Trabalho - MAE 0501

December 3, 2024

#Trabalho Individual - MAE 0501

Autor: Vítor Garcia Comissoli

NUSP: 11810411

Data: 02/12/2024

0.1 Links relevantes:

- Link para o GitHub: https://github.com/Vitor-Garcia-Comissoli/Codes_from_MAE0501
- Link para a pasta do Google Drive: https://drive.google.com/drive/folders/1Jow0NBWpHGlcvePy0uN-I8Oge6ie7mxE?usp=drive_link

0.2 Input dos arquivos necessários:

Input pelo Google Drive: Inicialmente, deve ser criado um atalho da pasta do Google Drive (fornecida através do link anteriormente) para o Drive do usuário. Em seguida, através do código a seguir, o Google Colab é conectado ao Drive, é montada uma pasta e alterado o diretório da pasta.

Input da pasta baixada localmente: Alternativamente, caso tenha baixado o .ipynb, estabelecer uma path para a pasta "Trabalho" (encontrada no repositório do GitHub).

[667]:
```
# Montando a pasta:
from google.colab import drive
drive.mount('/content/gdrive')

path = '/content/gdrive/My Drive/Décimo Semestre/MAE 0501/Trabalho'

# Descomentar essa linha, e comentar a anterior, após copiar a pasta para o␣
 ↪drive pessoal:
# path = '/content/gdrive/My Drive/Trabalho'

# Descomentar essa linha, e comentar as anteriores, para usar o arquivo baixado␣
 ↪nativamente:
# path = 'F:/Downloads/Trabalho'

# Alterando o diretório:
import os
os.chdir(path)
```

```
# Listando os arquivos do diretório:
!ls
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call
drive.mount("/content/gdrive", force_remount=True).
 Dicionário.xlsx           sao-paulo-properties-new.csv.csv  'Trabalho -
orientações.pdf'
 sao-paulo-properties.csv   submission.csv
```

[668]:
```
# Importação das bibliotecas

#!apt-get install texlive texlive-xetex texlive-latex-extra pandoc
#!pip install pypandoc
import os
import pandas as pd
import statsmodels.api as sm
import sklearn as sk
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score
from sklearn.metrics import mean_squared_error
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cluster import KMeans
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestRegressor
import numpy as np
import matplotlib.pyplot as plt
```

[669]:
```
treino = pd.read_csv('sao-paulo-properties.csv', sep=';', index_col="X")
%load_ext google.colab.data_table
%reload_ext google.colab.data_table
drive = True
treino
```

```
Output hidden; open in https://colab.research.google.com to view.
```

[670]:
```
teste = pd.read_csv('sao-paulo-properties-new.csv.csv', sep=';', index_col="X")
%load_ext google.colab.data_table
%reload_ext google.colab.data_table
drive = True
teste
```

```
The google.colab.data_table extension is already loaded. To reload it, use:
```

```
%reload_ext google.colab.data_table
```

[670]:

X	Condo	Size	Rooms	Toilets	Suites	Parking	Elevator	Furnished
1	180	52	1	2	1	1	1	0
2	0	65	2	2	1	1	0	0
3	337	55	2	2	1	1	1	0
4	720	68	3	2	1	1	0	0
5	857	113	3	2	1	2	1	0
…	…	…	…	…	…	…	…	…
1731	900	111	3	2	1	2	1	0
1732	980	52	2	1	0	1	0	1
1733	2100	172	3	4	3	3	1	0
1734	850	48	1	2	1	1	0	1
1735	2153	258	3	4	3	4	0	1

X	Swimming.Pool	New	District	Property.Type	Latitude
1	0	0	Artur Alvim/São Paulo	apartment	-2.354.984
2	0	0	Artur Alvim/São Paulo	apartment	-23.548.751
3	0	0	Belém/São Paulo	apartment	-23.546.104
4	1	0	Belém/São Paulo	apartment	-23.538.163
5	0	0	Belém/São Paulo	apartment	-23.538.134
…	…	…	…	…	…
1731	1	0	Brooklin/São Paulo	apartment	-23.602.792
1732	1	0	Brooklin/São Paulo	apartment	-236.150.791
1733	1	0	Brooklin/São Paulo	apartment	-236.123.367
1734	1	0	Brooklin/São Paulo	apartment	-236.186.739
1735	0	0	Brooklin/São Paulo	apartment	-236.194.808

X	Longitude
1	-46.484.137
2	-46.477.195
3	-46.594.579
4	-46.591.505
5	-46.594.059
…	…
1731	-466.947.674
1732	-466.802.535
1733	-466.856.745
1734	-466.853.961
1735	-466.840.282

[1735 rows x 14 columns]

Lidando com os dados faltantes:

Primeiramente, todos os valores iguais a 0 onde 0 não representa um valor válido para a variável, mas sim um dado faltante, serão subistituídos por NA.

