Maciej Zagórski
Informatyka, III rok

RAPORT Z PROJEKTU
Sprawdzenie działania modeli regresywnych

1. Przedmiot projektu

Przedmiotem projektu było sprawdzenie, jak działają modele regresywne uczenia maszynowego:

- model regresji liniowej,
- model regresji wielomianowej drugiego stopnia (dopasowanie wielomianu drugiego stopnia),
- model regresji wielomianowej trzeciego stopnia.

Ponadto dokonano sprawdzenia działania klasyfikacji dokonywanej przy pomocy regresji logistycznej (będącej właśnie algorytmem klasyfikacji).

Powyższego dokonano w oparciu o przykładowe zbiory danych „product.csv" oraz „product2.csv", zawierające wartości współczynników (cech, parametrów) „x" oraz „y". Zastosowanie modeli regresywnych w tym wypadku pozwala na przewidywanie (określenie), jaka powinna być wartość współczynnika „y" w przypadku produktu o współczynniku „x". Natomiast wykorzystanie modelu regresji logistycznej służy w tym wypadku kwalifikowaniu próbek o danych parametrach do określonej kategorii.

2. Realizacja projektu

2.1. Analiza i wstępne przygotowanie zbiorów danych

2.1.1. Uwagi ogólne

Każdy z wymienionych powyżej zbiorów danych – „product.csv" oraz „product2.csv" – zawiera informacje o pięciuset, jak można przypuszczać po nazwach plików, produktach. Na informacje te składają się wartości parametrów „x" oraz „y" dla każdego z produktów (rekordów). Zbiór „product2.csv", poza tymi samymi informacjami, co zbiór „product.csv", zawiera informację o etykiecie klasy każdego rekordu – tj. w kolumnie „good" informację o klasyfikacji każdego z produktów do grupy „0" lub „1". Produktów kwalifikowanych do grupy „0" jest przy tym w zbiorze mniej (184) niż produktów przyporządkowanych do grupy „1" (314).

Z danych zawartych w zbiorach nie wynika, czego one dotyczą (np. jakiego rodzaju produktów), za pomocą jakich cech zostały one opisane, ani w jakich jednostkach podane wartości zostały wyrażone. Zbiór „product2.csv" nie daje również odpowiedzi, co oznacza, że niektóre produkty są kwalifikowane do grupy „1", a niektóre do grupy „2". Można jedynie domyślać się, że wartość „1" w kolumnie „good" należy odczytywać jako *true*, to znaczy, że dany produkt jest produktem „dobrym" (właściwym); niemniej, niewiele to pomaga, nie mając wiedzy o tym, o jakie produkty i o jaką klasyfikację chodzi.

Zbiór „product.csv" zostanie wykorzystany do sprawdzenia działania modeli regresywnych uczenia maszynowego – regresji liniowej i regresji wielomianowej – podczas gdy zbiór „product2.csv" – do sprawdzenia klasyfikacji dokonanej przy pomocy regresji logistycznej – jako że zbiór ten zawiera informacje o etykietach klas poszczególnych rekordów, a regresja logistyczna jest algorytmem klasyfikacji.

Oba zbiory zawierają dane porządkowe – w postaci pierwszej, nienazwanej kolumny zawierającej liczbę porządkową rekordu. Aby takie dane nie wpłynęły nieadekwatnie na efekty zastosowania algorytmów, zostały one (po zaimportowaniu zbiorów danych do programu przy wykorzystaniu biblioteki *pandas*) usunięte.

```
data_product = pd.read_csv('../data/report_3_regression_models_product.csv')

data_product.drop([data_product.columns[0]], axis=1, inplace=True)
```

```
data_product_2 = pd.read_csv('../data/report_3_regression_models_product2.csv')

data_product_2.drop([data_product_2.columns[0]], axis=1, inplace=True)
```

2.1.2. Ocena kompletności i prawidłowości danych

Dane zawarte w zbiorach „product.csv" i „product2.csv" są kompletne, tzn.:

- w przypadku żadnego z produktów wartość cechy „x" lub „y" nie jest pusta – nie jest pustym, niewypełnionym polem,
- w zbiorze „product2.csv" każdy z produktów jest zakwalifikowany tylko do jednej z dwóch kategorii „good", tzn. „1" lub „0" – nie ma produktu, który nie zostałby zakwalifikowany do którejś z tych kategorii bądź też zostałby zakwalifikowany do innej kategorii, np. „2".

Okazuje się jednak, że próbki o takich samych wartościach „x" i „y" mogą występować w zbiorach kilkukrotnie – zwykle dwu- lub trzykrotnie, chociaż w jednym przypadku takie same wartości charakteryzują nawet dziewięć różnych rekordów. Takich „powtarzających" się wierszy (duplikatów) jest aż 257, a więc stanowią one ponad połowę każdego (500-elementowego) zbioru.

Ponadto w zbiorze „product2.csv" spośród tych 257 powtarzających się rekordów w 101 przypadkach występują takie same wartości „x" i „y", ale różne etykiety klas w kolumnie „good". Przykładowo, w przypadku dwóch duplikatów w jednym z nich wskazano klasę „0", a w drugim – „1"; w przypadku czterech duplikatów – dwa mogą być klasyfikowane do grupy „0", a dwa do grupy „1" etc. W tym znaczeniu dane wydają się nieprawidłowe – można byłoby oczekiwać, że rekordy o takich samych parametrach „x" i „y" będą klasyfikowane do tej samej kategorii „good".

2.2. Modele regresji liniowej i wielomianowej

2.2.1. Standaryzacja

W pierwszej kolejności dokonano standaryzacji danych ze zbioru „product.csv" poprzez wykorzystanie funkcji *StandardScaler* z modułu *sklearn.preprocessing* biblioteki *scikit-learn*. Nie powinno mieć przy tym znaczenia, czy standaryzację wykonano dla każdej z cech („x" i „y") indywidualnie, czy też dla całego zaimportowanego zbioru – zgodnie z dokumentacją biblioteki *scikit-learn* czynności w ramach standaryzacji są dokonywane niezależnie dla każdej cechy ze zbioru danych[1].

```
stdsc = StandardScaler()

data_product_std = stdsc.fit_transform(data_product)
```

[1] *„Centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the training set"* ([b.a.], *scikit-learn: machine learning in Python – scikit-learn 1.2.2 documentation*, https://scikit-learn.org/stable/, [dostęp: 23.05.2023 r.].

