# WUM_PD4_Szymon_Recko

May 3, 2021

## 1 PD 4

[23]:
```
pip install dalex
```

```
Requirement already satisfied: dalex in /usr/local/lib/python3.7/dist-packages
(1.1.0)
Requirement already satisfied: pandas>=1.1.2 in /usr/local/lib/python3.7/dist-
packages (from dalex) (1.1.5)
Requirement already satisfied: numpy>=1.18.4 in /usr/local/lib/python3.7/dist-
packages (from dalex) (1.19.5)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-
packages (from dalex) (56.0.0)
Requirement already satisfied: tqdm>=4.48.2 in /usr/local/lib/python3.7/dist-
packages (from dalex) (4.60.0)
Requirement already satisfied: plotly>=4.12.0 in /usr/local/lib/python3.7/dist-
packages (from dalex) (4.14.3)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-
packages (from pandas>=1.1.2->dalex) (2018.9)
Requirement already satisfied: python-dateutil>=2.7.3 in
/usr/local/lib/python3.7/dist-packages (from pandas>=1.1.2->dalex) (2.8.1)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages
(from plotly>=4.12.0->dalex) (1.15.0)
Requirement already satisfied: retrying>=1.3.3 in /usr/local/lib/python3.7/dist-
packages (from plotly>=4.12.0->dalex) (1.3.3)
```

[24]:
```python
from sklearn.svm import SVR
import pandas as pd
import numpy as np
from dalex import datasets
from sklearn.model_selection import train_test_split
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
```

# 2 Apartments

## 2.1 Wczytanie danych i podział na zbiory

[25]:

```
df_train=datasets.load_apartments()
df_test=datasets.load_apartments_test()
```

[26]:

```
X_train=df_train.drop(["m2_price"],axis=1)
y_train=df_train["m2_price"]
X_test=df_test.drop(["m2_price"],axis=1)
y_test=df_test["m2_price"]
```

[28]:

```
X_train=pd.get_dummies(X_train)
X_test=pd.get_dummies(X_test)
X_test, X_val, y_test, y_val=train_test_split(X_test,y_test,random_state = 123)
```

## 2.2 Podstawowy model

[29]:

```
model=SVR()
model.fit(X_train,y_train)
print("RMSE on test set: ",mean_squared_error(model.predict(X_test),y_test,squared=False))
```

```
RMSE on test set:  917.6127943909642
```

## 2.3 Standaryzacja

[30]:

```
scal1 = StandardScaler()
scal2 = StandardScaler()
scal1.fit(X_train)
scal2.fit(X_test)
X_train_scaled=scal1.transform(X_train)
X_test_scaled=scal2.transform(X_test)
```

[31]:

```
model_scaled=SVR()
model_scaled.fit(X_train_scaled,y_train)
print("RMSE on scaled test set: ",mean_squared_error(model_scaled.predict(X_test_scaled),y_test,squared=False))
```

```
RMSE on scaled test set:  887.3459683040384
```

## 2.4 Tuning hiperparametrów

Nie używam tutaj innych kerneli ponieważ trenowanie ich jest strasznie czasochłonne. Pociąga to za sobą, że sprawdzanie różnych wartości ‘degree’ nie ma sensu, ponieważ tylko kernel ‘poly’ bierze go pod uwagę.

[32]:

```
scal3 = StandardScaler()
scal3.fit(X_val)
X_val_scaled=scal1.transform(X_val)
model_tuning=SVR()
params={'gamma':[0.0001,0.001,0.01,0.1,1,'auto','scale'],
        'C':[0.1,0.5,1,10,100,1000]}
RS=RandomizedSearchCV(model_tuning,params,random_state=123,cv=5,n_iter=40,n_jobs=-1)
RS.fit(X_val,y_val)
RS.best_params_
```

[32]:

```
{'C': 1000, 'gamma': 0.01}
```

[33]:

```
print("RMSE with tuning on test set: ",mean_squared_error(RS.
 ↪predict(X_test),y_test,squared=False))
```

```
RMSE with tuning on test set:  709.3565608572167
```

# 3 Dragons

## 3.1 Wczytanie danych i podział na zbiory

[34]:

```
df_train=datasets.load_dragons()
df_test=datasets.load_dragons_test()
```

[35]:

```
X_train=df_train.drop(["life_length"],axis=1)
y_train=df_train["life_length"]
X_test=df_test.drop(["life_length"],axis=1)
y_test=df_test["life_length"]
X_test, X_val, y_test, y_val=train_test_split(X_test,y_test,random_state = 123)
```

[37]:

```
X_train=pd.get_dummies(X_train)
X_test=pd.get_dummies(X_test)
X_test, X_val, y_test, y_val=train_test_split(X_test,y_test,random_state = 123)
```

## 3.2 Podstawowy model

[38]:

```
model=SVR()
model.fit(X_train,y_train)
print("RMSE on test set: ",mean_squared_error(model.
 ↪predict(X_test),y_test,squared=False))
```

```
RMSE on test set:  501.78770081655085
```

### 3.3 Standaryzacja

[39]:
```
scal1 = StandardScaler()
scal2 = StandardScaler()
scal1.fit(X_train)
scal2.fit(X_test)
X_train_scaled=scal1.transform(X_train)
X_test_scaled=scal2.transform(X_test)
```

[40]:
```
model_scaled=SVR()
model_scaled.fit(X_train_scaled,y_train)
print("RMSE on scaled test set: ",mean_squared_error(model_scaled.
 ↪predict(X_test_scaled),y_test,squared=False))
```

```
RMSE on scaled test set:  457.32066298156246
```

### 3.4 Tuning hiperparametrów

[41]:
```
scal3 = StandardScaler()
scal3.fit(X_val)
X_val_scaled=scal1.transform(X_val)
model_tuning=SVR()
params={'gamma':[0.0001,0.001,0.01,0.1,1,'auto','scale'],
        'C':[0.1,0.5,1,10,100,1000]}
RS=RandomizedSearchCV(model_tuning,params,random_state=123,cv=3,n_jobs=-1)
RS.fit(X_val,y_val)
RS.best_params_
```

[41]:
```
{'C': 1000, 'gamma': 'auto'}
```

[42]:
```
print("RMSE with tuning on test set: ",mean_squared_error(RS.
 ↪predict(X_test),y_test,squared=False))
```

```
RMSE with tuning on test set:  493.83090058129267
```

## 4 Wnioski

Zauważamy, że standaryzacja danych i tuning hiperparametrów w różnym stopniu poprawiają dokładność modelu, w zależności od problemu którym się zajmujemy i ilości danych, które posiadamy.