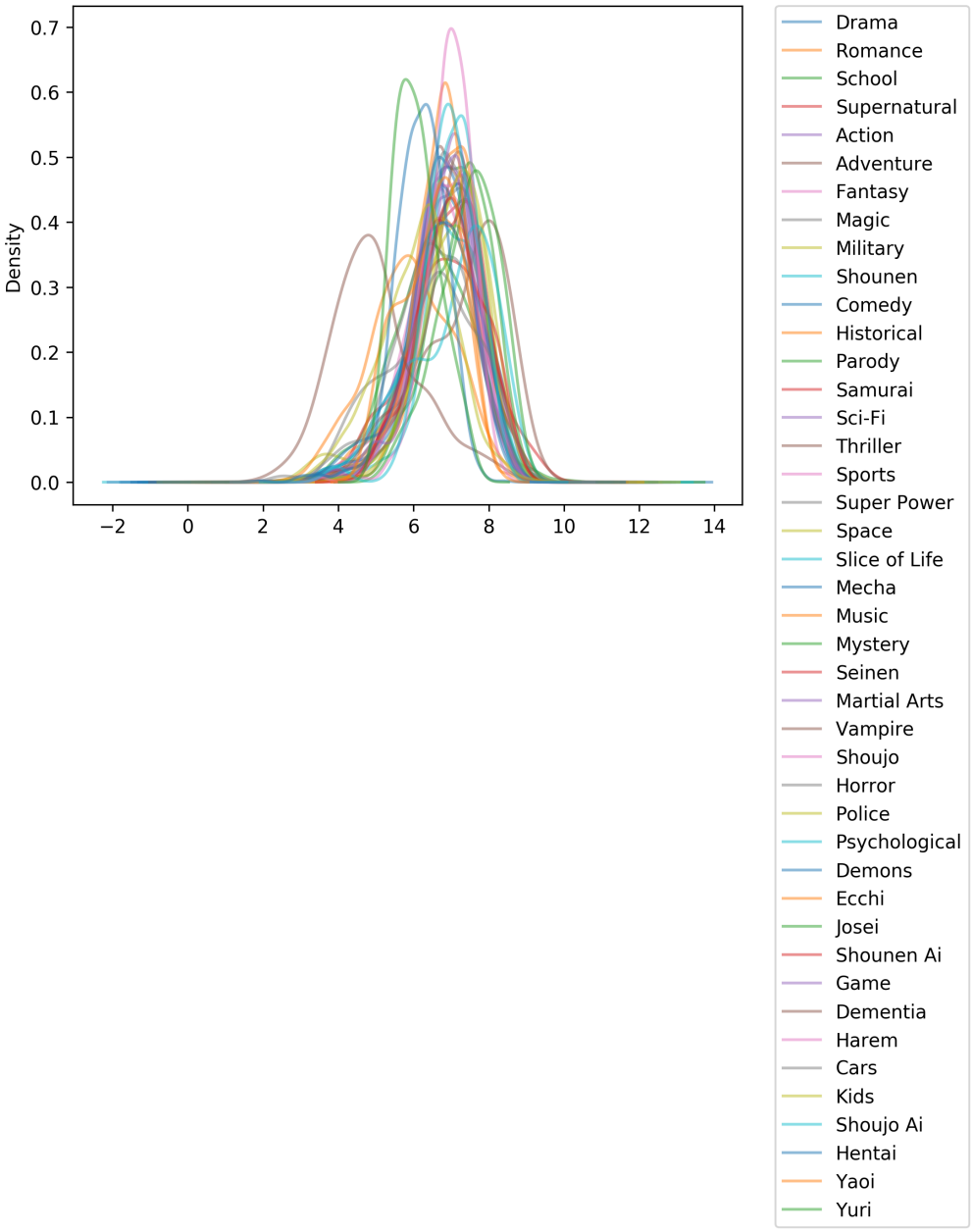
```
def generate_barplot_of_genre_frequencies():
    print('[generate_barplot_of_genre_frequencies] Function enter', '\n\
    ↪ n')

    df_all_genres_count = preprocessing_anime.generate_genre_frequencies
    ↪ ()

    # Plotting the bar plot of genres and their respective (absolute)
    ↪ frequencies
    df_all_genres_count.plot(x='Genre', y='Count', kind='bar')
    plt.tight_layout()
    plt.savefig('../podaci/slike/barplot_genre_count.png', dpi=150)
    plt.close()

    print('[generate_barplot_of_genre_frequencies] Function exit', '\n\n
    ↪ ')
```



Slika 2: Stubičasti dijagram apsolutnih učestalosti žanrova.

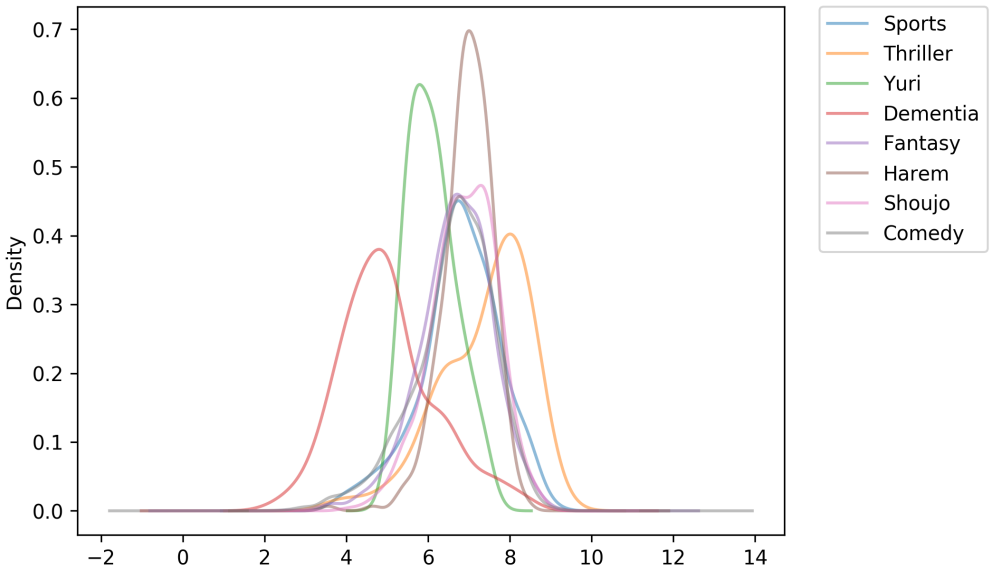
Sa slike 2 možemo videti da je najzastupljeniji žanr „Comedy", dok je najmanje zastupljen žanr „Yaoi". Takođe, vidimo da je razlika između prvog i drugog najzastupljenijeg žanra velika razlika, dok su ostale razlike između susednih žanrova značajno manje.

