pr_dom4

May 9, 2021

[124]:
```
from sklearn import svm
import dalex
import pandas as pd
from category_encoders import TargetEncoder
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.preprocessing import Normalizer
from sklearn.metrics import mean_squared_error,  make_scorer
import numpy as np
```

[125]:
```
apartments=dalex.datasets.load_apartments()
apartments
```

[125]:

	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 x 6 columns]

[126]:
```
cereal=pd.read_csv("cereal.csv")
print(cereal)
# Ze względu, że SVC wymaga liczb całkowitych jako target zastąpię rating␣
 ↪podłogą z wartości kolumny
cereal["rating"]=cereal["rating"].round(0).astype(int)
```

	name	mfr	type	calories	protein	fat	sodium	fiber	\
0	100% Bran	N	C	70	4	1	130	10.0	
1	100% Natural Bran	Q	C	120	3	5	15	2.0	
2	All-Bran	K	C	70	4	1	260	9.0	

```
3   All-Bran with Extra Fiber   K    C        50        4    0     140   14.0
4               Almond Delight   R    C       110        2    2     200    1.0
..                         ...  ..  ...       ...      ...  ...     ...    ...
72                    Triples   G    C       110        2    1     250    0.0
73                       Trix   G    C       110        1    1     140    0.0
74                 Wheat Chex   R    C       100        3    1     230    3.0
75                   Wheaties   G    C       100        3    1     200    3.0
76         Wheaties Honey Gold   G    C       110        2    1     200    1.0

    carbo  sugars  potass  vitamins  shelf  weight  cups     rating
0     5.0       6     280        25      3     1.0  0.33  68.402973
1     8.0       8     135         0      3     1.0  1.00  33.983679
2     7.0       5     320        25      3     1.0  0.33  59.425505
3     8.0       0     330        25      3     1.0  0.50  93.704912
4    14.0       8      -1        25      3     1.0  0.75  34.384843
..    ...     ...     ...       ...    ...     ...   ...        ...
72   21.0       3      60        25      3     1.0  0.75  39.106174
73   13.0      12      25        25      2     1.0  1.00  27.753301
74   17.0       3     115        25      1     1.0  0.67  49.787445
75   17.0       3     110        25      1     1.0  1.00  51.592193
76   16.0       8      60        25      1     1.0  0.75  36.187559

[77 rows x 16 columns]
```

0.1 Preprocessing

[127]:
```
te=TargetEncoder()
apartments['district']=te.
 ↪fit_transform(apartments['district'],apartments['m2_price'])

y=apartments['m2_price']
X=apartments.iloc[:,1:]
trainx,testx,trainy,testy=train_test_split(X,y,test_size=0.3,random_state=23)
```

```
C:\Users\Maciek\anaconda3\lib\site-packages\category_encoders\utils.py:21:
FutureWarning: is_categorical is deprecated and will be removed in a future
version.  Use is_categorical_dtype instead
  elif pd.api.types.is_categorical(cols):
```

[128]:
```
te=TargetEncoder()
cereal['mfr']=te.fit_transform(cereal['mfr'],cereal['rating'])
```

```
C:\Users\Maciek\anaconda3\lib\site-packages\category_encoders\utils.py:21:
FutureWarning: is_categorical is deprecated and will be removed in a future
version.  Use is_categorical_dtype instead
  elif pd.api.types.is_categorical(cols):
```

[129]:
```
te=TargetEncoder()
cereal['type']=te.fit_transform(cereal['type'],cereal['rating'])
```

```
C:\Users\Maciek\anaconda3\lib\site-packages\category_encoders\utils.py:21:
FutureWarning: is_categorical is deprecated and will be removed in a future
version.  Use is_categorical_dtype instead
  elif pd.api.types.is_categorical(cols):
```

1 Apartments

1.1 Standard parameters without standarisation

[130]:
```
svc=svm.SVC()
svc.fit(trainx,trainy)
ans=svc.predict(testx)
print(f"Mean squared error:  { mean_squared_error(ans,testy, squared=False)}")
```

```
Mean squared error:  932.263381954549
```

1.2 Standard parameters with normalisation

[131]:
```
scaler=Normalizer().fit(apartments.iloc[:,1:])
apartments2=pd.DataFrame(scaler.transform(apartments.iloc[:,1:]))
y=apartments['m2_price']
X=apartments2.iloc[:,1:]
trainx2,testx2,trainy2,testy2=train_test_split(X,y,test_size=0.
 ↪3,random_state=2123)
svc2=svm.SVC()
svc2.fit(trainx2,trainy2)
ans2=svc2.predict(testx2)
print(f"Mean squared error:  { mean_squared_error(ans2,testy2, squared=False)}")
```

```
Mean squared error:  917.2134738798088
```

Jak widać dzięki standaryzacji udało się osiągnąć lepszy wynik

1.3 Hyperparameters Random Search

[132]:
```
model_params={
    "kernel":["rbf","poly"],
    "gamma":['scale','auto'],
    "degree":[2,3,4,5,6]

}
```

