Uczenie liniowej hipotezy dla problemu przewidywania cen nieruchomości

KWD

Maria Guz, Karol Pątko

01.02.2020

Streszczenie

Projekt składa się z wprowadzenia opisującego dane wejściowe na których prowadzone są operacje i dla których generowany jest model predykcji. W dalszej części wstępu teoretycznego zostały opisane wykorzystane metody regresji oraz podstawowe informacje z dziedziny machine learning. Natomiast sekcja badań symulacjynych poświęcona jest testowaniu działania różnych algorytmów regresji, porównaniu otrzymanych wyników oraz analizie jakości algorytmów. W trakcie badań zmieniane są poszczególne parametry funkcji wykonujących algorytmy regresji.

Spis treści

Rozdział 1

Wprowadzenie

1.1 Opis problemu

1.1.1 Opis danych wejściowych

Problemem jest przewidywanie ceny nieruchomości na podstawie dostępnych danych, takich jak wskaźnik przestępczości, stężenie tlenków azotu czy wskaźnik dostępności do autostrad. W zestawie *Boston Housing Dataset*, zaimportowanym z biblioteki sklearn, znajduje się 506 instancji. Każda z nich posiada 14 atrybutów z czego 13 to cechy wpływające na cenę nieruchomości, a ostatni to wspomniana cena nieruchomości.By znaleźć najbliższą wartości realnej kwotę, oszacowaną przez jeden z algorytmów uczenia maszynowego będziemy porównywać różne typy regresji liniowej oraz wpływ wartości parametrów na skuteczność wytrenowanego modelu. Jako wskaźniki jakości algorytmu posłużą nam wartość błędu średniokwadratowego(*MSE*) oraz współczynnik determinacji(*r2 score*). Naszym zadaniem jest wytrenowanie takiego modelu, którego wartość błędu średniokwadratowego będzie jak najmniejsza, natomiast wartość współczynnika determinacji będzie jak największa.

1.1.2 Analiza eksploracyjna

Jedną z pierwszych użytych przez nas bibliotek jest zestaw narzędzi niezwykle pomocnych podczas analizy i eksploracji danych - pandas. Dzięki metodom zawartym w tej bilbliotece można wygodnie odczytać dane, by dokonać analizy i eksploracji zbioru uczącego algorytmu. Poniżej znajdują się pliki graficzne przedstawiające opis oraz wizualizację danych trenujących, zrealizowane przy pomocy poleceń *['DESCR']* i *D*ataFrame.describe().

```
Boston House Prices dataset
===========================

Notes
------

Data Set Characteristics:

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical
       predictive

    :Median Value (attribute 14) is usually the target

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land
          zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business
          acres per town
        - CHAS     Charles River dummy variable (= 1
          if tract bounds river; 0 otherwise)
        - NOX      nitric oxides concentration (parts
          per 10 million)
        - RM       average number of rooms per
          dwelling
        - AGE      proportion of owner-occupied units
          built prior to 1940
        - DIS      weighted distances to five Boston
          employment centres
        - RAD      index of accessibility to radial
          highways
        - TAX      full-value property-tax rate per
          $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the
          proportion of blacks by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied
          homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.
```

Statystyki opisowe

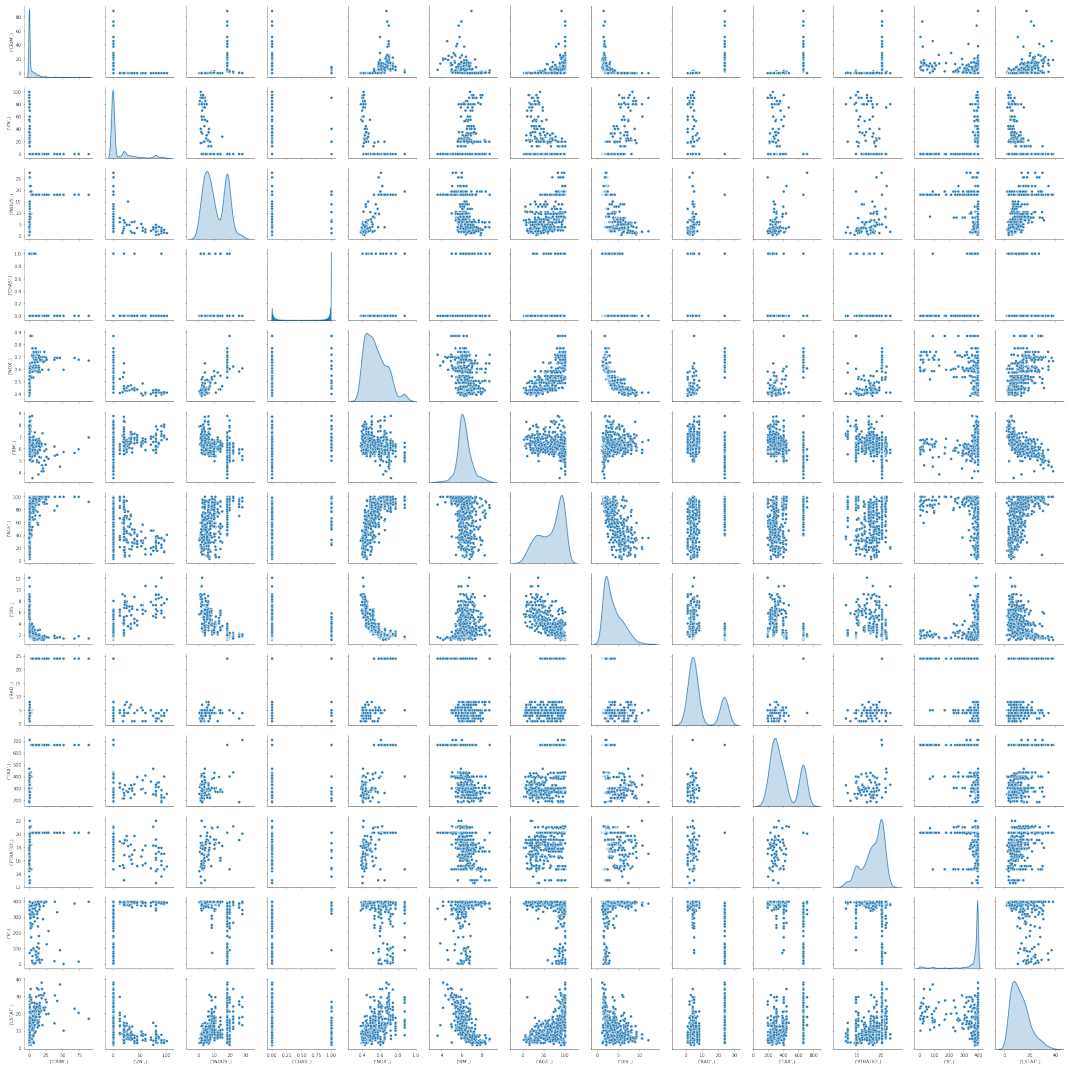
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.237154	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

Przykładowe zestawy cech dla instancji

	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
5	0.02985	0.0	2.18	0.0	0.458	6.430	58.7	6.0622	3.0	222.0	18.7	394.12	5.21
6	0.08829	12.5	7.87	0.0	0.524	6.012	66.6	5.5605	5.0	311.0	15.2	395.60	12.43
7	0.14455	12.5	7.87	0.0	0.524	6.172	96.1	5.9505	5.0	311.0	15.2	396.90	19.15
8	0.21124	12.5	7.87	0.0	0.524	5.631	100.0	6.0821	5.0	311.0	15.2	386.63	29.93
9	0.17004	12.5	7.87	0.0	0.524	6.004	85.9	6.5921	5.0	311.0	15.2	386.71	17.10
10	0.22489	12.5	7.87	0.0	0.524	6.377	94.3	6.3467	5.0	311.0	15.2	392.52	20.45

