UNIVERZITET U BEOGRADU
ELEKTROTEHNIČKI FAKULTET

SOFTVERSKO INŽENJERSTVO VELIKIH BAZA PODATAKA

Rešenje domaćeg zadatka

Profesor:

Dr Miroslav Bojović

Student:

Vladimir Janković
2023/3244

Beograd, jun 2024. godine

Sadržaj

Opsi domaćeg zadatka

Domaći zadatak predstavlja rešenje Kaggle takmičenja „Regression with a Mohs Hardness Dataset“, za koje je potrebno predvitedi vrednost tvrdoće minerala na osnovu određenih odlika. Podaci su dati u tabelarnom formatu u fajlu *train.csv*, gde je svaki podatak opisan sa 11 kolona koje predstavljaju odlike podatka i jedna kolona koja predstavlja izlaznu vrednost podatka koju je potrebno predvideti. Sve odlike podataka su kontinualne numeričke vrednosti, kao i izlazne vrednosti. S obzirom da je izlazna vrednost kontinualna, zadatak predstavlja regresioni problem, koji se rešava korišćenjem regresionih algoritama mašinskog učenja. Metrika koja se koristi u zadatku je medijana apsolutne greške:

$$MedAE(p, y) = median(|p_1 - y_1|, |p_2 - y_2|, \dots, |p_n - y_n|)$$

Rešenje projektnog zadatka je realizovano pomoću programskog jezika *Python*, koji sadrži veliku kolekciju biblioteka za mašinsko učenje. Za potrebe rešavanja domaćeg zadatka, iskorišćene su biblioteke *scikit-learn* i *xgboost*.

1. Analiza podataka

Analiza podataka je obrađenja pomoću Python biblioteka *pandas*, *matplotlib* i *seaborn*. Nakon učitavanja datoteke *train.csv*, potrebno je proveriti da li postoje nedostajuće vrednosti u kolonama za odlike. Pozivom funkcija *pandas.info* i *pandas.describe*, u konzoli se ispisuje sledeći sadržaj:

```
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   allelectrons_Total    10407 non-null  float64
 1   density_Total         10407 non-null  float64
 2   allelectrons_Average  10407 non-null  float64
 3   val_e_Average         10407 non-null  float64
 4   atomicweight_Average  10407 non-null  float64
 5   ionenergy_Average     10407 non-null  float64
 6   el_neg_chi_Average    10407 non-null  float64
 7   R_vdw_element_Average 10407 non-null  float64
 8   R_cov_element_Average 10407 non-null  float64
 9   zaratio_Average       10407 non-null  float64
 10  density_Average       10407 non-null  float64
 11  Hardness              10407 non-null  float64
```

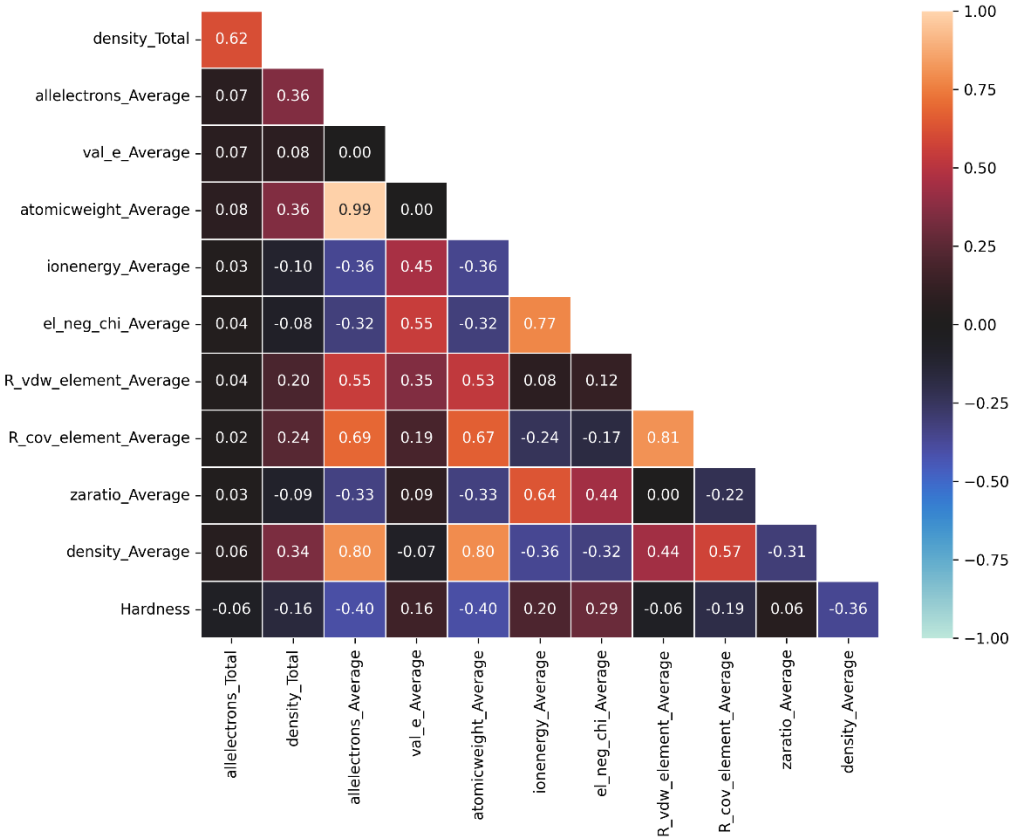
Slika 1. Informacije o tabeli podataka

Može se primetiti sa slike 1. da ne postoje nedostajući podaci u tabeli, što znači da nije potrebno veštački popunjavati tabelu vrednostima. Broj podataka je 10.407, što je dovoljno veliki skup za modele mašinskog učenja. Naredni korak u analizi podataka je provera koliko su odlike međusobno korelisane. Na slici 2. se nalazi implementacija funkcije pomoću koje se iscrtava korelaciona matrica i čuva lokalno.

```python
def plot_corr(data, name):
    data_corr = data.corr()
    mask = np.triu(np.ones_like(data_corr, dtype=bool))[1:, :-1]
    plt.figure(figsize=(10, 8))
    sns.heatmap(data=data_corr.iloc[1:, :-1], mask=mask, annot=True, fmt=".2f",
                vmin=-1, vmax=1, linecolor='white', linewidths=0.5, center=0)
    plt.tight_layout()
    plt.savefig(f'./{name}.png', dpi=300)
    plt.close()
```

