[WUM] Praca domowa nr4 Kacper Kurowski.ipynb

May 7, 2021

1 [WUM] PD4

1.1 Kacper Kurowski

Wczytajmy paczki

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns

from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_validate
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import (train_test_split, RandomizedSearchCV)
from matplotlib import pyplot as plt
from dalex.datasets import load_apartments

import warnings
warnings.filterwarnings('ignore')
np.random.seed = 42
import category_encoders as ce
```

1.2 Apartments

raz dane apartamentów. Przy okazji zakodujmy dystrykty przy pomocy zmiennej m2_price - załóżmy, że mniej więcej ceny apartamentów mocno zależą od tego, gdzie się znajdują

[2]:
```
ap = load_apartments()

target_encoder = ce.TargetEncoder(cols=['district'])
ap = target_encoder.fit_transform(ap, ap['m2_price'])
```

Podzielmy dane na zbiór testowy i treningowy

[3]:
```
y = np.array(ap['m2_price'])
X = ap.drop(['m2_price'],axis=1)
```

```
ap_X_train, ap_X_test, ap_y_train, ap_y_test = train_test_split(X, y,␣
↪test_size=0.25, random_state=1618)
```

Skorzystajmy z SVM na nieprzeskalowanych danych

[5]:
```
from sklearn.svm import SVR
param_grid_SVM_rbf = {
    'C': [1, 10, 50, 100, 150, 200, 300],
    'degree': [2, 3, 4, 5],
    'gamma': ['scale', 'auto', 0.5, 0.75, 1, 1.25, 1.5],
}

svr_unscal_rbf = SVR()
svr_unscal_grid_rbf = RandomizedSearchCV(
    svr_unscal_rbf, param_grid_SVM_rbf,
    scoring ='neg_root_mean_squared_error', n_iter=100, n_jobs=-1)
svr_unscal_grid_rbf.fit(ap_X_train, ap_y_train)
```

[5]:
```
RandomizedSearchCV(estimator=SVR(), n_iter=100, n_jobs=-1,
                   param_distributions={'C': [1, 10, 50, 100, 150, 200, 300],
                                        'degree': [2, 3, 4, 5],
                                        'gamma': ['scale', 'auto', 0.5, 0.75, 1,
                                                  1.25, 1.5]},
                   scoring='neg_root_mean_squared_error')
```

[6]:
```
y_test_hat = svr_unscal_grid_rbf.best_estimator_.predict(ap_X_test)
print("RMSE dla nieprzeskalowanych:", np.sqrt(mean_squared_error(ap_y_test,␣
↪y_test_hat)))
```

```
RMSE dla nieprzeskalowanych: 576.0607396413324
```

Uzyskaliśmy pewien wynik dla danych nieprzeskalowanych w metryce RMSE. Przeskalujmy następnie dane, by sprawdzić, czy przeskalowanie, którego użyli autorzy artykułu rzeczywiście poprawia uzyskiwane rezultaty

[7]:
```
ss = StandardScaler()

ap_X_train_scal = ap_X_train.copy()
ap_X_train_scal[ap_X_train_scal.columns.values] = ss.fit_transform(
    ap_X_train_scal[ap_X_train_scal.columns.values]
)
ap_X_test_scal = ap_X_test.copy()
ap_X_test_scal[ap_X_test_scal.columns.values] = ss.fit_transform(
    ap_X_test_scal[ap_X_test_scal.columns.values]
)
```

[8]:
```
svr_scal_rbf = SVR()
svr_scal_grid_rbf = RandomizedSearchCV(
    svr_unscaled_rbf, param_grid_SVM_rbf,
```

```
    scoring ='neg_root_mean_squared_error', n_iter=100, n_jobs=-1)
svr_scal_grid_rbf.fit(ap_X_train_scal, ap_y_train)
```

[8]:
```
RandomizedSearchCV(estimator=SVR(), n_iter=100, n_jobs=-1,
                   param_distributions={'C': [1, 10, 50, 100, 150, 200, 300],
                                        'degree': [2, 3, 4, 5],
                                        'gamma': ['scale', 'auto', 0.5, 0.75, 1,
                                                  1.25, 1.5]},
                   scoring='neg_root_mean_squared_error')
```

[9]:
```
y_test_hat_scal = svr_scal_grid_rbf.best_estimator_.predict(ap_X_test_scal)
print("RMSE dla przeskalowanych:", np.sqrt(mean_squared_error(ap_y_test,␣
 ↪y_test_hat_scal)))
```