Zestaw danych *Boston Housing Dataset* składa się z 506 obiektów. Każdy z nich posiada 13 cech oraz przewidywaną wartość nieruchomości, której oszacowanie będzie zadaniem wytrenowanego modelu. Dane są zróżnicowane. Pochodzą z różnych dziedzin życia i są o nieporównywalnych miarach. Należy sprowadzić je do jednej porównywalnej miaru statystycznej za pomocą standaryzacji danych. Pominięcie standaryzacji danych może skutkować wyuczeniem błędnego modelu. W celu wykonania stadaryzacji danych można skorzystać z dostępnego w bibliotece sklearn StandardScalera. Skaluje on dane do porównywalnej miary korzystając z wzoru **z = (x - u) / s**, gdzie u jest średnią próbek treningowych, a s jest standardowym odchyleniem próbek treningowych.

1.1.3 Wizualizacja danych



Rozdział 2

Opis metody

2.1 Wprowadzenie teoretyczne

2.1.1 Uczenie maszynowe

Uczenie maszynowe to budowanie modelu matematycznego i wykorzystanie metod matematycznych do przewidywania i samodoskonalenia się. Często takie programy wykorzystuje się do rozpoznawania mowy lub pisma.

Rodzaje uczenia maszynowego:

- **Uczenie nadzorowane** (*Supervised Learning*)
  Program uczy się na podstawie otrzymanych danych oraz odpowiedzi do nich. Ucząc się program generuje pewien wzorzec (model), który wykorzystywany jest później do przewidywania wyników na podstawie innych danych.
- **Uczenie częściowo nadzorowane** (*Semi - Supervised Learning*)
  Program otrzymuje zarówno dane wejściowe zawierające odpowiednie dane wyjściowe, jak i nieoznaczone czyli bez odpowiadającym im wyników.
- **Uczenie nienadzorowane** (*Unsupervised Learning*)
  Program nie posiada „klucza odpowiedzi" i musi sam analizować dane, szukać wzorców i odnajdywać relacje. Wraz ze wzrostem zbiorów danych prezentowane wnioski są coraz bardziej precyzyjne.
- **Uczenie wzmocnione** (*Reinforcement Learning*)
  Program otrzymuje gotowy zestaw dozwolonych działań, reguł i stwierdzeń. Działając w ich ramach dokonuje analizy i obserwuje ich skutki. Wykorzystuje reguły w taki sposób, aby osiągnąć pożądany efekt.

2.1.2 Regresja liniowa

Regresja liniowa pozwala przewidzieć wartość wyjściową poprzez obliczenie jej na podstawie cech wejściowych oraz stałej (punkt obciążenia).
Równanie prezentujące predykcję za pomocą modelu regresji liniowej:

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

Można go także przedstawić za pomocą formy wektorowej.

$$\hat{y} = h_\theta(\mathbf{x}) = \theta^T \cdot \mathbf{x}$$

Po wytrenowaniu modelu powinno się sprawdzić czy został on dobrze wytrenowany. Sprowadza się to do zmierzenia błędu na przykład za pomocą metody średnich kwadratów, która pozwala na znalezienie jak najbardziej optymalnej funckji dopasowanej do wczytanych danych.
Równanie pozwalające wyliczyć błąd MSE:

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

2.1.3 Regresja wielomianowa

W przypadku kiedy dane są nieliniowe i funkcja liniowa nie jest w stanie wystarczająco dobrze reprezentować podanych cech mamy możliwość utworzeniu modelu, który będzie funkcją drugiego bądź wyższego stopnia. Do korzystania z tej metody ilość cech jest rozszerzana (cechy są podnoszone do odpowiedniej potęgi w zależności od stpnia wielomianu). Technika ta jest typową techniką wykorzystywaną do badania zależności krzywoliniowych.

2.1.4 Regularyzowane modele liniowe

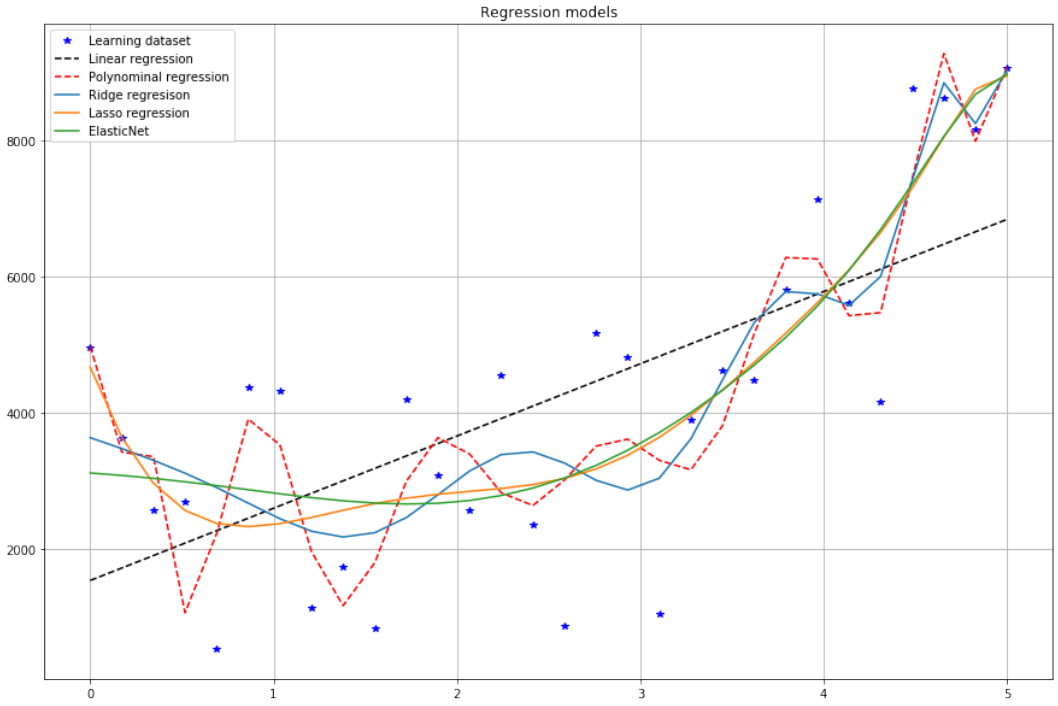
Po wyuczeniu modelu można dokonać jego regularyzacji. Polega ona na ograniczeniu wyjściowej funkcji aby dostosować ją do ogółu napotkanych danych przy późniejszym wykorzystaniu jeśli wyuczony model jest za bardzo "przystosowany" do danych trenujących. Najprostrzym sposobem na regularyzację modelu jest zmniejszenie stopnia wielomianu. W przypadku kiedy funkcja wyjściowa jest liniowa możemy regularyzować model poprzez ograniczenie wag. Regularyzację możemy osiągnąć za pomocą między innymi trzech niżej podanych metod

- **Ridge regression**
  Odmiana regresji liniowej w której do funkcji kosztu dodajemy człon regularyzacji. W związku z tym model musi utrzymać jak najmniejsze wartości wag oraz dopasować się do danych. Modyfikując parametr alfa możemy określić stopień regularyzacji modelu (jeśli alfa = 0 to dodatkowy człon jest bezużyteczny i ostatecznie mamy do czynienia ze standardową regresją liniową).