Slika 2. Pomoćna funkcija za iscrtavanje korelacione matrice

Funkcija *plot_corr* uzima kao prvi parametar podatak tipa *pandas.DataFrame* i računa vrednost korelacione matrice, koja je simetrična u odnosu na glavnu dijagonalu. Zbog toga je dovoljno prikazati samo donji ili gornji trougao matrice. Promenljiva *mask* predstavlja matricu čiji je gorinji trougao markiran, a pri iscrtavanju korelacione matrice, markirana polja se maskiraju, tj. neće biti prikazana. Biblioteka *seaborn* sadrži funkciju *heatmap* kojom se iscrtava korelaciona matrica u određenom formatu. Na slici 3. je prikazana korelaciona matrica za dati skup podataka.

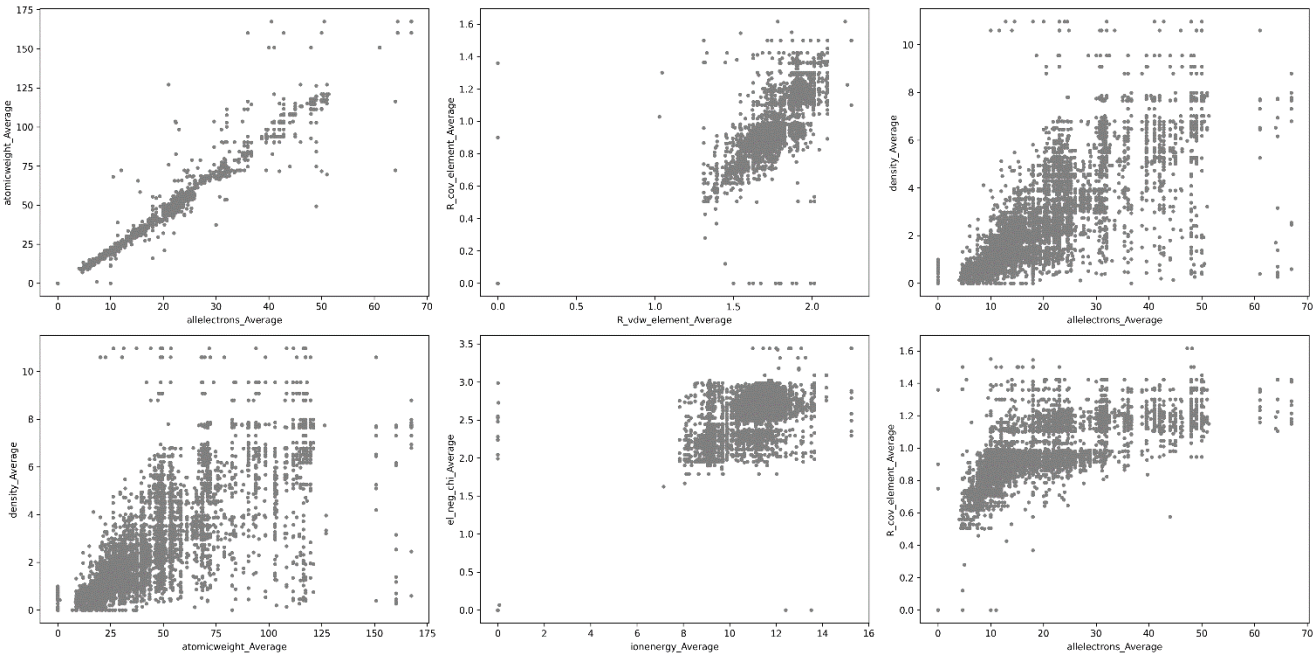


Slika 3. Korelaciona matrica

Analizom korelacione matrice se mogu primetiti sledeće stvari. Postoji velika korelacija između odlika *allelectrons_Average* i *atomicweight_Average* i to čak 0.99, što označava da su te dve odlike gotovo identične. Pored toga, postoje i druge visoke vrednosti korelacija, pa se delom koda na slici 4. grafički iscrtavaju odnosi između odlika sa visikom korelacijom, a na slici 5. su prikazani grafici zavisnosti takvih parova odlika.