2.2.2. Usunięcie próbek odstających

Następnie z tak „ustandaryzowanego” zbioru danych usunięto próbki odstające, stosując do tego metodę opartą o rozstęp ćwiartkowy – ze zbioru usunięto próbki, dla których wartość „y” była:

- większa lub równa trzeciemu kwartylowi (kwantylowi rzędu 0,75) dla całego zbioru wartości „y”, powiększonemu o wartość rozstępu ćwiartkowego przemnożonego przez współczynnik 1,5; albo
- mniejsza lub równa pierwszemu kwartylowi (kwantylowi rzędu 0,25) dla całego zbioru wartości „y”, pomniejszonemu o wartość rozstępu ćwiartkowego przemnożonego przez współczynnik 1,5.

W tym celu zdefiniowano własną funkcję *remove_outliers,* zwracającą zbiór danych z usuniętymi próbkami odstającymi (zbiór danych „nieodstających” – *inliers*) oraz zbiór danych zawierający tylko usunięte próbki odstające (*outliers*).

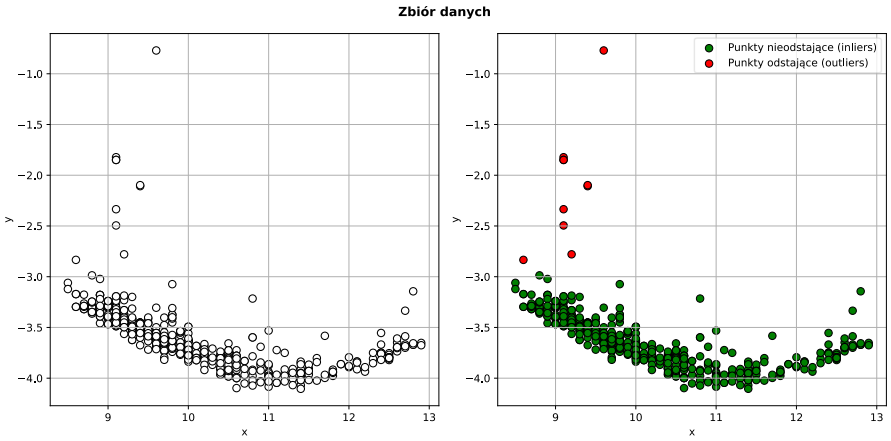
```
def remove_outliers(dataset):
    q1 = np.quantile(dataset[:, 1], 0.25)
    q3 = np.quantile(dataset[:, 1], 0.75)
    iqr = q3 - q1

    print("Q1: %.3f\nQ2: %.3f\nIQR: %.3f" % (q1, q3, iqr))

    inliers = dataset[dataset[:, 1] < q3 + 1.5 * iqr]
    inliers = inliers[inliers[:, 1] > q1 - 1.5 * iqr]

    outliers = dataset[dataset[:, 1] >= q3 + 1.5 * iqr]
    outliers = np.vstack((outliers, dataset[dataset[:, 1] <= q1 - 1.5 * iqr]))

    return inliers, outliers
```

```
Q1: -0.623
Q2: 0.541
IQR: 1.164
```

Poniżej zwizualizowano zbiór danych „product.csv” po dokonaniu standaryzacji i usunięciu próbek odstających. Wykorzystano w tym celu moduł *matplotlib.pyplot* z biblioteki *matplotlib.* Każdy punkt na wykresie to odrębna próbka ze zbioru. Aby wyrazić przy tym cechy „x” i „y” w wartościach rzeczywistych – aby „odwrócić” standaryzację tych danych – posłużono się metodą *inverse_transform*.

```
f, (plt1, plt2) = plt.subplots(1, 2, figsize=(12, 6))

plt1.scatter(data_product.x, data_product.y, c='white', marker='o',
             edgecolor='black', s=50)

plt1.set(xlabel="x", ylabel="y")
plt1.grid()

plt2.scatter(stdsc.inverse_transform(data_product_stand_inliers)[:, 0],
             stdsc.inverse_transform(data_product_stand_inliers)[:, 1], c='green',
             marker='o', edgecolor='black', s=50, label="Próbki nieodstające (inliers)")
plt2.scatter(stdsc.inverse_transform(data_product_stand_outliers)[:, 0],
             stdsc.inverse_transform(data_product_stand_outliers)[:, 1], c='red',
             marker='o', edgecolor='black', s=50, label="Próbki odstające (outliers)")

plt2.legend(scatterpoints=1)
plt2.set(xlabel="x", ylabel="y")
plt2.grid()
```

```
f.suptitle("Zbiór danych", weight='bold')
plt.tight_layout()
```



2.2.3. Podział na podzbiory i dopasowanie danych

Następnie dane podzielono na dwa podzbiory, tj.:

- podzbiór zawierający wartości parametru „x” dla każdej z próbek oraz
- podzbiór zawierający odpowiadające im wartości parametru „y” (także dla każdej z próbek).

Ponieważ dalsze funkcje z biblioteki *scikit-learn* będą wymagały użycia dwuwymiarowych tablic, do każdego podzbioru dodano nowy wymiar (przy pomocy funkcji *newaxis* z biblioteki *numpy*).

```
X_std = data_product_std[:, 0, np.newaxis]
y_std = data_product_std[:, 1, np.newaxis]
```

Na tym etapie przygotowano też zbiór z wartościami „x”, który posłuży do odpowiedniego przedstawienia wykresów funkcji wielomianowych – w przypadku modeli regresji wielomianowej drugiego i trzeciego stopnia. W tym celu, wykorzystując funkcję *arrange* z biblioteki *numpy*, wygenerowano równomiernie rozmieszczone wartości w zadanym przedziale – tu: w oparciu o minimalne i maksymalne wartości „x” występujące w zbiorze i przedział równy 0,1.

```
X_fit = np.arange(X_std.min(), X_std.max(), 0.1)[:, np.newaxis]

print(X_fit)
```

```
[[-1.48185406]
 [-1.38185406]
 [-1.28185406]
 [-1.18185406]
 [-1.08185406]
# [...]
```

