Regressão linear múltipla

Introdução

Como vimos na nota de aula anterior, utilizar a regressão linear simples nem sempre é uma boa opção. Sabemos que é raro o caso em que uma variável dependente é associada a uma e apenas uma variável independente.

Vamos voltar para o preço de um imóvel. Além da área desse imóvel, quais outros fatores poderiam influenciar o preço? Podemos citar alguns exemplos, como: criminalidade da região, renda per capta dos moradores do bairro, proximidade a áreas comerciais, idade do edifício, entre outros.

Ao invés de estabelecermos *n* modelos para cada uma das *n* variáveis independentes, podemos unir todas essas variáveis em uma única expressão:

preço = a0 + a1 * x1 + a2 * x2 + ... + an * xn + erro

Atenção às variáveis independentes

Antigamente, uma análise por regressão linear múltipla era mais simples, já que os dados disponíveis eram muito mais escassos. Atualmente, muitas vezes temos tantas informações que temos que filtrar quais variáveis entram no nosso modelo, ao invés de filtrar as variáveis que deveriam ser removidas.

Precisamos pensar em remover variáveis do nosso modelo, mesmo com os dados disponíveis, por dois principais motivos:

- "**Entra lixo, sai lixo**". Isso significa que, se você utilizar no modelo variáveis que não tenham nada a ver com a sua variável dependente, o seu resultado também pode não fazer muito sentido. Para isso, é comum fazer um estudo preliminar da correlação das candidatas a variáveis independentes, antes de começar a rodar o modelo.
- Um número muito grande de variáveis implica em um modelo mais complexo. Muitas vezes a justificativa pelo uso de determinadas variáveis pode se perder, dificultando discussões de negócio. O ideal é tentar manter variáveis que realmente contribuam para o modelo.

Um ponto importante de se destacar na regressão múltipla é que é importante que as variáveis independentes sejam também independentes entre si. Por exemplo, no nosso caso do preço dos imóveis, não faz sentido termos como variáveis independentes a renda per capta e a faixa socio-econômica (você consegue obter uma informação da outra).

Aplicação

Para aplicar um modelo de regressão linear múltipla, vamos utilizar um site chamado Kaggle. Esse site possui inúmeros conjuntos de dados, desafios de mineração de dados e ciência de dados, e competições entre os usuários. Ele é uma fonte excelente de dados para estudo e aplicação dos conceitos que estamos aprendendo.

Dentro do Kaggle, vamos pegar a base de dados do Relatório de Felicidade Mundial. Vamos tentar prever a pontuação de felicidade baseado em algumas variáveis:

- PIB per capta
- Apoio social, que mede o apoio de amigos e familiares. A ideia é que apoio social melhora a qualidade de vida
- Saúde, medido pelo número médio de anos que um recém-nascido pode esperar viver em plena saúde
- Liberdade individual
- Generosidade
- Corrupção

Import e análise dos dados

```
%matplotlib inline
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn import datasets, metrics
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split, cross_val_score


df_19 = pd.read_csv("happiness2019.csv")
df_19.describe()
```

	Overall rank	Score	GDP per capita	Social support	Healthy life expectancy	Freedom to make life choices	Generos
count	156.000000	156.000000	156.000000	156.000000	156.000000	156.000000	156.000
mean	78.500000	5.407096	0.905147	1.208814	0.725244	0.392571	0.184
std	45.177428	1.113120	0.398389	0.299191	0.242124	0.143289	0.095
min	1.000000	2.853000	0.000000	0.000000	0.000000	0.000000	0.000
25%	39.750000	4.544500	0.602750	1.055750	0.547750	0.308000	0.108
50%	78.500000	5.379500	0.960000	1.271500	0.789000	0.417000	0.177
75%	117.250000	6.184500	1.232500	1.452500	0.881750	0.507250	0.248

```
df_19.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 156 entries, 0 to 155
Data columns (total 9 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   Overall rank                  156 non-null    int64  
 1   Country or region             156 non-null    object 
 2   Score                         156 non-null    float64
 3   GDP per capita                156 non-null    float64
 4   Social support                156 non-null    float64
 5   Healthy life expectancy       156 non-null    float64
 6   Freedom to make life choices  156 non-null    float64
 7   Generosity                    156 non-null    float64
 8   Perceptions of corruption     156 non-null    float64
dtypes: float64(7), int64(1), object(1)
memory usage: 11.1+ KB
```