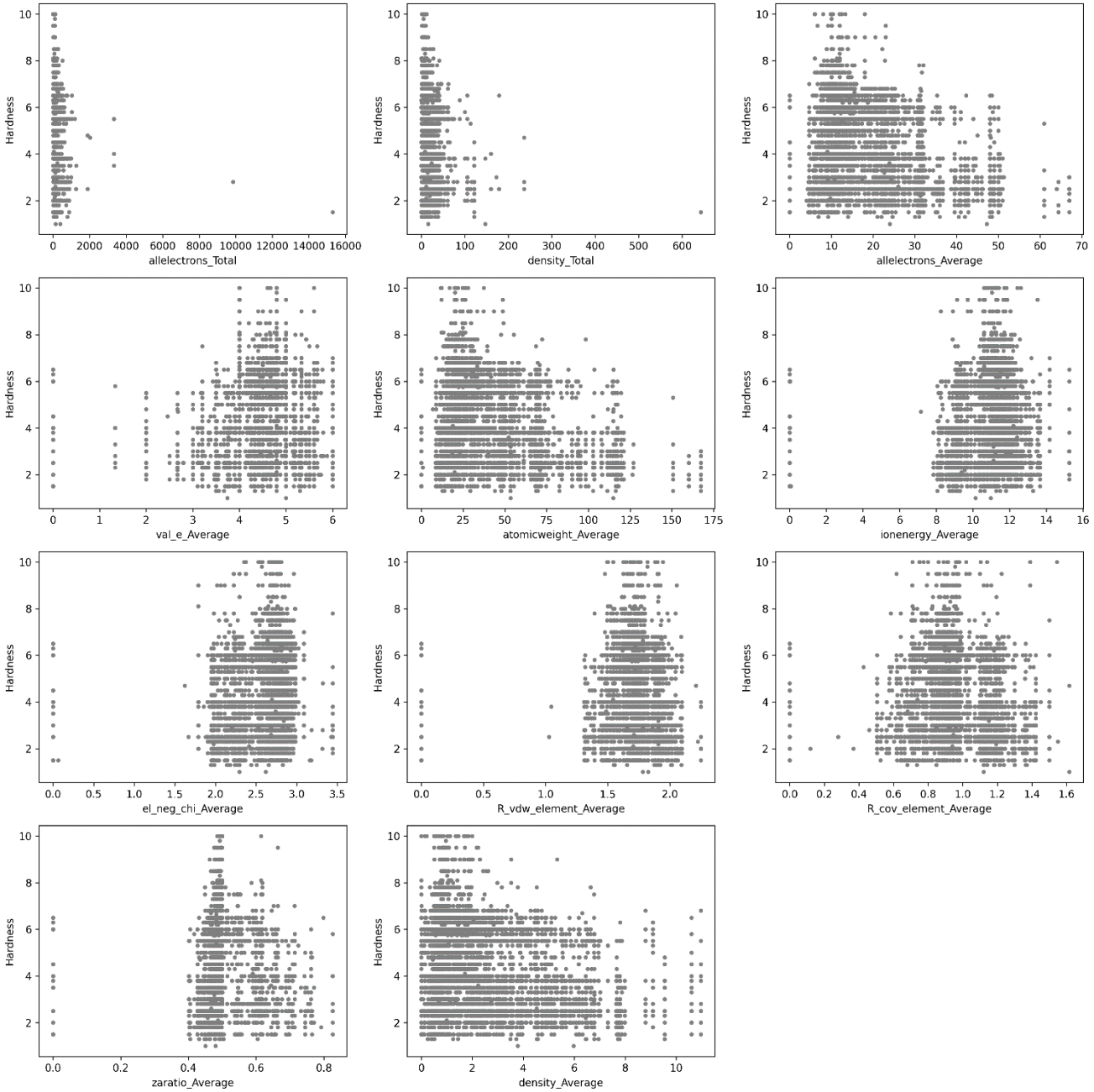
```
75  pairs = [
76      ('allelectrons_Average', 'atomicweight_Average'),
77      ('R_vdw_element_Average', 'R_cov_element_Average'),
78      ('allelectrons_Average', 'density_Average'),
79      ('atomicweight_Average', 'density_Average'),
80      ('ionenergy_Average', 'el_neg_chi_Average'),
81      ('allelectrons_Average', 'R_cov_element_Average'),
82  ]
83  plt.figure(figsize=(20, 10))
84  k = 1
85  for p in pairs:
86      x = data[p[0]].to_numpy()
87      y = data[p[1]].to_numpy()
88      plt.subplot(2, 3, k)
89      k += 1
90      plt.scatter(x=x, y=y, c='grey', marker='.')
91      plt.xlabel(p[0])
92      plt.ylabel(p[1])
93      plt.tight_layout()
94  plt.savefig(f'./HighCorr/HighCorr.png', dpi=300)
95  plt.close()
```

Slika 4. Deo koda za grafičko iscrtavanje odnosa odliak sa visokom korelacijom



Slika 5. Grafici odnosa odlika sa visokom korelacijom

Izlazna promenljiva u ovom skupu podataka je *Hardness* (tvrdoća minerala). Da bi se proverilo postojanje *outlier*-a ili određenih podataka koji mogu narušiti određenu pravilnost u skupu podataka, potrebno je vizuelizovati odnos između izlaza skupa podataka i svake odlike. Na slici 6. su prikazani grafici zavisnosti između svake odlike i izlaza, a deo koda zaslužan za iscrtavanje grafika sa slike 6. je prikazan na slici 7. Analizom ovih grafika se jasno može uočiti da postoje podaci koji predstavljaju *outlier*-e i koje je potrebno ukloniti iz skupa podataka. Na primer, za odliku *allelectrons_Total* skoro svi podaci imaju vrednosti u opsegu od 0 do 1000, za odliku *ionenergy_Average* u opsegu od 8 do 16, ...



Slika 6. Grafici zavisnosti svih odlika od izlaza

```
k = 1
plt.figure(figsize=(15, 15))
for col in data.columns:
    if col == 'Hardness':
        continue
    x = data[col].to_numpy()
    y = data['Hardness'].to_numpy()
    plt.subplot(4, 3, k)
    k += 1
    plt.scatter(x=x, y=y, c='grey', marker='.')
    plt.xlabel(col)
    plt.ylabel('Hardness')
    plt.tight_layout()
plt.savefig(f'./vsHardness/Hardness.png', dpi=300)
plt.close()
```