Além disso, serão geradas duas novas bases de dados, onde na primeira serão retiradas todas as linhas que contém NA, e na segundo, os valores de NA serão substituidos pela média (ou moda, no caso de variáveis categóricas).

[671]:
```
treino["Price"] = treino["Price"].replace(0, np.nan)
treino["Condo"] = treino["Condo"].replace(0, np.nan)
treino["Size"] = treino["Size"].replace(0, np.nan)
treino['District'] = treino['District'].replace('0', np.nan)
treino["Property.Type"] = treino["Property.Type"].replace('0', np.nan)
treino["Latitude"] = treino["Latitude"].replace('0', np.nan)
treino["Longitude"] = treino["Longitude"].replace('0', np.nan)

teste["Condo"] = teste["Condo"].replace(0, np.nan)
teste["Size"] = teste["Size"].replace(0, np.nan)
teste['District'] = teste['District'].replace('0', np.nan)
teste["Property.Type"] = teste["Property.Type"].replace('0', np.nan)
teste["Latitude"] = teste["Latitude"].replace('0', np.nan)
teste["Longitude"] = teste["Longitude"].replace('0', np.nan)
```

[672]:
```
# Defininfo função que subistitui NAs diferentemente, dependendo do tipo da␣
 ↪variável de cada coluna.
def fill_na_with_mean_or_mode(df):
    for col in df.columns:
        if pd.api.types.is_numeric_dtype(df[col]):
            # Replace NaN in numeric columns with the mean
            df.fillna({col: df[col].mean()}, inplace=True)
        else:
            # Replace NaN in non-numeric columns with the mode (most frequent␣
 ↪value)
            df.fillna({col: df[col].mode()[0]}, inplace=True)
    return df
```

[673]:
```
treino_1 = treino.dropna()
treino_2 = fill_na_with_mean_or_mode(treino)
```

[674]:
```
treino.shape
```

[674]:
```
(4047, 15)
```

[675]:
```
treino_1.shape
```

[675]:
```
(3420, 15)
```

[676]:
```
treino_2.shape
```

[676]: (4047, 15)

[677]:
```
treino_1
```

Output hidden; open in https://colab.research.google.com to view.

[678]:
```
treino_2
```

Output hidden; open in https://colab.research.google.com to view.

Analogamente ao caso anterior, temos:

[679]:
```
teste_1 = teste.dropna()
teste_2 = fill_na_with_mean_or_mode(teste)
```

[680]:
```
teste.shape
```

[680]: (1735, 14)

[681]:
```
teste_1.shape
```

[681]: (1480, 14)

[682]:
```
teste_2.shape
```

[682]: (1735, 14)

[683]:
```
teste_1
```

[683]:

X	Condo	Size	Rooms	Toilets	Suites	Parking	Elevator	Furnished	Swimming.Pool	New	District	Property.Type	Latitude
1	180.0	52	1	2	1	1	1	0	0	0	Artur Alvim/São Paulo	apartment	-2.354.984
3	337.0	55	2	2	1	1	1	0	0	0	Belém/São Paulo	apartment	-23.546.104
4	720.0	68	3	2	1	1	0	0	1	0	Belém/São Paulo	apartment	-23.538.163
5	857.0	113	3	2	1	2	1	0					
6	450.0	50	1	2	1	1	1	0					
…	…	…	…	…	…	…	…	…					
1731	900.0	111	3	2	1	2	1	0					
1732	980.0	52	2	1	0	1	0	1					
1733	2100.0	172	3	4	3	3	1	0					
1734	850.0	48	1	2	1	1	0	1					
1735	2153.0	258	3	4	3	4	0	1					

```
5                 0    0        Belém/São Paulo     apartment   -23.538.134
6                 1    0        Belém/São Paulo     apartment   -23.546.089
…               …  …                      …           …            …
1731              1    0     Brooklin/São Paulo     apartment   -23.602.792
1732              1    0     Brooklin/São Paulo     apartment  -236.150.791
1733              1    0     Brooklin/São Paulo     apartment  -236.123.367
1734              1    0     Brooklin/São Paulo     apartment  -236.186.739
1735              0    0     Brooklin/São Paulo     apartment  -236.194.808

          Longitude
X
1       -46.484.137
3       -46.594.579
4       -46.591.505
5       -46.594.059
6       -46.590.325
…              …
1731  -466.947.674
1732  -466.802.535
1733  -466.856.745
1734  -466.853.961
1735  -466.840.282

[1480 rows x 14 columns]
```

