Praca_domowa_2

March 23, 2021

1 Praca domowa nr 2

Bartosz Sawicki

[1]:
```
import category_encoders as ce
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split

import warnings
warnings.filterwarnings('ignore')
```

1.1 Wczytanie zbioru danych

[2]:
```
url = 'https://www.dropbox.com/s/360xhh2d9lnaek3/allegro-api-transactions.csv?
 ↪dl=1'

input_df = pd.read_csv(url)
input_df.head()
```

[2]:
```
   lp                 date     item_id  \
0   0  2016-04-03 21:21:08  4753602474
1   1  2016-04-03 15:35:26  4773181874
2   2  2016-04-03 14:14:31  4781627074
3   3  2016-04-03 19:55:44  4783971474
4   4  2016-04-03 18:05:54  4787908274

                                          categories  pay_option_on_delivery  \
0  ['Komputery', 'Dyski i napędy', 'Nośniki', 'No…                       1
1  ['Odzież, Obuwie, Dodatki', 'Bielizna damska',…                       1
2  ['Dom i Ogród', 'Budownictwo i Akcesoria', 'Śc…                       1
3  ['Książki i Komiksy', 'Poradniki i albumy', 'Z…                       1
```

```
4  ['Odzież, Obuwie, Dodatki', 'Ślub i wesele', '…                      1

   pay_option_transfer         seller   price  it_is_allegro_standard  \
0                    1    radzioch666   59.99                       1
1                    1  InwestycjeNET    4.90                       1
2                    1    otostyl_com  109.90                       1
3                    1        Matfel1   18.50                       0
4                    1      PPHU_RICO   19.90                       1

   it_quantity  it_is_brand_zone  it_seller_rating          it_location  \
0          997                 0             50177              Warszawa
1         9288                 0             12428              Warszawa
2          895                 0              7389                Leszno
3          971                 0             15006  Wola Krzysztoporska
4          950                 0             32975             BIAŁYSTOK

             main_category
0                Komputery
1  Odzież, Obuwie, Dodatki
2              Dom i Ogród
3        Książki i Komiksy
4  Odzież, Obuwie, Dodatki
```

[3]:
```
input_df.shape
```

[3]:
```
(420020, 14)
```

[4]:
```
input_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 420020 entries, 0 to 420019
Data columns (total 14 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   lp                      420020 non-null  int64
 1   date                    420020 non-null  object
 2   item_id                 420020 non-null  int64
 3   categories              420020 non-null  object
 4   pay_option_on_delivery  420020 non-null  int64
 5   pay_option_transfer     420020 non-null  int64
 6   seller                  420020 non-null  object
 7   price                   420020 non-null  float64
 8   it_is_allegro_standard  420020 non-null  int64
 9   it_quantity             420020 non-null  int64
 10  it_is_brand_zone        420020 non-null  int64
 11  it_seller_rating        420020 non-null  int64
 12  it_location             420020 non-null  object
```

```
 13  main_category           420020 non-null  object
dtypes: float64(1), int64(8), object(5)
memory usage: 44.9+ MB
```

[5]:
```
input_df['date'] = pd.to_datetime(input_df.date, format="%Y-%m-%d %H:%M:%S")
```

[6]:
```
df = input_df.drop(['lp', 'item_id', 'categories', 'date'], axis='columns')
```

Usuwamy niepotrzebne kolumny. Usuwamy kategorie i datę dla uproszczenia regresji.

1.2 Podział na zbiór treningowy i testowy

[7]:
```
columns = df.columns.drop('price')

X_train, X_test, y_train, y_test = train_test_split(
    df[columns],
    df['price'],
    test_size=0.33, random_state=42)
```

2 1. Kodowanie zmiennych kategorycznych

2.1 Target encoding `it_location`

[8]:
```
target_encoder = ce.TargetEncoder(cols=['it_location'])
target_encoder.fit_transform(X_train,y_train)
```