Slika 7. Deo koda za iscrtavanje odnosa odlika i izlaza

Na osnovu grafika sa slike 6. vrši se filtriranje podataka uklanjanjem neželjenih *outlier*-a. Na slici 8. je prikazan dao koda koji uklanja *outlier*-e iz skupa podataka. S obzirom da postoji velika korelacija između odlika *allelectrons_Average* i *atomicweight_Average* može se ukloniti jedna od ovih odlika (odabrana je odlika *allelectrons_Average* za uklanjaje). Dodatno, odlike *R_cov_element_Average* i *R_vdw_element_Average* imaju visoku koerlaciju pa je izvršeno objedinjavanje u jednu odliku *R_Average*, koja predstavlja proizvod vrednosti dveju odlika.

```
######################################################################
#                        DATA FILTERING                              #
######################################################################

filter_data = data.copy(deep=True)
filter_data = filter_data.drop(index=filter_data.loc[filter_data['allelectrons_Total'] > 750, 'allelectrons_Total'].index)
filter_data = filter_data.drop(index=filter_data.loc[filter_data['atomicweight_Average'] > 150, 'atomicweight_Average'].index)
filter_data = filter_data.drop(index=filter_data.loc[filter_data['atomicweight_Average'] < 8, 'atomicweight_Average'].index)
filter_data = filter_data.drop(index=filter_data.loc[filter_data['density_Total'] > 50, 'density_Total'].index)
filter_data = filter_data.drop(index=filter_data.loc[filter_data['val_e_Average'] < 2.5, 'val_e_Average'].index)
filter_data = filter_data.drop(index=filter_data.loc[filter_data['el_neg_chi_Average'] < 1.5, 'el_neg_chi_Average'].index)
filter_data = filter_data.drop(index=filter_data.loc[filter_data['el_neg_chi_Average'] > 3.05, 'el_neg_chi_Average'].index)
filter_data = filter_data.drop(index=filter_data.loc[filter_data['ionenergy_Average'] < 8, 'ionenergy_Average'].index)
filter_data = filter_data.drop(index=filter_data.loc[filter_data['R_cov_element_Average'] < 0.5, 'R_cov_element_Average'].index)
filter_data = filter_data.drop(index=filter_data.loc[filter_data['R_vdw_element_Average'] < 1.3, 'R_vdw_element_Average'].index)
filter_data = filter_data.drop(index=filter_data.loc[filter_data['R_vdw_element_Average'] > 2.2, 'R_vdw_element_Average'].index)
filter_data.insert(len(filter_data.columns) - 1, 'R_Average', filter_data['R_cov_element_Average'] * filter_data['R_vdw_element_Average'])
filter_data = filter_data.drop(columns=['R_cov_element_Average', 'R_vdw_element_Average'])
filter_data = filter_data.drop(index=filter_data.loc[filter_data['density_Average'] > 8, 'density_Average'].index)
filter_data = filter_data.drop(index=filter_data.loc[filter_data['zaratio_Average'] < 0.4, 'zaratio_Average'].index)
filter_data = filter_data.drop(index=filter_data.loc[(filter_data['allelectrons_Total'] > 300) & (filter_data['density_Total'] < 15), 'allelectrons_Total'].index)
filter_data = filter_data.drop(columns=['allelectrons_Average'])
```

Slika 8. Deo koda za uklanjanje *outlier*-a iz skupa podataka

2. Odabir modela mašinskog učenja

Nakon što su podaci filtrirani, sledeći korak je razdvajanje odlika i izlaza, potencijalna transformacija i skaliranje odlika, kako bi se unapredio rad modela i razdvajanje skupa podataka na skupove za obučavanje i testiranje, prikazano na slici 9. S obzirom da su sve odlike kontinualne numeričke vrednosti, odrađeno je skaliranje vrednosti odlika pomoću klase *StandardScaler*. Ova klasa vrši skaliranje tako što svaku vrednost oduzme sa srednjom vrednošću odlike i deli je sa standardnom devijacijom odlike. Skaliranje na ovaj način može ubrzati konvergenciju kod obučavanja modela. Pored toga je iskorišćena klasa *LocalOutlierFactor* koja pomaže u uklanjaju *outlier*-a koji nisu otkriveni tokom filtriranja podataka. Algoritam radi na principu najbližih komšija, gde se označavaju podaci koji poseduju veliki stepen izolacije od ostalih podataka. Funkcija *train_test_split* se koristi kako bi se formirali skupovi za obučavanje i tersiranje, gde je 75% podataka u skupu za obučavanje i 25% u skupu za testiranje. Pre odabira modela, napravljen je objekat klase *KFold*, koji će se koristiti za validaciju modela i pronalaženje najbolje kombinacije hiperparametara.

```
146  ##########################################################################
147  #                          MODEL SELECTION                               #
148  ##########################################################################
149
150
151  y = filter_data["Hardness"]
152  X = filter_data.drop(columns=['Hardness'])
153
154  st = StandardScaler()
155  X = st.fit_transform(X)
156
157  lof = LocalOutlierFactor(n_neighbors=20)
158  outliers = lof.fit_predict(X)
159  mask = outliers != -1
160  X, y = X[mask], y[mask]
161
162  x_train, x_test, y_train, y_test = train_test_split(X, y, train_size=0.75, random_state=123, shuffle=False)
163
164  kf = KFold(n_splits=5, shuffle=True, random_state=123)
```