- **Lasso regression**
  Regresja LASSO jest kolejną odmianą regresji liniowej. Podobnie jak w przypadku regresji grzbietowej dodawany jest człon regularyzacyjny. Podstawową właściwością tej metody jet dążenie do eliminacji wag najmniej istotnych cech (zamienia wartość wagi na 0). Przeprowadzane jest automatyczne dobieranie cech i generowanie modelu z niewielką ilością niezerowych wag.

- **ElasticNet**
  Metoda ta jest pośrednia pomiędzy wyżej wspomnianymi odmianami regresji. Dodatkowy człon regularyzacyjny jest tworzony przez połączenie obydwu członów regularyzacyjnych z metody grzbietowej oraz LASSO. Człon ten może być zmieniany za pomocą współczynnika proporcji r (jeśli r = 0 to metoda elastycznej siatki przyjmuje wzór metody grzbietowej, natomiast jeśli r = 1 wtedy metoda ta przyjmuje wzór metody LASSO).

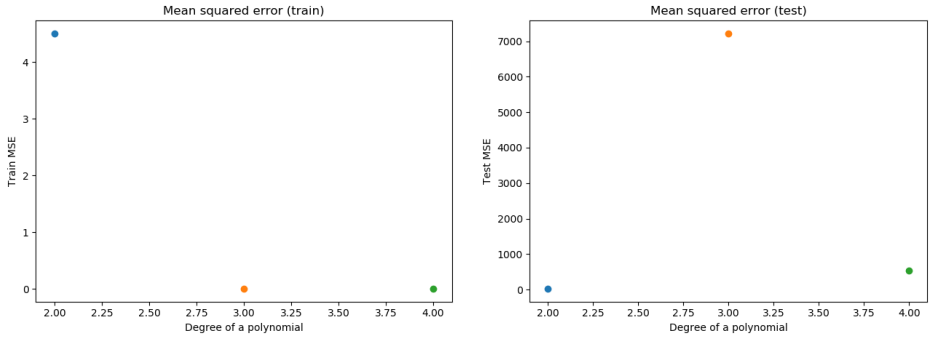
Porównanie modeli regresji dla losowych danych



2.2 Badania symulacyjne

2.2.1 Cechy wielomianowe

Niżej przedstawione wykresy przedstawiają wartość błędu średniokwadratowego w zależności od stopnia wielomianu opisującego cechy wielomianowe. Podczas generowania nowych cech należy kontrolować czy nie zachodzi zjawisko overfittingu. Na niżej ukazanych wykresach można zauważyć, że błąd na zbiorze testowym dla stopnia wielomianu = 3 jest nieporównawalnie większy od błędu stopnia 2. i 4. Jednak wykres przedstawiający błąd średniokwadratowy na zbiorze uczącym wygląda zupełnie inaczej. Wartość błędu średniokwadratowego przy wykorzystaniu cech wielomianowych trzeciego stopnia jest najmniejsza. Taka sytuacja jest jednoznaczna - algorytm za bardzo dopasował się do zawierającego błędy zestawu danych trenujących, natomiast nie radzi sobie z przewidywaniem wartości dla nowych, nieznanych danych. Algorytm jest przetrenowany - mamy do czynienia z overfittingiem. Ze względu na to zdecydowaliśmy się na generowanie cech wielomianowych stopnia drugiego. Innym sposobem niedopuszczenia do overfittingu jest wydłużenie fazy uczenia i jednoczesna optymalizacja. Takie rozwiązanie nazywa się regularyzacją i zapobiega przejściu w fazę overfittingu.




```
MSE( Second degree polynomial(train) ):  4.501167672798458
MSE( Second degree polynomial(test) ):  18.881917071216986
MSE( Third degree polynomial(train) ):  7.407636905915444e-24
MSE( Third degree polynomial(test) ):  7218.721693460779
MSE( Fourth degree polynomial(train) ):  5.524508462872734e-25
MSE( Fourth degree polynomial ):  527.7139590772728
```

2.2.2 Analiza jakości algorytmów trenujących

Analiza jakości algorytmów trenujących została zrealizowana na podstawie wartości błędu średniokwadratowego oraz R2 Score. W poniższych wynikach można zaobserwować, że najlepszym algorytmem regresji pod względem błędu MSE jest Ridge Regression. Nieznacznie od niego odbiega Polynomial Features. Najgorzej sprawdził się algorytmy Linear Regression oraz ElasticNet Regression. Błąd średniokwadratowy poniekąd przekłada się na współcznnik determinacji. Im jest on większy tym lepiej. W tym przypadku kolejność od najlepszej metody do najgoszej odpowiada kolejności w MSE.

Mean Squared Error

```
MSE( Linear Regression ):  29.133662243080355
MSE( Polynomial Features Regression ):  18.881917071216986
MSE( Ridge Regression ):  17.507718906216347
MSE( Lasso Regression ):  25.89366519184813
MSE( ElasticNet Regression ):  29.32117939744992
```

R2 Score

```
Linear Regression variance score: 0.68
Polynomial Regression variance score: 0.80
Ridge Regression variance score: 0.81
Lasso Regression variance score: 0.72
Elastic Regression variance score: 0.68
```

Cross Validation

Sprawdzian krzyżowy jest metodą statystyczną, któa polega na podziale prób statycznych na zbiory. Analiza danych zostaje przeprowadzona na części utworzonych podzbiorów (zbiór uczący). Na koniec pozostałe zbiory służą do testowania otrzymanych wyników (zbiór testowy, walidacyjny).

```
Linear Regression:  [ 0.75759644  0.861565    0.85879891  0.85883967
0.8087769   0.53015321  -0.00796695  0.81171264  0.84842609  0.85639315]

Polynomial Regression:   [ 0.75759644  0.861565    0.85879891  0.85883967
0.8087769   0.53015321 -0.00796695  0.81171264  0.84842609  0.85639315]

Ridge Regression:  [0.75517461 0.89546955 0.91572867 0.87815741 0.81990664
0.47404469 0.49586335 0.91060916 0.90495508 0.91385454]

Lasso Regression:  [0.62970388 0.78252054 0.8155522  0.84833661 0.79374736
0.08562462 0.68392288 0.91396189 0.85765783 0.78172395]

ElasticNet Regression:  [0.60683724 0.79994787 0.808602 0.81098932 0.80929697
0.19391193 0.77713428 0.91087007 0.831761   0.75559328]
```