Podobny rezultat można przy tym osiągnąć w oparciu o rzeczywiste wartości „x” – o ile wcześniej zostałyby one posortowane. Gdyby wykorzystany został zbiór „X_std” bez takiego posortowania, funkcja rysująca

wykres (zob. p. 3.1.2 niżej) „skakałaby” między kolejnymi wartościami parametru „x”, wykreślając tym sposobem „zygzak” bądź też (przy 450 próbkach) zamalowany kształt przypominający kształtem „banana”.

```
X_fit = np.sort(X_std.flatten())[:, np.newaxis]
```

2.2.4. Podział danych na dane treningowe i testowe

W dalszej kolejności dokonano losowego podziału danych na podzbiór danych treningowych (70%) oraz podzbiór danych testowych (30%) – wykorzystując funkcję *train_test_split* z modułu *sklearn.model_selection* (z biblioteki *scikit-learn*).

```
X_train, X_test, y_train, y_test = train_test_split(X_std, y_std, test_size=0.30,
                                                    random_state=0)
```

Ponieważ dane ze zbioru „product.csv” nie posiadają informacji o klasyfikacji poszczególnych próbek (etykietach klas), powyższa funkcja została użyta bez parametru *stratify* (por. p. 2.3.3 niżej).

2.2.5. Zastosowanie modeli regresywnych

Do podzbiorów danych przygotowanych zgodnie z powyższymi uwagami zastosowano następnie modele regresywne uczenia maszynowego wskazane w p. 1 wyżej:

- Model regresji liniowej, zaimplementowany w bibliotece *scikit-learn* w formie funkcji *LinearRegression* z modułu *sklearn.linear_model*:

```
lr = LinearRegression()
lr.fit(X_train, y_train)
```

- Model regresji liniowej wykorzystujący algorytm konsensu próby losowej (*RANdom SAmple Consensus,* RANSAC), dzięki któremu model ten jest bardziej odporny na próbki odstające; w celu jego zastosowania wykorzystano funkcję *RANSACRegressor* z modułu *sklearn.linear_model* z biblioteki *scikit-learn*:

```
ransac = RANSACRegressor(estimator=LinearRegression(), max_trials=100,
                         loss='absolute_error', random_state=0)
ransac.fit(X_train, y_train)
```

- Model regresji wielomianowej drugiego stopnia; w tym wypadku zastosowano najpierw funkcję transformującą *PolynomialFeatures* z modułu *sklearn.preprocessing* biblioteki *scikit-learn* – z parametrem *degree=2*, dla wielomianu drugiego stopnia – a następnie model regresji liniowej:

```
quadratic = PolynomialFeatures(degree=2)
X_train_quad = quadratic.fit_transform(X_train)
X_test_quad = quadratic.fit_transform(X_test)

lr_quadratic = LinearRegression()
lr_quadratic.fit(X_train_quad, y_train)
```

Funkcja transformującą *PolynomialFeatures* przekształca wartości „x" w taki sposób, aby były one dostosowane do regresji wielomianowej; powyżej dokonano takiego przekształcenia zarówno dla danych treningowych (*X_train*), jak i testowych (*X_test*), co zostanie wykorzystane przy ewaluacji zastosowanych modeli (por. p. 2.3.1 niżej).

- Model regresji wielomianowej – dopasowanie wielomianu trzeciego stopnia – analogicznie jak powyżej, wykorzystano funkcję *PolynomialFeatures* – z parametrem *degree=3* – i model regresji liniowej:

```
cubic = PolynomialFeatures(degree=3)
X_train_cubic = cubic.fit_transform(X_train)
X_test_cubic = cubic.fit_transform(X_test)

lr_cubic = LinearRegression()
lr_cubic.fit(X_train_cubic, y_train)
```

2.2.6. Ewaluacja zastosowanych modeli

W następnym kroku dokonano oceny zastosowanych modeli poprzez wyliczenie dla każdego z nich (dla rezultatów jego zastosowania):

- Wartości błędu średniokwadratowego (*Mean Squarred Error*, MSE) – uśrednionych wartości funkcji kosztu SSE, tj. sumy kwadratów błędu (*Sum of Squared Errors*). Jest to suma kwadratów różnic pomiędzy wartościami rzeczywistymi a wartościami przewidywanymi, podzielona przez liczbę próbek. A zatem, im mniejsza będzie wartość MSE (0 oznacza brak błędów), tym bardziej dokładny będzie zastosowany model – tym bardziej trafnych predykcji będzie dokonywał.
- Współczynnika determinacji R^2 – jest to standaryzowana wersja metody (współczynnika) MSE, liczona w oparciu o iloraz SSE i SST. SST jest całkowitą sumą kwadratów (*Sum of Squared Total*) – sumą kwadratów różnic pomiędzy wartościami rzeczywistymi a wartością średnią. Dla zestawu danych treningowych współczynnik R^2 przyjmuje wartości od 0 do 1 (gdzie 1 oznacza najlepsze dopasowanie – brak błędów, a więc MSE równy 0), natomiast dla danych testowych może być też wartością ujemną[2].

Powyższe współczynniki wyliczono przy tym odrębnie dla podzbioru zawierającego dane treningowe oraz dla podzbioru zawierającego dane testowe.

Do wyliczenia tych współczynników wykorzystano funkcje *mean_squared_error* oraz *r2_score* z modułu *sklearn.metrics* biblioteki *scikit-learn*, a także zdefiniowaną funkcję *score*.

```
def score(X, y, estimator, estimator_name):
    y_pred = estimator.predict(X)
    print(estimator_name + '\nMSE: %.3f,\nR^2: %.3f\n' % (mean_squared_error(y, y_pred),
                                                         r2_score(y, y_pred)))


score(X_train, y_train, lr, "LR: zbiór danych treningowych:")
score(X_test, y_test, lr, "LR: zbiór danych testowych:")

score(X_train, y_train, ransac, "RANSAC: zbiór danych treningowych:")
score(X_test, y_test, ransac, "RANSAC: zbiór danych testowych:")

score(X_train_quad, y_train, lr_quadratic, "^2: zbiór danych treningowych:")
score(X_test_quad, y_test, lr_quadratic, "^2: zbiór danych testowych:")
```

[2] S. Raschka, V. Mirjalili, *Python. Machine learning i deep learning. Biblioteki scikit-learn i TensorFlow 2*, wyd. 3, Gliwice 2021, s. 326.

```
score(X_train_cubic, y_train, lr_cubic, "^3: zbiór danych treningowych:")
score(X_test_cubic, y_test, lr_cubic, "^3: zbiór danych testowych:")
```

2.3. Model regresji logistycznej

2.3.1. Uwagi ogólne

W ramach realizacji projektu dokonano też sprawdzenia działania klasyfikacji dokonywanej przy pomocy regresji logistycznej. Wbrew temu, co mogłaby sugerować nazwa, regresja logistyczna nie jest modelem regresji, ale właśnie klasyfikacji. Służy on do rozwiązywania problemów o charakterze liniowym i binarnym (sprawdza się szczególnie w przypadku klas rozdzielnych liniowo)[3].

