Uczenie liniowej hipotezy dla problemu przewidywania cen nieruchomości

KWD

Anna Pawlikowska

Hubert Wąsik

23.01.2020

Spis treści

Rozdział 1

Teoretyczne zagadnienia projektu

1.1 Uczenie maszynowe

Uczeniem maszynowym określamy dziedzinę nauki programowania komputerów w sposób umożliwiający im uczenie się na podstawie otrzymanych danych, bez konieczności ich jawnego uczenia. Przykładowym oprogramowaniem wykorzystującym algorytmy uczenia maszynowego są oprogramowania wykorzystywane do rozpoznawania mowy, automatycznie nawigujące i sterujące np.: pojazdem lub statkiem kosmicznym bądź oprogramowania analizujące i klasyfikujące dane.

Uczenie maszynowe pozwala nam na rozwiązywanie problemów zbyt złożonych dla tradycyjnych metod lub dla takich, dla których nie istnieją jeszcze gotowe algorytmy. Korzystanie również z technik związanych z uczeniem maszynowym do analizowania dużej a nawet i olbrzymiej ilości danych może pomagać w odkrywaniu nieoczywistych wzorców. Proces taki nosi nazwę wydobywaniem danych.

Systemy uczenia maszynowego można podzielić w opraciu o stopień oraz rodzaje nadzorowania procesu uczenia:

1. uczenie nadzorowane
2. uczenie nienadzorowane
3. uczenie półnadzorowane
4. uczenie przez wzmacnianie

W poniższym projekcie wykorzystywane będą algorytmy regresji należące do algorytmów wykorzystywanych przez uczenie nadzorowane. W takim uczeniu dane uczące przekazywane są algorytmowi zawierając dołączone

rozwiązania problemu - etykiety. Typowym zadaniem takiego uczenia jest przewidywanie docelowej wartości numerycznej, takiej jak cena mieszkania lub procentowe prawdopodobieństwo zachorowania na daną chorobę. Taki typ zadania nosi nazwę regresji i aby wyuczyć dany system należy podać mu wiele przykładów wraz z etykietami i czynnikami prognostycznymi.

1.2 Regresja liniowa

Regresja liniowa służy do prognozowania wartości Y poprzez obliczenie wyważonej sumy cech wejściowych i stałej zwanej punktem obciążenia bądź punktem przecięcia. Poniżej zamieszczone zostało równanie prezentujące predykcję za pomocą modelu regresji liniowej:

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n \tag{1.1}$$

$\hat{y}$ - prognozowana wartość
n = liczba cech
x_i - wartość i-tej cechy
θ_j - j-ty parametr modelu

Trenowanie tak utworzonego modelu polega na zmierzeniu, jak dobrze model jest dopasowany do danych. W tym celu można wykorzystać metodę najmniejszych kwadratów, która pozwoli nam na znalezienie najbardziej optymalnej prostej. Za pomocą równania zamieszczonego poniżej wyliczamy błąd MSE hipotezy h wobec zestawu uczącego X.

$$MSE(X, h_\theta) = \frac{1}{m} \sum_{i=1}^{m} (\theta^T x^{(i)} - y^{(i)})^2 \tag{1.2}$$

1.3 Regresja wielomianowa

W przypadku, gdy dane nie układają się jednak liniowo na wykresie, możemy dodać potęgi każdej cechy w postaci nowej cechy, a następnie wytrenować model wobec tak rozszerzonego zestawu cech. Taką technikę nazywamy regresją wielomianową. Technika ta jest typową techniką wykorzystywaną do badania zależności krzywoliniowych.

1.4 Regularyzowane modele liniowe

W przypadku, gdy nasz otrzymany wyuczony model jest przetrenowany - algorytm “uczy się” poszczególnych przypadków, a nie ogólnych zasad -

należy dokonać jego regularyzacji, tzw. ograniczenia. Im mniej stopni swobody posiada nasz budowany model, tym trudniej jest go przetrenować wobec danych. Jednym z prostych sposób na regularyzację modelu np. wielomianowego jest po prostu zmniejszenie liczby stopni wielomianu. Natomiast w przypadku modelu liniowego regularyzację możemy otrzymać poprzez ograniczenie wag modelu. Taki zabieg osiągany jest przy wykorzystaniu regresji grzbietowej, LASSO oraz elastycznej siatki.

1.4.1 Regresja grzbietowa

Regresja grzbietowa stanowi regularyzowaną odmianę regresji liniowej, gdzie do funkcji kosztu zostaje dodany człon regularyzacyjny o postaci $\alpha \sum_{i=1}^{n} \theta_i^2$. Dzięki temu model jest zmuszony nie tylko do dopasowywania się do danych, ale również do utrzymywania jak najmniejszych wartości wag. Za pomocą hiperparametru określamy stopień regularyzacji modelu, gdzie w przypadku $\alpha = 0$ mamy do czynienia ze standardową regresją liniową.

$$J(\theta) = MSE(\theta) + \alpha \frac{1}{2} \sum_{i=1}^{n} \theta_i^2 \tag{1.3}$$

1.4.2 Regresja metodą LASSO

Regresja metodą LASSO stanowi kolejną regularyzowaną odmianę regresji liniowej. Dodawany jest tu człon regularyzacyjny do funkcji kosztu gdzie korzystamy z normy ℓ_1 wektora wag.

$$J(\theta) = MSE(\theta) + \alpha \sum_{i=1}^{n} |\theta_i| \tag{1.4}$$

Jedną z głównych własności regresji metodą LASSO jest to, iż dąży ona do całkowitej eliminacji wag w najmniej istotnych cechach poprzez zmianę ich wartości na 0. Metoda LASSO automatycznie przeprowadza dobór cech i generuje model rzadki zawierający niewiele niezerowych wag cech.

1.4.3 Regresja metodą elastycznej siatki

Metoda elastycznej siatki stanowi rozwiązanie pośrednie pomiędzy regresją grzbietową a regresją metodą LASSO - człon regularyzacyjny tworzy prosta kombinacja członów obydwu wspomnianych technik, która może być sterowana za pomocą współczynnika proporcji r. W przypadku gdy wartość r = 0, metoda elastycznej siatki staje się równoważna regresji grzbietowej, natomiast w przypadku r = 1 zachowuje się jak metoda LASSO. Można zaobserwować to na równaniu kosztu metody elastycznej siatki:

$$J(\theta) = MSE(\theta) + r\alpha \sum_{i=1}^{n} |\theta_i| + \frac{1-r}{2} \sum_{i=1}^{n} \theta_i^2 \tag{1.5}$$

Rozdział 2

Język oraz zastosowane biblioteki

2.1 Python

Głównym językiem programowania, z którego będziemy korzystać, jest Python. Jest to język programowania wysokiego poziomu ogólnego przeznaczenia, który posiada wiele bibliotek umożliwiających rozbudowanie jego funkcjonalności i wykorzystanie ich do stworzenia naszego modelu i nauczenia go.