[8]:

	pay_option_on_delivery	pay_option_transfer	seller	\
347640	1	1	RemoteWorld	
139240	1	1	sendobry	
218088	1	0	MDSportpl	
245097	1	1	www_marketbio_pl	
400785	1	1	asiagaw	
…	…	…	…	
259178	1	1	nowyelektronik2	
365838	1	1	eoryginalne_pl	
131932	1	1	TomAutCz	
146867	0	1	violetta_te	
121958	1	0	MadeIn_USA	

	it_is_allegro_standard	it_quantity	it_is_brand_zone	\
347640	0	991	0	
139240	1	931	0	
218088	1	0	0	
245097	1	27	0	
400785	0	82	0	
…	…	…	…	
259178	1	5	0	

```
365838                      1           5                 0
131932                      0           0                 0
146867                      0           0                 0
121958                      1         966                 0

        it_seller_rating  it_location                       main_category
347640              3342   136.872051                             RTV i AGD
139240             17671   119.166000                          Dla Dzieci
218088               649    77.615936                 Sport i Turystyka
245097             13146    85.522162                         Delikatesy
400785              5487    51.506531                       Dom i Ogród
…                   …         …                                   …
259178              7939    64.980640  Sprzęt estradowy, studyjny i DJ-ski
365838              6806    72.339750          Odzież, Obuwie, Dodatki
131932                71   117.521734                      Motoryzacja
146867                69    76.498167                          Zdrowie
121958             29471    69.396733                            Uroda

[281413 rows x 9 columns]
```

2.2 Kodowanie `main_category`

2.2.1 One-Hot

[9]:
```
one_hot_encoder = ce.OneHotEncoder(cols=['main_category'])
one_hot = one_hot_encoder.fit_transform(X_train,y_train)
```

One Hot Encoder dla każdej kategorii tworzy nową kolumnę i wstawia do niej 1 gdy obserwacja należy do tej kategorii, 0 w przeciwnym przypadku.

2.2.2 Hashing Difference Coding

[10]:
```
hashing_encoder = ce.HashingEncoder(cols=['main_category'])
hash_ = hashing_encoder.fit_transform(X_train,y_train)
```

Hashing encoding działa podobnie jak One Hot, ale przypisuje kategorie do kolumn na podstawie funkcji hashującej, której parametry można ustawić (w szczególności zbiór wartości), więc liczba wynikowych kolumn jest pod kontrolą programisty. Tym kodowaniem możemy kontrolować wymiary zakodowanego zbioru. Gdy ustawimy `n_components=len(df.columns)` otrzymamy kodowanie one-hot.

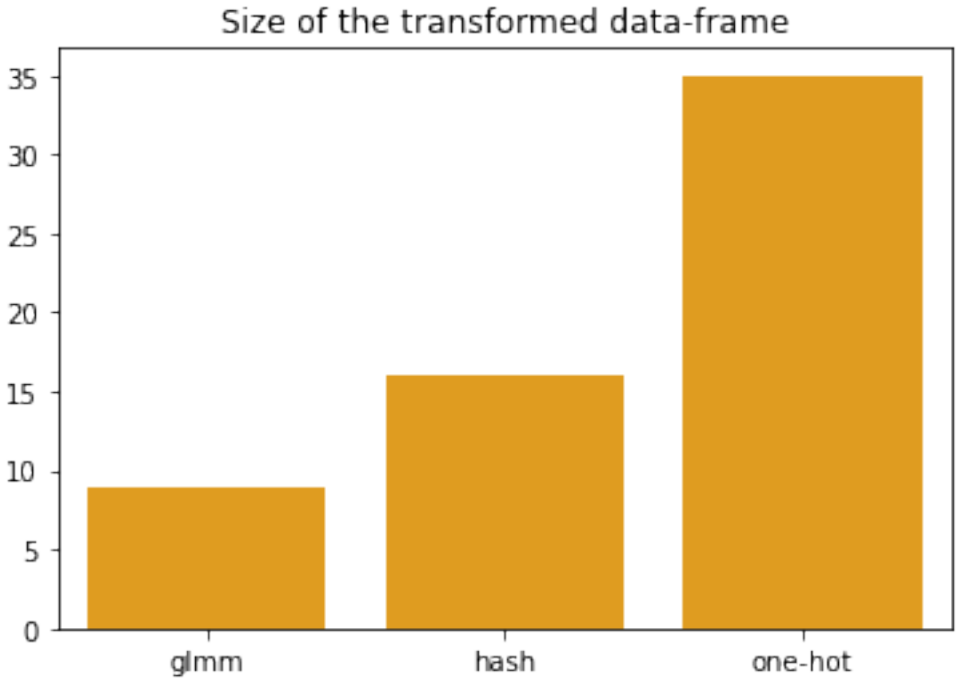
2.2.3 Generalized Linear Mixed Model Encoder

[11]:
```
glmm_encoder = ce.GLMMEncoder(cols=['main_category'])
glmm = glmm_encoder.fit_transform(X_train,y_train)
```

Generalized Linear Mixed Model Encoder to samo tuningująca się odmaina Target Encodera. Dla każdej kategorii wylicza regularyzowaną różnicę średniej kategorii od średniej całego zbioru.