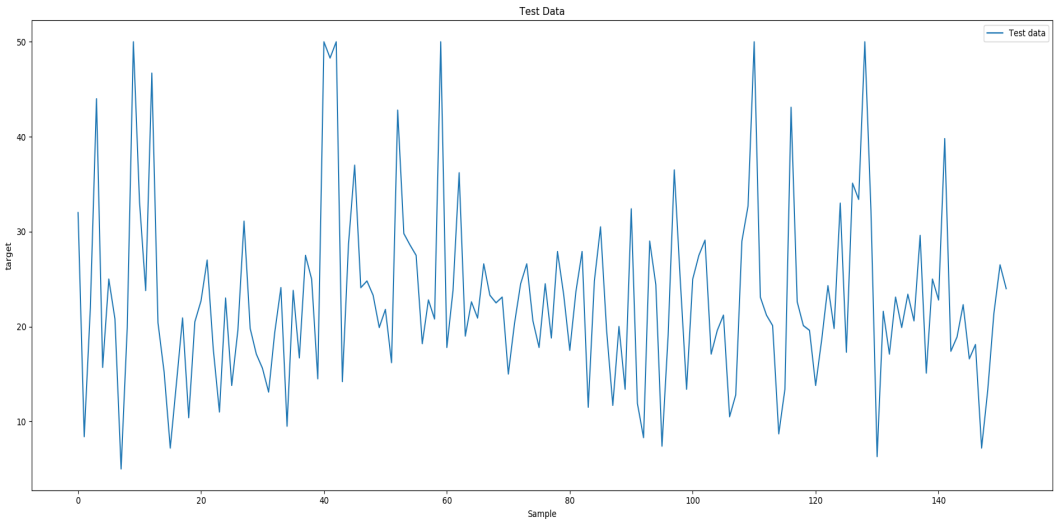
Wykresy

Sekcja ta zawiera wykresy przedstawiające wyniki przewidywania wartości nieruchomości na podstawie nowego zbioru uczącego przez algorytmy:

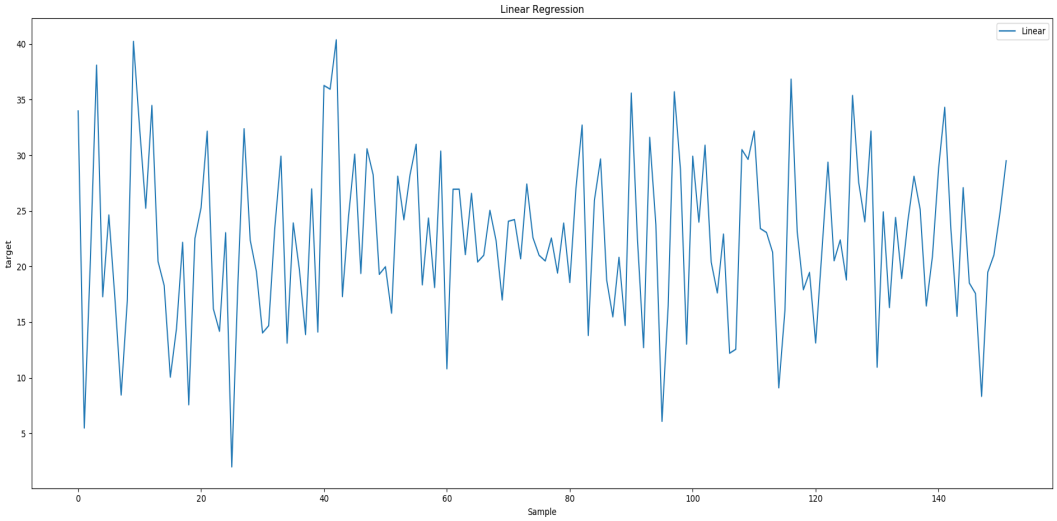
- Linear Regression
- Polynomial Regression
- Ridge
- Lasso
- ElasticNet

oraz wykres przedstawiający wyniki ze zbioru testującego i wykres wszystkich wyników służący porównaniu rezultatów.