Veza između žanrova i rejtinga Analiza koju ćemo sada sprovesti predstavlja nešto složeniji proces od prethodne dve analize. Koristićemo *nacrt gustine* (engl. *density plot*) da bismo uporedili kako se ponašaju žanrovi u odnosu na rejting, kao i međusobno. Definišemo narednu funkciju:



Slika 3: Nacrt gustine svih žanrova.

```
def generate_kde_genres_ratings(include_all=True):
    print('[generate_kde_genres_ratings] Function enter', '\n\n')

    # df_all_genres_ratings = preprocessing_anime.
    ↪ generate_anime_genres_ratings()
    df_all_genres_ratings = pandas.read_csv('../podaci/
    ↪ anime_genres_ratings.csv')

    if include_all:
        df_all_genres_ratings[all_genres].plot.kde(alpha=0.5)
        lgd = plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad
    ↪ =0.)
        plt.savefig('../podaci/slike/genre_rating/kde_all_genres.png',
    ↪ dpi=300, bbox_extra_artists=(lgd,), bbox_inches='tight')
        plt.close()
    else:
        selected_genres = ['Sports', 'Thriller', 'Yuri', 'Dementia', '
    ↪ Fantasy', 'Harem', 'Shoujo', 'Comedy']
        for genre in selected_genres:
            df_all_genres_ratings[[genre]].plot.hist(alpha=0.5, bins=10,
    ↪  xlim=(1, 11))
            plt.savefig('../podaci/slike/genre_rating/histogram_' +
    ↪ genre + '_rating_frequency.png', dpi=72)
            plt.close()

        df_all_genres_ratings[selected_genres].plot.kde(alpha=0.5)
        lgd = plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad
    ↪ =0.)
        plt.savefig('../podaci/slike/genre_rating/kde_selected_genres.
    ↪ png', dpi=300, bbox_extra_artists=(lgd,), bbox_inches='tight')

    print('[generate_kde_genres_ratings] Function exit', '\n\n')
```



Slika 4: Nacrt gustine odabranih žanrova.

Argument `include_all` određuje da li će biti iscrtan nacrt gustine za sve žanrove ili samo za (unapred) odabrane žanrove. Na slici 3 prikazan je nacrt koji se dobija pozivom

```
generate_kde_genres_ratings(True)
```

Trebalo bi da bude jasno da je ovaj prikaz težak za interpretaciju. Ipak, možemo uočiti nekoliko interesantnih žanrova, pa ih izdvojiti u poseban grafik. Na slici 4 prikazan je nacrt koji se dobija pozivom

```
generate_kde_genres_ratings(False)
```

Odavde već možemo da izvučemo neke hipoteze. Žanr „Harem" se u velikoj meri ponaša u skladu sa raspodelom rejtinga datoj na slici 1. Takođe, postoje žanrovi koji su međusobno slični po rejtingu, kao što su „Sports", „Comedy" i „Fantasy" (slično se ponaša i „Shoujo" žanr, sa malo boljim uspehom). Žanr „Thriller" ima posebno dobre ocene od ostalih odabranih žanrova. Za razliku od njega, žanrovi „Yuri" i „Dementia" imaju lošije ocene, s tim da je drugi od njih značajno lošije prošao kod publike.

3 Regresiona analiza

Sada kada smo izvršili detaljnu početnu analizu podataka, možemo preći na kreiranje modela koji će nam (nadajmo se) dati odgovore na dodatna pitanja. Jedno od pitanja koje svakako jeste značajno glasi:

- Da li se može predvideti rejting anime serijala na osnovu njegovih karakteristika?

Na ovo pitanje ćemo pokušati da dâmo odgovor regresionom analizom. Kao i do sada, koristićemo programski jezik Python.

Odabir regresionih modela i parametara Za potrebe regresije odlučeno je da se koriste Multilayer Perceptron neuronska mreža i Grebena regresija. Oba modela imaju veliki broj konfiguracija, pa ne bi bilo loše ispitati različite konfiguracije za svaki model. U tu svrhu možemo koristiti unakrsnu validaciju. Takođe, zbog različitosti vrednosnih skala, trebalo bi izvršiti standardizaciju podataka.

Postupak regresione analize Opišimo detaljnije način na koji će biti izvršena regresiona analiza. Opisanom postupku će uslediti programski kôd koji ga implementira. Procedura koju ćemo izvršiti sastoji se od sledećih koraka:

1. **Generisanje ulaznih podataka.** Skup podataka koji ćemo koristiti jeste onaj dobijen pozivom funkcije `generate_anime_genres`. Ukoliko je funkcija bila ranije pozivana, dostupna nam je datoteka anime_genres.csv koju možemo direktno pročitati.

2. **Potpuno pretprocesiranje.** Da bismo eliminisali redundantne redove i kolone, nad ulaznim podacima primenjujemo funkciju `additional_preprocessing`.

3. **Particionisanje podataka.** Kako pokušavamo da predvidimo rejting anime serijala, ciljna promenljiva je određena kolonom rating. Koristićemo sve ostale kolone kao nezavisne promenljive. Potrebno je izvršiti particionisanje, pri čemu je odabrana veličina od 70% za trening skup.

4. **Standardizacija podataka.** Napomenuli smo da vrednosti kolona dolaze iz različitih vrednosnih skala. Da bismo izbegli preveliki uticaj nekih kolona, vršimo standardizaciju pomoću `StandardScaler`-a.