[12]:
```
labels = ['glmm', 'hash', 'one-hot']
size = [len(x.columns) for x in [glmm, hash_, one_hot]]

sns.barplot(x=labels, y=size, color='orange').set_title("Size of the␣
 ↪transformed data-frame")
plt.show()
```



2.3 Budowa modelu regresji liniowej

dla sprawdzenia jak różne kodowanie wpływa na skuteczność

2.3.1 Musimy jeszcze zakodować `seller`

Użyjemy target encoder dlatego zmodyfikujemy już istniejący

[13]:
```
target_encoder = ce.TargetEncoder(cols=['it_location', 'seller'])
target_encoder.fit_transform(X_train,y_train)
```

[13]:

	pay_option_on_delivery	pay_option_transfer	seller
347640	1	1	76.498167
139240	1	1	43.174211
218088	1	0	175.589453
245097	1	1	14.180245
400785	1	1	12.906780
…	…	…	…

```
259178                        1                    1   10.799817
365838                        1                    1  181.042308
131932                        1                    1  155.088304
146867                        0                    1   76.498167
121958                        1                    0   14.005152

        it_is_allegro_standard  it_quantity  it_is_brand_zone  \
347640                       0          991                 0
139240                       1          931                 0
218088                       1            0                 0
245097                       1           27                 0
400785                       0           82                 0
…                          …           …                 …
259178                       1            5                 0
365838                       1            5                 0
131932                       0            0                 0
146867                       0            0                 0
121958                       1          966                 0

        it_seller_rating  it_location                        main_category
347640              3342   136.872051                            RTV i AGD
139240             17671   119.166000                          Dla Dzieci
218088               649    77.615936                 Sport i Turystyka
245097             13146    85.522162                        Delikatesy
400785              5487    51.506531                       Dom i Ogród
…                   …          …                                  …
259178              7939    64.980640  Sprzęt estradowy, studyjny i DJ-ski
365838              6806    72.339750          Odzież, Obuwie, Dodatki
131932                71   117.521734                      Motoryzacja
146867                69    76.498167                          Zdrowie
121958             29471    69.396733                            Uroda

[281413 rows x 9 columns]
```

[14]:
```
pipe_one_hot = Pipeline(
[
    ('transformer_target', target_encoder),
    ('transformer_one_hot', one_hot_encoder),
    ('linear-model', LinearRegression())
])

pipe_backward = Pipeline(
[
    ('transformer_target', target_encoder),
    ('transformer_hashing', hashing_encoder),
    ('linear-model', LinearRegression())
])
```

```
pipe_glmm = Pipeline(
[
    ('transformer_target', target_encoder),
    ('transformer_glmm', glmm_encoder),
    ('linear-model', LinearRegression())
])

pipes = [pipe_backward, pipe_glmm, pipe_one_hot]
```

[15]:
```
for pipe in pipes:
    pipe.fit(X_train, y_train)
    y_pred = pipe.predict(X_test)
    print(f' {pipe.steps[1][0]} RMSE : {np.sqrt(mean_squared_error(y_test,␣
 ↪y_pred)):.3f}')
```

```
 transformer_hashing RMSE : 496.848
 transformer_glmm RMSE : 496.771
 transformer_one_hot RMSE : 496.726
```

W tym przypadku najlepsze okazało się kodowanie one-hot, najgorsze natomiast kodowanie haszujące.