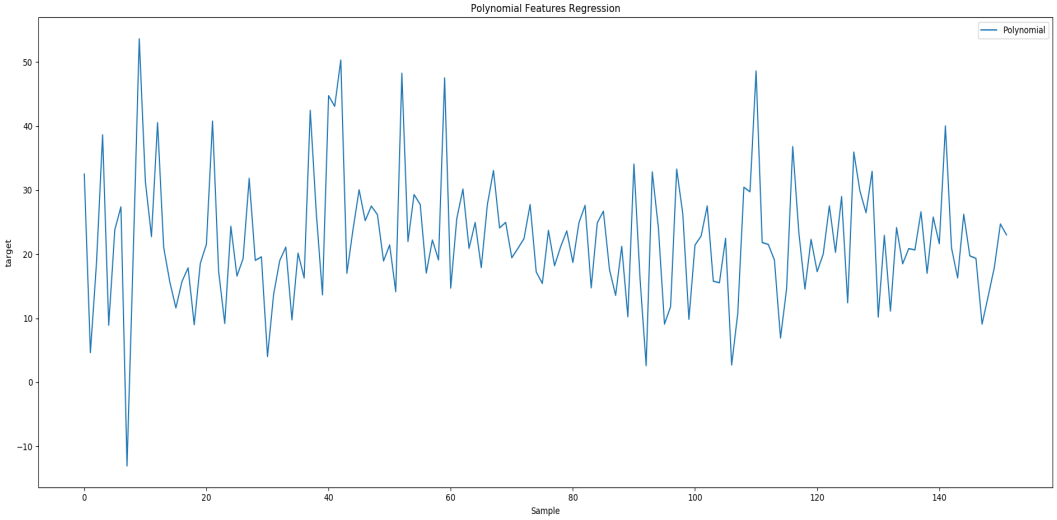
Wykres wyników testujących



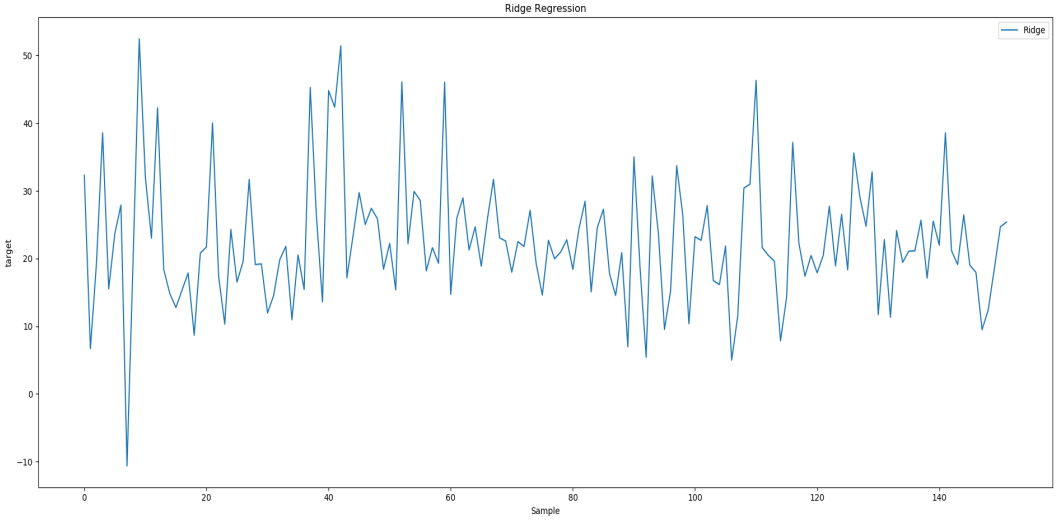
Linear Regression



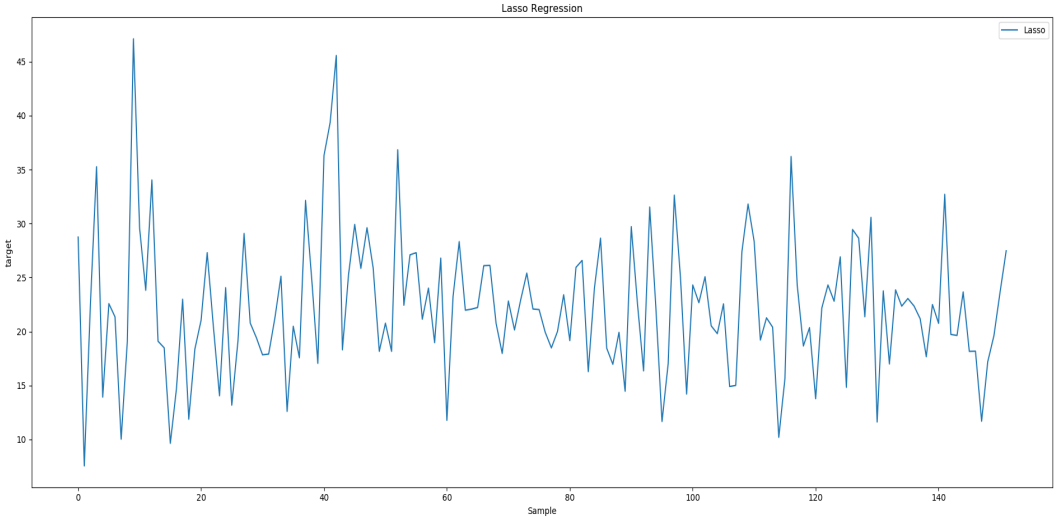
Polynomial Regression



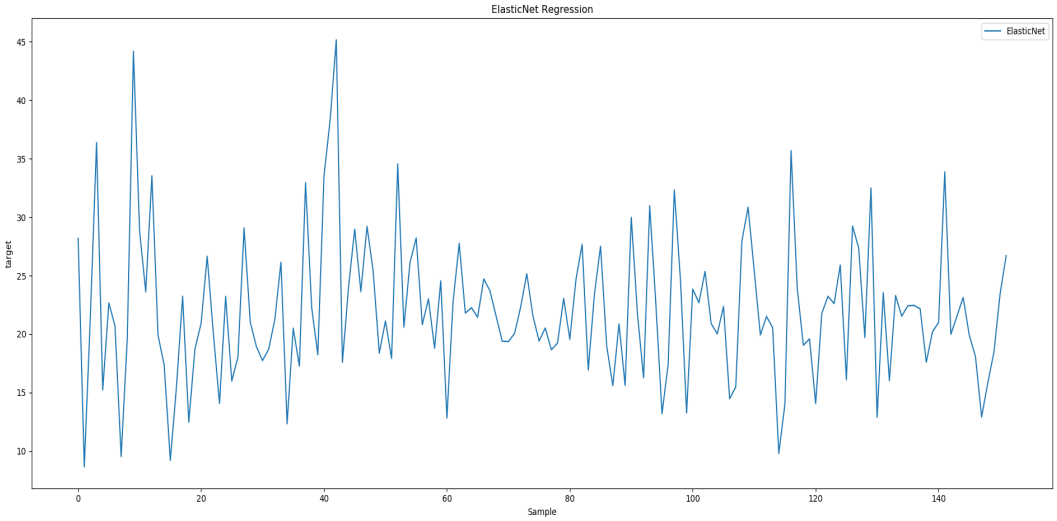
Ridge



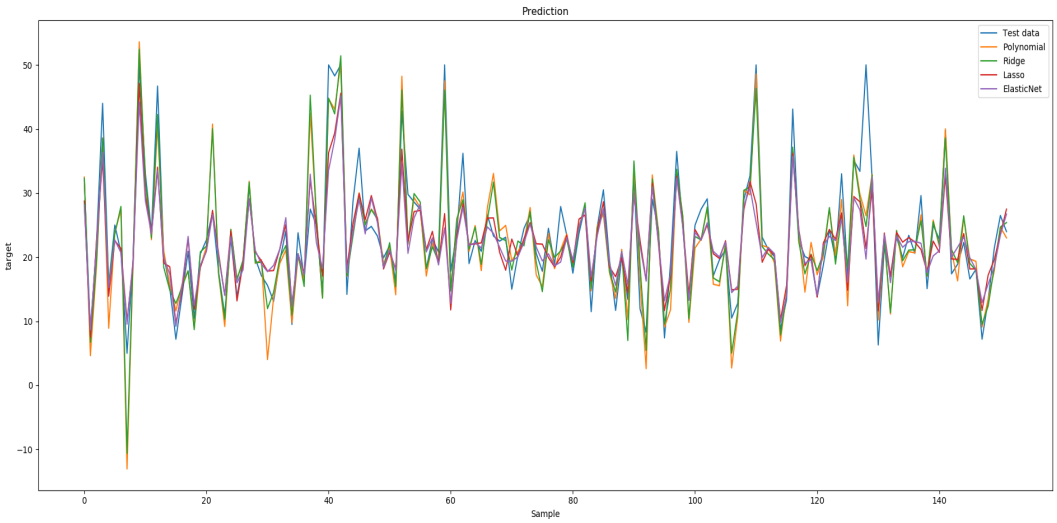
Lasso



ElasticNet



Wszystkie algorytmy



2.2.3 Zmiana parametru alpha

Poniższe wyniki odzwierciedlają zmiany skuteczności algorytmów regularyzacyjnych spowodowane modyfikacją jednego parametru.

Ridge

W algorytmie regresji grzbietowej parametr aplha odpowiada sile regularyzacji. Musi być dodatnią liczbą zmiennoprzecinkową. Siła regularyzacji rośnie proporcjonalnie do wartości parametru alpha. Poniższe wartości błędu średniokwadratowego odzwierciedlają skuteczność algorytmu w zależności od wartości alpha.

```
MSE( Ridge Regression | alpha = 0.1 ):  18.97574804359847
MSE( Ridge Regression | alpha = 0.2 ):  18.64484398510324
MSE( Ridge Regression | alpha = 0.5 ):  18.061055754020334
MSE( Ridge Regression | alpha = 0.3 ):  18.407985568391993
MSE( Ridge Regression | alpha = 0.4 ):  18.219457324023118
MSE( Ridge Regression | alpha = 0.5 ):  18.061055754020334
MSE( Ridge Regression | alpha = 0.6 ):  17.923828978533276
MSE( Ridge Regression | alpha = 0.7 ):  17.802672451238283
MSE( Ridge Regression | alpha = 0.8 ):  17.694335275513257
MSE( Ridge Regression | alpha = 0.9 ):  17.59656865058456
```

Lasso

W algorytmie regresji typu Lasso parametr aplha odpowiada stałej zwielokrotniającej warunek L1. Musi znajdować się w przedziale od 0 do 1. Wraz z wartością parametru alpha rośnie liczba wyzerowanych przez model najmniej istotnych cech. Alpha = 0 jest równoznaczne regresjo liniowej bez regularyzacji. Poniższe wartości błędu średniokwadratowego odzwierciedlają skuteczność algorytmu w zależności od wartości alpha.

