escola britânica de artes criativas & tecnologia

Analista de Dados

Aprendizado de Máquina - Regressão

Módulo | Análise de Dados: Aprendizado de Máquina, Regressão

Caderno de Aula

Professor André Perez

Tópicos

1. Regressão;
2. Dados;
3. Treino;
4. Avaliação;
5. Predição;

Aulas

0. Abordagens estatísticas

- **Descritiva**: foco no passado para entender o **presente**.
- **Preditiva**: foca no passado para inferir o **futuro**.

1. Regressão

1.1. Motivação

> Dado a **altura** de um jogador, qual deve ser seu **peso**?

Queremos uma equação matemática que represente esta relação. Uma possível equação seria a equação linear de primero grau:

> $y = f(x) = \textbf{a}x + \textbf{b}$

O número $\textbf{a}$ é chamado de coeficiente angular e controla a inclinação da reta, já o número $\textbf{b}$ é chamado de coeficiente linear e indica o deslocamento horizontal da reta. A idéia é predizer o peso que um atleta deve ter dado a sua altura, ou seja:

> $peso = f(altura) = \textbf{a}(altura) + \textbf{b}$

Qual o melhor valor de $\textbf{a}$ e $\textbf{b}$ para esse conjunto de dados?

1.2. Regressão Linear

A regressão linear é um abordagem estatística que busca encontrar a relaçao entre um atributo alvo y (variável resposta) e um conjunto de atributos preditores x_i através de uma linha reta (em uma ou mais dimensões), relação essa preferencialmente **causal**. De maneira geral, busca encontrar $\textbf{a}_i$ e $\textbf{b}$ tal que:

> $y = f(x_i) = (\sum_{i=1}^{n} \textbf{a}_ix_i) + \textbf{b}$

Para apenas uma dimensão ou um atributo, temos:

> $y = f(x_1) = \textbf{a}_1x_1 + \textbf{b}$

- **Exemplo**: peso como função da altura.

> $peso = f(altura) = \textbf{a}(altura) + \textbf{b}$

Através do **treino** do modelo, encontra-se os valores de $\textbf{a}$ e $\textbf{b}$ que melhor se ajustam a um conjunto de dados.

1.3. Pacote Scikit-Learn

Pacote Python para ciência de dados e *machine learning*. A documentação pode ser encontrada neste link. Possuí diversos modelos para aprendizado supervisionado, não supervisionado, etc. além de métodos auxiliares. Para regressão linear, temos:

In [ ]:
```
from sklearn.linear_model import LinearRegression
```

In [ ]:
```
model = LinearRegression()
```

2. Dados

2.1. Pré-processamento

Neste módulo, vamos utilizar dados sobre o salário mensal em dólares americados de jogadores da NBA em 2020. O conjunto de dados está neste link e é uma cópia do conjunto de dados do Kaggle, presente neste link.

In [ ]:
```
!wget -q "https://raw.githubusercontent.com/andre-marcos-perez/ebac-course-
```

In [ ]:
```
import numpy as np
import pandas as pd
import seaborn as sns
```

In [ ]:
```
nba = pd.read_csv("nba.csv")
```

Vamos conhecer um pouco melhor o conjunto de dados.

In [ ]:
```
nba.head()
```

In [ ]:
```
nba.info()
```

In [ ]:
```
nba.describe().T
```

In [ ]:
```
nba.drop(["rating", "draft_year"], axis=1).describe()
```

Vamos selecionar os atributos que podem estar relacionados com o salário de um jogador e manipular um a um. Como estamos modelando o problema como se o peso fosse uma função exclusiva da altura de um jogador, temos:

In [ ]:
```
data = nba[["weight", "height"]]
```

In [ ]:
```
data.head()
```

- **Height**: Atributo numérico, formatar e padronizar.

In [ ]:
```
data[['height']].head()
```

In [ ]:
```
data['height'] = data['height'].apply(
    lambda height: float(height.split(sep='/')[-1].strip())
)
```

In [ ]:
```
data[['height']].describe().T
```

- **Weight**: Variável resposta numérica, formatar.