5. **Odabir konfiguracija algoritama za unakrsnu validaciju.** Za svaki regresioni algoritam treba odabrati parametre. Odabrani su sledeći parametri:

(a) neuronska mreža (MLPRegressor):
- 'solver': ['sgd'],
- 'learning_rate': ['constant', 'adaptive'],
- 'learning_rate_init': [0.01, 0.001],
- 'hidden_layer_sizes': [(10,), (100,)],
- 'momentum': [0],
- 'activation': ['identity', 'logistic', 'tanh']

(b) grebena regresija (Ridge):
- 'alpha': [0.001, 0.01, 0.2, 0.5, 1],
- 'fit_intercept': [True, False],
- 'solver': ['auto', 'sag'],
- 'tol': [1e-2, 1e-3, 1e-4, 1e-5]

6. **Odabir ocena greške za regresiju.** Za ocene greške odabrane su srednjekvadratna greška (`mean_squared_error`)

$$\mathrm{MSE}(\texttt{y\_true}, \texttt{y\_pred}) = \frac{1}{n}\sum_{i=0}^{n-1}(\texttt{y\_true}_i - \texttt{y\_pred}_i)^2$$

i koeficijent determinacije (`r2_score`)

$$R^2(\texttt{y\_true}, \texttt{y\_pred}) = 1 - \frac{\sum_{i=0}^{n-1}(\texttt{y\_true}_i - \texttt{y\_pred}_i)^2}{\sum_{i=0}^{n-1}(\texttt{y\_true}_i - \texttt{avg(y)})^2},$$

gde je $\texttt{y\_pred}_i$ predviđena vrednost i-tog uzorka, $\texttt{y\_true}_i$ stvarna vrednost i-tog uzorka, `avg(y)` uzoračka sredina.

7. **Istraživanje modela unakrsnom validacijom.** Ovaj korak predstavlja glavni korak u istraživanju. On se sastoji od nekoliko manjih koraka.
   (a) Izvršiti unakrsnu validaciju za odgovarajući model i oceni greške. Ovaj korak možemo obmotati u izračunavanje vremena izvršavanja programskog kôda.
   (b) Izračunati ocene greške za svaku konfiguraciju.
   (c) Prikazati najbolju konfiguraciju i izračunati ocene greške za trening i test skup.
   (d) Sačuvati rezultate.

Programski kôd Opisani postupak se preslikava u naredni programski kôd.

```
import pandas
import preprocessing_anime as pp
import time
from sklearn import preprocessing
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.neural_network import MLPRegressor
from sklearn.linear_model import Ridge

# df_anime = pp.additional_preprocessing(pp.generate_anime_genres())
df_anime = pp.additional_preprocessing(pandas.read_csv('../podaci/
    ↪ anime_genres.csv'))

features = df_anime.columns.drop('rating').tolist()
```

```
x = df_anime[features]
y = df_anime['rating']

x_train_original, x_test_original, y_train, y_test = train_test_split(x,
    ↪  y, train_size=0.7)

scaler = preprocessing.StandardScaler().fit(x_train_original)
x_train = pandas.DataFrame(scaler.transform(x_train_original))
x_train.columns = features
x_test = pandas.DataFrame(scaler.transform(x_test_original))
x_test.columns = features

df_model = pandas.DataFrame()

parameters_grid = [[{
    # Parameters for MLPRegressor()
    'solver': ['sgd'],
    'learning_rate': ['constant', 'adaptive'],
    'learning_rate_init': [0.01, 0.001],
    'hidden_layer_sizes': [(10,), (100,)],
    'momentum': [0],
    'activation': ['identity', 'logistic', 'tanh']
}], [{
    # Parameters for Ridge()
    'alpha': [0.001, 0.01, 0.2, 0.5, 1],
    'fit_intercept': [True, False],
    'solver': ['auto', 'sag'],
    'tol': [1e-2, 1e-3, 1e-4, 1e-5]
}]]
scores = ['neg_mean_squared_error', 'r2']
regressor_grid = [MLPRegressor(), Ridge()]
models = ['MLPRegressor', 'Ridge']

for i in range(0, 2):
    for score in scores:
        start = time.time()
        reg = GridSearchCV(regressor_grid[i], parameters_grid[i], cv=5,
    ↪ scoring=score)
        reg.fit(x_train, y_train)
        end = time.time()

        print('[anime_regression] Execution time of cross validation (in
    ↪  seconds): %.2f' % (end - start), '\n')

        print('Scoring for each regression model: ')
        means = reg.cv_results_['mean_test_score']
        stds = reg.cv_results_['std_test_score']
        for mean, std, params in zip(means, stds, reg.cv_results_['
    ↪ params']):
            print("%0.3f (+/-%0.03f) za %s" % (mean, std * 2, params))
        print()

        print('Best parameters:', reg.best_params_, '\n')
        print('Scoring results for model with best parameters: ')
        y_true_train, y_pred_train = y_train, reg.predict(x_train)
```

```
        y_true_test, y_pred_test = y_test, reg.predict(x_test)

        r2_train = r2_score(y_true_train, y_pred_train)
        r2_test = r2_score(y_true_test, y_pred_test)
        print('Training results:')
        print('R^2 score:', r2_train, '\n')
        print('Test results:')
        print('R^2 score:', r2_train, '\n')

        mse_train = mean_squared_error(y_true_train, y_pred_train)
        mse_test = mean_squared_error(y_true_test, y_pred_test)
        print('Training results:')
        print('MSE score:', mse_train, '\n')
        print('Test results:')
        print('MSE score:', mse_test, '\n')

        df_model.append({
            'model': models[i],
            'cv_scoring': score,
            'parameters': reg.best_params_,
            'R2_score_train': r2_train,
            'R2_score_test': r2_test,
            'MSE_score_train': mse_train,
            'MSE_score_test': mse_train,
            'cv_time': (end - start)
        })

with open('../podaci/anime_regression_output.csv', 'w', encoding='utf-8'
    ↪ ) as csv_file:
    csv = df_model.to_csv(index=False)
    csv_file.write(csv)
```