Model regresji logistycznej pozwala nie tylko na przewidywanie klasyfikacji (etykiet klas) poszczególnych próbek ze zbiorów danych, ale również na wskazanie, z jakim prawdopodobieństwem określona próbka przynależy do danej klasy.

2.3.2. Podział zbioru danych i ich standaryzacja

W celu zastosowania modelu regresji logistycznej w pierwszej kolejności postanowiono przygotować dwa zbiory danych – zbiór „data_product_2", będący prostym importem danych ze zbioru „product2.csv" (z usuniętą kolumną z liczbą porządkową poszczególnych rekordów), oraz zbiór „data_product_2_clean", w którym usunięto te duplikaty rekordów, które miały takie same wartości cech „x" i „y", ale różne klasyfikacji w ramach kolumny „good" – traktując takie rekordy jako dane nieprawidłowe (uwagi w tym zakresie przedstawiono w p. 2.1.2 wyżej). Zbiory te będą określone dalej jako (odpowiednio) „dane niezmodyfikowane" oraz „dane »wyczyszczone«".

Aby przygotować takie zbiory danych posłużono się biblioteką *pandas* oraz zawartymi w niej funkcjami, m.in. *duplicated, drop_duplicates* i *merge*. W pierwszej kolejności uzyskano zbiór rekordów zawierających wszystkie duplikaty, tj. rekordy o takich samych parametrach „x" i „y" (257 rekordów). Następnie z takiego zbioru usunięto te rekordy, które mają również taką samą klasę w kolumnie „good" („dobre duplikaty"), uzyskując tym sposobem zbiór rekordów mających takie same wartości „x" i „y", ale różne wartości „good" („złe duplikaty"). Następnie rekordy o takich parametrach „x" i „y", jak w uzyskanym zbiorze, usunięto z kopii zbioru „data_product_2" – uzyskując „wyczyszczony" zbiór „data_product_2_clean", zawierający 399 rekordów (usunięto 101 rekordów uznanych za nieprawidłowe).

```
data_product_2 = pd.read_csv('../data/report_3_regression_models_product2.csv')

data_product_2.drop([data_product_2.columns[0]], axis=1, inplace=True)

data_product_2_temp = data_product_2.copy()
data_product_2_temp = data_product_2_temp[data_product_2_temp.duplicated(subset=
                                                                         ['x', 'y'],
                                                                         keep=False)]
data_product_2_temp.drop_duplicates(keep='first', inplace=True)
data_product_2_temp = data_product_2_temp[data_product_2_temp.duplicated(subset=
                                                                         ['x', 'y'],
                                                                         keep=False)]

data_product_2_clean = data_product_2.merge(data_product_2_temp, how="outer",
                                            indicator=True)
data_product_2_clean = data_product_2_clean.loc[data_product_2_clean["_merge"]
                                                == "left_only"].drop("_merge", axis=1)
```

[3] S. Raschka, V. Mirjalili, dz. cyt., s. 82.

```
print(data_product_2)
print(data_product_2_clean)
```

```
        x         y  good
0     8.8 -3.339879     1
1     9.5 -3.535988     1
2    10.1 -3.681798     1
3     9.9 -3.536677     1
4     9.9 -3.536677     1
..    ...       ...   ...
495  12.5 -3.807757     1
496   8.6 -3.296743     0
497  12.5 -3.800713     1
498  10.0 -3.775396     0
499   8.6 -3.296743     0

[500 rows x 3 columns]
        x         y  good
0     8.8 -3.339879     1
1     8.8 -3.339879     1
2     9.5 -3.535988     1
3     9.5 -3.535988     1
4    10.1 -3.681798     1
..    ...       ...   ...
495  10.7 -3.907848     1
496  12.5 -3.807757     1
497   8.6 -3.296743     0
498   8.6 -3.296743     0
499  12.5 -3.800713     1

[399 rows x 3 columns]
```

Następnie każdy z tych zbiorów podzielono na dwa odrębne podzbiory:

- podzbiór „X” („X_clean”) zawierający informacje o parametrach „x” i „y” dla poszczególnych rekordów,
- podzbiór „y” („y_clean”) zawierający informację o etykietach klas próbek.

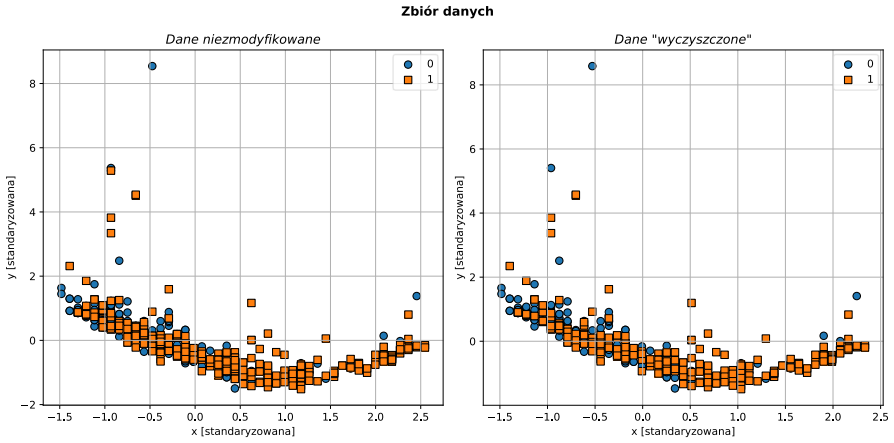
```
X = data_product_2.drop('good', axis=1)
X_clean = data_product_2_clean.drop('good', axis=1)

y = data_product_2['good']
y_clean = data_product_2_clean['good']
```

Dokonano również standaryzacji danych znajdujących się w podzbiorach „X” oraz „X_clean” (tylko, gdyż zbiory „y” i „y_clean” zawierają informacje o etykietach klasy rekordów); więcej uwag w zakresie standaryzacji przedstawiono w p. 2.2.1 wyżej.

```
stdsc = StandardScaler()

X_std = stdsc.fit_transform(X)
X_clean_std = stdsc.fit_transform(X_clean)
```