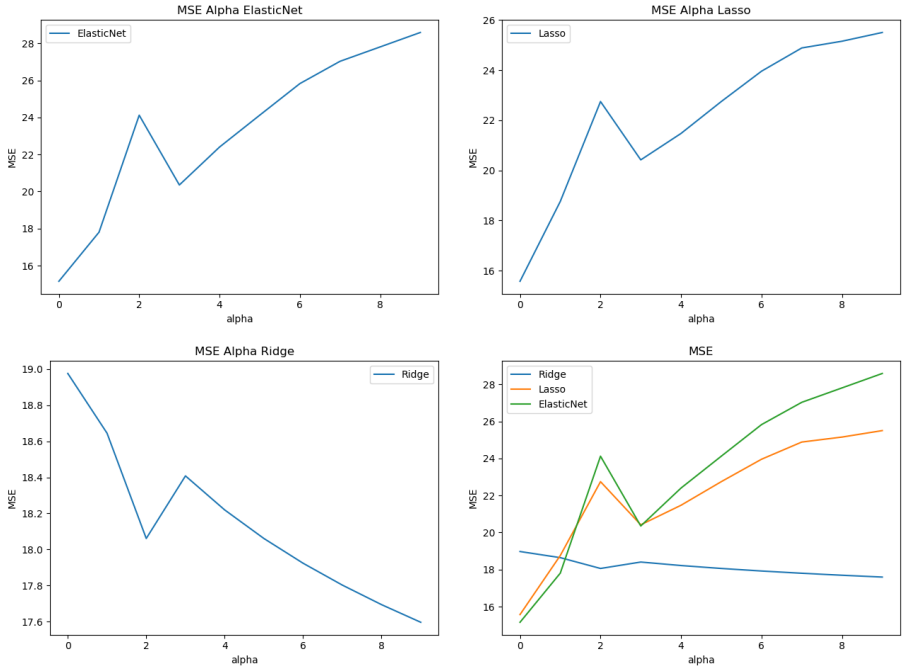
```
MSE( Lasso Regression | alpha = 0.1 ):  15.571662021982975
MSE( Lasso Regression | alpha = 0.2 ):  18.757730750265882
MSE( Lasso Regression | alpha = 0.5 ):  22.74762403461749
MSE( Lasso Regression | alpha = 0.3 ):  20.41861211785669
MSE( Lasso Regression | alpha = 0.4 ):  21.473741958799515
MSE( Lasso Regression | alpha = 0.5 ):  22.74762403461749
MSE( Lasso Regression | alpha = 0.6 ):  23.95869936202824
MSE( Lasso Regression | alpha = 0.7 ):  24.88707722101838
MSE( Lasso Regression | alpha = 0.8 ):  25.15598337232744
MSE( Lasso Regression | alpha = 0.9 ):  25.505300530867405
```

ElasticNet

W algorytmie regresji typu Lasso parametr aplha odpowiada stałej zwielokrotniającej dodatkowych warunków. Wraz z wartością parametru alpha rośnie liczba wyzerowanych przez model najmniej istotnych cech. Alpha = 0 jest równoznaczne regresjo liniowej bez regularyzacji. Poniższe wartości błędu średniokwadratowego odzwierciedlają skuteczność algorytmu w zależności od wartości alpha.

```
MSE( ElasticNet Regression | alpha = 0.1 ):  15.155320778111095
MSE( ElasticNet Regression | alpha = 0.2 ):  17.806643195596656
MSE( ElasticNet Regression | alpha = 0.5 ):  24.122191763441037
MSE( ElasticNet Regression | alpha = 0.3 ):  20.35489519302588
MSE( ElasticNet Regression | alpha = 0.4 ):  22.40232244416601
MSE( ElasticNet Regression | alpha = 0.5 ):  24.122191763441037
MSE( ElasticNet Regression | alpha = 0.6 ):  25.828241641606777
MSE( ElasticNet Regression | alpha = 0.7 ):  27.032181934267747
MSE( ElasticNet Regression | alpha = 0.8 ):  27.81010733551388
MSE( ElasticNet Regression | alpha = 0.9 ):  28.589619318818933
```

Niżej zamieszczone zostały wykresy ukazujące zmianę wartości błędu średniokwadratowego w zależności od wartości parametru alpha



2.2.4 Wpływ parametrów konfiguracyjnych

Grid search

Przeszukiwanie parametrów GridSearch można w prosty i wygodny sposób znaleźć parametry, które powiny być bardziej optymalne. Stosując GridSearch do wcześniej wykorzystywanych algorytmów regresjii otrzymaliśmy następujące parametry:

```
{{'alpha': 9, 'copy_X': True, 'fit_intercept': True, 'solver': 'svd'}
Ridge(alpha=9, copy_X=True, fit_intercept=True, max_iter=None, normalize=False,
      random_state=None, solver='svd', tol=0.001)
```

```
{'alpha': 0.1, 'max_iter': 1000, 'normalize': False, 'positive': False,
    'precompute': False, 'selection': 'random', 'tol': 0.01, 'warm_start': False}
Lasso(alpha=0.1, copy_X=True, fit_intercept=True, max_iter=1000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='random', tol=0.01, warm_start=False)
```

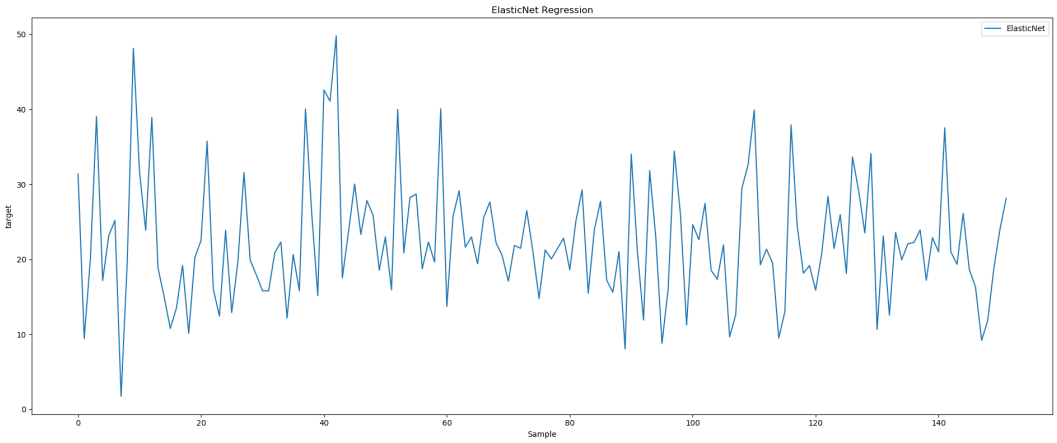
```
{'alpha': 0.1, 'l1_ratio': 0.1, 'normalize': False, 'positive': False,
    'precompute': False, 'selection': 'random', 'warm_start': True}
ElasticNet(alpha=0.1, copy_X=True, fit_intercept=True, l1_ratio=0.1,
```