2.2 Zastosowane biblioteki

2.2.1 Scikit-Learn

Scikit-Learn jest bezpłatną biblioteką wykorzystywaną do nauki algorytmów uczenia maszynowego. Pozwala na stosowanie algorytmów klasyfikacji, regresji oraz grupowania, zwiększania gradientu i wielu innych. Jest również przystosowane do współdziałania z innymi bibliotekami wykorzystywanych przy uczeniu maszynowym, takie jak NumPy, MatPlotLib czy Pandas.

2.2.2 Pandas

Biblioteka Pandas pozwala na łatwą manipulację i prezentację danych. Oferuje ona struktury danych oraz możliwość wykorzystywania operacji do manipulowania tabelami numerycznymi czy szeregami czasowymi.

2.2.3 MatPlotLib

Matplotlib jest biblioteką wykorzystywaną do tworzenia wykresów w środowisku Python. Zawiera ona API "pylab" zaprojektowane tak aby było jak najbardziej podobne do MATLABa, przez co jest łatwy do nauczenia przez

jego użytkowników. Matplotlib został napisany i jest utrzymywany głównie przez Johna Huntera i jest dostępny na licencji przypominającej licencję BSD.

Rozdział 3

Tworzenie i analiza modeli

3.1 Analiza całokształtu projektu

Zadaniem projektu jest utworzenie modelu cen mieszkań w Bostonie przy wykorzystaniu różnych metod regresji. Dane dot. mieszkań zawierają informacje, takie jak wskaźnik przestępczości na mieszkańca miasta, odsetek terenów mieszkalnych przeznaczonych pod działki o powierzchni ponad 25 000 stóp kwadratowych czy nawet stosunek uczniów do nauczycieli według miast. Zakładamy, iż model nauczy się z tych danych i przy użyciu tych metryk będzie w stanie przewidywać medianę cen mieszkań w dowolnym mieście w Bostonie. W celu sprawdzenia dokładności modelu obliczone zostaną odpowiednio MSE oraz współczynnik determinacji R^2.

3.2 Wykorzystywane dane

3.2.1 Źródło danych

Dane mieszkań w projekcie będą pobrane za pomocą modułu datasets z sklearn'u.

```
from sklearn.datasets import load_boston
boston_market_data = load_boston()
```

3.2.2 Wizualizacja danych

Za pomocą instrukcji

```
print(boston_market_data['DESCR'])
```

otrzymujemy następujący opis:

- liczba instancji: 506

- liczba atrybutów: 13 predykcyjnych cech liczbowych oraz 14ty atrybut będący medianą ceny, jest to też szukany atrybut
- Opis cech:

1. CRIM - wskaźnik przestępczości na mieszkańca
2. ZN - odsetek terenów mieszkalnych przeznaczonych pod działki o powierzchni ponad 25 000 stóp kwadratowych
3. INDUS - odsetek niedetalicznych akrów biznesowych
4. CHAS - domy znajdujące się bliżej rzeki Charles mają wyższe ceny, jest to zmienna fikcyjna
5. NOX - stężenie tlenków azotu (części na 10 milionów)
6. RM - średnia liczba pokoi na mieszkanie
7. AGE - odsetek jednostek zajmowanych przez właścicieli
8. DIS - ważone odległości do pięciu centrów zatrudnienia w Bostonie
9. RAD - wskaźnik dostępności do autostrad
10. TAX - pełnowartościowa stawka podatku od nieruchomości za 10.000 USD
11. PTRATIO - stosunek liczby uczniów do nauczycieli
12. B - 1000 (Bk - 0,63)^2
13. LSTAT - % mieszkańców niższego statusu
14. MEDV - Wartość szukana

W celu wyświetlenia wartości znajdujących się w pierwszych oraz ostatnich wierszach, podliczenia danych i wyświetlenia informacji na ich temat wykorzystaliśmy bibliotekę pandas umożliwiającą wizualizację danych w wygodny sposób:

```
import pandas as pd
market_p = pd.DataFrame(boston_market_data.data, \
            columns=[boston_market_data.feature_names])
market_p
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33
...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.67
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.08
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.64
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.48
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.88

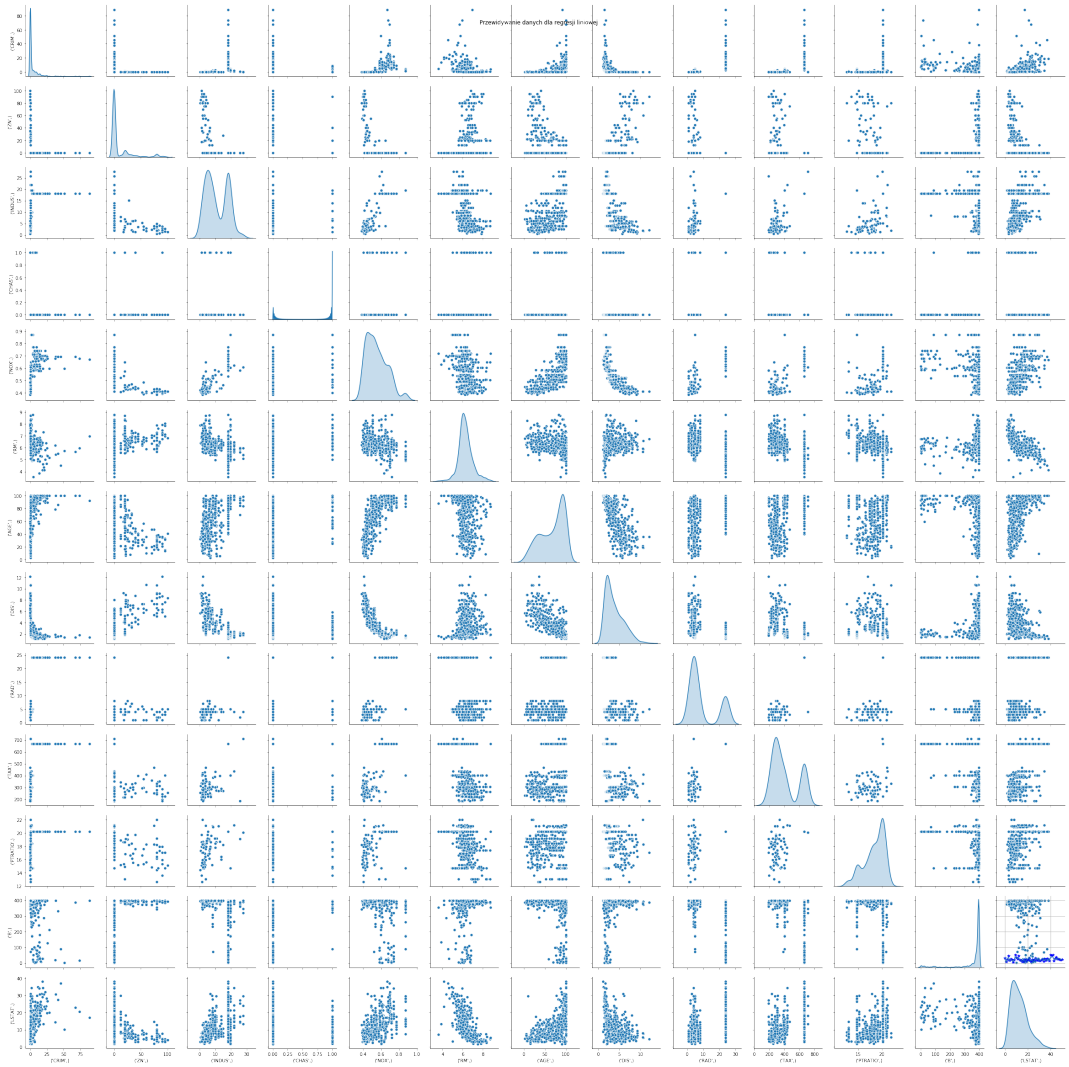
Powyższa tabela jest wynikiem działania przedstawionego wyżej fragmentu kodu. Potwierdza ona prawdziwość uzyskanych wcześniej informacji oraz pozwala na podejrzenie wartości poszczególnych cech.