```
RMSE dla przeskalowanych: 207.31293351525915
```

jak widzimy, przeskalowanie danych znacząco poprawiło uzsykiwane rezultaty - tym razem błąd wynosi jedynie 40% poprzednio uzyskanego. Oznacza to, że autorzy artykułu słusznie przeskaowali dane.

1.3 Aus Weather

Dokonajmy następnie podobnej analizy na zbiorze weatherAUS, który juz w jednej z poprzednich prac domowych analizowaliśmy. W tamtej pracy domowej dokonaliśmy kodowania zmiennych, z którego korzystam poniżej.

[12]:
```
aus_wheather = pd.read_csv( "/home/kurowskik/kaggle/weatherAUS.csv", sep = ",",␣
 ↪header=0)
```

[13]:
```
del aus_wheather["Evaporation"]
del aus_wheather["Sunshine"]
del aus_wheather["Cloud9am"]
del aus_wheather["Cloud3pm"]

direction_to_encoding = {
    "N" : [1.0,0.0,0.0,0.0],
    "NNW" : [0.75,0.25,0.0,0.0],
    "NW" : [0.5,0.5,0.0,0.0],
    "WNW" : [0.25,0.66,0.0,0.0],
    "W" : [0.0,1.0,0.0,0.0],
    "WSW" : [0.0,0.75,0.25,0.0],
    "SW" : [0.0,0.5,0.5,0.0],
    "SSW" : [0.0,0.75,0.66,0.0],
    "S" : [0.0,0.0,1.0,0.0],
    "SSE" : [0.0,0.0,0.75,0.25],
    "SE" : [0.0,0.0,0.5,0.5],
    "ESE" : [0.0,0.0,0.25,0.75],
    "E" : [0.0,0.0,0.0,1.0],
    "ENE" : [0.25,0.0,0.0,0.75],
```

```
    "NE" : [0.5,0.0,0.0,0.5],
    "NNE" : [0.75,0.66,0.0,0.25],
    "nan" : [0.0,0.0,0.0,0.0]
}

GustDir = pd.DataFrame(
    aus_wheather["WindGustDir"].fillna("nan").map(direction_to_encoding).
 ↪tolist(),
    columns=['WindGustDirN','WindGustDirW','WindGustDirS','WindGustDirE'],
    index = aus_wheather.index)
aus_wheather = aus_wheather.merge(GustDir, left_index=True, right_index=True)

GustDir9am = pd.DataFrame(
    aus_wheather["WindDir9am"].fillna("nan").map(direction_to_encoding).
 ↪tolist(),
    columns=['WindDir9amN','WindDir9amW','WindDir9amS','WindDir9amE'],
    index = aus_wheather.index)

aus_wheather = aus_wheather.merge(GustDir9am, left_index=True, right_index=True)
GustDir3pm = pd.DataFrame(
    aus_wheather["WindDir3pm"].fillna("nan").map(direction_to_encoding).
 ↪tolist(),
    columns=['WindDir3pmN','WindDir3pmW','WindDir3pmS','WindDir3pmE'],
    index = aus_wheather.index)
aus_wheather = aus_wheather.merge(GustDir3pm, left_index=True, right_index=True)

def encode_dates(x):
    tmp = x.split("-")
    return [float( tmp[0]), float(tmp[1]), float(tmp[2]) ]

dates = pd.DataFrame(
    aus_wheather['Date'].map( encode_dates).tolist(),
    columns=["Year", "Month", "Day"],
    index = aus_wheather.index)
aus_wheather = aus_wheather.merge(dates, left_index=True, right_index=True)

def encodeRain(x):
    if x == "Yes":
        return 1
    elif x == "No":
        return 0

aus_wheather['RainTomorrow'] = aus_wheather['RainTomorrow'].map( encodeRain)
aus_wheather['RainToday'] = aus_wheather['RainToday'].map( encodeRain)

tmp = aus_wheather['Location'].map( lambda x: sum(bytearray(x, 'utf-8'))+len(x)␣
 ↪) # Kodujemy lokację, niestety nieróżnowartościowo
```

