Regressão linear simples

Introdução

Imagino que todos conheçam o termo **correlação**. Essa medida leva em conta a força do relacionamento linear entre duas variáveis. Um índice de correlação próximo de 1 indica que, se uma variável tem uma alta, a outra também possui uma tendência de alta. Um índice próximo de -1 indica o oposto (uma variável tem uma alta, enquanto que a outra tem uma tendência de queda).

Para a maioria das aplicações, no entanto, saber que tal relacionamento linear existe não é o bastante. Nós queremos conseguir entender a natureza do relacionamento. É aí que usamos os modelos de regressão, começando pela regressão linear simples.

Antes de começarmos a falar sobre os modelos, é bom ficar bastante atento ao fato de que **correlação não significa causalidade**! Ou seja, o fato de dois dados apresentarem um índice alto de correlação, seja ele positivo ou negativo, não significa que um dado cause alteração no outro. Esse é um assunto muito delicado, então toda análise baseada em regressão precisa ser embasada, para que a regressão faça sentido.

Exemplos bem interessantes dessa "máxima" de que correlação é diferente de causalidade estão apresentados no site spurious correlations.

Os algoritmos de regressão também são considerados modelos supervisionados, já que possuímos dados sobre um alvo e estamos tentando estabelecer uma função que conecte esses dados.

Conceitos iniciais

Vamos pensar num cenário em que temos dados sobre a área de um imóvel e o preço de venda. Faz sentido imaginarmos que, quanto maior a área de um imóvel, maior será o preço de venda. Claro que outros fatores influenciam esse preço, mas vamos focar, no momento, na área do imóvel.

A regressão linear simples indica que, caso você desenhe a relação entre dois dados em um espaço bidimensional, você terá uma reta.

Ou seja, podemos considerar que a relação entre dois dados distintos atende a função:

preço = a * área + b

Nessa expressão, *a* é a inclinação da reta, enquanto que *b* é o ponto no eixo y (dos preços) cortado pela reta.

Chamamos a variável que se encontra no eixo "x" de *variável independente*, ou seja, ela é a suposta causa da variável no eixo "y", que é a *variável dependente*.

Sabemos que, em uma base de dados significativa, é extremamente improvável obtermos precisamente a relação mostrada para todos os pares de dados, conhecidos valores "a" e "b" (desenhar um eixo com uma reta e indicar os possíveis pares de dados). Portanto, podemos acertar a nossa função para o seguinte:

preço = a * área + b + e

Onde "e" é um termo de erro, que esperamos que seja pequeno. Ele normalmente contabiliza outros fatores não contemplados pelo nosso modelo, e o nosso objetivo é encontrar valores de "a" e "b" de forma que, no geral, o termo "e" seja o mínimo possível para todos os nossos pares de dados.

Portanto, temos que saber qual é o erro total sobre todo o conjunto de dados. Mas não podemos simplesmente adicionar os erros, pois erros positivos e negativos podem se anular e falsear o resultado. Por isso, nós somamos os erros ao quadrado, que são sempre positivos.

Dessa forma, o nosso problema passa a ser escolher valores "a" e "b" de forma que a soma dos quadrados dos erros seja a menor possível.

Não vamos entrar no mérito aqui de como calculamos matemáticamente essa soma mínima.

Aplicação

Para aplicar esse modelo de regressão linear vamos usar a scikit-learn em uma base de dados disponível no GitHub.

Passo 1: 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.linear_model import LinearRegression
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn import metrics


df = pd.read_csv("HousingPrices.csv")
```

Agora vamos explorar os dados. A primeira coisa é identificar o tamanho da base.

```
df.shape
```

```
(1460, 2)
```

O conjunto de dados possui 1460 linhas e duas colunas. Vamos ver como o nosso conjunto de dados está organizado.