[684]:
```
teste_2
```

[684]:
```
             Condo  Size  Rooms  Toilets  Suites  Parking  Elevator  Furnished  \
X
1       180.000000    52      1        2       1        1         1          0
2       894.999367    65      2        2       1        1         0          0
3       337.000000    55      2        2       1        1         1          0
4       720.000000    68      3        2       1        1         0          0
5       857.000000   113      3        2       1        2         1          0
…              …   …    …      …      …       …      …        …
1731    900.000000   111      3        2       1        2         1          0
1732    980.000000    52      2        1       0        1         0          1
1733   2100.000000   172      3        4       3        3         1          0
1734    850.000000    48      1        2       1        1         0          1
1735   2153.000000   258      3        4       3        4         0          1

       Swimming.Pool  New              District Property.Type      Latitude  \
X
1                  0    0  Artur Alvim/São Paulo     apartment    -2.354.984
2                  0    0  Artur Alvim/São Paulo     apartment   -23.548.751
3                  0    0        Belém/São Paulo     apartment   -23.546.104
4                  1    0        Belém/São Paulo     apartment   -23.538.163
```

```
5                 0    0        Belém/São Paulo     apartment   -23.538.134
…               …  …                      …           …           …
1731              1    0     Brooklin/São Paulo     apartment   -23.602.792
1732              1    0     Brooklin/São Paulo     apartment  -236.150.791
1733              1    0     Brooklin/São Paulo     apartment  -236.123.367
1734              1    0     Brooklin/São Paulo     apartment  -236.186.739
1735              0    0     Brooklin/São Paulo     apartment  -236.194.808

          Longitude
X
1       -46.484.137
2       -46.477.195
3       -46.594.579
4       -46.591.505
5       -46.594.059
…              …
1731   -466.947.674
1732   -466.802.535
1733   -466.856.745
1734   -466.853.961
1735   -466.840.282

[1735 rows x 14 columns]
```

Como existem valores para as variáveis “District”, “Latitude” e “Longitude” que estão presentes na base de testes, mas não estão presentes na base de treino, e vice versa, optou-se por retirá-las da modelagem dos modelos, para viabilizar o cálculo das predições de cada modelo.

[685]:
```
X1 = treino_1.drop(columns=['Price', 'Latitude', 'Longitude', 'District'])
y1 = treino_1['Price']

X2 = treino_2.drop(columns=['Price', 'Latitude', 'Longitude', 'District'])
y2 = treino_2['Price']
```

[686]:
```
X1_prev = teste_1.drop(columns=['Latitude', 'Longitude', 'District'])
X2_prev = teste_2.drop(columns=['Latitude', 'Longitude', 'District'])
```

[687]:
```
X1_transformed = pd.get_dummies(X1, columns=["Property.Type"], drop_first=False)
X2_transformed = pd.get_dummies(X2, columns=['Property.Type'], drop_first=False)
X1_prev_transformed = pd.get_dummies(X1_prev, columns=['Property.Type'],␣
 ↪drop_first=False)
X2_prev_transformed = pd.get_dummies(X2_prev, columns=['Property.Type'],␣
 ↪drop_first=False)
```

[688]:
```
X1_train, X1_test, y1_train, y1_test = train_test_split(X1_transformed, y1,␣
 ↪test_size=0.3, random_state=42)
```

```
X2_train, X2_test, y2_train, y2_test = train_test_split(X2_transformed, y2,␣
↪test_size=0.3, random_state=42)
```

0.3 Modelo 1: Regressão linear

[689]:
```
model1 = LinearRegression()
model2 = LinearRegression()
```

[690]:
```
model1.fit(X1_train, y1_train)
model2.fit(X2_train, y2_train)
```

[690]:
```
LinearRegression()
```

Seguem abaixo os R^2s de ambos os modelos

[691]:
```
model1.score(X1_train, y1_train)
```

[691]:
```
0.6212161852159397
```

[692]:
```
model2.score(X2_train, y2_train)
```

[692]:
```
0.6189172466423825
```

Disso, observou-se um desempenho melhor retirando valores nulos que subistituindo-os pela média. Realizou-se então a predição utilizando o primeiro modelo.