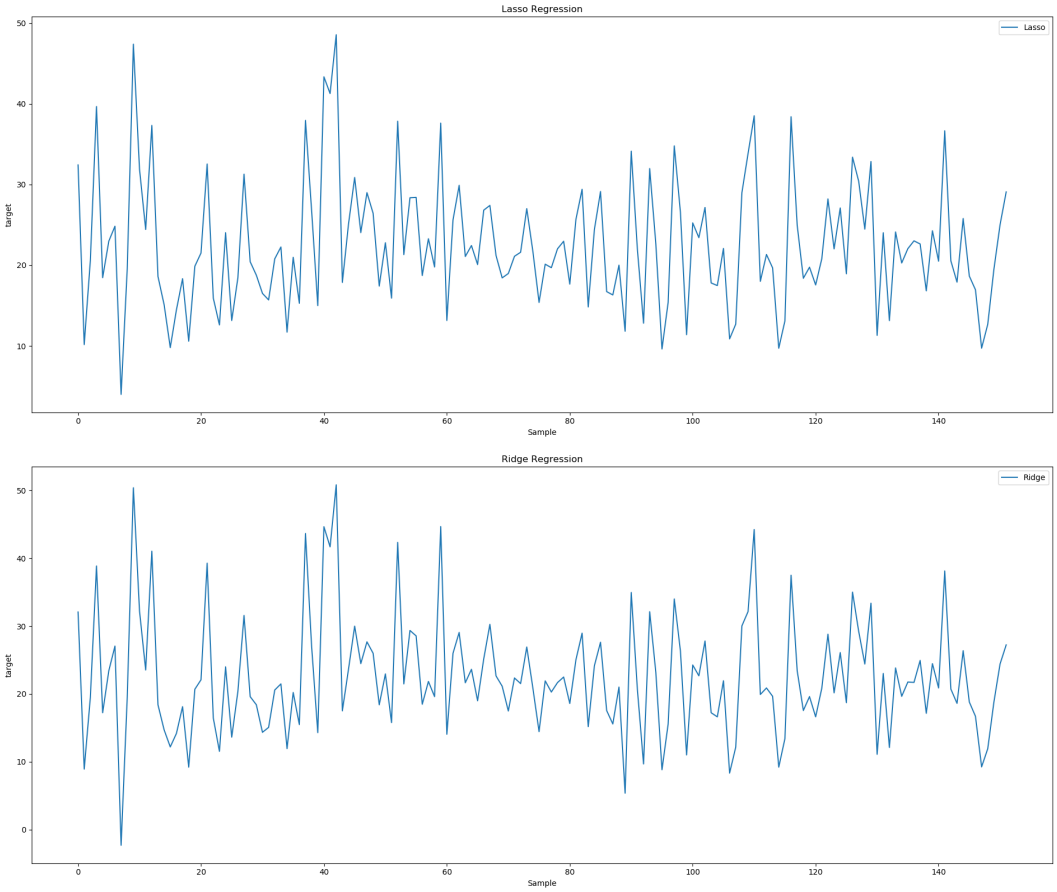
```
        max_iter=1000, normalize=False, positive=False, precompute=False,
        random_state=None, selection='random', tol=0.0001, warm_start=True)

}
```

Ponowne trenowanie modelu

Wykorzystując parametry zwrócone w wyniku przeszukiwania GridSearch dokonaliśmy ponownej analizy jakości uzyskanych algorytmów. Wartości błędu średniokwadratowego, współczynnika determinacji oraz wykresy zostały przedstawione niżej.

```
Ridge Regression variance score: 0.83
Lasso Regression variance score: 0.83
Elastic Regression variance score: 0.83

MSE Ridge:  15.909491147585078
MSE Lasso:  15.635056528063478
MSE ElasticNet:  15.660792061437585
```




Lasso Regression
Lasso
target
Sample
0
20
40
60
80
100
120
140
10
20
30
40
50

Ridge Regression
Ridge
target
Sample
0
20
40
60
80
100
120
140
0
10
20
30
40
50

Rozdział 3

Podsumowanie

Problem przewidywania wartości nieruchomości mieszkań na podstawie zestawu danych *Boston Housing Dataset*, można rozwiązać na różne sposoby. Przy wyborze odpowiedniego algorytmu należy jednak pamiętać o zapoznaniu się ze zbiorem uczącym, danymi, na podstawie których będzie uczył się model. Następnie należy zdecydować czy przeprowadzenie standaryzacji oraz normalizacji danych jest konieczne. Narzędzia takie jak GridSearch czy Cross Validation są niezwykle pomocne w wyborze odpowiednich parametrów dla danego algorytmu. Obliczenie MSE oraz variance score jest natomiast sposobem na wybór odpowiedniego spośród wielu algorytmów. Niestety często dane uczące zawierają błędy co także wpływa na wytrenowanie modelu. Dzięki analizie jakości wytrenowanego modelu, dobrym przygotowaniu danych uczących oraz poprawnej interpretacji wyników możemy znaleźć najlepsze rozwiązanie dla danego problemu.

Bibliografia

[1] Aurelien Geron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow. Concepts, Tools, and Techniques to Build Intelligent Systems, 2017.*

[2] *Scikit-learn Documentation, [online], https://scikit-learn.org/stable/*

Dodatek A

Kod programu

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

# ============================================================


mse = lambda alg: mean_squared_error(y_test, alg.predict(x_test_poly))
prnt = lambda alg, mse, al: print("MSE( ", alg, " | alpha = ", al, " ):
", mse, sep="")
doArray = lambda title, algh, alg, X: [alg(title, x, algh) for x in X]
coef = lambda alg: print(alg.coef_)
fit = lambda alg: alg.fit(x_train_poly, y_train)


# ------------------------------------------------------
```

```
def MSE(title, alg, al):
    par = mse(alg)
    prnt(title, par, al)
    return par


def TestAlpha(title, al, alg):
    par = alg(alpha=al)
    par.fit(x_train_poly, y_train)
    mse = MSE(title, par, al)
    return mse


def doPlot(*X, title, labelX, labelY):
    fig = plt.figure(figsize=(19,8))
    for (x,y) in X:
        plt.plot(x, label=y)
    plt.legend()
    plt.title(title)
    plt.xlabel(labelX)
    plt.ylabel(labelY)
    plt.show()


def doFit(*X):
    for x in X:
        x.fit(x_train_poly, y_train)

def doFitLinear(*X):
    for x in X:
        x.fit(x_train, y_train)


def doGridSearch(parameters, alg):
    model = GridSearchCV(alg, parameters, cv=3)
    model.fit(x_train_poly, y_train)
    print(model.best_params_)
    print(model.best_estimator_)
```

```
# boston =load_boston()
# print(boston['DESCR'])
# data_desc = pd.DataFrame(boston.data, columns=[boston.feature_names])
# data_desc.describe()
# sb.pairplot(data_desc, diag_kind="kde", palette="husl")

# ----STANDARD SCALLER----

x, y = load_boston(return_X_y=True)

scaler = StandardScaler()
x = scaler.fit_transform(x)

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3,
random_state=1010)
pf = PolynomialFeatures(2)

x_train_poly = pf.fit_transform(x_train)
x_test_poly = pf.fit_transform(x_test)

# ----POLYNOMIAL DEGREE----

# pf = PolynomialFeatures(2)
# pf3 = PolynomialFeatures(3)
# pf4 = PolynomialFeatures(4)
#
# x_train_poly = pf.fit_transform(x_train)
# x_test_poly = pf.fit_transform(x_test)
#
# x_train_poly3 = pf3.fit_transform(x_train)
# x_test_poly3 = pf3.fit_transform(x_test)
#
# x_train_poly4 = pf4.fit_transform(x_train)
# x_test_poly4 = pf4.fit_transform(x_test)
#
# polynomialF = LinearRegression()
# polynomialF.fit(x_train_poly, y_train)
# mse2train = mean_squared_error(y_train, polynomialF.predict(x_train_poly))
# mse2test = mean_squared_error(y_test, polynomialF.predict(x_test_poly))
# prnt("Second degree polynomial(train)", mse2train)
# prnt("Second degree polynomial(test)", mse2test)
#
# polynomialF = LinearRegression()
```

```
# polynomialF.fit(x_train_poly3, y_train)
# mse3train = mean_squared_error(y_train, polynomialF.predict(x_train_poly3))
# mse3test = mean_squared_error(y_test, polynomialF.predict(x_test_poly3))
# prnt("Third degree polynomial(train)", mse3train)
# prnt("Third degree polynomial(test)", mse3test)
#
# polynomialF = LinearRegression()
# polynomialF.fit(x_train_poly4, y_train)
# mse4train = mean_squared_error(y_train, polynomialF.predict(x_train_poly4))
# mse4test = mean_squared_error(y_test, polynomialF.predict(x_test_poly4))
# prnt("Fourth degree polynomial(train)", mse4train)
# prnt("Fourth degree polynomial", mse4test)
#
# doPlot(mse2train,mse3train,mse4train, title="Mean squared error (train)",
labelX='Degree of a polynomial', labelY='Train MSE')
#
# doPlot(mse2test,mse3test,mse4test, title="Mean squared error (test)",
labelX='Degree of a polynomial', labelY='Test MSE')