Poniżej zwizualizowano zbiór danych „product2.csv” po dokonaniu powyższych działań oraz standaryzacji (odrębnie dla danych niezmodyfikowanych i dla danych „wyczyszczonych”). Wykorzystano w tym celu moduł *matplotlib.pyplot* z biblioteki *matplotlib.* Każdy punkt na wykresie to odrębna próbka ze zbioru.

```
f, (plt1, plt2) = plt.subplots(1, 2, figsize=(12, 6))

plt1.scatter(ma.array(X_std[:, 0], mask=y), ma.array(X_std[:, 1], mask=y),
             c='#1f77b4', marker='o', edgecolor='black', s=50, label="0")
plt1.set(xlabel="x [standaryzowana]", ylabel="y [standaryzowana]")
plt1.scatter(ma.array(X_std[:, 0], mask=np.logical_not(y)),
             ma.array(X_std[:, 1], mask=np.logical_not(y)),
             c='#ff7f0e', marker='s', edgecolor='black', s=50, label="1")
plt1.set(xlabel="x [standaryzowana]", ylabel="y [standaryzowana]")

plt1.legend(scatterpoints=1)
plt1.set_title("Dane niezmodyfikowane", style="italic")
plt1.grid()

plt2.scatter(ma.array(X_clean_std[:, 0], mask=y_clean),
             ma.array(X_clean_std[:, 1], mask=y_clean),
             c='#1f77b4', marker='o', edgecolor='black', s=50, label="0")
plt2.set(xlabel="x [standaryzowana]", ylabel="y [standaryzowana]")
plt2.scatter(ma.array(X_clean_std[:, 0], mask=np.logical_not(y_clean)),
             ma.array(X_clean_std[:, 1], mask=np.logical_not(y_clean)),
             c='#ff7f0e', marker='s', edgecolor='black', s=50, label="1")
plt2.set(xlabel="x [standaryzowana]", ylabel="y [standaryzowana]")

plt2.legend(scatterpoints=1)
plt2.set_title("Dane \"wyczyszczone\"", style="italic")
plt2.grid()

f.suptitle("Zbiór danych", weight='bold')
plt.tight_layout()
```



2.3.3. Podział danych na dane treningowe i testowe

W następnym kroku dokonano losowego podziału danych na podzbiór danych treningowych (70%) oraz podzbiór danych testowych (30%). Taki podział wykonano odrębnie dla danych niezmodyfikowanych („X_std" i „y") oraz danych „wyczyszczonych" („X_clean_std" i „y_clean"), wykorzystując do tego funkcję *train_test_split* z parametrem *stratify=y* (*stratify=y_clean*). Dzięki zastosowaniu tego parametru podzbiory zwracane przez funkcję zachowują takie same proporcje etykiet klas, jak zbiór wejściowy.

```
X_train, X_test, y_train, y_test = train_test_split(X_std, y, test_size=0.30,
                                                    random_state=0, stratify=y)
```

```
X_clean_train, X_clean_test, y_clean_train, y_clean_test = train_test_split(X_clean_std,
                                                                            y_clean,
                                                                            test_size
                                                                            =0.30,
                                                                            random_state
                                                                            =0,
                                                                            stratify=
                                                                            y_clean)
```

2.3.4. Zastosowanie modelu regresji logistycznej

Do tak przygotowanych podzbiorów danych zastosowano model regresji logistycznej, wykorzystując do tego funkcję *LogisticRegression* z modułu *sklearn.linear_model* biblioteki *scikit-learn*. Wskazaną funkcję zastosowano używając m.in. następujących parametrów:

- *C=0.1*, co oznacza, że zastosowano wysoką regularyzację modelu (im niższy parametr C, tym większa regularyzacja),
- *multi_class='ovr'*, ponieważ klasyfikacja poszczególnych próbek w analizowanym przypadku ma charakter binarny (każda próbka należy do kategorii „0" lub „1").

```
logreg = LogisticRegression(C=0.1, random_state=0, solver='lbfgs', multi_class='ovr')

logreg.fit(X_train, y_train)

# [...]

logreg.fit(X_clean_train, y_clean_train)
```

2.3.1. Ewaluacja zastosowanego modelu

Do oceny rezultatów zastosowania regresji logistycznej bardziej odpowiednie wydają się sposoby ewaluacji właściwe algorytmom klasyfikującym. Wobec tego określono procentową dokładność zastosowanego modelu, stosując metodę *score* (z biblioteki *scikit-learn*) oraz zdefiniowaną funkcję *score_logreg.*

```
def score_logreg(X, y, estimator, estimator_name):
    y_pred = estimator.predict(X)
    print(estimator_name + "Score: %.3f" % (logreg.score(X, y)))
```

Powyższą wartość wyliczono przy tym odrębnie dla danych treningowych oraz testowych, w odniesieniu do każdego z zestawów danych – tj. danych niezmodyfikowanych i danych „wyczyszczonych".

W dalszym kroku dokonano oceny w odniesieniu do każdego z tych zestawów danych, ale usuwając z nich kolumny zawierające informację o wartości cechy „y" – chcąc ocenić, czy model zadziała gorzej, jeżeli w zestawach danych pod uwagę zostanie wzięta tylko kolumna zawierająca informacje o wartości „x".

```
logreg.fit(X_train, y_train)

score_logreg(X_train, y_train, logreg, "LogReg: dane niezmodyfikowane: zbiór danych"
                                       "treningowych: ")
score_logreg(X_test, y_test, logreg, "LogReg: dane niezmodyfikowane: zbiór danych"
                                     "testowych: ")
```

```
logreg.fit(X_clean_train, y_clean_train)

score_logreg(X_clean_train, y_clean_train, logreg, "LogReg: dane \"wyczyszczone\":"
                                                   "zbiór danych treningowych: ")
score_logreg(X_clean_test, y_clean_test, logreg, "LogReg: dane \"wyczyszczone\":"
                                                 "zbiór danych testowych: ")

# [...]

X_train = np.delete(X_train, 0, 1)
X_test = np.delete(X_test, 0, 1)
logreg.fit(X_train, y_train)

score_logreg(X_train, y_train, logreg, "LogReg: dane niezmodyfikowane (bez \"y\"):"
                                       "zbiór danych treningowych: ")
score_logreg(X_test, y_test, logreg, "LogReg: dane niezmodyfikowane(bez \"y\"):"
                                     "zbiór danych testowych: ")

X_clean_train = np.delete(X_clean_train, 0, 1)
X_clean_test = np.delete(X_clean_test, 0, 1)
logreg.fit(X_clean_train, y_clean_train)

score_logreg(X_clean_train, y_clean_train, logreg, "LogReg: dane \"wyczyszczone\" (bez"
                                                   "\"y\"): zbiór danych treningowych:")
score_logreg(X_clean_test, y_clean_test, logreg, "LogReg: dane \"wyczyszczone\" (bez"
                                                 "\"y\"): zbiór danych testowych: ")
```