```
market_p.describe()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.237154	18.455534	356.674032	12.653063
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.537116	2.164946	91.294864	7.141062
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.600000	0.320000	1.730000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.000000	17.400000	375.377500	6.950000
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.000000	19.050000	391.440000	11.360000
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.000000	20.200000	396.225000	16.955000
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000	396.900000	37.970000

Dodatkowo, możemy też wywołać metodę *describe*, aby zobaczyć tabelę z dodatkowymi informacjami na temat cech zbioru danych, takie jak minimalne i maksymalne wartości cech oraz kwintały.

```
import matplotlib.pyplot as plt
import seaborn as sb
%matplotlib inline
sb.pairplot(market_p, diag_kind="kde")
```



Wreszcie, możemy także skorzystać z bibliotek MatPlotLib and Seaborn aby przedstawić poszczególne cechy zbioru danych w postaci różnego typu wykresów, co pozwala na lepsze zorientowanie się w rozkładzie danych wejściowych.

3.2.3 Przygotowywanie danych

Standaryzacja danych pozwala na korektę danych w taki sposób, by tworzyły one jednolity zapis. Standaryzacja zestawu danych jest częstym wymogiem dla wielu estymatorów uczenia maszynowego. W przeciwnym wypadku mogą wyuczyć się one błędnego modelu. W celu przygotowania danych przeprowadziliśmy standaryzację wykorzystując StandardScaler, która usuwa średnią i skaluje do wariacji jednostkowej. Wynik próbkowy obliczany

jest jako:

$$z = \frac{x - u}{s} \tag{3.1}$$

gdzie U jest średnią próbek treningowych bądź posiada wartość zero w przypadku, gdy $with\_mean = False$. Wartość S jest natomiast standardowym odchyleniem próbek treningowych bądź posiada wartość 1, jeśli $with\_std = False$. Środkowanie i skalowanie danych odbywa się niezależnie dla każdej cechy poprzez obliczenie odpowiednich statystyk próbek w zbiorze uczącym. Średnia i odchylenie standardowe są następnie zapisywane do wykorzystania w późniejszych danych za pomocą transformacji. Wykonana instrukcja:

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
boston_scaled_data = scaler.fit_transform(boston_market_data['data'])
```

W następnym kroku przeprowadziliśmy podział danych na zbiór uczący i testowy:

```
from sklearn.model_selection import train_test_split

boston_train_data, boston_test_data, \
boston_train_target, boston_test_target = \
train_test_split(boston_scaled_data,boston_market_data['target'],\
test_size=0.1, random_state=10)
```

3.3 Tworzenie modelu

3.3.1 Regresja liniowa

W celu utworzenia modelu przy wykorzystaniu regresji liniowej użyliśmy następującego polecenia:

```
from sklearn.linear_model import LinearRegression

linear_regression = LinearRegression(normalize=True)
linear_regression.fit(boston_train_data, boston_train_target)
```

3.3.2 Regresja wielomianowa

W celu utworzenia modelu przy wykorzystaniu regresji wielomianowej użyliśmy następującego polecenia:

```
from sklearn.preprocessing import PolynomialFeatures

pt = PolynomialFeatures(2)
```

```
lr_forPoly = LinearRegression(normalize = True)
housig_train_poly = pt.fit_transform(boston_train_data)
housig_test_poly = pt.fit_transform(boston_test_data)
lr_forPoly.fit(housig_train_poly, boston_train_target)
```

3.3.3 Regresja grzbietowa

W celu utworzenia modelu przy wykorzystaniu regresji grzbietowej użyliśmy następującego polecenia:

```
from sklearn.linear_model import Ridge

ridge = Ridge(alpha=0.5, solver="cholesky")
ridge.fit(boston_train_data,boston_train_target)
```

3.3.4 Regresja metodą LASSO

W celu utworzenia modelu przy wykorzystaniu regresji metodą LASSO użyliśmy następującego polecenia:

```
from sklearn.linear_model import Lasso

lasso = Lasso(alpha=0.5, max_iter=1e5)
lasso.fit(boston_train_data,boston_train_target)
```

3.3.5 Regresja elastycznej siatki

W celu utworzenia modelu przy wykorzystaniu regresji elastycznej siatki użyliśmy następującego polecenia:

```
from sklearn.linear_model import ElasticNet

elastic = ElasticNet(alpha=0.5, l1_ratio=0.5, max_iter=1e5)
elastic.fit(boston_train_data,boston_train_target)
```

3.4 Ocena otrzymanych modeli

W celu oceny modeli zaimportowane zostały dwie metody:

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

3.4.1 Model regresji liniowej

Do odczytania trafności modelu oraz MSE wykorzystaliśmy następujące komendy:

```
print("Średni błąd kwadratowy naszego modelu dla regresji \\
liniowej wynosi: %.2f" % )
mean_squared_error(boston_test_target, \
linear_regression.predict(boston_test_data)))
print('Trafność modelu regresji liniowej: %.2f' % )
r2_score(boston_test_target, linear_regression.predict(boston_test_data)))
```

Oraz otrzymaliśmy następujący wynik:

```
Średni błąd kwadratowy naszego modelu dla regresji liniowej wynosi: 27.38
Trafność modelu regresji liniowej: 0.78
```

3.4.2 Model regresji wielomianowej

Do odczytania trafności modelu oraz MSE wykorzystaliśmy następujące komendy:

```
print("Średni błąd kwadratowy naszego modelu dla regresji \\
wielomianowej wynosi: %.2f" % )
mean_squared_error(boston_test_target, lr_forPoly.predict(housig_test_poly)))
print('Trafność modelu regresji wielomianowej: %.2f' % )
r2_score(boston_test_target, lr_forPoly.predict(housig_test_poly)))
```

Oraz otrzymaliśmy następujący wynik:

```
Średni błąd kwadratowy naszego modelu dla regresji wielomianowej wynosi: 16.41
Trafność modelu regresji wielomianowej: 0.87
```

3.4.3 Model regresji grzbietowej

Do odczytania trafności modelu oraz MSE wykorzystaliśmy następujące komendy:

```
print("Średni błąd kwadratowy naszego modelu dla regresji \\
grzbietowej wynosi: %.2f" % )
mean_squared_error(boston_test_target, ridge.predict(boston_test_data)))
print('Trafność modelu regresji grzbietowej: %.2f' % )
r2_score(boston_test_target, ridge.predict(boston_test_data)))
```

Oraz otrzymaliśmy następujący wynik:

```
Średni błąd kwadratowy naszego modelu dla regresji grzbietowej wynosi: 27.41
Trafność modelu regresji grzbietowej: 0.78
```

3.4.4 Model regresji metodą LASSO

Do odczytania trafności modelu oraz MSE wykorzystaliśmy następujące komendy:

```
print("Średni błąd kwadratowy naszego modelu dla regresji \\
LASSO wynosi: %.2f" %)
mean_squared_error(boston_test_target, lasso.predict(boston_test_data)))
print('Trafność modelu regresji LASSO: %.2f' % )
r2_score(boston_test_target, lasso.predict(boston_test_data)))
```

Oraz otrzymaliśmy następujący wynik:

```
Średni błąd kwadratowy naszego modelu dla regresji LASSO wynosi: 33.70
Trafność modelu regresji LASSO: 0.73
```

3.4.5 Model regresji elastycznej siatki

Do odczytania trafności modelu oraz MSE wykorzystaliśmy następujące komendy:

```
print("Średni błąd kwadratowy naszego modelu dla regresji \\
elastycznej siatki wynosi: %.2f" %)
mean_squared_error(boston_test_target, elastic.predict(boston_test_data)))
print('Trafność modelu regresji elastycznej siatki: %.2f' % )
r2_score(boston_test_target, elastic.predict(boston_test_data)))
```

Oraz otrzymaliśmy następujący wynik:

```
Średni błąd kwadratowy naszego modelu dla regresji elastycznej siatki wynosi: 35.89
Trafność modelu regresji elastycznej siatki: 0.71
```

3.5 Regularyzacja modelu

Po otrzymaniu modeli można je dostroić. Jedną z metod umożliwiającą regularyzację jest własnoręczne dobieranie wartości hiperparametrów, dopóki nie uzyskane będą ich znakomite kombinacje. Niemniej operacja ta byłaby zbyt długa i żmudna, stąd z pomocą przychodzi metoda przeszukiwania siatki.

W tej metodzie wystarczy podać jedynie interesujące nas hiperparametry oraz ich proponowane wartości, a wszystkie ich kombinacje zostaną ocenione za pomocą hiperparametrów dla modelu elastycznej siatki. Oszacowane zostaną wszystkie podane kombinacje dla wartości hiperparametrów, a następnie zwrócona zostanie najlepsza kombinacja.

W celu otrzymania najlepszych parametrów dla wykorzystywanych przez nas w projekcie modeli wykonaliśmy następujące instrukcje:

```
parameters=[{'alpha':[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.7, 0.9], \
'l1_ratio':[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.7, 0.9]}]
model = GridSearchCV(ElasticNet(), parameters, cv=3)
model.fit(boston_train_data,boston_train_target)
print(model.best_params_)
model.best_estimator_
```

Oraz otrzymaliśmy następujący wynik:

```
{'alpha': 0.1, 'l1_ratio': 0.1}
```

Podaną wartość alpha oraz $l1\_ration$ wprowadziliśmy do nowo utworzonych modeli: grzbietowej, LASSO oraz elastycznej siatki. Przeprowadzone zostały ponownie testy jakościowe poprzez obliczenie trafności modelu oraz MSE i otrzmaliśmy następujące wyniki:

```
Średni błąd kwadratowy naszego modelu dla
regresji grzbietowej przy pomocy regularyzacji wynosi: 27.38
Trafność modelu regresji grzbietowej przy pomocy regularyzacji: 0.78

Średni błąd kwadratowy naszego modelu dla
regresji LASSO przy pomocy regularyzacji  wynosi: 28.56
Trafność modelu regresji LASSO przy pomocy regularyzacji: 0.77

Średni błąd kwadratowy naszego modelu dla
regresji elastycznej siatki przy pomocy regularyzacji wynosi: 29.41
Trafność modelu regresji elastycznej siatki przy pomocy regularyzacji: 0.77
```

3.6 Test modeli

W celu przetestowania przewidywania wartości dla danego ID mieszkania znajdującego się w zbiorze danych wykorzystaliśmy następujące polecenie:

```
id = 2
print()
print("Wartosc rzeczywista: " + str(boston_test_target[id]))

linear_regression_prediction = \
linear_regression.predict(boston_test_data[id,:].reshape(1,-1))
print()
print("Regresja liniowa: ")
print("Otrzymalismy: " + str(linear_regression_prediction))

lr_forPoly_pre = lr_forPoly.predict(housig_test_poly[id,:].reshape(1,-1))
print()
```