# ----REGRESSION----

linear = LinearRegression()
polynomialF = LinearRegression()
ridge = Ridge()
lasso = Lasso()
elastic = ElasticNet()

doFitLinear(linear)
doFit(polynomialF, ridge, lasso, elastic)

predictLinear = linear.predict(x_test)
predictPolynomialF = polynomialF.predict(x_test_poly)
predictRidge = ridge.predict(x_test_poly)
predictLasso = lasso.predict(x_test_poly)
predictElastic = elastic.predict(x_test_poly)

# ------------------------------------------------------

doPlot((y_test, 'Test data'), title="Test Data", labelX='Sample',
labelY='target')

doPlot((predictLinear, 'Linear'), title="Linear Regression",
labelX='Sample', labelY='target')
doPlot((predictPolynomialF, 'Polynomial'), title="Polynomial Features
```

```
Regression", labelX='Sample', labelY='target')
doPlot((predictRidge, 'Ridge'), title="Ridge Regression", labelX='Sample',
labelY='target')
doPlot((predictLasso, 'Lasso'), title="Lasso Regression", labelX='Sample',
labelY='target')
doPlot((predictElastic, 'ElasticNet'), title="ElasticNet Regression",
labelX='Sample', labelY='target')
doPlot((y_test, 'Test data'), (predictPolynomialF, 'Polynomial'),
(predictRidge, 'Ridge'), (predictLasso, 'Lasso'), (predictElastic,
'ElasticNet'), title="Prediction", labelX='Sample', labelY='target')


prnt("Linear Regression", mean_squared_error(y_test, predictLinear), "-")
MSE("Polynomial Features Regression", polynomialF, "-")
MSE("Ridge Regression", ridge, "default")
MSE("Lasso Regression", lasso, "default")
MSE("ElasticNet Regression", elastic, "default")

# ----R2 SCORE----

# score = linear.score(x_test, y_test)
# print("Linear Regression variance score: %.2f" % score)
# score = polynomialF.score(x_test_poly, y_test)
# print("Polynomial Regression variance score: %.2f" % score)
# score = ridge.score(x_test_poly, y_test)
# print("Ridge Regression variance score: %.2f" % score)
# score = lasso.score(x_test_poly, y_test)
# print("Lasso Regression variance score: %.2f" % score)
# score = elastic.score(x_test_poly, y_test)
# print("Elastic Regression variance score: %.2f" % score)

# ----CROSS VALIDATION----

# print("CROSS VALIDATION".center(100), "\n")
# scores = cross_val_score(linear, x_train_poly, y_train, cv=10)
# print("Linear Regression: |  ", scores)
# scores = cross_val_score(polynomialF, x_train_poly, y_train, cv=10)
# print("Polynomial Regression: |  ", scores)
# scores = cross_val_score(ridge, x_train_poly, y_train, cv=10)
# print("Ridge Regression: |  ", scores)
# scores = cross_val_score(lasso, x_train_poly, y_train, cv=10)
# print("Lasso Regression: |  ", scores)
# scores = cross_val_score(elastic, x_train_poly, y_train, cv=10)
# print("ElasticNet Regression: |  ", scores)
```

```
# -----ALPHA----

# alpha = [0.1, 0.2, 0.5, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
#
# mseAlphaRidge = doArray("Ridge Regression", Ridge, TestAlpha, alpha)
# mseAlphaLasso = doArray("Lasso Regression", Lasso, TestAlpha, alpha)
# mseAlphaElastic = doArray("ElasticNet Regression", ElasticNet,
TestAlpha, alpha)

# doPlot((mseAlphaRidge, 'Ridge'), title="MSE Alpha Ridge",
labelX='alpha', labelY='MSE')
# doPlot((mseAlphaLasso, 'Lasso'), title="MSE Alpha Lasso",
labelX='alpha', labelY='MSE')
# doPlot((mseAlphaElastic, 'ElasticNet'),  title="MSE Alpha ElasticNet",
labelX='alpha', labelY='MSE')
#
# doPlot((mseAlphaRidge, 'Ridge'), (mseAlphaLasso, 'Lasso'),
(mseAlphaElastic, 'ElasticNet'), title="MSE", labelX='alpha', labelY='MSE')

# ---- GRID SEARCH ----

# doGridSearch([{'alpha': [0.1, 5, 7, 8, 9, 10, 30], 'copy_X':
[True, False], 'fit_intercept': [True, False], 'solver':['auto', 'svd',
'cholesky', 'lsqr', 'sparse_cg', 'sag', 'saga']}], ridge)

# doGridSearch([{'alpha': [ 0.1, 0.3, 0.4, 0.9], 'max_iter': [500,1000],
'normalize':[True,False], 'positive':[True, False], 'precompute':[True,
False], 'selection':['cyclic', 'random'], 'tol': [1e-4, 1e-6, 1e-2],
'warm_start':[True, False]}], Lasso())

# doGridSearch([{'alpha': [0.1, 0.5, 0.9], 'l1_ratio': [0.1, 0.2, 0.5, 0.9],
'normalize':[True, False], 'positive':[True, False], 'precompute':
[True, False], 'selection':['cyclic', 'random'], 'warm_start':
[True, False]}], ElasticNet())

# ----PREDICT v2----

ridge = Ridge(alpha=9, copy_X=True, fit_intercept=True, max_iter=None,
normalize=False,
      random_state=None, solver='svd', tol=0.001)

lasso=Lasso(alpha=0.1, copy_X=True, fit_intercept=True, max_iter=1000,
      normalize=False, positive=False, precompute=False, random_state=None,
```

```
      selection='random', tol=0.01, warm_start=False)

elastic=ElasticNet(alpha=0.1, copy_X=True, fit_intercept=True, l1_ratio=0.1,
max_iter=1000, normalize=False, positive=False, precompute=False,
random_state=None, selection='random', tol=0.0001, warm_start=True)

doFit( ridge, lasso, elastic)

score = ridge.score(x_test_poly, y_test)
print("Ridge Regression variance score: %.2f" % score)
score = lasso.score(x_test_poly, y_test)
print("Lasso Regression variance score: %.2f" % score)
score = elastic.score(x_test_poly, y_test)
print("Elastic Regression variance score: %.2f" % score)

predictRidge = ridge.predict(x_test_poly)
predictLasso = lasso.predict(x_test_poly)
predictElastic = elastic.predict(x_test_poly)

doPlot((predictRidge, 'Ridge'), title="Ridge Regression", labelX='Sample',
labelY='target')
doPlot((predictLasso, 'Lasso'), title="Lasso Regression", labelX='Sample',
labelY='target')
doPlot((predictElastic, 'ElasticNet'), title="ElasticNet Regression",
labelX='Sample', labelY='target')

print("MSE Ridge: ", mean_squared_error(y_test, predictRidge))
print("MSE Lasso: ", mean_squared_error(y_test, predictLasso))
print("MSE ElasticNet: ", mean_squared_error(y_test, predictElastic))
```