Rezultat izvršavanja – (MLPRegressor, MSE)

```
[anime_regression] Execution time of cross validation (in seconds): 1373.46

Scoring for each regression model:
-0.672 (+/-0.097) za {'activation': 'identity', 'hidden_layer_sizes': (10,), 'learning_rate':
-0.669 (+/-0.092) za {'activation': 'identity', 'hidden_layer_sizes': (10,), 'learning_rate':
-0.665 (+/-0.097) za {'activation': 'identity', 'hidden_layer_sizes': (10,), 'learning_rate':
-0.680 (+/-0.093) za {'activation': 'identity', 'hidden_layer_sizes': (10,), 'learning_rate':
-0.675 (+/-0.086) za {'activation': 'identity', 'hidden_layer_sizes': (100,), 'learning_rate'
-0.666 (+/-0.097) za {'activation': 'identity', 'hidden_layer_sizes': (100,), 'learning_rate'
-0.666 (+/-0.098) za {'activation': 'identity', 'hidden_layer_sizes': (100,), 'learning_rate'
-0.671 (+/-0.098) za {'activation': 'identity', 'hidden_layer_sizes': (100,), 'learning_rate'
-0.611 (+/-0.093) za {'activation': 'logistic', 'hidden_layer_sizes': (10,), 'learning_rate':
-0.659 (+/-0.093) za {'activation': 'logistic', 'hidden_layer_sizes': (10,), 'learning_rate':
-0.606 (+/-0.091) za {'activation': 'logistic', 'hidden_layer_sizes': (10,), 'learning_rate':
-0.659 (+/-0.096) za {'activation': 'logistic', 'hidden_layer_sizes': (10,), 'learning_rate':
-0.658 (+/-0.099) za {'activation': 'logistic', 'hidden_layer_sizes': (100,), 'learning_rate'
-0.682 (+/-0.096) za {'activation': 'logistic', 'hidden_layer_sizes': (100,), 'learning_rate'
-0.658 (+/-0.095) za {'activation': 'logistic', 'hidden_layer_sizes': (100,), 'learning_rate'
-0.679 (+/-0.095) za {'activation': 'logistic', 'hidden_layer_sizes': (100,), 'learning_rate'
-0.563 (+/-0.066) za {'activation': 'tanh', 'hidden_layer_sizes': (10,), 'learning_rate': 'co
-0.708 (+/-0.102) za {'activation': 'tanh', 'hidden_layer_sizes': (10,), 'learning_rate': 'co
```

```
-0.567 (+/-0.081) za {'activation': 'tanh', 'hidden_layer_sizes': (10,), 'learning_rate': 'ad
-0.700 (+/-0.147) za {'activation': 'tanh', 'hidden_layer_sizes': (10,), 'learning_rate': 'ad
-0.564 (+/-0.082) za {'activation': 'tanh', 'hidden_layer_sizes': (100,), 'learning_rate': 'c
-0.647 (+/-0.082) za {'activation': 'tanh', 'hidden_layer_sizes': (100,), 'learning_rate': 'c
-0.574 (+/-0.083) za {'activation': 'tanh', 'hidden_layer_sizes': (100,), 'learning_rate': 'a
-0.652 (+/-0.082) za {'activation': 'tanh', 'hidden_layer_sizes': (100,), 'learning_rate': 'a

Best parameters: {'activation': 'tanh', 'hidden_layer_sizes': (10,), 'learning_rate': 'consta

Scoring results for model with best parameters:
Training results:
R^2 score: 0.508445585859

Test results:
R^2 score: 0.508445585859

Training results:
MSE score: 0.508514904363

Test results:
MSE score: 0.517743306732
```

Rezultat izvršavanja – (MLPRegressor, r2)

```
[anime_regression] Execution time of cross validation (in seconds): 1366.08

Scoring for each regression model:
0.351 (+/-0.054) za {'activation': 'identity', 'hidden_layer_sizes': (10,), 'learning_rate':
0.351 (+/-0.051) za {'activation': 'identity', 'hidden_layer_sizes': (10,), 'learning_rate':
0.355 (+/-0.049) za {'activation': 'identity', 'hidden_layer_sizes': (10,), 'learning_rate':
0.353 (+/-0.051) za {'activation': 'identity', 'hidden_layer_sizes': (10,), 'learning_rate':
0.348 (+/-0.059) za {'activation': 'identity', 'hidden_layer_sizes': (100,), 'learning_rate':
0.356 (+/-0.050) za {'activation': 'identity', 'hidden_layer_sizes': (100,), 'learning_rate':
0.356 (+/-0.051) za {'activation': 'identity', 'hidden_layer_sizes': (100,), 'learning_rate':
0.354 (+/-0.053) za {'activation': 'identity', 'hidden_layer_sizes': (100,), 'learning_rate':
0.410 (+/-0.050) za {'activation': 'logistic', 'hidden_layer_sizes': (10,), 'learning_rate':
0.364 (+/-0.052) za {'activation': 'logistic', 'hidden_layer_sizes': (10,), 'learning_rate':
0.409 (+/-0.051) za {'activation': 'logistic', 'hidden_layer_sizes': (10,), 'learning_rate':
0.366 (+/-0.049) za {'activation': 'logistic', 'hidden_layer_sizes': (10,), 'learning_rate':
0.361 (+/-0.050) za {'activation': 'logistic', 'hidden_layer_sizes': (100,), 'learning_rate':
0.344 (+/-0.034) za {'activation': 'logistic', 'hidden_layer_sizes': (100,), 'learning_rate':
0.365 (+/-0.046) za {'activation': 'logistic', 'hidden_layer_sizes': (100,), 'learning_rate':
0.347 (+/-0.048) za {'activation': 'logistic', 'hidden_layer_sizes': (100,), 'learning_rate':
0.465 (+/-0.045) za {'activation': 'tanh', 'hidden_layer_sizes': (10,), 'learning_rate': 'con
0.319 (+/-0.029) za {'activation': 'tanh', 'hidden_layer_sizes': (10,), 'learning_rate': 'con
0.459 (+/-0.048) za {'activation': 'tanh', 'hidden_layer_sizes': (10,), 'learning_rate': 'ada
0.313 (+/-0.053) za {'activation': 'tanh', 'hidden_layer_sizes': (10,), 'learning_rate': 'ada
0.455 (+/-0.046) za {'activation': 'tanh', 'hidden_layer_sizes': (100,), 'learning_rate': 'co
0.379 (+/-0.036) za {'activation': 'tanh', 'hidden_layer_sizes': (100,), 'learning_rate': 'co
0.453 (+/-0.046) za {'activation': 'tanh', 'hidden_layer_sizes': (100,), 'learning_rate': 'ad
0.372 (+/-0.048) za {'activation': 'tanh', 'hidden_layer_sizes': (100,), 'learning_rate': 'ad

Best parameters: {'activation': 'tanh', 'hidden_layer_sizes': (10,), 'learning_rate': 'consta

Scoring results for model with best parameters:
```

```
Training results:
R^2 score: 0.516124324397

Test results:
R^2 score: 0.516124324397

Training results:
MSE score: 0.500571220244

Test results:
MSE score: 0.51439068573
```

