Praca domowa 4

Bartosz Siński

In [91]:
```
import dalex
import pandas as pd
import numpy as np
np.set_seed = 42
from sklearn.model_selection import train_test_split
```

Zbiór apratments z DALEX

Na zbiorze danych będziemy przewidywac cenę metra kwadratowego mieszkania, przy użyciu Epsilon-Support Vector Regression.

In [92]:
```
df_apartments = dalex.datasets.load_apartments()
df_apartments_test = dalex.datasets.load_apartments_test()
```

In [93]:
```
df_apartments
```

Out[93]:

	m2_price	construction_year	surface	floor	no_rooms	district
1	5897	1953	25	3	1	Srodmiescie
2	1818	1992	143	9	5	Bielany
3	3643	1937	56	1	2	Praga
4	3517	1995	93	7	3	Ochota
5	3013	1992	144	6	5	Mokotow
...	...	...	...	...	...	...
996	6355	1921	44	2	2	Srodmiescie
997	3422	1921	48	10	2	Bemowo
998	3098	1980	85	3	3	Bemowo
999	4192	1942	36	7	1	Zoliborz
1000	3327	1992	112	6	5	Mokotow

1000 rows × 6 columns

In [94]:
```
df_apartments = pd.get_dummies(df_apartments)
df_apartments_test = pd.get_dummies(df_apartments_test)
```

In [95]:
```
X = df_apartments.drop('m2_price',axis=1)
y = df_apartments[['m2_price']]
X_test = df_apartments_test.drop('m2_price',axis=1)
y_test = df_apartments_test[['m2_price']]
X_train, X_val, y_train, y_val = train_test_split(X, y,random_state = 42)
```

SVR z domyślnymi parametrami, standaryzacja i normalizacja danych

In [62]:
```
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
first_regression = SVR(kernel="poly",degree=5)
first_regression.fit(X_train,y_train)
mean_squared_error(first_regression.predict(X_val),y_val)
```

Out[62]:
```
841489.535078139
```

Sprawdzimy czy standaryzacja poprawi wynik.

In [63]:
```
from sklearn import preprocessing
scaler = preprocessing.StandardScaler().fit(X)
scaler2 = preprocessing.StandardScaler().fit(X_test)
X_scaled = scaler.transform(X)
y_scaled = y
X_test_scaled =scaler2.transform(X_test)
y_test_scaled = y_test
X_train_s, X_val_s, y_train_s, y_val_s = train_test_split(X_scaled, y_scaled,random_st
```

In [64]:
```
second_regression = SVR()
second_regression.fit(X_train_s,y_train_s)
mean_squared_error(first_regression.predict(X_val_s),y_val_s)
```

Out[64]:
```
840406.3177960722
```

A następnie znormalizujmy nasze dane.

In [96]:
```
X_n = preprocessing.normalize(X,axis=0)
y_n = y
X_test_n = preprocessing.normalize(X_test,axis=0)
y_test_n = y_test
X_train_n, X_val_n, y_train_n, y_val_n = train_test_split(X_n, y_n,random_state = 42)
```

In [97]:
```
third_regression = SVR()
third_regression.fit(X_train_n,y_train_n)
mean_squared_error(third_regression.predict(X_val_n),y_val_n)
```

Out[97]:
```
793595.4930103313
```

SVR z tuningiem hiperparametrów

In [73]:
```
import warnings
warnings.filterwarnings("ignore")
from sklearn.model_selection import RandomizedSearchCV
fourth_regression = SVR()
grid = { 'C' : [0.001,0.01,0.1,10,100,1000],
            'kernel' : ["rbf"],
            'gamma' : [0.0001,0.001,0.01,0.1,1,10,100,'auto','scale']
}
fourth_regression = RandomizedSearchCV(estimator = fourth_regression,param_distributio
fourth_regression.fit(X_train,y_train)
mean_squared_error(third_regression.predict(X_val),y_val)
```

```
Fitting 5 folds for each of 54 candidates, totalling 270 fits
```

Out[73]:
```
842256.1704445934
```

In [74]:
```
fourth_regression.best_params_
```

Out[74]:
```
{'kernel': 'rbf', 'gamma': 0.001, 'C': 1000}
```

Do trenowania modelu wykorzystaliśmy jedynie jądro gaussowskie, ponieważ inne jądra powodowały znaczące wydłużenie się czasu trenowania modelu.