Slika 9. Deo koda za formiranje skupova za obučavanje i testiranje

Naredni korak je odabir i provera performansi raznih modela mašinskog učenja nad ovim skupom podataka. Za rešavanje problema predviđanja izlazne vrednosti, koriste se regresioni modeli algoritama mašinskog učenja. Biblioteka *scikit-learn* je popularna za rad sa mašinskim učenjem i sadrži veliki broj gotovih klasa regresivnih modela. Takođe, iskorišćena je i biblioteka *xgboost*, čiji modeli za predviđanje su među najboljima i vrlo su popularni u *Kaggel* takmičenjima. U nastavku se analizira svaki model kao i njegove performanse u predviđanju izlazne vrednosti. Svi modeli su smešteni u listu sa nazivom *models* i za svaki model će se izvršiti kod prikazan na slici 10.

```
166 > models = [...]
199
200 for name, model, params in models:
201     t = time.time()
202     grid = GridSearchCV(model, params, cv=kf, scoring='neg_median_absolute_error')
203     grid.fit(x_train, y_train)
204     best_model = grid.best_estimator_
205     pred = best_model.predict(x_test)
206     cv_results = cross_val_score(best_model, x_train, y_train, cv=kf, scoring='neg_median_absolute_error')
207     t = time.time() - t
208
209     print(f"\n\n{name}:")
210     if len(params) != 0:
211         print(f"Best Hyperparameters: {grid.best_params_}")
212     print(f"Training Score: {best_model.score(x_train, y_train):.3f}")
213     print(f"Median Absolute Error on Test Set: {median_absolute_error(y_test, pred):.3f}")
214     print(f"Cross-Validation Median Absolute Error: {-cv_results.mean():.3f} ± {cv_results.std():.3f}")
215     print(f'Time required: {t:.3f}s')
```

Slika 10. Kod za obučavanje i proveru performansi modela

Za svaki model se vrši pretraga po mreži (GridSearch) kako bi se odredila najbolja kombinacija hiperparametara za model. Tokom pretrage se koristi unakrsna validacija, jer su moguće promene optimalnih hiperparametara u zavisnosti od podele podataka za skupove obučavanja i validaciju. Odabirom najboljih hiperparametara, vrši se obučavanje modela i predviđanje izlaza. Međutim, rezultati obučavanja i predviđanja se mogu razlikovati u zavisnosti od podele podataka, pa se i u ovom slučaju koristi unakrsna validacija za proveru performansi modela. Meri se i vreme obučavanja modela.

2.1. Linearni modeli

Hipoteza linearnih modela ima oblik linearne kombinacije odlika i težinskih parametara. Cilj liearnih modela je minimizacija funkcije greške koja je oblika srednje kvadratne greške. Minimizacija funkcije greške se vrši pomoću gradijentnog spusta, gde se u svakoj iteraciji ažuriraju težinski parametri dok ne konvergiraju ili se obradi maksimalan broj iteracija. Od linearnih modela iz biblioteke *scikit-learn* razmatraju se *LinearRegression, Ridge, Lasso, ElasticNet* i *SGDRegressor*.

- **Linearna Regresija**

Najjednostavniji regresioni algoriram je Linearna regresija. S obzirom da je broj odlika veći od jedan, model predstavlja višestruku linearnu regresiju. Na slici 11. je prikazana instanca klase *LinearRegression*, kao i njene performanse u predviđanju izlaza.

```
167     ('Linear Regression', LinearRegression(), {}),
```

```
Linear Regression:
Training Score: 0.265
Median Absolute Error on Test Set: 0.950
Cross-Validation Median Absolute Error: 0.918 ± 0.022
Time required: 0.040s
```

Slika 11. Linearna Regresija

- **Grebena Regresija**

Grebena regresija predstavlja Linearnu regresiju sa L2 regularizacijom. Hiperparametar *alpha* označava jačinu regularizacije. Regularizacija utiče na funkciju greške i ažuriranje težinskih parametara. L2 regularizaciona funkcija je kvadrat druge norme vektora parametara:

$$\|w\|_2^2 = \sum_{i=1}^{n} {w_i}^2$$

Na slici 12. je prikazana instanca klase *Ridge*, kao i njene performanse u predviđanju izlaza, sa najboljim hiperparametrom.

```
168     ('Ridge Regression', Ridge(), {
169         'alpha': [0.0001, 0.001, 0.01, 0.1],
170     }),
```

```
Ridge Regression:
Best Hyperparameters: {'alpha': 0.0001}
Training Score: 0.265
Median Absolute Error on Test Set: 0.950
Cross-Validation Median Absolute Error: 0.918 ± 0.022
Time required: 0.079s
```

Slika 12. Grebena Regresija

- **LASSO Regresija**

LASSO regresija predstavlja Linearnu regresiju sa L1 regularizacijom. Hiperparametar *alpha* označava jačinu regularizacije. Regularizacija utiče na funkciju greške i ažuriranje težinskih parametara. L1 regularizaciona funkcija je prva norma vektora parametara:

$$\|w\|_1 = \sum_{i=1}^{n} |w_i|$$

Na slici 13. je prikazana instanca klase *Lasso*, kao i njene performanse u predviđanju izlaza, sa najboljim hiperparametrom.

```
171     ('Lasso Regression', Lasso(random_state=123), {
172         'alpha': [0.0001, 0.001, 0.01, 0.1],
173     }),
```

```
Lasso Regression:
Best Hyperparameters: {'alpha': 0.001}
Training Score: 0.265
Median Absolute Error on Test Set: 0.950
Cross-Validation Median Absolute Error: 0.918 ± 0.022
Time required: 0.127s
```

Slika 13. LASSO Regresija

- # ElasticNet Regresija

ElasticNet regresija predstavlja Linearnu regresiju sa L1 i L2 regularizacijom. Hiperparametar *alpha* označava jačinu regularizacije, dok hiperparametar *L1_ratio* označava u kojoj meri se koriste L1 i L2 regularizacije. Vrednost hiperparametra *L1_ratio* je između 0 i 1, pa se L1 koristi u *L1_ratio* meri, a L2 u *(1 - L1_ratio)* meri. Regularizacija utiče na funkciju greške i ažuriranje težinskih parametara. Na slici 14. je prikazana instanca klase *ElasticNet*, kao i njene performanse u predviđanju izlaza, sa najboljim hiperparametrima.

```
174     ('ElasticNet Regression', ElasticNet(random_state=123), {
175         'alpha': [0.0001, 0.001, 0.01, 0.1],
176         'l1_ratio': [0.1, 0.5, 0.7, 0.9]
177     }),
```

```
ElasticNet Regression:
Best Hyperparameters: {'alpha': 0.001, 'l1_ratio': 0.9}
Training Score: 0.265
Median Absolute Error on Test Set: 0.950
Cross-Validation Median Absolute Error: 0.918 ± 0.022
Time required: 0.463s
```