```
print("Regresja wielomianowa: ")
print("Otrzymalismy: " + str(lr_forPoly_pre))

ridge_pre = ridge.predict(boston_test_data[id,:].reshape(1,-1))
print()
print("Regresja grzbietowa: ")
print("Otrzymalismy: " + str(ridge_pre))

lasso_pre = lasso.predict(boston_test_data[id,:].reshape(1,-1))
print()
print("Regresja LASSO: ")
print("Otrzymalismy: " + str(lasso_pre))

elsatic_pre = elastic.predict(boston_test_data[id,:].reshape(1,-1))
print()
print("Regresja elastycznej siatki: ")
print("Otrzymalismy: " + str(elsatic_pre))

ridge_reg_pre = ridge_reg.predict(boston_test_data[id,:].reshape(1,-1))
print()
print("Regresja grzbietowa po regularyzacji: ")
print("Otrzymalismy: " + str(ridge_reg_pre))

lasso_reg_pre = lasso_reg.predict(boston_test_data[id,:].reshape(1,-1))
print()
print("Regresja LASSO po regularyzacji: ")
print("Otrzymalismy: " + str(lasso_reg_pre))

elastic_reg_pre = elastic_reg.predict(boston_test_data[id,:].reshape(1,-1))
print()
print("Regresja elastycznej siatki po regularyzacji: ")
print("Otrzymalismy: " + str(elastic_reg_pre))
```

Oraz otrzymaliśmy następujący wynik:

```
Wartosc rzeczywista: 22.0

Regresja liniowa:
Otrzymalismy: [21.0195741]

Regresja wielomianowa:
Otrzymalismy: [20.125]
```

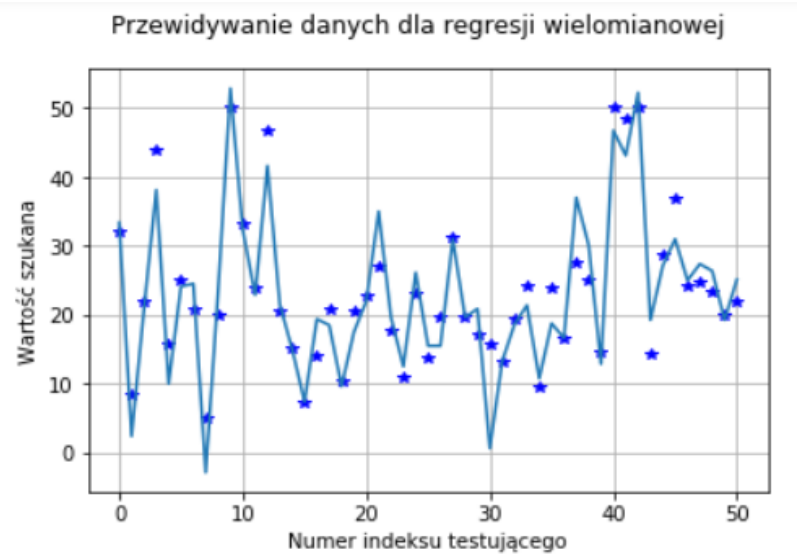
```
Regresja grzbietowa:
Otrzymalismy: [21.03616056]

Regresja LASSO:
Otrzymalismy: [22.79925667]

Regresja elastycznej siatki:
Otrzymalismy: [22.68488368]

Regresja grzbietowa po regularyzacji:
Otrzymalismy: [21.02292773]

Regresja LASSO po regularyzacji:
Otrzymalismy: [21.73694708]

Regresja elastycznej siatki po regularyzacji:
Otrzymalismy: [21.76059102]
```

Po przeprowadzeniu testu na jednej danej oraz otrzymaniu procentowej dokładności utwórzonych modeli, przeszliśmy do utworzenia wykresów dla wszystkich 51 jednostek testujących naszego modelu, na którym niebieskimi punktami oznaczono wartości docelowe, natomiast nebieską linią wartości otrzymane za pomocą modeli.

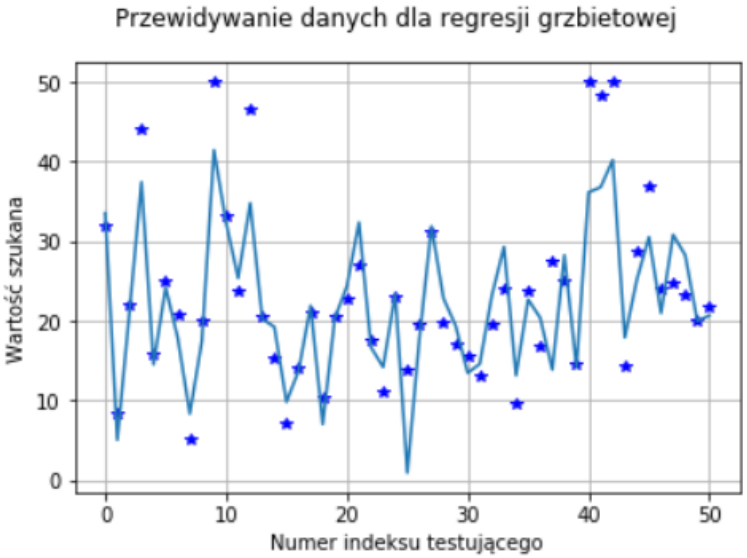
Wykorzystany kod:

```
x = []
y = []
y2 = []

for i in range(51):
    x.append(i)
    y.append(boston_test_target[i])
    y2.append(linear_regression.predict(boston_test_data[i].reshape(1,-1)))

plt.grid()
plt.plot(x, y, '*b')
plt.plot(x, y2)
plt.suptitle('Przewidywanie danych dla regresji liniowej')
plt.xlabel('Numer indeksu testującego')
plt.ylabel('Wartość szukana')
plt.show()
```

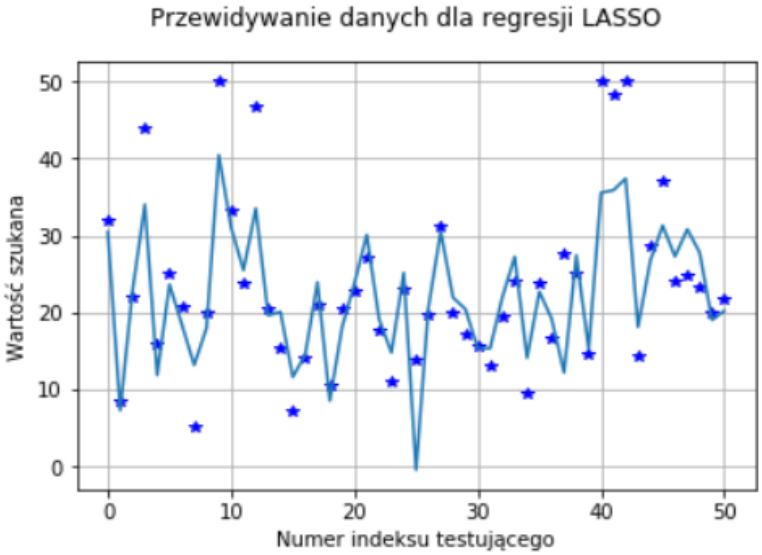