Rezultat izvršavanja – (Ridge, MSE)

```
[anime_regression] Execution time of cross validation (in seconds): 125.88

Scoring for each regression model:
-0.664 (+/-0.098) za {'alpha': 0.001, 'fit_intercept': True, 'solver': 'auto', 'tol': 0.01}
-0.664 (+/-0.098) za {'alpha': 0.001, 'fit_intercept': True, 'solver': 'auto', 'tol': 0.001}
-0.664 (+/-0.098) za {'alpha': 0.001, 'fit_intercept': True, 'solver': 'auto', 'tol': 0.0001}
-0.664 (+/-0.098) za {'alpha': 0.001, 'fit_intercept': True, 'solver': 'auto', 'tol': 1e-05}
-0.664 (+/-0.098) za {'alpha': 0.001, 'fit_intercept': True, 'solver': 'sag', 'tol': 0.01}
-0.664 (+/-0.098) za {'alpha': 0.001, 'fit_intercept': True, 'solver': 'sag', 'tol': 0.001}
-0.664 (+/-0.098) za {'alpha': 0.001, 'fit_intercept': True, 'solver': 'sag', 'tol': 0.0001}
-0.664 (+/-0.098) za {'alpha': 0.001, 'fit_intercept': True, 'solver': 'sag', 'tol': 1e-05}
-43.041 (+/-0.453) za {'alpha': 0.001, 'fit_intercept': False, 'solver': 'auto', 'tol': 0.01}
-43.041 (+/-0.453) za {'alpha': 0.001, 'fit_intercept': False, 'solver': 'auto', 'tol': 0.001}
-43.041 (+/-0.453) za {'alpha': 0.001, 'fit_intercept': False, 'solver': 'auto', 'tol': 0.000
-43.041 (+/-0.453) za {'alpha': 0.001, 'fit_intercept': False, 'solver': 'auto', 'tol': 1e-05}
-43.040 (+/-0.451) za {'alpha': 0.001, 'fit_intercept': False, 'solver': 'sag', 'tol': 0.01}
-43.041 (+/-0.453) za {'alpha': 0.001, 'fit_intercept': False, 'solver': 'sag', 'tol': 0.001}
-43.041 (+/-0.453) za {'alpha': 0.001, 'fit_intercept': False, 'solver': 'sag', 'tol': 0.0001}
-43.041 (+/-0.453) za {'alpha': 0.001, 'fit_intercept': False, 'solver': 'sag', 'tol': 1e-05}
-0.664 (+/-0.098) za {'alpha': 0.01, 'fit_intercept': True, 'solver': 'auto', 'tol': 0.01}
-0.664 (+/-0.098) za {'alpha': 0.01, 'fit_intercept': True, 'solver': 'auto', 'tol': 0.001}
-0.664 (+/-0.098) za {'alpha': 0.01, 'fit_intercept': True, 'solver': 'auto', 'tol': 0.0001}
-0.664 (+/-0.098) za {'alpha': 0.01, 'fit_intercept': True, 'solver': 'auto', 'tol': 1e-05}
-0.664 (+/-0.098) za {'alpha': 0.01, 'fit_intercept': True, 'solver': 'sag', 'tol': 0.01}
-0.664 (+/-0.098) za {'alpha': 0.01, 'fit_intercept': True, 'solver': 'sag', 'tol': 0.001}
-0.664 (+/-0.098) za {'alpha': 0.01, 'fit_intercept': True, 'solver': 'sag', 'tol': 0.0001}
-0.664 (+/-0.098) za {'alpha': 0.01, 'fit_intercept': True, 'solver': 'sag', 'tol': 1e-05}
-43.041 (+/-0.453) za {'alpha': 0.01, 'fit_intercept': False, 'solver': 'auto', 'tol': 0.01}
-43.041 (+/-0.453) za {'alpha': 0.01, 'fit_intercept': False, 'solver': 'auto', 'tol': 0.001}
-43.041 (+/-0.453) za {'alpha': 0.01, 'fit_intercept': False, 'solver': 'auto', 'tol': 0.0001}
-43.041 (+/-0.453) za {'alpha': 0.01, 'fit_intercept': False, 'solver': 'auto', 'tol': 1e-05}
-43.040 (+/-0.450) za {'alpha': 0.01, 'fit_intercept': False, 'solver': 'sag', 'tol': 0.01}
-43.041 (+/-0.453) za {'alpha': 0.01, 'fit_intercept': False, 'solver': 'sag', 'tol': 0.001}
-43.041 (+/-0.453) za {'alpha': 0.01, 'fit_intercept': False, 'solver': 'sag', 'tol': 0.0001}
-43.041 (+/-0.453) za {'alpha': 0.01, 'fit_intercept': False, 'solver': 'sag', 'tol': 1e-05}
-0.664 (+/-0.098) za {'alpha': 0.2, 'fit_intercept': True, 'solver': 'auto', 'tol': 0.01}
-0.664 (+/-0.098) za {'alpha': 0.2, 'fit_intercept': True, 'solver': 'auto', 'tol': 0.001}
-0.664 (+/-0.098) za {'alpha': 0.2, 'fit_intercept': True, 'solver': 'auto', 'tol': 0.0001}
-0.664 (+/-0.098) za {'alpha': 0.2, 'fit_intercept': True, 'solver': 'auto', 'tol': 1e-05}
-0.664 (+/-0.098) za {'alpha': 0.2, 'fit_intercept': True, 'solver': 'sag', 'tol': 0.01}
```

```
-0.664 (+/-0.098) za {'alpha': 0.2, 'fit_intercept': True, 'solver': 'sag', 'tol': 0.001}
-0.664 (+/-0.098) za {'alpha': 0.2, 'fit_intercept': True, 'solver': 'sag', 'tol': 0.0001}
-0.664 (+/-0.098) za {'alpha': 0.2, 'fit_intercept': True, 'solver': 'sag', 'tol': 1e-05}
-43.041 (+/-0.453) za {'alpha': 0.2, 'fit_intercept': False, 'solver': 'auto', 'tol': 0.01}