In [ ]:
```
data[['weight']].head()
```

In [ ]:
```
data['weight'] = data['weight'].apply(
    lambda weight: float(weight.split(sep='/')[-1].split(sep='kg')[0].strip
)
```

In [ ]:
```
data[['weight']].describe().T
```

O resultado do pré-processamento nos trás um dado limpo e pronto para ser utilizado no treino do modelo.

In [ ]:
```
data.head()
```

2.2. Treino / Teste

De maneira geral, um modelo de aprendizagem supervisionada precisa ser treinado com um conjunto de dados e avaliado com outro, assim conseguimos enter um pouco melhor a capacidade do modelo em **generalizar** as predições com dados não visto, que é a situação real em que será utilizado. Para tanto, dividimos nossa base de dados em duas: uma maior de **treino** e uma menor de **testes**.

In [ ]:
```
from sklearn.model_selection import train_test_split
```

In [ ]:
```
predictors_train,
predictors_test,
target_train,
target_test = train_test_split(
    data.drop(['weight'], axis=1),
    data['weight'],
    test_size=0.25,
    random_state=123
)
```

- **Variáveis preditoras (predictors)**

In [ ]:
```
predictors_train.head()
```

In [ ]:
```
predictors_train.shape
```

In [ ]:
```
predictors_test.head()
```

In [ ]:
```
predictors_test.shape
```

- **Variável resposta (target)**

In [ ]:
```
target_train.head()
```

In [ ]:
```
target_train.shape
```

In [ ]:
```
target_test.head()
```

In [ ]:
```
target_test.shape
```

3. Treino

O treino de modelos de aprendizagem supervisionada consiste na etapa de calculo dos coeficientes do modelo baseado na associação da variável resposta com os variáveis preditoras através do uso de um ou mais algoritmos. No caso da regressão linear, estamo interessados em definir os valores de $\textbf{a}_i$ e $\textbf{b}$:

> $y = f(x_i) = (\sum_{i=1}^{n} \textbf{a}_ix_i) + \textbf{b}$

3.1. Algoritmo

O treino de um modelo de regressão linear é feito através do uso do método de gradiente (explicação neste link). A explicação do algoritmo foge do escopo desse curso mas a idéia é que busca-se minimizar a diferença entre os pontos e a reta definida por $\textbf{a}_i$ e $\textbf{b}$, ou seja, encontrar os valores de $\textbf{a}_i$ e $\textbf{b}$ que define a reta que esta mais "perto" de todos os pontos da base de dados de treino.

In [ ]:
```
model = model.fit(predictors_train, target_train)
```

In [ ]:
```
model.__dict__
```

In [ ]:
```
a = model.coef_
print(a)
```

In [ ]:
```
b = model.intercept_
print(b)
```

Logo, temos:

> $y = f(x) = \textbf{a}x + \textbf{b}$

> $peso = f(altura) = \textbf{a}(altura) + \textbf{b}$

> $peso = f(altura) = \textbf{88.9746} * (altura) - \textbf{80.2631}$

Com o modelo treinado, estamos prontos para fazer predições.

In [ ]:
```
data.head(1)
```

In [ ]:
```
altura = 2.06
peso = (a * altura) + b

print(peso)
```

In [ ]:
```
altura = np.array([2.06])
peso = model.predict(altura.reshape(-1, 1))

print(peso)
```

4. Avaliação

Para enter o poder preditivo do modelo de aprendizagem supervisionada, precisamos avaliar sua capacidade de generalização, ou seja, avalivar as predições em dados "não vistos" na etapa de treino. Comparamos então as predições com os dados reais através de uma métrica.

- **Salário predito**

In [ ]:
```
target_predicted = model.predict(predictors_test)
```

In [ ]:
```
target_predicted[0:5]
```

In [ ]:
```
target_predicted.shape
```

- **Salário teste**

In [ ]:
```
target_test[0:5]
```

In [ ]:
```
target_test.shape
```

4.1. RMSE

Do inglês *root mean square error* ou raíz quadrada do erro quadrático médio, o RMSE mede a diferença média absoluta entre os valores preditos com os valores reais. O resultado pode ser interpretado com uma faixa de valor em que a predição varia do valor real, portanto, quando menor, melhor. Contudo, a definição de "menor" é particular para cada variável resposta devido a diferenças de escala.