[137]:
```

```

```
rs=RandomizedSearchCV(estimator = svm.SVC(),param_distributions = model_params,␣
 ↪cv=3,n_jobs=-1,n_iter=100,␣
 ↪random_state=1613,verbose=1,scoring='neg_root_mean_squared_error')
rs.fit(trainx2,trainy2)
```

```
C:\Users\Maciek\anaconda3\lib\site-
packages\sklearn\model_selection\_search.py:278: UserWarning: The total space of
parameters 20 is smaller than n_iter=100. Running 20 iterations. For exhaustive
searches, use GridSearchCV.
  warnings.warn(
C:\Users\Maciek\anaconda3\lib\site-
packages\sklearn\model_selection\_split.py:670: UserWarning: The least populated
class in y has only 1 members, which is less than n_splits=3.
  warnings.warn(("The least populated class in y has only %d"
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.

Fitting 3 folds for each of 20 candidates, totalling 60 fits

[Parallel(n_jobs=-1)]: Done  26 tasks      | elapsed:    3.9s
[Parallel(n_jobs=-1)]: Done  60 out of  60 | elapsed:    5.8s finished
```

[137]:
```
RandomizedSearchCV(cv=3, estimator=SVC(), n_iter=100, n_jobs=-1,
                   param_distributions={'degree': [2, 3, 4, 5, 6],
                                        'gamma': ['scale', 'auto'],
                                        'kernel': ['rbf', 'poly']},
                   random_state=1613, scoring='neg_root_mean_squared_error',
                   verbose=1)
```

[139]:
```
print(f"Mean squared error:  { rs.best_score_}")
print(f"Hyperparameters:  { rs.best_params_}")
```

```
Mean squared error:  -628.3103133478154
Hyperparameters:  {'kernel': 'poly', 'gamma': 'scale', 'degree': 3}
```

Jak widać dostrojenie hiperparametrów dość mocno wpłynęło na poprawę wyniku.

2 Cereals

Baza danych pobrana z: https://www.kaggle.com/crawford/80-cereals , w której targetem jest rating

[142]:
```
y=cereal['rating']
X=cereal.iloc[:,1:-1]
trainx,testx,trainy,testy=train_test_split(X,y,test_size=0.3,random_state=23)
```

2.1 Standard parameters without standarisation

[143]:
```
svc=svm.SVC()
svc.fit(trainx,trainy)
ans=svc.predict(testx)
print(f"Mean squared error:  { mean_squared_error(ans,testy, squared=False)}")
```

```
Mean squared error:  11.98262631201246
```

2.2 Standard parameters with normalisation

[144]:
```
scaler=Normalizer().fit(apartments.iloc[:,1:])
cereal2=pd.DataFrame(scaler.transform(cereal.iloc[:,1:]))
X=cereal2.iloc[:,1:-1]
y=cereal["rating"]
trainx2,testx2,trainy2,testy2=train_test_split(X,y,test_size=0.
 ↪3,random_state=2123)
svc2=svm.SVC()
svc2.fit(trainx2,trainy2)
ans2=svc2.predict(testx2)
print(f"Mean squared error:  { mean_squared_error(ans2,testy2, squared=False)}")
```

```
Mean squared error:  13.083067937860243
```

Jak można się było spodziewać standaryzacja polepszyła wynik również w drugiej bazie danych.

2.3 Hyperparameters Random Search

[145]:
```
model_params={
    "kernel":["rbf","poly"],
    "gamma":['scale','auto'],
    "degree":[2,3,4,5,6]

}
```

[146]:
```
rs=RandomizedSearchCV(estimator = svm.SVC(),param_distributions = model_params,␣
 ↪cv=3,n_iter=100,␣
 ↪random_state=1613,verbose=1,scoring='neg_root_mean_squared_error')
rs.fit(trainx2,trainy2)
```

```
C:\Users\Maciek\anaconda3\lib\site-
packages\sklearn\model_selection\_search.py:278: UserWarning: The total space of
parameters 20 is smaller than n_iter=100. Running 20 iterations. For exhaustive
searches, use GridSearchCV.
  warnings.warn(
C:\Users\Maciek\anaconda3\lib\site-
packages\sklearn\model_selection\_split.py:670: UserWarning: The least populated
class in y has only 1 members, which is less than n_splits=3.
```

```
  warnings.warn(("The least populated class in y has only %d"
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

Fitting 3 folds for each of 20 candidates, totalling 60 fits

[Parallel(n_jobs=1)]: Done  60 out of  60 | elapsed:    0.1s finished
```

[146]:
```
RandomizedSearchCV(cv=3, estimator=SVC(), n_iter=100,
                   param_distributions={'degree': [2, 3, 4, 5, 6],
                                        'gamma': ['scale', 'auto'],
                                        'kernel': ['rbf', 'poly']},
                   random_state=1613, scoring='neg_root_mean_squared_error',
                   verbose=1)
```

[147]:
```
print(f"Mean squared error:  { rs.best_score_}")
print(f"Hyperparameters:  { rs.best_params_}")
```

```
Mean squared error:  -10.193508870281567
Hyperparameters:  {'kernel': 'poly', 'gamma': 'scale', 'degree': 6}
```

Bez większych niespodzianek random search poprawił wynik.