Slika 14. ElasticNet Regresija

- # Stohastički Gradijentni Spust

Stohastički gradijentni spust predstavlja poseban način obučavanje Linearne regresije. Grupni grdijentni spust prolazi kroz sav skup podataka da bi napravio jedan korak u obučavanju, što je skupa operacija ako je broj podataka velik. Za razliku na grupni, stohastičku gradijentni spust pravi korak u obučavanju nakon svake obrade podatka, što dovodi do brže konvergencije. Hiperparametar *alpha* označava jačinu regularizacije, dok hiperparametar *L1_ratio* označava u kojoj meri se koriste L1 i L2 regularizacije. Na slici 15. je prikazana instanca klase *SGDRegressor*, kao i njene performanse u predviđanju izlaza, sa najboljim hiperparametrima.

```
178     ('SGD Regression', SGDRegressor(random_state=123), {
179         'alpha': [0.0001, 0.001, 0.01, 0.1],
180         'penalty': ['l2', 'l1', 'elasticnet'],
181         'l1_ratio': [0.1, 0.5, 0.7, 0.9, 0.95, 0.99, 1],
182     }),
```

```
SGD Regression:
Best Hyperparameters: {'alpha': 0.01, 'l1_ratio': 0.7, 'penalty': 'elasticnet'}
Training Score: 0.261
Median Absolute Error on Test Set: 0.935
Cross-Validation Median Absolute Error: 0.908 ± 0.031
Time required: 5.010s
```

Slika 15. Stohastički Gradijentni Spust

2.2. Modeli stabla odlučivanja

Stabla odlučivanja predstavljaju modele koji se najčešće koriste za klasifikaciju podataka, ali se mogu koristiti i u regresionim problemima. Strukturu stabla čine čvorovi i grane, gde čvorovi predstavljaju ulazne parametre a grane moguće vrednosti tih parametara. Stabla odlučivanja se grade od korena ka listovima gde se skup podataka za obučavanje u svakom nivou stabla deli pod određenim kriterijumom. Iz biblioteke *scikit-learn* je iskorišćena klasa *DecisionTreeRegressor*.

- **Regresivno stablo odlučivanja**

Kod regresionog stabla odlučivanja, listovi ne predstavljaju klase vec usrednjene vrednosti izlaza podataka koji pripadaju tom listu. Hiperparametar *max_depth* označava maksimalnu dubinu/visinu stabla, a *min_samples_split* označava najmanji broj podataka potreban da se čvor razgrana. Na slici 16. je prikazana instanca klase *DecisionTreeRegressor*, kao i njene performanse u predviđanju izlaza, sa najboljim hiperparametrima.

```
183     ('DecisionTree Regressor', DecisionTreeRegressor(random_state=123), {
184         'max_depth': [None, 10, 20],
185         'min_samples_split': [2, 10, 20]
186     }),
```

```
DecisionTree Regressor:
Best Hyperparameters: {'max_depth': 10, 'min_samples_split': 2}
Training Score: 0.635
Median Absolute Error on Test Set: 0.613
Cross-Validation Median Absolute Error: 0.637 ± 0.036
Time required: 2.530s
```

Slika 16. Regresivno Stablo Odlučivanja

2.3. Ensemble modeli

Ensemble modeli predstavljaju modele mašinskog učenja koji kombinuju više manjih modela u okviru jednog većeg. Svaki manji model uči i predviđa izlaz, dok veći model na neki način kombijuje sva ta predviđanja u jedno konačno. U rešavanju problema se razmatraju dve najpoznatije *ensemble* metode:

- *Random Forest*
- *Gradient-Boosted Trees*

Pored toga, postoje tehnike kombinovanja bilo kojih modela za predviđanje izlaza i u vidu glasanja se donosi konačno predviđanje izlaza podatka.

Iz biblioteke *scikit-learn* su iskorišćene klase *DecisionTreeRegressor, GradienBoostingRegressor, HistGradientBoostingRegressor* i *VotingRegressor*, a iz biblioteke *xgboost* klasa *XGBRegressor*.

- **Random Forest**

Ova *ensemble* metoda koristi skup Regresivnih stabala odlučivanja tokom obučavanja i izlaz predstavlja srednju vrednost predviđanja svih stabla odlučivanja. Svako stablo odlučivanja se može trenirati nad zasebnim skupom podataka, što može dovesti do umanjenja efekta preterane prilagođenosti. Hiperparametar *n_estimators* označava broj stabala odlučivanja koji se koriste, *max_depth* označava maksimalnu dubinu/visinu stabla, a *min_samples_split* označava najmanji broj podataka potreban da se čvor razgrana. Na slici 17. je prikazana instanca klase *RandomForestRegressor*, kao i njene performanse u predviđanju izlaza, sa najboljim hiperparametrima.

```
    ('RandomForest Regressor', RandomForestRegressor(
        random_state=123,
        n_estimators=100,
        max_depth=10,
        min_samples_split=2
    ), {}),
```

```
RandomForest Regressor:
Training Score: 0.652
Median Absolute Error on Test Set: 0.669
Cross-Validation Median Absolute Error: 0.632 ± 0.021
Time required: 28.433s
```