3. Prezentacja wyników

3.1. Modele regresji liniowej i wielomianowej

3.1.1. Uzyskane wartości współczynników MSE i R^2

Wartości współczynników MSE i R^2 (zaokrąglone do trzech miejsc po przecinku) uzyskane w odniesieniu do zastosowanych modeli regresji liniowej i wielomianowej przedstawiono w tabeli poniżej – odrębnie dla poszczególnych modeli regresywnych i podzbiorów danych (treningowych i testowych).

	Model regresji	**Zbiór danych**	**MSE**	**R^2**
1.	Liniowej	Treningowych	0.296	0.465
2.		Testowych	0.212	0.541
3.	Liniowej (RANSAC)	Treningowych	0.414	0.252
4.		Testowych	0.319	0.311
5.	Wielomianowej drugiego stopnia	Treningowych	0.117	0.788
6.		Testowych	0.086	0.813
7.	Wielomianowej trzeciego stopnia	Treningowych	0.107	0.806
8.		Testowych	0.080	0.827

3.1.2. Wizualizacja efektów zastosowania modeli regresji

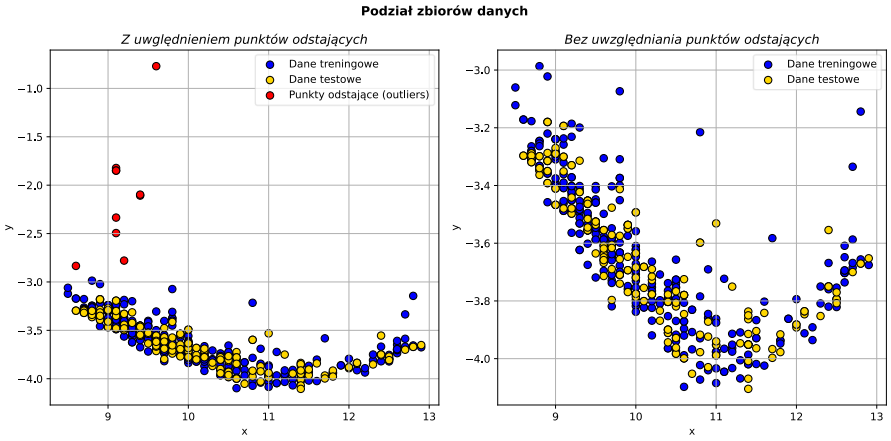
Efekty zastosowania modeli regresywnych zostały zwizualizowane poniżej w formie wykresu punktowego – przy wykorzystaniu modułu *matplotlib.pyplot* z biblioteki *matplotlib.* Każdy punkt na wykresie to odrębny

rekord z zestawu danych. Linie regresji zostały przy tym wykreślone w oparciu o przygotowany zbiór „X_fit" z dopasowanymi danymi (zob. p. 2.2.3 wyżej).

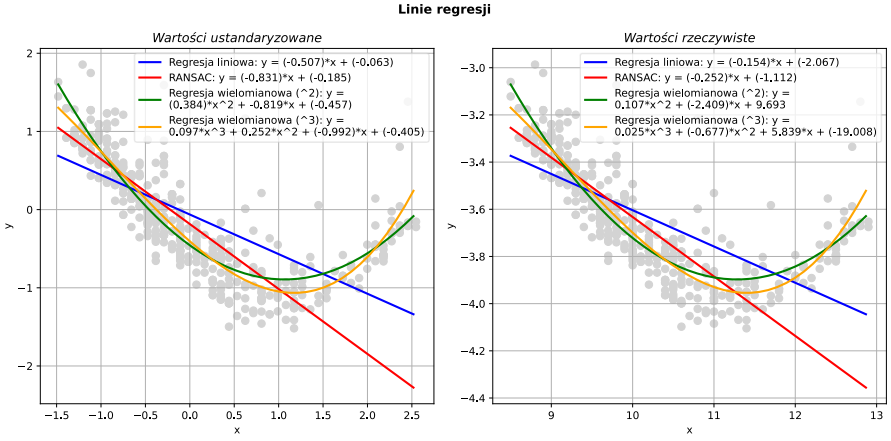
```
X_fit = np.arange(X_std.min(), X_std.max(), 0.1)[:, np.newaxis]
# X_fit = np.sort(X_std.flatten())[:, np.newaxis]
# [...]

y_lr = lr.predict(X_fit)
y_ransac = ransac.predict(X_fit)
y_quadratic = lr_quadratic.predict(quadratic.fit_transform(X_fit))
y_cubic = lr_cubic.predict(cubic.fit_transform(X_fit))
```

Ponadto, aby wyrazić cechy „x" i „y" w wartościach rzeczywistych – aby „odwrócić" standaryzację tych danych – wykorzystano metodą *inverse_ transform* (zob. p. 2.2.2 wyżej).

Ze względu na czytelność na osobnym wykresie przedstawiono podział podzbiorów danych, tzn. poszczególne rekordy w podziale na podzbiór treningowy i podzbiór testowy.



Na następnej parze wykresów przedstawiono linie regresji dla poszczególnych modeli regresywnych, wraz z wzorami uzyskanych modeli – z wartościami „x" i „y" oraz współczynnikami (wagowymi i punktu przecięcia z osią y), wyrażonymi w wartościach ustandaryzowanych oraz w wartościach rzeczywistych.