Vamos apenas incluir o ano como coluna ao final dos nossos dados. Caso uníssemos tabelas dos outros anos, poderíamos usar o ano como uma variável independente.

```
df_19["Year"] = 2019
df_19.head()
```

	Overall rank	Country or region	Score	GDP per capita	Social support	Healthy life expectancy	Freedom to make life choices	Generosity
0	1	Finland	7.769	1.340	1.587	0.986	0.596	0.153
1	2	Denmark	7.600	1.383	1.573	0.996	0.592	0.252
2	3	Norway	7.554	1.488	1.582	1.028	0.603	0.271
3	4	Iceland	7.494	1.380	1.624	1.026	0.591	0.354
4	5	Netherlands	7.488	1.396	1.522	0.999	0.557	0.322

Agora vamos criar uma coluna com os nossos alvos. Vamos usar a função `pd.qcut` para dividir as nossas entradas de acordo com faixas de valores que vamos estabelecer.

```
target = ["Top", "Top-Mid", "Low-Mid", "Low"]
target_n = [4, 3, 2, 1]

df_19["target"] = pd.qcut(df_19["Overall rank"], len(target), labels=target)
df_19["target_n"] = pd.qcut(df_19["Overall rank"], len(target), labels=target_n)
```

Vamos terminar de limpar o nosso DataFrame em um df final:

```
finaldf = df_19
finaldf.dropna(inplace=True)
```

```
finaldf.isnull().any()
```

```
Overall rank                    False
Country or region               False
Score                           False
GDP per capita                  False
Social support                  False
Healthy life expectancy         False
Freedom to make life choices    False
Generosity                      False
Perceptions of corruption       False
Year                            False
target                          False
target_n                        False
dtype: bool
```

```
# Renomeando as colunas para ficar mais fácil
finaldf.columns = ["Rank", "Country", "Score", "GDP", "Support", "Health", "Freedom", "Generosity", "Corr
finaldf.head()
```

	Rank	Country	Score	GDP	Support	Health	Freedom	Generosity	Corruption
0	1	Finland	7.769	1.340	1.587	0.986	0.596	0.153	0.393
1	2	Denmark	7.600	1.383	1.573	0.996	0.592	0.252	0.410
2	3	Norway	7.554	1.488	1.582	1.028	0.603	0.271	0.341
3	4	Iceland	7.494	1.380	1.624	1.026	0.591	0.354	0.118
4	5	Netherlands	7.488	1.396	1.522	0.999	0.557	0.322	0.298

Não é o caso nesse problema, mas vamos supor que a coluna `Corruption` possua dados inválidos. Poderíamos substituir esses dados inválidos pela média dos demais dados, para não prejudicar o cálculo (ou poderíamos até mesmo remover esses dados).

```
finaldf.Corruption.fillna((finaldf.Corruption.mean()), inplace=True)
finaldf.head()
```

	Rank	Country	Score	GDP	Support	Health	Freedom	Generosity	Corruption
0	1	Finland	7.769	1.340	1.587	0.986	0.596	0.153	0.393
1	2	Denmark	7.600	1.383	1.573	0.996	0.592	0.252	0.410
2	3	Norway	7.554	1.488	1.582	1.028	0.603	0.271	0.341
3	4	Iceland	7.494	1.380	1.624	1.026	0.591	0.354	0.118
4	5	Netherlands	7.488	1.396	1.522	0.999	0.557	0.322	0.298

Para completar a preparação dos dados, vamos criar um dataframe vazio, para incluir as nossas métricas:

- Raiz quadrada da média dos erros (RMSE)

- R-quadrado
- R-quadrado ajustado
- Média dos R-quadrados obtido pelo Cross-Validation

Lembrando, ter um R-quadrado próximo de 1.0 e um valor baixo no RMSE significa que o modelo está melhor ajustado.

```
evaluation = pd.DataFrame(
    {
        "Model": [],
        "Details": [],
        "Root Mean Squared Error (RMSE)": [],
        "R-squared (training)": [],
        "Adjusted R-squared (training)": [],
        "R-squared (test)": [],
        "Adjusted R-squared (test)": [],
        "5-Fold Cross Validation": []
    }
)
```