```
x = []
y = []
y2 = []

for i in range(51):
    x.append(i)
    y.append(boston_test_target[i])
    y2.append(lr_forPoly.predict(housig_test_poly[i].reshape(1,-1)))
plt.grid()
plt.plot(x, y, '*b')
plt.plot(x, y2)
plt.suptitle('Przewidywanie danych dla regresji wielomianowej')
plt.xlabel('Numer indeksu testującego')
plt.ylabel('Wartość szukana')
plt.show()
```




```
x = []
y = []
y2 = []

for i in range(51):
    x.append(i)
    y.append(boston_test_target[i])
    y2.append(ridge.predict(boston_test_data[i].reshape(1,-1)))

plt.grid()
plt.plot(x, y, '*b')
plt.plot(x, y2)
plt.suptitle('Przewidywanie danych dla regresji grzbietowej')
plt.xlabel('Numer indeksu testującego')
plt.ylabel('Wartość szukana')
plt.show()
```

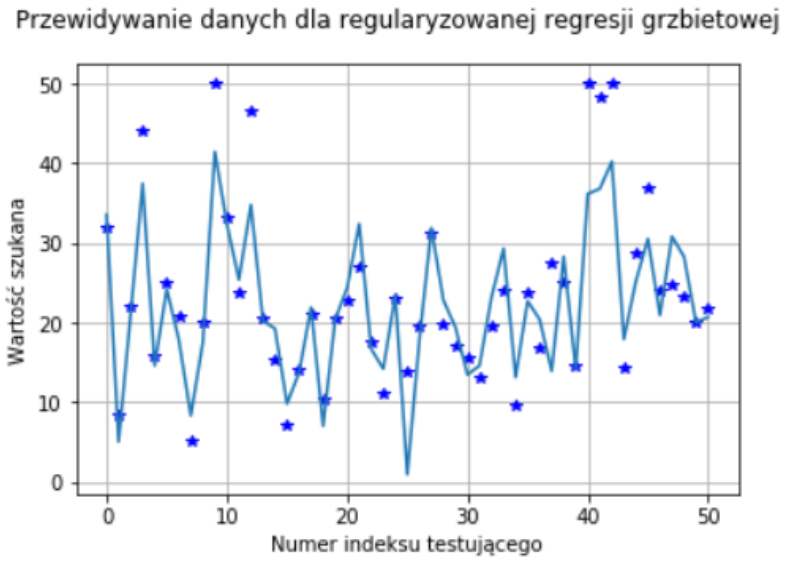



```
x = []
y = []
y2 = []

for i in range(51):
    x.append(i)
    y.append(boston_test_target[i])
    y2.append(lasso.predict(boston_test_data[i].reshape(1,-1)))

plt.grid()
plt.plot(x, y, '*b')
plt.plot(x, y2)
plt.suptitle('Przewidywanie danych dla regresji LASSO')
plt.xlabel('Numer indeksu testującego')
plt.ylabel('Wartość szukana')
plt.show()
```

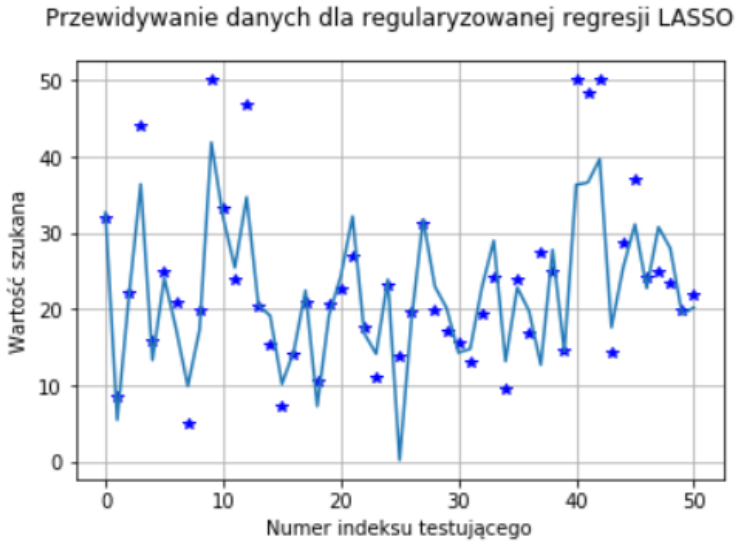



```
x = []
y = []
y2 = []

for i in range(51):
    x.append(i)
    y.append(boston_test_target[i])
    y2.append(elastic.predict(boston_test_data[i].reshape(1,-1)))

plt.grid()
plt.plot(x, y, '*b')
plt.plot(x, y2)
plt.suptitle('Przewidywanie danych dla regresji elastycznej siatki')
plt.xlabel('Numer indeksu testującego')
plt.ylabel('Wartość szukana')
plt.show()
```


```
x = []
y = []
y2 = []

for i in range(51):
    x.append(i)
    y.append(boston_test_target[i])
    y2.append(ridge_reg.predict(boston_test_data[i].reshape(1,-1)))

plt.grid()
plt.plot(x, y, '*b')
plt.plot(x, y2)
plt.suptitle('Przewidywanie danych dla regularyzowanej regresji grzbietowej')
plt.xlabel('Numer indeksu testującego')
plt.ylabel('Wartość szukana')
plt.show()
```




```
x = []
y = []
y2 = []

for i in range(51):
    x.append(i)
    y.append(boston_test_target[i])
    y2.append(lasso_reg.predict(boston_test_data[i].reshape(1,-1)))

plt.grid()
plt.plot(x, y, '*b')
plt.plot(x, y2)
plt.suptitle('Przewidywanie danych dla regularyzowanej regresji LASSO')
plt.xlabel('Numer indeksu testującego')
plt.ylabel('Wartość szukana')
plt.show()
```