3.2. Model regresji logistycznej

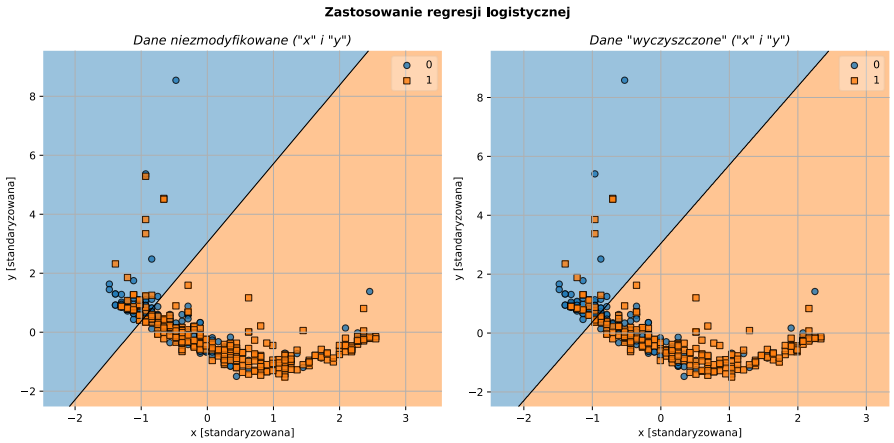
3.2.1. Procentowa dokładność zastosowanego modelu

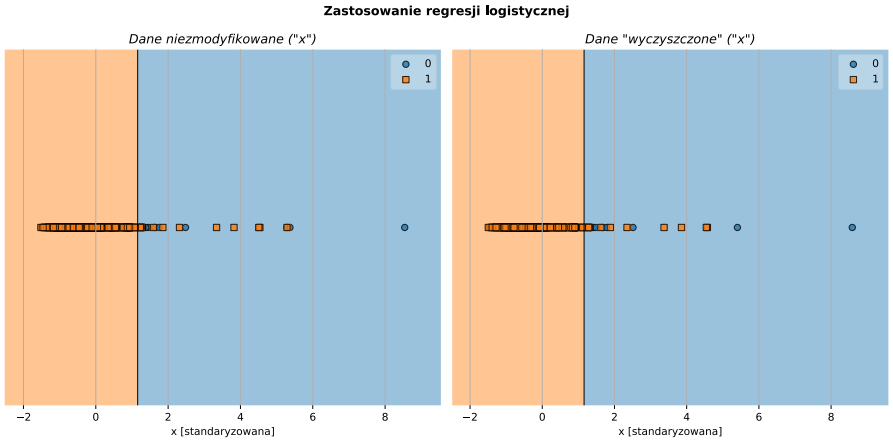
Uzyskane wartości procentowej dokładności zastosowanego modelu regresji logistycznej przedstawiono w tabeli poniżej – odrębnie dla poszczególnych zestawów danych – niezmodyfikowanych oraz „wyczyszczonych", w tym także z uwzględnieniem parametru „y" w zbiorach danych oraz bez jego uwzględniania – a także w podziale na dane treningowe oraz testowe.

	Zestaw danych	**Zbiór danych**	***Score***
1.	Niezmodyfikowane	Treningowych	0.666
2.		Testowych	0.667
3.	„Wyczyszczone"	Treningowych	0.685
4.		Testowych	0.783
5.	Niezmodyfikowane bez wartości „y"	Treningowych	0.629
6.		Testowych	0.620
7.	„Wyczyszczone" bez wartości „y"	Treningowych	0.652
8.		Testowych	0.633

3.2.2. Wizualizacja efektów zastosowania modelu regresji logistycznej

Efekty zastosowania modelu regresji logistycznej zwizualizowano również poniżej w formie wykresu punktowego – przy wykorzystaniu funkcji *plot*_decision_regions z modułu *mlxtend.plotting* biblioteki *mlxtend* oraz modułu *matplotlib.pyplot* z biblioteki *matplotlib.* Każdy punkt na wykresie to odrębny rekord z odpowiedniego zestawu danych.



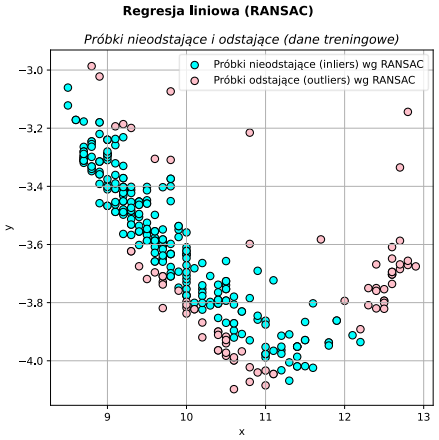


4. Komentarze i wnioski

4.1. Modele regresji liniowej i wielomianowej

W oparciu o wykonane czynności i uzyskane wyniki w odniesieniu do zastosowanych modeli regresji liniowej i wielomianowej można wskazać, że:

- **Najlepsze wyniki uzyskano w rezultacie zastosowania modelu regresji wielomianowej trzeciego stopnia** – w jego przypadku:
  - współczynnik MSE zbliżał się do 0 (0.107 dla danych treningowych, 0.080 dla danych testowych; wartość 0 oznaczałaby brak błędów),
  - współczynnik R^2 wyniósł powyżej 0.8 (0.806 dla danych treningowych, 0.827 dla danych testowych; wartość 1 oznaczałaby najlepsze dopasowanie – brak błędów).
- Natomiast **najgorsze wyniki uzyskano w rezultacie zastosowania modelu regresji liniowej z wykorzystaniem algorytmu RANSAC**: MSE o wartościach 0.414 dla danych treningowych i 0.319 dla danych testowych (wartości ok. cztery razy większe od najlepszego rozwiązania) i R^2 o wartościach 0.252 dla danych treningowych i 0.311 dla danych testowych (wartości ok. trzy razy mniejsze o najlepszego rozwiązania).
- Powyższe rezultaty nie zaskakują w tym znaczeniu, że **już po wizualizacji danych w formie wykresu punktowego (zob. p. 2.2.2 wyżej) widać, że próbki układają się w formie „łuku”** – co oznacza, że lepiej dopasowane będą do nich modele regresji wielomianowej niż modele wykorzystujące regresję liniową.
- **Brak dopasowania modeli regresji liniowej do tak układających się danych ilustruje zastosowanie modelu wykorzystującego algorytm RANSAC.** Model ten (jak wskazano w p. 2.2.5 wyżej) jest bardziej odporny na próbki odstające w tym znaczeniu, że już w ramach swojego działania wyodrębnia on próbki odstające (*outliers*). Pomimo dokonania w ramach projektu takiego podziału – usunięcia próbek odstających ze zbiorów danych (przy wykorzystaniu metody opartej o rozstęp ćwiartkowy – zob. p. 2.2.2 wyżej) – zastosowany algorytm uznał kolejne próbki za dane odstające i nie wziął ich pod uwagę przy stosowaniu modelu regresji liniowej. Poniższy wykres punktowy ilustruje te „dodatkowe” próbki uznane za odstające – jak widać za takowe algorytm uznał próbki po prawej stronie, po „wygięciu” łuku (tak jakby takiego wygięcia nie było).