Slika 17. Random Forest

- **Gradient Boosting**

Ova *ensemble* metoda koristi skup Regresivnih stabala odlučivanja tokom obučavanja, ali ne od jednom. Algoritam počinje od jednostavnog stabla ili čak i lista, pa u svakoj iteraciji računa funkciju greške i na osnovu nje konstruiše novo stablo čije predviđanje utiče na konačno predviđanje u određenoj meri. Posle određenog broja iteracija, formira se „šuma“ stabala koji utiču na konačno predviđanje izlaza. Na taj način se postižu preciznija predviđanja. Hiperparametar *n_estimators* označava broj stabala odlučivanja koji se kreiraju iterativno, *max_depth* označava maksimalnu dubinu/visinu stabla, a *learning_rate* označava uticaj predviđanja novih stabala. Na slici 18. je prikazana instanca klase *GradentBoostingRegressor*, njene performanse u predviđanju izlaza, sa najboljim hiperparametrima.

```
    ('Gradient Boosting Regressor', GradientBoostingRegressor(
        random_state=123,
        n_estimators=100,
        max_depth=7,
        learning_rate=0.1
    ), {}),
```

```
Gradient Boosting Regressor:
Training Score: 0.801
Median Absolute Error on Test Set: 0.695
Cross-Validation Median Absolute Error: 0.646 ± 0.019
Time required: 29.956s
```

Slika 18. Gradient Boosting

- # Histogram-based Gradient Boosting

Model vrlo sličan kao *Gradient Boosting*. Jedina razlika je algoritam pomoću kog se granaju čvorovi u stablima. Brže obučavanje se postiže ovim modelom, nego kod *Gradient Boosting*-a, pa se koristi kad je broj podataka u skupu za obučavanje velik (preko 10.000 podataka). Hiperparametar *max_iter* označava broj stabala odlučivanja koji se kreiraju iterativno, *max_depth* označava maksimalnu dubinu/visinu stabla, a *learning_rate* označava uticaj predviđanja novih stabala. Na slici 19. je prikazana instanca klase *HistGradientBoostingRegressor*, njene performanse u predviđanju izlaza sa najboljim hiperparametrima.

```
('HistGradientBoosting Regression', HistGradientBoostingRegressor(
    random_state=123,
    max_depth=10,
    max_iter=300
), {
    'learning_rate': [0.01, 0.1, 0.2]
}),
```

```
HistGradientBoosting Regression:
Best Hyperparameters: {'learning_rate': 0.1}
Training Score: 0.803
Median Absolute Error on Test Set: 0.673
Cross-Validation Median Absolute Error: 0.667 ± 0.024
Time required: 24.878s
```

Slika 19. Histogram-based Gradient Boosting

- # Regresivno glasanje

Iako *ensemble* metode kombinuju veliki broj stabla odlučivanja, postoje meta-modeli koji kombinuju rezultate predviđanja skupa modela. Klasa *VotingRegressor* kao parametar uzima listu modela koji se treniraju nad istim skupom podataka i čiji se rezultati predviđanja usrednjavaju, što postaje konačni rezultat predviđanja. Na slici 20. je prikazana instanca klase *VotingRegressor*, koja koristi tri različita modela za predviđanje izlaza.

```
('Voting Regressor', VotingRegressor(estimators=[
    ('dt', DecisionTreeRegressor(random_state=123, max_depth=10, min_samples_split=2)),
    ('hgb', HistGradientBoostingRegressor(random_state=123, max_depth=10, max_iter=300)),
    ('gb', GradientBoostingRegressor(random_state=123, n_estimators=100, max_depth=7))
]), {}),
```

```
Voting Regressor:
Training Score: 0.778
Median Absolute Error on Test Set: 0.658
Cross-Validation Median Absolute Error: 0.643 ± 0.029
Time required: 41.693s
```

Slika 20. Regresivno glasanje

- **Extreme Gradient Boosting**

Model predstavlja naprednu implementaciju Gradient Boosting modela, time što je fleksibilniji i efikasniji. Često se koristi u realnom svetu i na takmičenjima za mašinsko učenje. Uključuje L1 i L2 regularizacije, paralelno procesiranje, odsecanje grana stabla, ugrađena unakrsna validacija, ... Na slici 21. je prikazana instanca klase *XGBRegressor* i njene performanse sa odabranim hiperparametrima.

```
211     ('XGBoost Regressor', XGBRegressor(
212         objective='reg:squarederror',
213         random_state=123,
214         max_depth=7
215     ), {
216         'n_estimators': [50, 100, 200],
217         'learning_rate': [0.01, 0.1, 0.2]
218     }),
```

```
XGBoost Regressor:
Best Hyperparameters: {'learning_rate': 0.1, 'n_estimators': 200}
Training Score: 0.878
Median Absolute Error on Test Set: 0.677
Cross-Validation Median Absolute Error: 0.646 ± 0.010
Time required: 21.977s
```

Slika 21. Extreme Gradient Boosting

2.4. Najbliži susedi

Kao i stabla odlučivanja, najčešće se koriste u klasifikaciji podataka, ali se mogu koristiti i u regresivnim problemima. Iz biblioteke *scikit-learn* je upotrebljena klase *KNeighborsRegressor*.