```
x = []
y = []
y2 = []

for i in range(51):
    x.append(i)
    y.append(boston_test_target[i])
    y2.append(elastic_reg.predict(boston_test_data[i].reshape(1,-1)))

plt.grid()
plt.plot(x, y, '*b')
plt.plot(x, y2)
plt.suptitle('Przewidywanie danych dla regularyzowanej regresji \\
elastycznej siatki')
plt.xlabel('Numer indeksu testującego')
plt.ylabel('Wartość szukana')
plt.show()
```


Przewidywanie danych dla regularyzowanej regresji elastycznej siatki
Wartość szukana
Numer indeksu testującego
0
10
20
30
40
50

Rozdział 4

Wnioski

Nazwa modelu	Wartość R2	Wartość MSE
Regresja liniowa	0.78	27.38
Regresja wielomianowa	0.87	16.41
Regresja grzbietowa	0.78	27.41
Regresja LASSO	0.73	33.70
Regresja elastycznej siatki	0.71	35.89
Regresja grzbietowa prze pomocy regularyzacji	0.78	27.38
Regresja LASSO przy pomocy regularyzacji	0.77	28.56
Regresja elastycznej siatki przy pomocy regularyzacji	0.77	29.41

Utworzone przez nas modele prawidłowo przewidują szacowane wartości kosztów mieszkań w bostonie. Najlepszy wynik uzyskał model otrzymany poprzez regresję wielomianową, z trafnością 87%. Należy również zauważyć, iż regularyzacja modelu niewiele zmieniła dla regresji grzbietowej, dla regresji LASSO precyzja zwiększyła się aż o 4%, natomiast dla regresji elastycznej siatki aż o 6%.

Rozdział 5

Kod źródłowy

```
from sklearn.datasets import load_boston
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt
import seaborn as sb
import pandas as pd
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.metrics import accuracy_score

boston_market_data = load_boston()

#Prezentacja danych
#%matplotlib inline
#market_p = pd.DataFrame(boston_market_data.data, \
#            columns=[boston_market_data.feature_names])
#market_p
#market_p.describe()
#sb.pairplot(market_p, diag_kind="kde")

#Standaryzacja danych
scaler = StandardScaler()
boston_scaled_data = scaler.fit_transform(boston_market_data['data'])
```

```
#Załadowanie danych dla regresji liniowych i wielomianowej
boston_train_data, boston_test_data, \
boston_train_target, boston_test_target = \
train_test_split(boston_scaled_data,boston_market_data['target'],
test_size=0.1, random_state=1010)

#Utworzenie regresji liniowej
linear_regression = LinearRegression(normalize=True)
linear_regression.fit(boston_train_data, boston_train_target)

#Utworzenie regresji wielomianowej
pt = PolynomialFeatures(2)
lr_forPoly = LinearRegression(normalize = True)
housig_train_poly = pt.fit_transform(boston_train_data)
housig_test_poly = pt.fit_transform(boston_test_data)
lr_forPoly.fit(housig_train_poly, boston_train_target)

#Utworzenie regresji grzbietowej
ridge = Ridge(alpha=0.5, solver="cholesky")
ridge.fit(boston_train_data,boston_train_target)

#Utworzenie regresji LASSO
lasso = Lasso(alpha=0.5, max_iter=1e5)
lasso.fit(boston_train_data,boston_train_target)

#Utworzenie regresji elastycznej siatki
elastic = ElasticNet(alpha=0.5, l1_ratio=0.5, max_iter=1e5)
elastic.fit(boston_train_data,boston_train_target)

#Wyregulowanie modelu - metoda przeszukiwania siatki
parameters=[{'alpha':[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.7, 0.9], \
'l1_ratio':[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.7, 0.9]}]
model = GridSearchCV(ElasticNet(), parameters, cv=3)
model.fit(boston_train_data,boston_train_target)
print(model.best_params_)
model.best_estimator_

#Utworzenie regresji grzbietowej przy pomocy regularyzacji
ridge_reg = Ridge(alpha=0.1, solver="cholesky")
ridge_reg.fit(boston_train_data,boston_train_target)

#Utworzenie regresji LASSO przy pomocy regularyzacji
lasso_reg = Lasso(alpha=0.1, max_iter=1e5)
```

```
lasso_reg.fit(boston_train_data,boston_train_target)

#Utworzenie regresji elastycznej siatki przy pomocy regularyzacji
elastic_reg = ElasticNet(alpha=0.1, l1_ratio=0.4, max_iter=1e5)
elastic_reg.fit(boston_train_data,boston_train_target)

########## Sprawdzanie MSSE oraz R2
#Regresja liniowa
print()
print("Średni błąd kwadratowy naszego modelu dla regresji \\
liniowej wynosi: %.2f" % )
mean_squared_error(boston_test_target, \
linear_regression.predict(boston_test_data)))
print('Trafność modelu regresji liniowej: %.2f' % )
r2_score(boston_test_target, linear_regression.predict(boston_test_data)))

#Regresja wielomianowa
print()
print("Średni błąd kwadratowy naszego modelu dla regresji \\
wielomianowej wynosi: %.2f" % )
mean_squared_error(boston_test_target, lr_forPoly.predict(housig_test_poly)))
print('Trafność modelu regresji wielomianowej: %.2f' % )
r2_score(boston_test_target, lr_forPoly.predict(housig_test_poly)))

#Regresja grzbietowa
print()
print("Średni błąd kwadratowy naszego modelu dla regresji \\
grzbietowej wynosi: %.2f" %)
mean_squared_error(boston_test_target, ridge.predict(boston_test_data)))
print('Trafność modelu regresji grzbietowej: %.2f' % )
r2_score(boston_test_target, ridge.predict(boston_test_data)))

#Regresja LASSO
print()
print("Średni błąd kwadratowy naszego modelu dla regresji \\
LASSO wynosi: %.2f" % )
mean_squared_error(boston_test_target, lasso.predict(boston_test_data)))
print('Trafność modelu regresji LASSO: %.2f' % )
r2_score(boston_test_target, lasso.predict(boston_test_data)))

#Regresja elastycznej siatki
print()
print("Średni błąd kwadratowy naszego modelu dla regresji \\
elastycznej siatki wynosi: %.2f" % )
```

```
mean_squared_error(boston_test_target, elastic.predict(boston_test_data)))
print('Trafność modelu regresji elastycznej siatki: %.2f' % )
r2_score(boston_test_target, elastic.predict(boston_test_data)))

#Regresja grzbietowa przy pomocy regularyzacji
print()
print("Średni błąd kwadratowy naszego modelu dla regresji \\
grzbietowej przy pomocy regularyzacji wynosi: %.2f" % )
mean_squared_error(boston_test_target, ridge_reg.predict(boston_test_data)))
print('Trafność modelu regresji grzbietowej przy pomocy regularyzacji: %.2f' % )
r2_score(boston_test_target, ridge_reg.predict(boston_test_data)))