3 2. Uzupełnianie braków

[16]:
```
from sklearn.impute import KNNImputer
```

[17]:
```
input_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 420020 entries, 0 to 420019
Data columns (total 14 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   lp                      420020 non-null  int64
 1   date                    420020 non-null  datetime64[ns]
 2   item_id                 420020 non-null  int64
 3   categories              420020 non-null  object
 4   pay_option_on_delivery  420020 non-null  int64
 5   pay_option_transfer     420020 non-null  int64
 6   seller                  420020 non-null  object
 7   price                   420020 non-null  float64
 8   it_is_allegro_standard  420020 non-null  int64
 9   it_quantity             420020 non-null  int64
 10  it_is_brand_zone        420020 non-null  int64
 11  it_seller_rating        420020 non-null  int64
 12  it_location             420020 non-null  object
```

```
 13  main_category           420020 non-null  object
dtypes: datetime64[ns](1), float64(1), int64(8), object(4)
memory usage: 44.9+ MB
```

Aby skrócić czas obliczeń ograniczylem zbiór danych do 1000 obserwacji.

Poniżej dla `i`=1,...10 wykonano 10 razy imputację z parametrem `n_neighbors`=`i` z jedną brakującą kolumną (`it_seller_rating`).

[18]:
```
errors = {}

def remove_and_impute(n_neighbors):
    df2 = input_df[['price', 'it_seller_rating', 'it_quantity']].sample(1000,␣
 ↪random_state=997).copy(deep=True).reset_index()

    removed = df2['it_seller_rating'].sample(frac=.1)
    df2.loc[removed.index,'it_seller_rating'] = np.nan

    imputer = KNNImputer(n_neighbors=n_neighbors, weights="uniform")
    imputed = imputer.fit_transform(df2)

    if errors.get(n_neighbors-1) is None:
        errors.update({n_neighbors-1 :[]})
    errors.get(n_neighbors-1).append(np.sqrt(mean_squared_error(imputed[removed.
 ↪index,2], removed)))


for i in range(10):
    for j in range(10):
        remove_and_impute(i+1)

errors_df = pd.DataFrame(errors) # cols = n_neighbors-1
std1 = errors_df.describe().loc['std']
```

Następnie usunięto 2 kolumny (`it_seller_rating`, `it_location`) i powtórzono wcześniejszą procedurę

[19]:
```
errors_2 = {}

def remove_and_impute(n_neighbors):
    df2 = input_df[['price', 'it_seller_rating', 'it_quantity']].sample(1000,␣
 ↪random_state=997).copy(deep=True).reset_index()

    removed = df2.sample(frac=.1)
    df2.loc[removed.index,['it_seller_rating', 'it_location']] = np.nan

    imputer = KNNImputer(n_neighbors=n_neighbors, weights="uniform")
    imputed = imputer.fit_transform(df2)
```

```
    if errors_2.get(n_neighbors-1) is None:
        errors_2.update({n_neighbors-1 :[]})
    errors_2.get(n_neighbors-1).append(np.
 ↪sqrt(mean_squared_error(imputed[removed.index,2],␣
 ↪removed['it_seller_rating'])))


for i in range(10):
    for j in range(10):
        remove_and_impute(i+1)


errors_2df = pd.DataFrame(errors_2)
std2 = errors_2df.describe().loc['std']
```