- Exemplo:

Um RMSE igual a 100 kg significa que a predição varia, na média, entre +/- 100 kg. Se a variável resposta for o peso de um carro (toneladas), esse valor é excelente. Contudo, se a variável resposta for o peso de uma motocicleta (centenas de kilos), esse valor pode ser muito alto.

O RMSE é definido como:

> $RMSE(y,\hat{y}) = \sqrt{\frac{\sum_{i=1}^{n} (y_i-\hat{y}_i)^{2}}{n}}$

In [ ]:
```
from sklearn.metrics import mean_squared_error
```

In [ ]:
```
rmse = np.sqrt(mean_squared_error(target_test, target_predicted))
print(rmse)
```

Para facilitar sua interpretação, vamos colocar numa gráfico os valores de reais de teste e os valores preditos.

In [ ]:
```
test_data = pd.concat(
    [
        predictors_test,
        pd.DataFrame(target_test, columns=['weight'])
    ], axis=1
).reset_index(drop=True)
test_data['predicted'] = False
```

In [ ]:
```
test_data.head()
```

In [ ]:
```
prediction_data = pd.concat(
    [
        predictors_test.reset_index(drop=True),
        pd.DataFrame(target_predicted, columns=['weight'])
    ], axis=1
).reset_index(drop=True)
prediction_data['predicted'] = True
```

In [ ]:
```
prediction_data.tail()
```

In [ ]:
```
prediction = pd.concat(
    [test_data, prediction_data]
).reset_index(drop=True)
```

In [ ]:
```
with sns.axes_style('whitegrid'):

  # peso = 88.9746 * altura - 80.2631
  sns.scatterplot(data=prediction, x='height', y='weight', hue='predicted'
```

4.2. Comparação

Um dos objetivos de métricas de avaliação de modelos é comparar a qualidade de diferentes modelos. Para ilustrar, vamos construir uma segunda versão do modelo que utiliza um atributo a mais, a posição do jogador. A hipótese é que posições diferentes exigem alturas e pesos diferentes.

In [ ]:
```
data = nba[['weight', 'height', 'position']]
data.head()
```

- **Position**: Atributo categórico nominal, one-hot encoding.

In [ ]:
```
data["position"].drop_duplicates()
```

In [ ]:
```
data['position_f'] = data['position'].apply(
    lambda sex: 1 if sex == 'F' else 0
)
data['position_g'] = data['position'].apply(
    lambda sex: 1 if sex == 'G' else 0
)
data['position_c'] = data['position'].apply(
    lambda sex: 1 if sex == 'C' else 0
)
data['position_fg'] = data['position'].apply(
    lambda sex: 1 if sex == 'F-G' else 0
)
data['position_fc'] = data['position'].apply(
    lambda sex: 1 if sex == 'F-C' else 0
)
data['position_gf'] = data['position'].apply(
    lambda sex: 1 if sex == 'G-F' else 0
)
data['position_cf'] = data['position'].apply(
    lambda sex: 1 if sex == 'C-F' else 0
)
```

In [ ]:
```
data[[
    'position',
    'position_f',
    'position_g',
    'position_c',
    'position_fg',
    'position_fc',
    'position_gf',
    'position_cf'
]].head()
```

In [ ]:
```
data = data.drop(['position'], axis=1)
```

- **Height**: Atributo numérico, formatar e padronizar.

In [ ]:
```
data['height'] = data['height'].apply(
    lambda height: float(height.split(sep="/")[-1].strip())
)

altura_media = data['height'].mean()
altura_desvio_padrao = data['height'].std()

data['height'] = data['height'].apply(
    lambda height: (height - altura_media) / altura_desvio_padrao
)
```

- **Weight**: Variável resposta numérica, formatar.