```
aus_wheather['Location'] = tmp

del aus_wheather["Date"]
del aus_wheather["WindGustDir"]
del aus_wheather["WindDir9am"]
del aus_wheather["WindDir3pm"]

aus_wheather.fillna('0', inplace=True)
```

jako, że danych jest bardzo dużo, zdecydowałem się ograniczyć liczbę wierszy do 1000 - w ten sposób tyle samo wierszy ile w pierwszym zbiorze danych, więc obliczenia nie są przytłaczające.

[15]:
```
aus_smaller = aus_wheather.sample( 1000, random_state = 1618)
```

[16]:
```
y = np.array(aus_smaller['RainTomorrow'])
X = aus_smaller.drop(['RainTomorrow'],axis=1)


aus_X_train, aus_X_test, aus_y_train, aus_y_test = train_test_split(X, y,␣
↪test_size=0.25, random_state=1618)
```

[18]:
```
param_grid_SVM_poly = {
    'kernel': ['poly'],
    #'gamma': ['scale', 'auto', 0.5, 1, 1.5], niestety z tym nie potrafi␣
↪przeliczyc...
    'degree': [2, 3, 5, 6],
    'C': [1, 10, 100, 150, 200, 300],
}

svr_unscal_poly = SVR()
svr_unscal_grid_poly = RandomizedSearchCV(
    svr_unscal_poly, param_grid_SVM_poly,
    scoring ='neg_root_mean_squared_error', n_iter=50, n_jobs=-1)
svr_unscal_grid_poly.fit(aus_X_train, aus_y_train)
```

[18]:
```
RandomizedSearchCV(estimator=SVR(), n_iter=50, n_jobs=-1,
                   param_distributions={'C': [1, 10, 100, 150, 200, 300],
                                        'degree': [2, 3, 5, 6],
                                        'kernel': ['poly']},
                   scoring='neg_root_mean_squared_error')
```

[19]:
```
y_test_hat_aus = svr_unscal_grid_poly.best_estimator_.predict(aus_X_test)
print("RMSE dla nieprzeskalowanych:", np.sqrt(mean_squared_error(aus_y_test,␣
↪y_test_hat_aus)))
```

```
RMSE dla nieprzeskalowanych: 0.4043850366973795
```

Podobnie jak w przypadku zbioru danych apartamentów, przeskalujmy dane

[22]:
```
ss = StandardScaler()

aus_X_train_scal = aus_X_train.copy()
aus_X_train_scal[aus_X_train_scal.columns.values] = ss.fit_transform(
    aus_X_train_scal[aus_X_train_scal.columns.values]
)
aus_X_test_scal = aus_X_test.copy()
aus_X_test_scal[aus_X_test_scal.columns.values] = ss.fit_transform(
    aus_X_test_scal[aus_X_test_scal.columns.values]
)
```

[24]:
```
svr_scal_poly = SVR()
svr_scal_grid_poly = RandomizedSearchCV(
    svr_scal_poly, param_grid_SVM_poly,
    scoring ='neg_root_mean_squared_error', n_iter=50, n_jobs=-1)
svr_scal_grid_poly.fit(aus_X_train_scal, aus_y_train)
```

[24]:
```
RandomizedSearchCV(estimator=SVR(), n_iter=50, n_jobs=-1,
                   param_distributions={'C': [1, 10, 100, 150, 200, 300],
                                        'degree': [2, 3, 5, 6],
                                        'kernel': ['poly']},
                   scoring='neg_root_mean_squared_error')
```

[26]:
```
y_test_hat_scal = svr_scal_grid_poly.best_estimator_.predict(aus_X_test_scal)
print("RMSE dla przeskalowanych:", np.sqrt(mean_squared_error(aus_y_test,␣
 ↪y_test_hat_scal)))
```

```
RMSE dla przeskalowanych: 0.4218219346120553
```

Tym razem przeskaowanie nie daje tak dobrych rezultatów jak w przypadku zbioru danych apartments. Być może jest to kwestia liczby zmiennych, których w zbiorze ausWeather jest znacznie więcej