[693]:
```
y_pred1 = model1.predict(X1_prev_transformed)
```

[694]:
```
y_pred1 = pd.Series(y_pred1, index=X1_prev_transformed.index)
```

[695]:
```
y_pred1 = pd.DataFrame(y_pred1)
y_pred1
```

[695]:

X	0
1	1667.220379
3	1174.867934
4	1620.768762
5	2832.023586
6	2156.316928
…	…
1731	3013.934542
1732	2653.761255
1733	7597.737632
1734	3751.680702
1735	10891.935157

```
[1480 rows x 1 columns]
```

[696]:
```
y_pred1_test = model1.predict(X1_test)
```

0.4 Modelo 2: Regressão Lasso

[697]:
```
model_lasso1 = Lasso()
model_lasso2 = Lasso()
```

[698]:
```
model_lasso1.fit(X1_train, y1_train)
model_lasso2.fit(X2_train, y2_train)
```

[698]:
```
Lasso()
```

Seguem abaixo os R^2s de ambos os modelos

[699]:
```
model_lasso1.score(X1_train, y1_train)
```

[699]:
```
0.621214583319974
```

[700]:
```
model_lasso2.score(X2_train, y2_train)
```

[700]:
```
0.6189129462654426
```

Disso, observou-se um desempenho melhor retirando valores nulos que subistituindo-os pela média. Realizou-se então a predição utilizando o primeiro modelo.

[701]:
```
y_pred_lasso1 = model_lasso1.predict(X1_prev_transformed)
```

[702]:
```
y_pred_lasso1 = pd.Series(y_pred_lasso1, index=X1_prev_transformed.index)
```

[703]:
```
y_pred_lasso1 = pd.DataFrame(y_pred_lasso1)
y_pred_lasso1
```

[703]:

X	0
1	1669.144838
3	1180.630108
4	1621.723593
5	2839.411034
6	2155.902825
…	…
1731	3018.248763
1732	2648.625041
1733	7598.716749
1734	3740.952585
1735	10880.727304

```
[1480 rows x 1 columns]
```

[704]:
```
y_pred_lasso1_test = model_lasso1.predict(X1_test)
```

0.5 Modelo 3: Random Forest

[705]:
```
rf1 = RandomForestRegressor()
rf2 = RandomForestRegressor()
```

[706]:
```
rf1.fit(X1_train, y1_train)
rf2.fit(X2_train, y2_train)
```

[706]:
```
RandomForestRegressor()
```

Seguem abaixo os R^2s de ambos os modelos

[707]:
```
rf1.score(X1_train, y1_train)
```

[707]:
```
0.9443832783317399
```

[708]:
```
rf2.score(X2_train, y2_train)
```

[708]:
```
0.9366127332791013
```

Disso, observou-se um desempenho melhor retirando valores nulos que subistituindo-os pela média (mesmo que a diferença tenha sido pequena). Realizou-se então a predição utilizando o primeiro modelo.

[709]:
```
y_pred_rf1 = rf1.predict(X1_prev_transformed)
```

[710]:
```
y_pred_rf1 = pd.Series(y_pred_rf1, index=X1_prev_transformed.index)
```

[711]:
```
y_pred_rf1 = pd.DataFrame(y_pred_rf1)
y_pred_rf1
```

[711]:

X	0
1	1013.666667
3	1128.271429
4	2148.900000
5	2680.020000
6	2551.550000
…	…
1731	1952.200000
1732	2901.490000
1733	6870.300000
1734	2945.050000

```
1735  11611.000000

[1480 rows x 1 columns]
```

[712]:
```
y_pred_rf1_test = rf1.predict(X1_test)
```

0.6 Modelo 4: KNN

[713]:
```
knn1 = KNeighborsClassifier(n_neighbors=1)
knn2 = KNeighborsClassifier(n_neighbors=1)
```

[714]:
```
scores_1 = pd.DataFrame(cross_val_score(knn1, X1_train, y1_train, cv=100))
scores_2 = pd.DataFrame(cross_val_score(knn2, X2_train, y2_train, cv=100))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py:776:
UserWarning: The least populated class in y has only 1 members, which is less
than n_splits=100.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py:776:
UserWarning: The least populated class in y has only 1 members, which is less
than n_splits=100.
  warnings.warn(
```

[715]:
```
scores_1
```

[715]:
```
           0
0   0.208333
1   0.125000
2   0.000000
3   0.041667
4   0.125000
..       …
95  0.086957
96  0.086957
97  0.130435
98  0.086957
99  0.086957

[100 rows x 1 columns]
```

[716]:
```
scores_2
```

[716]:
```
           0
0   0.068966
1   0.206897
2   0.103448
3   0.068966
```

```
4   0.137931
..       …
95  0.142857
96  0.000000
97  0.035714
98  0.107143
99  0.035714

[100 rows x 1 columns]
```