-43.041 (+/-0.453) za {'alpha': 0.2, 'fit_intercept': False, 'solver': 'auto', 'tol': 0.001}
-43.041 (+/-0.453) za {'alpha': 0.2, 'fit_intercept': False, 'solver': 'auto', 'tol': 0.0001}
-43.041 (+/-0.453) za {'alpha': 0.2, 'fit_intercept': False, 'solver': 'auto', 'tol': 1e-05}
-43.039 (+/-0.452) za {'alpha': 0.2, 'fit_intercept': False, 'solver': 'sag', 'tol': 0.01}
-43.041 (+/-0.453) za {'alpha': 0.2, 'fit_intercept': False, 'solver': 'sag', 'tol': 0.001}
-43.041 (+/-0.453) za {'alpha': 0.2, 'fit_intercept': False, 'solver': 'sag', 'tol': 0.0001}
-43.041 (+/-0.453) za {'alpha': 0.2, 'fit_intercept': False, 'solver': 'sag', 'tol': 1e-05}
-0.664 (+/-0.098) za {'alpha': 0.5, 'fit_intercept': True, 'solver': 'auto', 'tol': 0.01}
-0.664 (+/-0.098) za {'alpha': 0.5, 'fit_intercept': True, 'solver': 'auto', 'tol': 0.001}
-0.664 (+/-0.098) za {'alpha': 0.5, 'fit_intercept': True, 'solver': 'auto', 'tol': 0.0001}
-0.664 (+/-0.098) za {'alpha': 0.5, 'fit_intercept': True, 'solver': 'auto', 'tol': 1e-05}
-0.664 (+/-0.098) za {'alpha': 0.5, 'fit_intercept': True, 'solver': 'sag', 'tol': 0.01}
-0.664 (+/-0.098) za {'alpha': 0.5, 'fit_intercept': True, 'solver': 'sag', 'tol': 0.001}
-0.664 (+/-0.098) za {'alpha': 0.5, 'fit_intercept': True, 'solver': 'sag', 'tol': 0.0001}
-0.664 (+/-0.098) za {'alpha': 0.5, 'fit_intercept': True, 'solver': 'sag', 'tol': 1e-05}
-43.041 (+/-0.453) za {'alpha': 0.5, 'fit_intercept': False, 'solver': 'auto', 'tol': 0.01}
-43.041 (+/-0.453) za {'alpha': 0.5, 'fit_intercept': False, 'solver': 'auto', 'tol': 0.001}
-43.041 (+/-0.453) za {'alpha': 0.5, 'fit_intercept': False, 'solver': 'auto', 'tol': 0.0001}
-43.041 (+/-0.453) za {'alpha': 0.5, 'fit_intercept': False, 'solver': 'auto', 'tol': 1e-05}
-43.040 (+/-0.453) za {'alpha': 0.5, 'fit_intercept': False, 'solver': 'sag', 'tol': 0.01}
-43.041 (+/-0.453) za {'alpha': 0.5, 'fit_intercept': False, 'solver': 'sag', 'tol': 0.001}
-43.041 (+/-0.453) za {'alpha': 0.5, 'fit_intercept': False, 'solver': 'sag', 'tol': 0.0001}
-43.041 (+/-0.453) za {'alpha': 0.5, 'fit_intercept': False, 'solver': 'sag', 'tol': 1e-05}
-0.664 (+/-0.098) za {'alpha': 1, 'fit_intercept': True, 'solver': 'auto', 'tol': 0.01}
-0.664 (+/-0.098) za {'alpha': 1, 'fit_intercept': True, 'solver': 'auto', 'tol': 0.001}
-0.664 (+/-0.098) za {'alpha': 1, 'fit_intercept': True, 'solver': 'auto', 'tol': 0.0001}
-0.664 (+/-0.098) za {'alpha': 1, 'fit_intercept': True, 'solver': 'auto', 'tol': 1e-05}
-0.664 (+/-0.098) za {'alpha': 1, 'fit_intercept': True, 'solver': 'sag', 'tol': 0.01}
-0.664 (+/-0.098) za {'alpha': 1, 'fit_intercept': True, 'solver': 'sag', 'tol': 0.001}
-0.664 (+/-0.098) za {'alpha': 1, 'fit_intercept': True, 'solver': 'sag', 'tol': 0.0001}
-0.664 (+/-0.098) za {'alpha': 1, 'fit_intercept': True, 'solver': 'sag', 'tol': 1e-05}
-43.041 (+/-0.453) za {'alpha': 1, 'fit_intercept': False, 'solver': 'auto', 'tol': 0.01}
-43.041 (+/-0.453) za {'alpha': 1, 'fit_intercept': False, 'solver': 'auto', 'tol': 0.001}
-43.041 (+/-0.453) za {'alpha': 1, 'fit_intercept': False, 'solver': 'auto', 'tol': 0.0001}
-43.041 (+/-0.453) za {'alpha': 1, 'fit_intercept': False, 'solver': 'auto', 'tol': 1e-05}
-43.040 (+/-0.451) za {'alpha': 1, 'fit_intercept': False, 'solver': 'sag', 'tol': 0.01}
-43.041 (+/-0.453) za {'alpha': 1, 'fit_intercept': False, 'solver': 'sag', 'tol': 0.001}
-43.041 (+/-0.453) za {'alpha': 1, 'fit_intercept': False, 'solver': 'sag', 'tol': 0.0001}
-43.041 (+/-0.453) za {'alpha': 1, 'fit_intercept': False, 'solver': 'sag', 'tol': 1e-05}

Best parameters: {'alpha': 1, 'fit_intercept': True, 'solver': 'auto', 'tol': 0.01}

Scoring results for model with best parameters:
Training results:
R^2 score: 0.365773657825

Test results:
R^2 score: 0.365773657825

Training results:
MSE score: 0.656109554624
```

```
Test results:
MSE score: 0.653075897292
```

