PD4

May 11, 2021

1 PD4 - Jan Smoleń

[1]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
import dalex as dx
import pickle
np.random.seed = 46
import shap
import xgboost as xgb
from sklearn.metrics import accuracy_score
import warnings
import plotly
warnings.filterwarnings('ignore')
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
```

1.1 Wczytywanie danych

[2]:
```
aps=dx.datasets.load_apartments()
from sklearn.preprocessing import OneHotEncoder
aps=dx.datasets.load_apartments()
Xa=aps.drop("district", axis=1)
ya=aps["district"]
aps.head()
```

[2]:

	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

Ponieważ mamy użyć SVM, to potraktujemy to zadanie jako klasyfikację ze względu na dzielnicę.

[3]:
```
wines=pd.read_csv("winequality-red.csv")
wines["is_good"] = wines.apply(lambda row: 1 if row.quality > 5 else 0, axis =␣
↪1)
Xw = wines.drop(["quality", "is_good"], axis = 1)
yw = wines[["is_good"]]
wines.head()
```

[3]:
```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.4              0.70         0.00             1.9      0.076
1            7.8              0.88         0.00             2.6      0.098
2            7.8              0.76         0.04             2.3      0.092
3           11.2              0.28         0.56             1.9      0.075
4            7.4              0.70         0.00             1.9      0.076

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                 11.0                  34.0   0.9978  3.51       0.56
1                 25.0                  67.0   0.9968  3.20       0.68
2                 15.0                  54.0   0.9970  3.26       0.65
3                 17.0                  60.0   0.9980  3.16       0.58
4                 11.0                  34.0   0.9978  3.51       0.56

   alcohol  quality  is_good
0      9.4        5        0
1      9.8        5        0
2      9.8        5        0
3      9.8        6        1
4      9.4        5        0
```

Ramkę danych o winie potraktujemy jako zadanie klasyfikacji - czy wino jest dobre (ma ocenę powyżej 5) czy nie.

1.2 Podział na zbiory testowe i treningowe

[4]:
```
from sklearn.model_selection import train_test_split
Xw_train, Xw_test, yw_train, yw_test = train_test_split(Xw, yw, test_size = 0.
↪2, random_state = 1613)
```

[5]:
```
Xa_train, Xa_test, ya_train, ya_test = train_test_split(Xa, ya, test_size = 0.
↪2, random_state = 1613)
```

1.3 SVM

[6]:
```
from sklearn.svm import SVC
svm_a=SVC()
svm_a.fit(Xa_train, ya_train)
svm_w=SVC()
svm_w.fit(Xw_train, yw_train)
```

[6]: SVC()

1.3.1 Bazowe wyniki, bez standaryzacji

[7]:
```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import accuracy_score
```

[8]:
```
ya_preds=svm_a.predict(Xa_test)
accuracy_score(ya_test, ya_preds)
```

[8]: 0.195

[9]:
```
yw_preds=svm_w.predict(Xw_test)
accuracy_score(yw_test, yw_preds)
```

[9]: 0.6625

Bazowe SVM na surowych zbiorach osiąga bardzo słabe wyniki.

1.3.2 Po standaryzacji

[10]:
```
Xa=(Xa-Xa.mean())/Xa.std()
Xw=(Xw-Xw.mean())/Xw.std()
Xw_train, Xw_test, yw_train, yw_test = train_test_split(Xw, yw, test_size = 0.
 ↪2, random_state = 1613)
Xa_train, Xa_test, ya_train, ya_test = train_test_split(Xa, ya, test_size = 0.
 ↪2, random_state = 1613)
svm_a.fit(Xa_train, ya_train)
svm_w.fit(Xw_train, yw_train)
```

[10]: SVC()

[11]:
```
ya_preds=svm_a.predict(Xa_test)
accuracy_score(ya_test, ya_preds)
```

[11]: 0.295

[12]:
```
yw_preds=svm_w.predict(Xw_test)
accuracy_score(yw_test, yw_preds)
```

[12]: 0.76875

Samo standaryzowanie bardzo polepszyło wyniki naszych modeli - o ponad 10%.

1.4 Trening

[18]:
```
svm_a_tuned=SVC(random_state=42)
c=[]  # wartości parametru C
gamma=[]  #wartości parametru gamma
for i in range(-4, 5):      # orientacyjne wartości na podstawie informacji␣
 ↪znalezionych w internecie
    c.append(10**i)
for i in range(-4, 5):
    gamma.append(10**i)
gamma.append("auto")
gamma.append("scale")
params = [{'C': c,
        'gamma': gamma,
        'kernel': ["rbf", "linear"]}]
from sklearn.model_selection import RandomizedSearchCV
rs_svm_a=RandomizedSearchCV(svm_a_tuned, param_distributions=params,␣
 ↪scoring='accuracy', cv=4, n_jobs=2)
#gs_svm=GridSearchCV(svm_a_tuned, param_grid=params, scoring='accuracy', cv=4,␣
 ↪n_jobs=2)
rs_svm_a.fit(Xa_train, ya_train)
rs_svm_a.best_params_
```

[18]:
```
{'kernel': 'rbf', 'gamma': 0.01, 'C': 10000}
```

[19]:
```
rs_svm_a_acc=accuracy_score(gs_svm.predict(Xa_test),ya_test)
rs_svm_a_acc
```

[19]:
```
0.285
```

[23]:
```
svm_w_tuned=SVC(random_state=42)
c=[]  # wartości parametru C
gamma=[]  #wartości parametru gamma
for i in range(-4, 5):      # orientacyjne wartości na podstawie informacji␣
 ↪znalezionych w internecie
    c.append(10**i)
for i in range(-4, 5):
    gamma.append(10**i)
gamma.append("auto")
gamma.append("scale")
params = [{'C': c,
        'gamma': gamma,
        'kernel': ["rbf", "linear"]}]
rs_svm_w=RandomizedSearchCV(svm_w_tuned, param_distributions=params,␣
 ↪scoring='accuracy', cv=4, n_jobs=2)
rs_svm_w.fit(Xw_train, yw_train)
rs_svm_w.best_params_
```

[23]: {'kernel': 'rbf', 'gamma': 'scale', 'C': 10}

[24]:
```
rs_svm_w_acc=accuracy_score(gs_w_svm.predict(Xw_test),yw_test)
rs_svm_w_acc
```

[24]: 0.76875

Po dosyć długim czasie trenowania, dostaliśmy wyniki takie same albo nawet nieznacznie gorsze od bazowego SVM na wystandaryzowanych danych.

[ ]:
```
```