```
df.head()
```

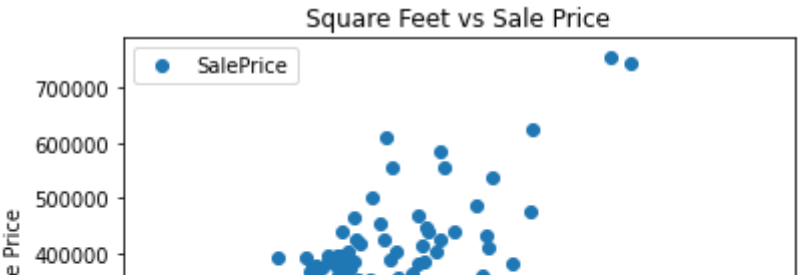
	SquareFeet	SalePrice
0	1710	208500
1	1262	181500
2	1786	223500
3	1717	140000
4	2198	250000

```
df.describe()
```

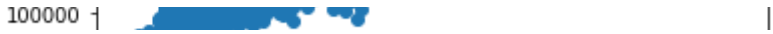
	SquareFeet	SalePrice
count	1460.000000	1460.000000
mean	1515.463699	180921.195890
std	525.480383	79442.502883
min	334.000000	34900.000000
25%	1129.500000	129975.000000
50%	1464.000000	163000.000000
75%	1776.750000	214000.000000
max	5642.000000	755000.000000

Podemos também desenhar nossos dados em um gráfico 2D e tentar identificar manualmente alguma relação entre os dados.

```
df.plot(x="SquareFeet", y="SalePrice", style="o")
plt.title("Square Feet vs Sale Price")
plt.xlabel("Square Feet")
plt.ylabel("Sale Price")
plt.show()
```



Pelo gráfico acima conseguimos observar claramente uma relação positiva linear entre a área do imóvel e o preço de venda.



Passo 2: tratamento e organização dos dados

Agora já temos algumas ideias com relação aos detalhes dos dados. O próximo passo é dividir os dados nos "atributos" e nos "alvos". Atributos são as variáveis independentes, e os alvos são as variáveis dependentes cujos valores serão previstos. No nosso conjunto de dados temos apenas duas colunas: nós queremos prever o preço de venda baseado na área do imóvel.

```
sq_feet = df.iloc[:, :-1].values
sale_price = df.iloc[:, 1].values
```

Os atributos foram armazenados na variável `sq_feet`. Especificamos o `-1` como a faixa para as colunas já que nós queremos conter todas as colunas do DataFrame exceto a última, que é a de preços. De forma análoga, a variável `sale_price` contém o nosso alvo.

Agora que temos os atributos e os alvos, o próximo passo é dividir os dados em treino e teste.

```
sq_feet_train, sq_feet_test, sale_price_train, sale_price_test = train_test_split(
    sq_feet, sale_price, test_size=0.2, random_state=0
)
```

A função acima dividiu 80% dos nossos dados no conjunto de treinamento, enquanto que 20% erá usado para os nossos testes.

Passo 3: divisão dos dados e treino do modelo

Já dividimos os nossos dados em treino e teste, portanto agora é hora de treinar o nosso modelo.

```
lin_reg = LinearRegression().fit(sq_feet_train, sale_price_train)
scores = cross_val_score(lin_reg, sq_feet_train, sale_price_train)
```

Cross validation (CV) é uma técnica muito utilizada para avaliação de desempenho de modelos de aprendizado de máquina. O CV consiste em particionar os dados em conjuntos (partes),

onde um conjunto é utilizador para treino e outro conjunto é utilizado para teste e avaliação do desempenho do modelo.

A utilização do CV tem altas chances de detectar se o seu modelo está sobrejustado ao seus dados de treinamento. Existe mais de um método de aplicação de CV. A função `cross_val_score` utiliza o método chamado **k-fold**.

K-fold consiste em dividir a base de dados de forma aleatória em K subconjuntos (com K definido previamente) com aproximadamente a mesma quantidade de amostras em cada um deles. A cada iteração, treino e teste, um conjunto formado por K - 1 subconjuntos são utilizados para treinamento e o subconjunto restante será utilizado para teste gerando um resultado de métrica para avaliação. Esse processo garante que cada subconjunto será utilizado para teste em algum momento da avaliação do modelo.