Rezultat izvršavanja – (Ridge, r2)

```
[anime_regression] Execution time of cross validation (in seconds): 127.02

Scoring for each regression model:
0.358 (+/-0.053) za {'alpha': 0.001, 'fit_intercept': True, 'solver': 'auto', 'tol': 0.01}
0.358 (+/-0.053) za {'alpha': 0.001, 'fit_intercept': True, 'solver': 'auto', 'tol': 0.001}
0.358 (+/-0.053) za {'alpha': 0.001, 'fit_intercept': True, 'solver': 'auto', 'tol': 0.0001}
0.358 (+/-0.053) za {'alpha': 0.001, 'fit_intercept': True, 'solver': 'auto', 'tol': 1e-05}
0.358 (+/-0.053) za {'alpha': 0.001, 'fit_intercept': True, 'solver': 'sag', 'tol': 0.01}
0.358 (+/-0.053) za {'alpha': 0.001, 'fit_intercept': True, 'solver': 'sag', 'tol': 0.001}
0.358 (+/-0.053) za {'alpha': 0.001, 'fit_intercept': True, 'solver': 'sag', 'tol': 0.0001}
0.358 (+/-0.053) za {'alpha': 0.001, 'fit_intercept': True, 'solver': 'sag', 'tol': 1e-05}
-40.733 (+/-4.311) za {'alpha': 0.001, 'fit_intercept': False, 'solver': 'auto', 'tol': 0.01}
-40.733 (+/-4.311) za {'alpha': 0.001, 'fit_intercept': False, 'solver': 'auto', 'tol': 0.001
-40.733 (+/-4.311) za {'alpha': 0.001, 'fit_intercept': False, 'solver': 'auto', 'tol': 0.000
-40.733 (+/-4.311) za {'alpha': 0.001, 'fit_intercept': False, 'solver': 'auto', 'tol': 1e-05
-40.733 (+/-4.309) za {'alpha': 0.001, 'fit_intercept': False, 'solver': 'sag', 'tol': 0.01}
-40.733 (+/-4.311) za {'alpha': 0.001, 'fit_intercept': False, 'solver': 'sag', 'tol': 0.001}
-40.733 (+/-4.311) za {'alpha': 0.001, 'fit_intercept': False, 'solver': 'sag', 'tol': 0.0001
-40.733 (+/-4.311) za {'alpha': 0.001, 'fit_intercept': False, 'solver': 'sag', 'tol': 1e-05}
0.358 (+/-0.053) za {'alpha': 0.01, 'fit_intercept': True, 'solver': 'auto', 'tol': 0.01}
0.358 (+/-0.053) za {'alpha': 0.01, 'fit_intercept': True, 'solver': 'auto', 'tol': 0.001}
0.358 (+/-0.053) za {'alpha': 0.01, 'fit_intercept': True, 'solver': 'auto', 'tol': 0.0001}
0.358 (+/-0.053) za {'alpha': 0.01, 'fit_intercept': True, 'solver': 'auto', 'tol': 1e-05}
0.358 (+/-0.053) za {'alpha': 0.01, 'fit_intercept': True, 'solver': 'sag', 'tol': 0.01}
0.358 (+/-0.053) za {'alpha': 0.01, 'fit_intercept': True, 'solver': 'sag', 'tol': 0.001}
0.358 (+/-0.053) za {'alpha': 0.01, 'fit_intercept': True, 'solver': 'sag', 'tol': 0.0001}
0.358 (+/-0.053) za {'alpha': 0.01, 'fit_intercept': True, 'solver': 'sag', 'tol': 1e-05}
-40.733 (+/-4.311) za {'alpha': 0.01, 'fit_intercept': False, 'solver': 'auto', 'tol': 0.01}
-40.733 (+/-4.311) za {'alpha': 0.01, 'fit_intercept': False, 'solver': 'auto', 'tol': 0.001}
-40.733 (+/-4.311) za {'alpha': 0.01, 'fit_intercept': False, 'solver': 'auto', 'tol': 0.0001
-40.733 (+/-4.311) za {'alpha': 0.01, 'fit_intercept': False, 'solver': 'auto', 'tol': 1e-05}
-40.732 (+/-4.310) za {'alpha': 0.01, 'fit_intercept': False, 'solver': 'sag', 'tol': 0.01}
-40.733 (+/-4.311) za {'alpha': 0.01, 'fit_intercept': False, 'solver': 'sag', 'tol': 0.001}
-40.733 (+/-4.311) za {'alpha': 0.01, 'fit_intercept': False, 'solver': 'sag', 'tol': 0.0001}
-40.733 (+/-4.311) za {'alpha': 0.01, 'fit_intercept': False, 'solver': 'sag', 'tol': 1e-05}
0.358 (+/-0.053) za {'alpha': 0.2, 'fit_intercept': True, 'solver': 'auto', 'tol': 0.01}
0.358 (+/-0.053) za {'alpha': 0.2, 'fit_intercept': True, 'solver': 'auto', 'tol': 0.001}
0.358 (+/-0.053) za {'alpha': 0.2, 'fit_intercept': True, 'solver': 'auto', 'tol': 0.0001}
0.358 (+/-0.053) za {'alpha': 0.2, 'fit_intercept': True, 'solver': 'auto', 'tol': 1e-05}
0.358 (+/-0.053) za {'alpha': 0.2, 'fit_intercept': True, 'solver': 'sag', 'tol': 0.01}
0.358 (+/-0.053) za {'alpha': 0.2, 'fit_intercept': True, 'solver': 'sag', 'tol': 0.001}
0.358 (+/-0.053) za {'alpha': 0.2, 'fit_intercept': True, 'solver': 'sag', 'tol': 0.0001}
0.358 (+/-0.053) za {'alpha': 0.2, 'fit_intercept': True, 'solver': 'sag', 'tol': 1e-05}
-40.733 (+/-4.311) za {'alpha': 0.2, 'fit_intercept': False, 'solver': 'auto', 'tol': 0.01}
-40.733 (+/-4.311) za {'alpha': 0.2, 'fit_intercept': False, 'solver': 'auto', 'tol': 0.001}
-40.733 (+/-4.311) za {'alpha': 0.2, 'fit_intercept': False, 'solver': 'auto', 'tol': 0.0001}
-40.733 (+/-4.311) za {'alpha': 0.2, 'fit_intercept': False, 'solver': 'auto', 'tol': 1e-05}
-40.732 (+/-4.309) za {'alpha': 0.2, 'fit_intercept': False, 'solver': 'sag', 'tol': 0.01}
-40.733 (+/-4.311) za {'alpha': 0.2, 'fit_intercept': False, 'solver': 'sag', 'tol': 0.001}
```

```
-40.733 (+/-4.311) za {'alpha': 0.2, 'fit_intercept': False, 'solver': 'sag', 'tol': 0.0001}
-40.733 (+/-4.311) za {'alpha': 0.2, 'fit_intercept': False, 'solver': 'sag', 'tol': 1e-05}
0.358 (+/-0.053) za {'alpha': 0.5, 'fit_intercept': True, 'solver': 'auto', 'tol': 0.01}
0.358 (+/-0.053) za {'alpha': 0.5, 'fit_intercept': True, 'solver': 'auto', 'tol': 0.001}
0.358 (+/-0.053) za {'alpha': 0.5, 'fit_intercept': True, 'solver': 'auto', 'tol': 0.0001}
0.358 (+/-0.053) za {'alpha': 0.5, 'fit_intercept': True, 'solver': 'auto', 'tol': 1e-05}
0.358 (+/-0.053) za {'alpha': 0.5, 'fit_intercept': True, 'solver': 'sag', 'tol': 0.01}
0.358 (+/-0.053) za {'alpha': 0.5, 'fit_intercept': True, 'solver': 'sag', 'tol': 0.001}
0.358 (+/-0.053) za {'alpha': 0.5, 'fit_intercept': True, 'solver': 'sag', 'tol': 0.0001}
0.358 (+/-0.053) za {'alpha': 0.5, 'fit_intercept': True, 'solver': 'sag', 'tol': 1e-05}
-40.733 (+/-4.311) za {'alpha': 0.5, 'fit_intercept': False, 'solver': 'auto', 'tol': 0.01}
-40.733 (+/-4.311) za {'alpha': 0.5, 'fit_intercept': False, 'solver': 'auto', 'tol': 0.001}
-40.733 (+/-4.311) za {'alpha': 0.5, 'fit_intercept': False, 'solver': 'auto', 'tol': 0.0001}
-40.733 (+/-4.311) za {'alpha': 0.5, 'fit_intercept': False, 'solver': 'auto', 'tol': 1e-05}
-40.731 (+/-4.307) za {'alpha': 0.5, 'fit_intercept': False, 'solver': 'sag', 'tol': 0.01}
-40.733 (+/-4.311) za {'alpha': 0.5, 'fit_intercept': False, 'solver': 'sag', 'tol': 0.001}
-40.733 (+/-4.311) za {'alpha': 0.5, 'fit_intercept': False, 'solver': 'sag', 'tol': 0.0001}
-40.733 (+/-4.311) za {'alpha': 0.5, 'fit_intercept': False, 'solver': 'sag', 'tol': 1e-05}
0.358 (+/-0.053) za {'alpha': 1, 'fit_intercept': True, 'solver': 'auto', 'tol': 0.01}
0.358 (+/-0.053) za {'alpha': 1, 'fit_intercept': True, 'solver': 'auto', 'tol': 0.001}
0.358 (+/-0.053) za {'alpha': 1, 'fit_intercept': True, 'solver': 'auto', 'tol': 0.0001}
0.358 (+/-0.053) za {'alpha': 1, 'fit_intercept': True, 'solver': 'auto', 'tol': 1e-05}
0.358 (+/-0.053) za {'alpha': 1, 'fit_intercept': True, 'solver': 'sag', 'tol': 0.01}
0.358 (+/-0.053) za {'alpha': 1, 'fit_intercept': True, 'solver': 'sag', 'tol': 0.001}
0.358 (+/-0.053) za {'alpha': 1, 'fit_intercept': True, 'solver': 'sag', 'tol': 0.0001}
0.358 (+/-0.053) za {'alpha': 1, 'fit_intercept': True, 'solver': 'sag', 'tol': 1e-05}
-40.733 (+/-4.311) za {'alpha': 1, 'fit_intercept': False, 'solver': 'auto', 'tol': 0.01}
-40.733 (+/-4.311) za {'alpha': 1, 'fit_intercept': False, 'solver': 'auto', 'tol': 0.001}
-40.733 (+/-4.311) za {'alpha': 1, 'fit_intercept': False, 'solver': 'auto', 'tol': 0.0001}
-40.733 (+/-4.311) za {'alpha': 1, 'fit_intercept': False, 'solver': 'auto', 'tol': 1e-05}
-40.732 (+/-4.308) za {'alpha': 1, 'fit_intercept': False, 'solver': 'sag', 'tol': 0.01}
-40.733 (+/-4.310) za {'alpha': 1, 'fit_intercept': False, 'solver': 'sag', 'tol': 0.001}
-40.733 (+/-4.311) za {'alpha': 1, 'fit_intercept': False, 'solver': 'sag', 'tol': 0.0001}
-40.733 (+/-4.311) za {'alpha': 1, 'fit_intercept': False, 'solver': 'sag', 'tol': 1e-05}

Best parameters: {'alpha': 1, 'fit_intercept': True, 'solver': 'auto', 'tol': 0.01}

Scoring results for model with best parameters:
Training results:
R^2 score: 0.365773657825

Test results:
R^2 score: 0.365773657825

Training results:
MSE score: 0.656109554624

Test results:
MSE score: 0.653075897292
```