- **K Najbližih Suseda**

Glavni hiperparametar ovog modela je *n_neighbors* koji predstavlja broj suseda koji se razmatra u predikcionoj odluci. Kod problema regresije, najbliži sisedi usrednjavaju svoju izlaznu vrednosti koja postaje predviđena izlazna vrednosti podatka. Broj suseda može biti $\sqrt{N}$, koren broja podataka za obučavanje i poželjno je da bude neparna vrednost. Implementacija razmatra nekoliko vrednsoti broja komšija kako bi se poredile performanse modela. Na slici 22. su prikazane instance klase *KNeighborsRegressor* i njene performanse sa različitim vrednostima broja suseda.

```python
print("\n\nK Nearest Neighbors Regression:")
knn_dict = {
    "K": [],
    "Score": [],
    "MedAE": [],
    "CV_MedAE": []
}

k = int(np.sqrt(len(x_train)))
k += 1 - k % 2

for n in range(5, k+1, 6):
    t1 = time.time()

    model = KNeighborsRegressor(n_neighbors=n)
    cv_scores = cross_val_score(model, x_train, y_train, cv=kf, scoring='neg_median_absolute_error')
    model.fit(x_train, y_train)
    pred = model.predict(x_test)

    knn_dict["K"].append(n)
    knn_dict["Score"].append(f"{model.score(x_train, y_train):.3f}")
    knn_dict["MedAE"].append(f"{median_absolute_error(y_test, pred):.3f}")
    knn_dict["CV_MedAE"].append(f"{-cv_scores.mean():.3f} ± {cv_scores.std():.3f}")

    t2 = time.time() - t1
    print(f"Training model for K = {n}, t = {t2:.3f}s")

print("\nResults:")
knn_results = pd.DataFrame(data=knn_dict)
print(knn_results)
```

```
Results:
     K  Score  MedAE       CV_MedAE
0    5  0.559  0.740  0.728 ± 0.035
1   11  0.489  0.736  0.702 ± 0.031
2   17  0.461  0.747  0.710 ± 0.036
3   23  0.446  0.761  0.710 ± 0.027
4   29  0.436  0.769  0.725 ± 0.017
5   35  0.426  0.763  0.732 ± 0.019
6   41  0.418  0.765  0.732 ± 0.018
7   47  0.412  0.779  0.734 ± 0.013
8   53  0.408  0.774  0.735 ± 0.012
9   59  0.404  0.763  0.738 ± 0.013
10  65  0.400  0.765  0.747 ± 0.014
11  71  0.395  0.768  0.750 ± 0.015
12  77  0.390  0.764  0.750 ± 0.017
```

Slika 22. K Najbližih Suseda

2.5. Neuralne mreže

Poslednja vrsta modela koja se razmatra su modeli neurlnih mreža. Iz biblioteke *scikit-learn* je upotrebljena klase *MLPRegressor*.

- **Multi-layer Perceptron**

Sastoji se iz tri glavna sloja. Ulazni sloj koji prima odlike podatka, skriveni sloj koji predstavlja jedan ili više slojeva između ulaznog i izlaznog, gde mreža uči da prepozna šablone i odlike, i izlazni sloj koji za regresivne probleme ima samo jedan neuron koji predviđa izlaznu vrednost podatka. Svaki neuronprima ulazni signal od prethodnog sloja koji se množi sa određenim težinskom parametrom i dovodi do aktivacione funkcije. Aktivaciona funkcija određuje izlaznu vrednost neurona koja nastavlja put ka sledećem neuronima u nizu, dok ne stigne na izlaz. Nakon predikcije se vrši propagacija unazan kako bi se ažurirala mreža na osnovu funkcije greške.

Hiperparametri su *hidden_layers_size* koja označava broj neruona u skrivenom sloju, *max_iter* koji označava broj ciklusa rada algoritma za ceo skup podataka za obučavanje, *activation* koji označava koja aktivaciona funkcija se koristi u neuronu. Na slici 23. je prikazana instanca klase *MLPRegressor* i njene performanse sa sa najboljim vrednostima hiperparametara.

```
219     ('MLP Regressor', MLPRegressor(
220         random_state=123,
221         max_iter=500,
222         hidden_layer_sizes=(50, 50),
223         activation='relu'
224     ), {}),
```

```
MLP Regressor:
Training Score: 0.543
Median Absolute Error on Test Set: 0.760
Cross-Validation Median Absolute Error: 0.747 ± 0.019
Time required: 92.457s
```

Slika 23. Multi-layer Perceptron

3. Analiza performansi modela

Performanse se posmatraju na nivou metrike, preciznosti i brzine obučavanja. Za metriku i brzinu obučavanja su poželjne što niže vrednosti, a za preciznost predviđanja što veće vrednosti. Na osnovu rezultata rada svih algoritama, mogu se zaključiti sledeće osobine modela.

Jednostavniji modeli, tj. Linearni modeli za predviđanje imaju identične rezultate predviđanja i metrike ali i najslabije performanse u vidu R^2 rezultata i metrike za unakrsnu validaciju, iako su vremenski najbrži. Iz te grupe se izdvaja model Stohastičkog gradijentnog spusta koji ima malo bolji rezultat metrike u odnosu na ostale Linearne modele.

U sredini rangiranja po performansama se nalaze model najbližih suseda i neuralne mreže. Iako se povećanjem broja suseda smanjuje preciznost i povećava metrika, smanjuje se greška od šuma i ispravlja se metrika na stvarnu vrednost. Neuralne mreže imaju bolje performanse od linearnih modela zbog određenog stepena nelinernosti u podacima, jer mogu modelirati kompleksne nelinearne odnose odlika.

Najbolje performanse imaju *ensemble* modeli i stabla odlučivanja. Model *XGBRegressor* ima ubedljivo najveću preciznost od svih modela, iako je metrika slična kao kod ostalih *ensemble* modela.

Što se tiče brzine izvršavanja, jednostavniji modeli se najbrže obučavaju, dok se komplikovaniji modeli obučavaju sporije. Najduže vreme obučavanje je kod model neuralnih mreža, zbog velikog broja epoha (reda stotina ili hiljada). Jedna epoha podrazumeva obučavanje modela nad celim skupom podataka za obučavanje. Dodatno, pretraga po mreži za određivanje najboljih hiperparametara je vrlo skupa operacija koja troši dosta vremena, što dovodi do sporog obučavanja celokupnog procesa, iako je modelu stanju da se brzo obučava. Povećanje broja suseda koji se proveravaju takođe dovodi do sporijeg obučavanja modela.