Wyniki

In [98]:
```
print("Base SVR: " + str(mean_squared_error(first_regression.predict(X_test),y_test))
print("SVR ze standaryzacją zmiennych: " + str(mean_squared_error(second_regression.pr
print("SVR ze normalizacją zmiennych: " + str(mean_squared_error(third_regression.pred
print("SVR z tuningiem hiperparametrów: " + str(mean_squared_error(fourth_regression.p
```

```
Base SVR: 828032.3549650194
SVR ze standaryzacją zmiennych: 788480.0857148026
SVR ze normalizacją zmiennych: 807940.4887160986
SVR z tuningiem hiperparametrów: 557875.8919504094
```

W tym przypadku ręczna standaryzacja poprawiła RMSE naszego wyniku lepiej niż normlalizacja. Tuning hiperparametrów także w znaczący sposób poprawił jakość predykcji naszego modelu, pomimo rozważania przez nas jedynie jądra gaussowskiego

Wybrany przeze mnie zbiór danych - Mobile Price Classification

Zbiór zawiera informacje o telefonach komórkowych, gdzie trzeba zaklasyfikować telefon do danego zakresu cenowego. https://www.kaggle.com/iabhishekofficial/mobile-price-classification

In [100…
```
df_phones = pd.read_csv("./src/phones_train.csv")
```

In [101…
```
df_phones
```

Out[101…

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...
0	842	0	2.2	0	1	0	7	0.6	188	2	...
1	1021	1	0.5	1	0	1	53	0.7	136	3	...
2	563	1	0.5	1	2	1	41	0.9	145	5	...
3	615	1	2.5	0	0	0	10	0.8	131	6	...
4	1821	1	1.2	0	13	1	44	0.6	141	2	...
...	...	...	...	...	...	...	...	...	...	...	...
1995	794	1	0.5	1	0	1	2	0.8	106	6	...
1996	1965	1	2.6	1	0	0	39	0.2	187	4	...
1997	1911	0	0.9	1	1	1	36	0.7	108	8	...
1998	1512	0	0.9	0	4	1	46	0.1	145	5	...
1999	510	1	2.0	1	5	1	45	0.9	168	6	...

2000 rows × 21 columns

In [102…
```
X = df_phones.drop('price_range',axis=1)
y = df_phones[['price_range']]
X_train, X_test,y_train,y_test = train_test_split(X, y,random_state = 42)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,random_state = 42)
```

Bazowy SVC, standaryzacja danych i normalizacja danych

In [103…
```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
first_classification = SVC(random_state=42)
first_classification.fit(X_train,y_train)
accuracy_score(first_classifiaction.predict(X_val),y_val)
```

Out[103…
```
0.96
```

Następnie klasyfikacja z wcześniejsza standaryzacją danych.

In [104…
```
X_scaled = preprocessing.StandardScaler().fit_transform(X)
X_train_s, X_test_S,y_train_s,y_test_S = train_test_split(X_scaled, y,random_state = 4
X_train_s, X_val_s, y_train_s, y_val_s = train_test_split(X_train_s, y_train_s,random_
```

In [105…
```
second_classification = SVC(random_state=42)
second_classification.fit(X_train_s,y_train_s)
accuracy_score(second_classification.predict(X_val_s),y_val_s)
```

Out[105…
```
0.8506666666666667
```

I normalizacją.

In [106…
```
X_n = preprocessing.normalize(X,axis=0)
X_train_n, X_test_n,y_train_n,y_test_n = train_test_split(X_n, y,random_state = 42)
X_train_n, X_val_n, y_train_n, y_val_n = train_test_split(X_train_n, y_train_n,random_
```

In [107…
```
third_classification = SVC(random_state=42)
third_classification.fit(X_train_n,y_train_n)
accuracy_score(third_classification.predict(X_val_n),y_val_n)
```