- Z uzyskanych rezultatów zastosowania modeli regresywnych wynika, że **w odniesieniu do żadnego z modeli nie doszło do jego przetrenowania** – w każdym przypadku współczynniki MSE i R^2 uzyskane dla danych testowych nie są gorsze od współczynników wyliczonych dla danych treningowych.
- **Brak pełniejszych informacji na temat tego, czego dotyczą analizowane zbiory danych, utrudnia oceną ich jakości** – w jaki sposób traktować np. pojawiające się w zbiorach rekordy o takich samych parametrach „x" i „y" (zob. w tym zakresie uwagi w p. 2.1.2 wyżej). A takie zduplikowane rekordy mogą wpływać np. na wyodrębnienie próbek odstających – przy wykorzystaniu do tego metody opartej o rozstęp ćwiartkowy – a w rezultacie i na rezultaty zastosowania określonych modeli.

4.2. Model regresji logistycznej

W oparciu o wykonane czynności i uzyskane wyniki w odniesieniu do zastosowanego modelu regresji logistycznej można wskazać, że:

- **Najlepsze wyniki uzyskano w rezultacie zastosowania regresji logistycznej w odniesieniu do danych „wyczyszczonych"** – w tym przypadku procentowa dokładność zastosowanego modelu wyniosła 0.685 dla danych treningowych i 0.783 dla danych testowych.
- Natomiast **najgorsze wyniki uzyskano w rezultacie zastosowania tej regresji w odniesieniu do zestawów danych, z których usunięto wartości „y".** Niemniej, należy zwrócić uwagę, że w przypadku danych niezmodyfikowanych różnice te nie były znaczące (dane niezmodyfikowane, treningowe – 0.666, testowe – 0.667; dane niezmodyfikowane bez wartości „y", treningowe – 0.629, testowe – 0.620).
- W przypadku usunięcia z zestawów danych wartość „y" uzyskanie lepszych rezultatów zastosowania modelu było możliwe dla danych „wyczyszczonych" poprzez zmniejszenie regularyzacji modelu (zwiększenia wartości parametru C w stosowanym algorytmie). Można byłoby na tej podstawie uznać, że **im mniej cech, w oparciu o które działa model, tym regularyzacja powinna być mniejsza** – nie jest wtedy konieczna, a może skutkować nadmiernym dopasowaniem modelu (jego przetrenowaniem). Niemniej, w przypadku danych niezmodyfikowanych zwiększenie parametru C (zmniejszenie regularyzacji) poskutkowało uzyskaniem gorszych wyników.

```
logreg.set_params(C=10)

X_train = np.delete(X_train, 0, 1)
X_test = np.delete(X_test, 0, 1)
logreg.fit(X_train, y_train)

score_logreg(X_train, y_train, logreg, "LogReg: dane niezmodyfikowane (bez \"y\"):"
                                       "zbiór danych treningowych: ")
score_logreg(X_test, y_test, logreg, "LogReg: dane niezmodyfikowane(bez \"y\"):"
                                     "zbiór danych testowych: ")

X_clean_train = np.delete(X_clean_train, 0, 1)
X_clean_test = np.delete(X_clean_test, 0, 1)
logreg.fit(X_clean_train, y_clean_train)

score_logreg(X_clean_train, y_clean_train, logreg, "LogReg: dane \"wyczyszczone\" (bez"
                                                   "\"y\"): zbiór danych treningowych:")
score_logreg(X_clean_test, y_clean_test, logreg, "LogReg: dane \"wyczyszczone\" (bez"
                                                 "\"y\"): zbiór danych testowych: ")
```

```
LogReg: dane niezmodyfikowane (bez "y"):zbiór danych treningowych: Score: 0.634
LogReg: dane niezmodyfikowane(bez "y"):zbiór danych testowych: Score: 0.607
LogReg: dane "wyczyszczone" (bez"y"): zbiór danych treningowych: Score: 0.695
LogReg: dane "wyczyszczone" (bez"y"): zbiór danych testowych: Score: 0.725
```

- Z uzyskanych rezultatów zastosowania regresji logistycznej z parametrem C równym 0.1 (wysoka regularyzacja) wynika, że w przypadku zestawów danych pozbawionych wartości „y" **doszło do przetrenowania modelu** – zarówno przy danych niezmodyfikowanych, jak i „wyczyszczonych" uzyskane rezultaty zastosowania modelu są lepsze dla danych treningowych niż dla danych testowych.
- Podobnie jak w przypadku modeli regresji liniowej i wielomianowej, brak pełniejszych informacji o zbiorach danych utrudniał ocenę ich jakości – i ich odpowiednie przetworzenie w taki sposób, aby zwiększyć skuteczność stosowanego modelu bez jednoczesnego usunięcia ze zbiorów istotnych informacji. Jednocześnie można wskazać, że zastosowany model **uzyskiwał lepsze rezultaty w przypadku danych „wyczyszczonych",** a więc takich danych, które nie zawierały „sprzecznych" informacji (różne etykiety klasy przy tej samej wartości parametrów „x" i „y").

5. Źródła

[b.a.], *Matplotlib – Visualization with Python*, https://matplotlib.org/, [dostęp: 26.05.2023 r.]

[b.a.], *mlxtend*, https://rasbt.github.io/mlxtend/, [dostęp: 26.05.2023 r.]

[b.a.], *pandas - Python Data Analysis Library*, https://pandas.pydata.org/, [dostęp: 26.05.2023 r.]

[b.a.], *scikit-learn: machine learning in Python – scikit-learn 1.2.2 documentation*, https://scikit-learn.org/stable/, [dostęp: 26.05.2023 r.]

S. Raschka, V. Mirjalili, *Python. Machine learning i deep learning. Biblioteki scikit-learn i TensorFlow 2*, wyd. 3, Gliwice 2021.