```
print(scores)
print(f"Score: {round(scores.mean(), 2)} (+/- {round(scores.std() * 2, 2)}))")
```

```
[0.51542273 0.49093565 0.48083647 0.55585545 0.55633684]
Score: 0.52 (+/- 0.06))
```

Esse `score` fornecido pela função `cross_val_score` é, por default, o **coeficiente de determinação**, ou **coeficiente R^2**. Basicamente ele é a soma dos erros quadrados. Se tivéssemos em um problema de classificação, poderíamos incluir o argumento `scoring="accuracy"` , por exemplo, para pegar os indicadores de acurácia dos modelos.

O coeficiente R^2 varia entre 0 e 1, onde valores mais próximos de 1 tendem a indicar um maior ajuste do modelo aos dados. O valor encontrado, de média 0.52, indica que nosso modelo não está tão bem ajustado aos dados. Faz sentido, já que existem inúmeros outros fatores que podem influenciar o preço dos imóveis, como renda per capta média na região, ano de construção, criminalidade do bairro, etc.

Voltando ao nosso modelo, podemos ver os coeficientes de inclinação da reta e interseção no eixo `y` utilizando os atributos abaixo:

```
print(lin_reg.coef_)
print(lin_reg.intercept_)
```

```
[110.26434426]
13330.293444921088
```

Isso significa que a nossa equação de formação dos preços com base na área dos imóveis é, desconsiderando os erros:

preço = 110.26 * área + 13330.29

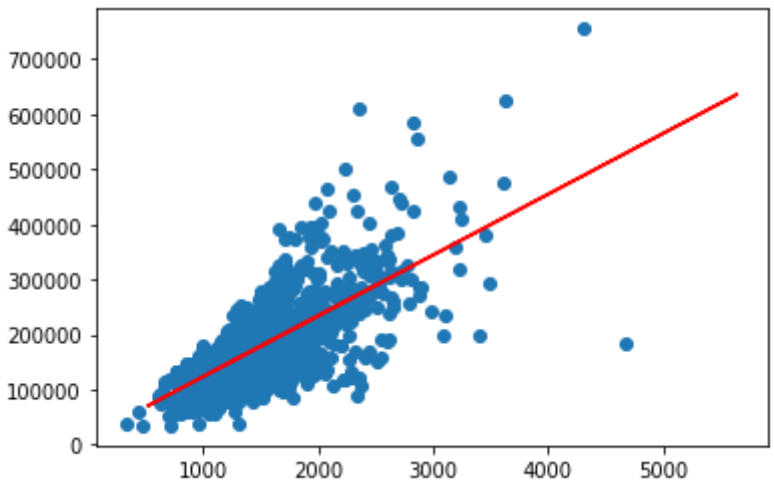
Com `área` sendo dada em pés quadrados.

Passo 4: análise dos resultados

Agora que treinamos nosso algoritmo, vamos fazer umas previsões. Usaremos nossos dados de teste para ver o quão acurado está o nosso modelo.

```
sale_price_pred = lin_reg.predict(sq_feet_test)
```

```
plt.scatter(sq_feet_train, sale_price_train)
plt.plot(sq_feet_test, sale_price_pred, color="red")
plt.show()
```



O `sale_price_pred` é um array do numpy que contém todos os valores previstos para os valores de input na série `sq_feet_test`.

Para compararmos os alvos reais com os previstos, podemos executar o seguinte:

```
df_compare = pd.DataFrame({"Actual": sale_price_test, "Predicted": sale_price_pred})
df_compare.head()
```

	Actual	Predicted
0	200624	290645.119259
1	133000	187327.428687
2	110000	145978.299590
3	192000	236284.797539
4	88000	133738.957377

Deu para ver que, realmente, o nosso modelo não é tão acurado assim. No entanto, deu para ter uma ideia do valor do imóvel baseado na sua área.

Avaliação

A última etapa é avaliar o desempenho do algoritmo. Para os algoritmos de regressão, utilizamos três indicadores:

- Média dos erros absolutos;
- Média dos erros quadrados;
- Raiz quadrada dos erros quadrados.

```
print(
  "Mean Absolute Error:",
  metrics.mean_absolute_error(
    sale_price_test,
    sale_price_pred
  )
)
print("Mean Sqaured Error:",
  metrics.mean_squared_error(
    sale_price_test,
    sale_price_pred
  )
)
print("Root Mean Squared Error:",
  np.sqrt(
    metrics.mean_squared_error(
      sale_price_test,
      sale_price_pred
    )
  )
)
```

```
Mean Absolute Error: 39364.76724953735
Mean Sqaured Error: 3913788296.4027987
Root Mean Squared Error: 62560.277304394986
```

Vimos lá em cima que a média dos preços de venda é da ordem de 180.000,00. Isso indica que a nossa raiz da média dos erros quadrados é muito maior que 10% dessa média, o que indica um algoritmo mediano. Certamente a nossa análise seria beneficiada pela inclusão de novos atributos.