[717]:
```
knn1.fit(X1_train, y1_train)
knn2.fit(X2_train, y2_train)
```

[717]:
```
KNeighborsClassifier(n_neighbors=1)
```

As acurácias obtidas para os modelos foram as seguintes:

[718]:
```
knn1.score(X1_train, y1_train)
```

[718]:
```
0.9782790309106099
```

[719]:
```
knn2.score(X2_train, y2_train)
```

[719]:
```
0.9544491525423728
```

Disso, observou-se um desempenho melhor retirando valores nulos que subistituindo-os pela média. Realizou-se então a predição utilizando o primeiro modelo.

[720]:
```
y_pred_knn1 = knn1.predict(X1_prev_transformed)
```

[721]:
```
y_pred_knn1 = pd.Series(y_pred_knn1, index=X1_prev_transformed.index)
```

[722]:
```
y_pred_knn1 = pd.DataFrame(y_pred_knn1)
y_pred_knn1
```

[722]:
```
            0
X
1         700
3        1100
4        2500
5        3501
6        1950
…         …
1731     1300
1732     2700
1733     8000
1734     1750
1735    13000
```

```
[1480 rows x 1 columns]
```

[723]:
```
y_pred_knn1_test = knn1.predict(X1_test)
```

0.7 Escolha do Modelo Final

Para comparar esses 4 modelos entre sí, se usará o Erro Quadrático Médio (EQM) de cada um, como calculado abaixo:

[724]:
```
mse_linear = mean_squared_error(y1_test, y_pred1_test)
mse_lasso = mean_squared_error(y1_test, y_pred_lasso1_test)
mse_rf = mean_squared_error(y1_test, y_pred_rf1_test)
mse_knn = mean_squared_error(y1_test, y_pred_knn1_test)

print("EQM para a Regressão Linear:", mse_linear)
print("EQM para a Regressão LASSO:", mse_lasso)
print("EQM para Random Forest:", mse_rf)
print("EQM para KNN:", mse_knn)
```

```
EQM para a Regressão Linear: 4404706.527614403
EQM para a Regressão LASSO: 4403100.097854282
EQM para Random Forest: 4412931.166297071
EQM para KNN: 9959859.411306042
```

Dados os 4 modelos gerados anteriormente, optou-se pela escolha do modelo 3 (Random Forest) como o modelo final, uma vez que o EQM do mesmo se mostrou próximo ao EQM tanto da regressão Linear quanto da regressão Lasso, além de muito menor que o EQM de KNN.

Como os EQMs dos 3 primeiros modelos estão muito próximos um do outro, e o valor de R^2 oriundo do modelo de Random Forest se mostrou muito superior aos valores obtidos pelas regressões, decidiu-se usar esse modelo como o modelo final.

Vale ressaltar que o modelo de KNN também apresentou um resultado muito bom (observando somente a acurácia, e ignorando o EQM), enquanto os dois modelos de regressão (Linear e Lasso) se mostraram bem menos eficientes (Quanto ao R^2) para essa base de dados.

0.8 Exportando a previsão do Modelo Final para .CSV

[725]:
```
# Escolha qual modelo exportar para o .CSV (por default está o modelo final␣
↪escolhido, por Random Forest)

#predictions = y_pred1.copy()
#predictions = y_pred_lasso1.copy()
predictions = y_pred_rf1.copy()
#predictions = y_pred_knn1.copy()
predictions = predictions.rename(columns={0: 'pred'})
```

[726]:
```
predictions
```

[726]:
```
              pred
X
1      1013.666667
3      1128.271429
4      2148.900000
5      2680.020000
6      2551.550000
…              …
1731   1952.200000
1732   2901.490000
1733   6870.300000
1734   2945.050000
1735  11611.000000

[1480 rows x 1 columns]
```

[727]:
```
predictions = predictions.reset_index()
```

[728]:
```
predictions.to_csv('/content/gdrive/My Drive/Décimo Semestre/MAE 0501/Trabalho/
  ↪submission.csv', index = False)
#predictions.to_csv('/content/gdrive/My Drive/Trabalho', index = False)
#predictions.to_csv('F/Downloads/submission.csv', index = False)
```

[732]:
```
#!jupyter nbconvert --to PDF "Trabalho - MAE 0501.ipynb"
```

```
[NbConvertApp] Converting notebook Trabalho - MAE 0501.ipynb to PDF
[NbConvertApp] Writing 86662 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 90483 bytes to Trabalho - MAE 0501.pdf
```