Definir uma função para calcular o R-quadrado ajustado

O R-quadrado aumenta conforme o aumento do número de atributos. Por conta disso, às vezes é necessário uma forma mais robusta de calcular o desempenho entre modelos. Essa forma é representada no R-quadrado ajustado, calculado como:

$^R^2 = R^2 - (n - 1) * (1 - R^2) / (n - k - 1)$

Onde `k` é o número de parâmetros, e `n` é o número de observações.

```
def adjusted_r2(r2, n, k):
    return r2 - (n - 1) * (1 - r2) / (n - k - 1)
```

Definir o modelo de Regressão Linear Múltipla

Não vamos focar em modelos de Regressão Linear Simples. É um exercício interessante utilizar uma das variáveis independentes que discutimos e aplicar num modelo de regressão simples para identificar se o modelo fica ou não ajustado.

Vamos então partir para a aplicação do modelo de regressão linear múltipla.

```
train_data, test_data = train_test_split(finaldf, train_size=0.8, random_state=3)

independent_var = ["GDP", "Support", "Health", "Freedom", "Generosity", "Corruption"]
lin_reg = LinearRegression()
lin_reg.fit(train_data[independent_var], train_data["Score"])

print(f"Intercept: {lin_reg.intercept_}")
print(f"Coefficients: {lin_reg.coef_}")
```

```
Intercept: 1.8392422407690097
Coefficients: [0.76539426 1.10295366 1.01929667 1.57742356 0.26413816 0.83081881]
```

```
pred = lin_reg.predict(test_data[independent_var])


rmse = metrics.mean_squared_error(test_data["Score"], pred)
r2_train = lin_reg.score(train_data[independent_var], train_data["Score"])
ar2_train = adjusted_r2(
    lin_reg.score(
        train_data[independent_var],
        train_data["Score"]
    ),
    train_data.shape[0],
    len(independent_var)
)

r2_test = lin_reg.score(
    test_data[independent_var],
    test_data["Score"]
)
ar2_test = adjusted_r2(
    lin_reg.score(
        test_data[independent_var],
        test_data["Score"],
    ),
    test_data.shape[0],
    len(independent_var)
)

cross_val = cross_val_score(lin_reg, finaldf[independent_var], finaldf["Score"], cv=5).mean()

r = evaluation.shape[0]
evaluation.loc[r] = ["Multiple Linear Regression-1", "Selected features", rmse, r2_train, ar2_train, r2_t
evaluation.sort_values(by="5-Fold Cross Validation", ascending=False)
```

	Model	Details	Root Mean Squared Error (RMSE)	R-squared (training)	Adjusted R-squared (training)	R-squared (test)	Adjust sq (
0	Multiple Linear	Selected features	0.233369	0.758398	0.504407	0.832271	0.6

Analisar a correlação entre variáveis independentes

Como já falamos, é importante analisar a correlação entre as variáveis, para ter maior confiança sobre os nossos dados. Lembrando que o ideal é que as variáveis independentes sejam, de fato, independentes entre si.