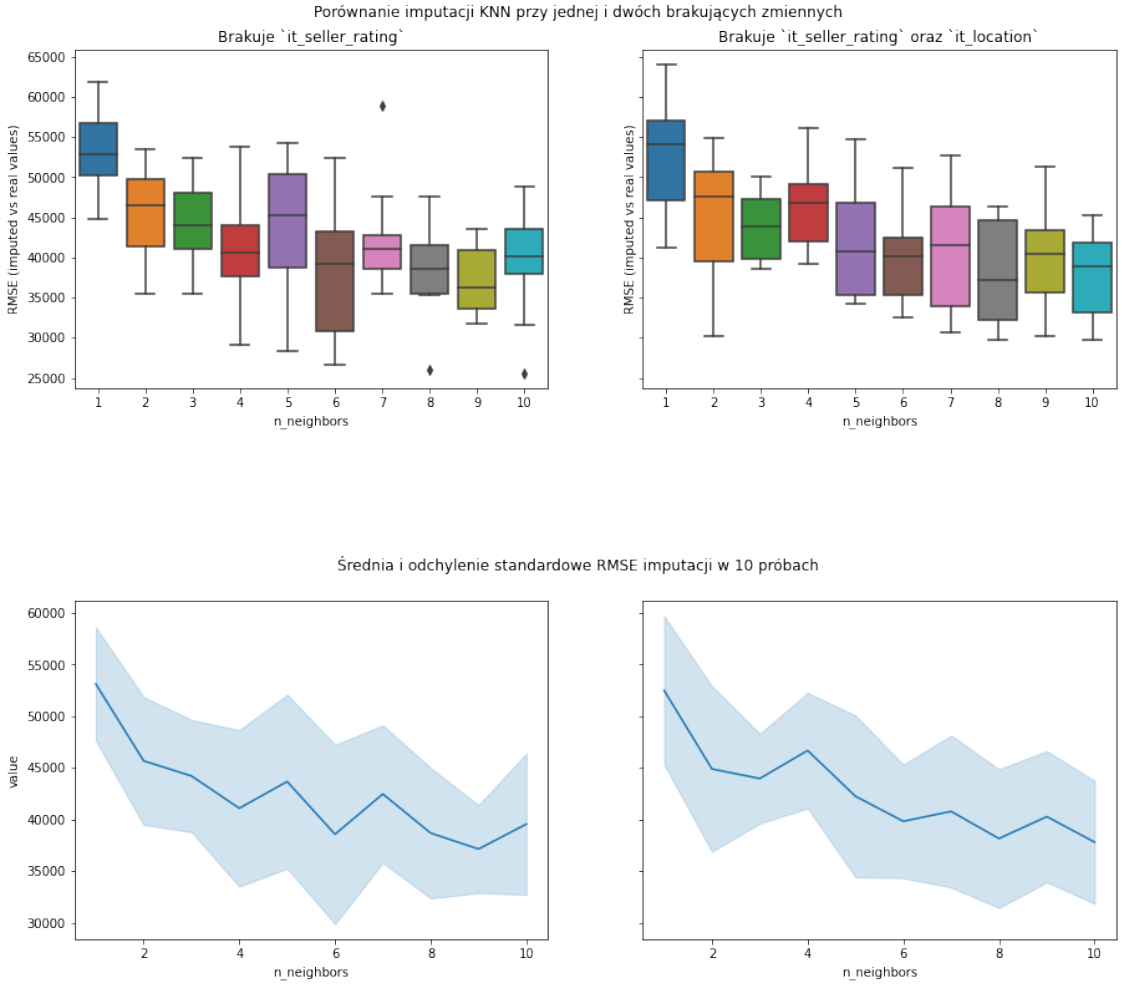
[20]:
```
fig, axes = plt.subplots(1, 2, sharex=True, sharey=True, figsize=(15,5))
fig.suptitle('Porównanie imputacji KNN przy jednej i dwóch brakujących␣
 ↪zmiennych')
axes[0].set_title('Brakuje `it_seller_rating`')
axes[1].set_title('Brakuje `it_seller_rating` oraz `it_location`')

ax = sns.boxplot(ax = axes[0], data = errors_df)
ax.set_xticklabels(labels = [i+1 for i in range(len(errors_df.columns))])
ax.set_xlabel('n_neighbors')
ax.set_ylabel('RMSE (imputed vs real values)')


ax3 = sns.boxplot(ax = axes[1], data = errors_2df)
ax3.set_xticklabels(labels = [i+1 for i in range(len(errors_2df.columns))])
ax3.set_xlabel('n_neighbors')
ax3.set_ylabel('RMSE (imputed vs real values)')


fig, axes = plt.subplots(1, 2, sharex=True, sharey=True, figsize=(15,5))
fig.suptitle('Średnia i odchylenie standardowe RMSE imputacji w 10 próbach')
ax2 = sns.lineplot(ax=axes[0], data = pd.melt(errors_df),x = pd.
 ↪melt(errors_df)['variable']+1, y = 'value', ci='sd')
ax4 = sns.lineplot(ax=axes[1], data = pd.melt(errors_2df),x = pd.
 ↪melt(errors_df)['variable']+1, y = 'value', ci='sd')
ax2.set_xlabel('n_neighbors')
ax4.set_xlabel('n_neighbors')
```

[20]:
```
Text(0.5, 0, 'n_neighbors')
```



Wyniki imputacji są podobne, niezależnie od tego czy opieramy się na jednej czy na dwóch kolumnach