#Regresja LASSO przy pomocy regularyzacji
print()
print("Średni błąd kwadratowy naszego modelu dla regresji \\
LASSO przy pomocy regularyzacji  wynosi: %.2f" % )
mean_squared_error(boston_test_target, lasso_reg.predict(boston_test_data)))
print('Trafność modelu regresji LASSO przy pomocy regularyzacji: %.2f' % )
r2_score(boston_test_target, lasso_reg.predict(boston_test_data)))

#Regresja elastycznej siatki przy pomocy regularyzacji
print()
print("Średni błąd kwadratowy naszego modelu dla regresji \\
elastycznej siatki przy pomocy regularyzacji wynosi: %.2f" % )
mean_squared_error(boston_test_target, elastic_reg.predict(boston_test_data)))
print('Trafność modelu regresji \\
elastycznej siatki przy pomocy regularyzacji: %.2f' % )
r2_score(boston_test_target, elastic_reg.predict(boston_test_data)))

#TEST
print()
print("#############")
id = 2
print("    TEST    ")
print("#############")

print()
print("Wartosc rzeczywista: " + str(boston_test_target[id]))

linear_regression_prediction = \
linear_regression.predict(boston_test_data[id,:].reshape(1,-1))
print()
print("Regresja liniowa: ")
print("Otrzymalismy: " + str(linear_regression_prediction))
```

```
lr_forPoly_pre = lr_forPoly.predict(housig_test_poly[id,:].reshape(1,-1))
print()
print("Regresja wielomianowa: ")
print("Otrzymalismy: " + str(lr_forPoly_pre))

ridge_pre = ridge.predict(boston_test_data[id,:].reshape(1,-1))
print()
print("Regresja grzbietowa: ")
print("Otrzymalismy: " + str(ridge_pre))

lasso_pre = lasso.predict(boston_test_data[id,:].reshape(1,-1))
print()
print("Regresja LASSO: ")
print("Otrzymalismy: " + str(lasso_pre))

elsatic_pre = elastic.predict(boston_test_data[id,:].reshape(1,-1))
print()
print("Regresja elastycznej siatki: ")
print("Otrzymalismy: " + str(elsatic_pre))

ridge_reg_pre = ridge_reg.predict(boston_test_data[id,:].reshape(1,-1))
print()
print("Regresja grzbietowa po regularyzacji: ")
print("Otrzymalismy: " + str(ridge_reg_pre))

lasso_reg_pre = lasso_reg.predict(boston_test_data[id,:].reshape(1,-1))
print()
print("Regresja LASSO po regularyzacji: ")
print("Otrzymalismy: " + str(lasso_reg_pre))

elastic_reg_pre = elastic_reg.predict(boston_test_data[id,:].reshape(1,-1))
print()
print("Regresja elastycznej siatki po regularyzacji: ")
print("Otrzymalismy: " + str(elastic_reg_pre))

x = []
y = []
y2 = []

for i in range(51):
    x.append(i)
    y.append(boston_test_target[i])
    y2.append(linear_regression.predict(boston_test_data[i].reshape(1,-1)))
```

```
plt.grid()
plt.plot(x, y, '*b')
plt.plot(x, y2)
plt.suptitle('Przewidywanie danych dla regresji liniowej')
plt.xlabel('Numer indeksu testującego')
plt.ylabel('Wartość szukana')
plt.show()

x = []
y = []
y2 = []

for i in range(51):
    x.append(i)
    y.append(boston_test_target[i])
    y2.append(lr_forPoly.predict(housig_test_poly[i].reshape(1,-1)))
plt.grid()
plt.plot(x, y, '*b')
plt.plot(x, y2)
plt.suptitle('Przewidywanie danych dla regresji wielomianowej')
plt.xlabel('Numer indeksu testującego')
plt.ylabel('Wartość szukana')
plt.show()


x = []
y = []
y2 = []

for i in range(51):
    x.append(i)
    y.append(boston_test_target[i])
    y2.append(ridge.predict(boston_test_data[i].reshape(1,-1)))

plt.grid()
plt.plot(x, y, '*b')
plt.plot(x, y2)
plt.suptitle('Przewidywanie danych dla regresji grzbietowej')
plt.xlabel('Numer indeksu testującego')
plt.ylabel('Wartość szukana')
plt.show()
```

```
x = []
y = []
y2 = []

for i in range(51):
    x.append(i)
    y.append(boston_test_target[i])
    y2.append(lasso.predict(boston_test_data[i].reshape(1,-1)))

plt.grid()
plt.plot(x, y, '*b')
plt.plot(x, y2)
plt.suptitle('Przewidywanie danych dla regresji LASSO')
plt.xlabel('Numer indeksu testującego')
plt.ylabel('Wartość szukana')
plt.show()


x = []
y = []
y2 = []

for i in range(51):
    x.append(i)
    y.append(boston_test_target[i])
    y2.append(elastic.predict(boston_test_data[i].reshape(1,-1)))

plt.grid()
plt.plot(x, y, '*b')
plt.plot(x, y2)
plt.suptitle('Przewidywanie danych dla regresji elastycznej siatki')
plt.xlabel('Numer indeksu testującego')
plt.ylabel('Wartość szukana')
plt.show()



x = []
y = []
y2 = []

for i in range(51):
    x.append(i)
    y.append(boston_test_target[i])
```

```
    y2.append(ridge_reg.predict(boston_test_data[i].reshape(1,-1)))

plt.grid()
plt.plot(x, y, '*b')
plt.plot(x, y2)
plt.suptitle('Przewidywanie danych dla regularyzowanej regresji grzbietowej')
plt.xlabel('Numer indeksu testującego')
plt.ylabel('Wartość szukana')
plt.show()


x = []
y = []
y2 = []

for i in range(51):
    x.append(i)
    y.append(boston_test_target[i])
    y2.append(lasso_reg.predict(boston_test_data[i].reshape(1,-1)))

plt.grid()
plt.plot(x, y, '*b')
plt.plot(x, y2)
plt.suptitle('Przewidywanie danych dla regularyzowanej regresji LASSO')
plt.xlabel('Numer indeksu testującego')
plt.ylabel('Wartość szukana')
plt.show()



x = []
y = []
y2 = []

for i in range(51):
    x.append(i)
    y.append(boston_test_target[i])
    y2.append(elastic_reg.predict(boston_test_data[i].reshape(1,-1)))

plt.grid()
plt.plot(x, y, '*b')
plt.plot(x, y2)
plt.suptitle('Przewidywanie danych dla regularyzowanej \\
regresji elastycznej siatki')
```

```
plt.xlabel('Numer indeksu testującego')
plt.ylabel('Wartość szukana')
plt.show()
```