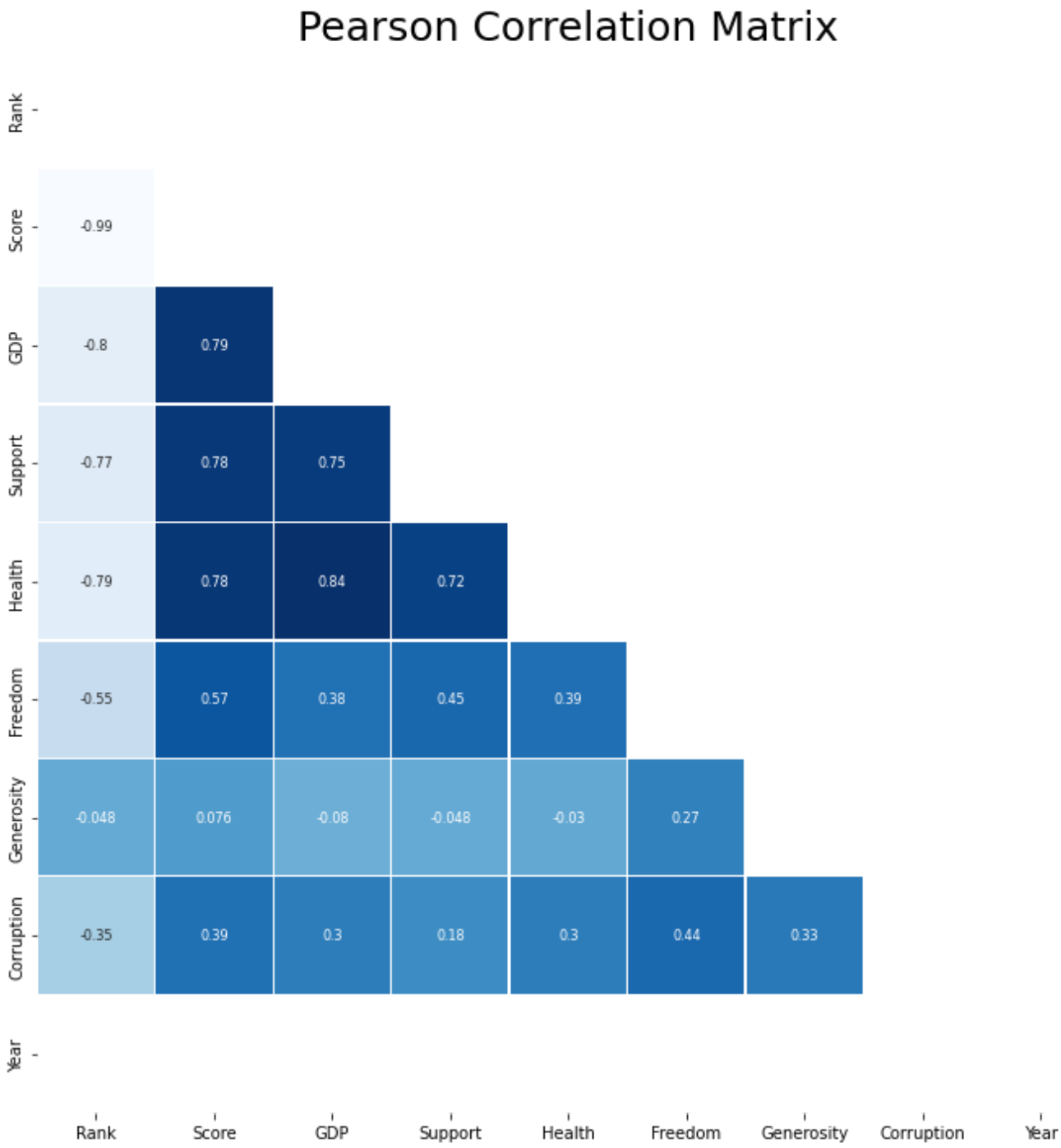
```
# Retorna um array de zeros com o mesmo shape e tipo do array dado
mask = np.zeros_like(finaldf.corr(), dtype=np.bool)

# Retorna os índices apenas do triângulo superior do array
mask[np.triu_indices_from(mask)] = True


plt.subplots(figsize=(16, 12))
plt.title("Pearson Correlation Matrix", fontsize=25)
```

```
sns.heatmap(
    finaldf.corr(),
    linewidths=0.25,
    square=True,  # definido para que cada célula seja quadrada
    cmap="Blues",
    linecolor="w",
    annot=True,  # definido para anotar o valor em cada célula
    annot_kws={"size": 8},  # argumentos da anotação
    mask=mask,  # os dados não são exibidos onde mask está como True
    cbar_kws={"shrink": 0.9}  # argumentos da color bar
)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f02dba695d0>
```



Apesar de termos algumas variáveis altamente correlacionadas (`Score`, `GDP`, `Support` e `Health`, principalmente), no geral os nossos dados possuem uma correlação não muito significativa.

Como exercício de fixação, é interessante alterar o DataFrame que utilizamos para remover algumas dessas variáveis independentes, e analisar os resultados do modelo gerado a partir daí. Quais foram as alterações observadas com relação às métricas utilizadas?