In [ ]:
```
data["weight"] = data["weight"].apply(
    lambda weight: float(weight.split(sep="/")[-1].split(sep="kg")[0].strip
)
```

- **Treino**

In [ ]:
```
predictors_train,
predictors_test,
target_train,
target_test = train_test_split(
    data.drop(['weight'], axis=1),
    data['weight'],
    test_size=0.25,
    random_state=123
)
```

In [ ]:
```
model_v2 = model.fit(predictors_train, target_train)
model_v2.__dict__
```

Dessa vez, estamos buscando uma reta multidimensional, portanto:

> $y = f(x_i) = (\sum_{i=1}^{n} \textbf{a}_ix_i) + \textbf{b}$

In [ ]:
```
a = model_v2.coef_
print(a)
```

In [ ]:
```
b = model_v2.intercept_
print(b)
```

> $peso = f(altura, posicao_f, ...) = \textbf{a}_1(altura) + \textbf{a}_2(posicao_f) + ... + \textbf{b}$

> $peso = f(altura, posicao_f, ...) = \textbf{3.5103}(altura) + \textbf{0.4282}(posicao_f) + ... + \textbf{99.0711}$

- **Avaliação**

In [ ]:
```
target_predicted = model_v2.predict(predictors_test)
```

In [ ]:
```
rmse_v2 = np.sqrt(mean_squared_error(target_test, target_predicted))
print(rmse_v2)
```

Com o RMSE para ambas as versões do modelo, podemos compara-los.

> **RMSE v1**: 7.680474067138796
>
> **RMSE v2**: 7.267852030433409

In [ ]:
```
print(f"Melhoria de {round(100 * (1 - (rmse_v2 / rmse)), 2)}%")
```

Podemos observar a melhoria de performance num gráfico.

In [ ]:
```
test_data = pd.concat(
    [
        predictors_test,
        pd.DataFrame(target_test, columns=['weight'])
    ], axis=1
).reset_index(drop=True)
test_data['predicted'] = False

prediction_data = pd.concat(
    [
        predictors_test.reset_index(drop=True),
        pd.DataFrame(target_predicted, columns=['weight'])
    ], axis=1
).reset_index(drop=True)
prediction_data['predicted'] = True

prediction_v2 = pd.concat(
    [test_data, prediction_data]
).reset_index(drop=True)
```

In [ ]:
```
with sns.axes_style('whitegrid'):

  # peso = 88.9746 * altura - 80.2631
  sns.scatterplot(
      data=prediction,
      x='height',
      y='weight',
      hue='predicted'
  )
```

In [ ]:
```
with sns.axes_style('whitegrid'):

  # peso = 3.51 * altura + 0.42 * posicao_f + ... + 99.07
  sns.scatterplot(
      data=prediction_v2,
      x='height',
      y='weight',
      hue='predicted'
  )
```

5. Predição

Com o modelo treinado, avaliado e selecionado, podemos utiliza-lo para resolver os problemas reais que motivaram sua construção, para tanto para criar um exemplo pré-processado e utilizar o modelo para realizar a predição.

> **Atenção**: O exemplo precisa seguir o mesmo pré-processamento realizado na construção do modelo.

- **Exemplo**: Pivô com 2.05m e 99 kg.

In [ ]:
```
data.head(1)
```

In [ ]:
```
altura_padronizada = (2.05 - altura_media) / altura_desvio_padrao
print(altura_padronizada)
```

In [ ]:
```
jogador = np.array([altura_padronizada, 0, 0, 1, 0, 0, 0, 0])
```

In [ ]:
```
peso = model_v2.predict(jogador.reshape(1, -1))
print(peso)
```

Conclui-se que o jogador **precisa** ganhar peso para atuar como pivô.

- **Exemplo**: Atacante com 2.05m e 99 kg.

In [ ]:
```
data.head(1)
```

In [ ]:
```
altura_padronizada = (2.05 - altura_media) / altura_desvio_padrao
print(altura_padronizada)
```

In [ ]:
```
jogador = np.array([altura_padronizada, 1, 0, 0, 0, 0, 0, 0])
```

In [ ]:
```
peso = model_v2.predict(jogador.reshape(1, -1))
print(peso)
```

Conclui-se que o jogador **não precisa** ganhar peso para atuar como atacante.