Out[107…
```
0.8
```

In [109…
```
pd.DataFrame(X_train_n)
```

Out[109…

	0	1	2	3	4	5	6	7	8	9
0	0.026698	0.000000	0.036252	0.031327	0.062149	0.000000	0.026717	0.011592	0.017005	0.017655
1	0.013051	0.000000	0.010358	0.000000	0.025591	0.030964	0.006072	0.019320	0.016541	0.017655
2	0.024060	0.000000	0.025894	0.031327	0.007312	0.000000	0.032790	0.030912	0.015304	0.030896
3	0.012728	0.031782	0.022010	0.000000	0.000000	0.030964	0.016395	0.034775	0.023188	0.035310
4	0.031122	0.031782	0.027189	0.000000	0.029247	0.000000	0.035826	0.003864	0.014068	0.022068
...	...	...	...	...	...	...	...	...	...	...
1120	0.025200	0.031782	0.009063	0.031327	0.014623	0.030964	0.022467	0.023184	0.014841	0.022068
1121	0.031156	0.031782	0.029778	0.000000	0.010968	0.030964	0.026717	0.007728	0.022879	0.030896
1122	0.027974	0.000000	0.036252	0.000000	0.000000	0.030964	0.026717	0.038639	0.021179	0.035310
1123	0.017764	0.031782	0.036252	0.031327	0.025591	0.000000	0.020038	0.023184	0.019942	0.017655
1124	0.011298	0.000000	0.019421	0.000000	0.018279	0.030964	0.003643	0.034775	0.025353	0.008827

1125 rows × 20 columns

SVC z tuningiem hiperparametrów

In [114…
```
import warnings
warnings.filterwarnings("ignore")
from sklearn.model_selection import RandomizedSearchCV
fourth_classification = SVC(random_state=42)
grid = { 'C' : [0.001,0.01,0.1,10,100,1000],
            'kernel' : ["rbf","poly"],
            'gamma' : [0.0001,0.001,0.01,0.1,1,10,100,'auto','scale'],
          'degree' : [1,2,3,4,5]
}
fourth_classification = RandomizedSearchCV(estimator = fourth_classification,param_dis
fourth_classification.fit(X_train,y_train)
accuracy_score(fourth_classification.predict(X_val),y_val)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

Out[114…
```
0.96
```

In [111…
```
fourth_classification.best_params_
```

Out[111…
```
{'kernel': 'poly', 'gamma': 0.0001, 'degree': 2, 'C': 0.001}
```

Na zbiorze walidacyjnym widać, że nasz tuning, nie podniósł za bardzo accuracy naszego modelu. Może to być spowodowane bardzo mocnym rozstrzałem w wartościach dobieranych parametrów. Spróbujemy więc jeszcze raz z wartościami bliżej tych, które przed chwilą otrzymalismy.

In [115…
```
import warnings
warnings.filterwarnings("ignore")
from sklearn.model_selection import RandomizedSearchCV
fifth_classification = SVC(random_state=42)
grid = { 'C' : [0.0001,0.0005,0.001,0.002,0.005,0.7,0.01],
            'kernel' : ["poly"],
            'gamma' : [0.00001,0.00005,0.0001,0.0002,0.0005,0.0007,0.001,'auto','sca
          'degree' : [1,2,3]
}
fifth_classification = RandomizedSearchCV(estimator = fifth_classification,param_distr
fifth_classification.fit(X_train,y_train)
accuracy_score(fifth_classification.predict(X_val),y_val)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

Out[115…
```
0.976
```

In [113…
```
fifth_classification.best_params_
```

Out[113…
```
{'kernel': 'poly', 'gamma': 0.0005, 'degree': 1, 'C': 0.7}
```

Wyniki

In [116…
```
print("Base SVC: " + str(accuracy_score(first_classification.predict(X_test),y_test))
print("SVC ze standaryzacją zmiennych: " + str(accuracy_score(second_classification.pr
print("SVC ze normalizacją zmiennych: " + str(accuracy_score(third_classification.pred
print("SVC z tuningiem hiperparametrów: " + str(accuracy_score(fourth_classification.p
print("SVC z tuningiem hiperparametrów++: " + str(accuracy_score(fifth_classification
```

```
Base SVC: 0.948
SVC ze standaryzacją zmiennych: 0.864
SVC ze normalizacją zmiennych: 0.818
SVC z tuningiem hiperparametrów: 0.962
SVC z tuningiem hiperparametrów++: 0.98
```

Standaryzacja zdecydowanie obniżyła wynik accuracy naszego modelu. Może to być spowodowane istnieniem w naszych danych wielu zmiennych przyjmujących jedynie wartości 0 i 1, które są dalekie od rozkładu normlanego. Normalizacja zmiennych też obniżyła accuracy modelu. Z kolei pierwszy tuning hiperparametrów podniósł accuracy o 1,4%. Kolejny tuning z dokładniejszymi parametrami dodatkowo podniósł accuracy naszego modelu na zbiorze testowym.