Analiza rezultata Detaljan uvid dobijamo analizom prethodnog rezultata izvršavanja. Međutim, mi ćemo na ovom mestu analizirati podatke dobijene u datoteci anime_regression_output.csv. Ovu datoteku čine podaci o najboljim modelima koji su konstruisani pri svakoj unakrsnoj validaciji. Pogledajmo mogući sadržaj ove datoteke (ispisan kao pandas.DataFrame).

```
          model              cv_scoring  \
```

```
0  MLPRegressor  neg_mean_squared_error
1  MLPRegressor                      r2
2         Ridge  neg_mean_squared_error
3         Ridge                      r2

                                          parameters  R2_score_train  \
0  {'activation': 'tanh', 'hidden_layer_sizes': (...         0.508446
1  {'activation': 'tanh', 'hidden_layer_sizes': (...         0.516124
2  {'alpha': 1, 'fit_intercept': True, 'solver': ...         0.365774
3  {'alpha': 1, 'fit_intercept': True, 'solver': ...         0.365774

   R2_score_test  MSE_score_train  MSE_score_test      cv_time
0       0.505073         0.508515        0.508515  1373.459475
1       0.508278         0.500571        0.500571  1366.075627
2       0.375704         0.656110        0.656110   125.880620
3       0.375704         0.656110        0.656110   127.019831
```

Na osnovu rezultata vidimo da se bolje ponaša neuronska mreža u odnosu na grebenu regresiju. Ipak, koeficijent determinacije bi trebalo biti bolji. Potrebno je izvršiti dodatnu analizu (na primer, promenom izbora konfiguracija algoritma neuronske mreže).

Komentar Prilikom izvršavanja prethodnog programskog koda, dobijena su upozorenja oblika

```
D:\Users\Admin\Anaconda3\lib\site-packages\sklearn\neural_network\
    ↪ multilayer_perceptron.py:563: ConvergenceWarning: Stochastic
    ↪ Optimizer: Maximum iterations reached and the optimization hasn't
    ↪  converged yet.
  % (), ConvergenceWarning)
```

za svaku konfiguraciju pri konstrukciji modela neuronske mreže.