

Um guia completo para o pré-processamento de dados em machine learning



Caíque Coelho

Jun 16, 2019 · 15 min read

A bala de prata do pré-processamento para qualquer dataset!

A utilização de técnicas de inteligência artificial para resolver diversos problemas é um processo que em si possui inúmeros etapas que podem ser aplicadas em um ciclo a fim de chegarmos em um resultado satisfatório.

Dentro deste ciclo a etapa pré-processamento talvez seja a mais importante a fim de se obter um bom resultado, o pré-processamento nada mais é do que o processo de preparação, organização e estruturação dos nossos dados além de ser o momento ideal para escolhermos quais dados fazem sentido fazerem parte do nosso dataset.

A ideia deste artigo é ser um ponto de referência para técnicas de pré-processamento que possam ser aplicadas em qualquer tipo de dados que mais tarde serão utilizados em técnicas de classificação com machine learning.

Antes de colocarmos a mão na massa é importante ressaltar a importância desta fase, pois a qualidade dos seus dados pode influenciar diretamente no resultado do seu modelo, muitas das vezes acabamos pensando que o problema da nossa solução é o algoritmo usado para gerar o modelo, porém o grande vilão é o seu próprio conjunto de dados que podem possuir muitos atributos com valores faltantes, outliers e escalas de valores contradizentes e por fim nenhum modelo será capaz de trabalhar com esses dados e gerar resultados de qualidade. Tenha em mente:

A qualidade do resultado do seu modelo começa com a qualidade dos dados que você está “inputando” na etapa de treino!

Sem mais delongas vamos ao que interessa, mão na massa!

Vocês podem ter acesso ao dataset de exemplo usado aqui e a todo o código acessando o projeto no github:

CaiqueCoelho/Preprocessing-Dataset-Template

A template to preprocessing your golden dataset before to put your data in your best model ...

[github.com](#)

Para começar vamos importar algumas bibliotecas importantes para esse trabalho. Numpy que irá nos permitir trabalhar de forma eficaz com vetores e matrizes além de facilitar alguns cálculos matemáticos e Pandas que irá nos permitir manipular de forma muito fácil o nosso dataset.

```
import numpy as np
import pandas as pd
```

Vamos em seguida carregar o nosso dataset de forma simples com o pandas

```
dataset = pd.read_csv('fake_data_2.csv')
```

Para visualizarmos os nossos dados vamos usar a função head que irá nos trazer os 5 primeiros dados do nosso dataset

```
dataset.head()
```

	cargo	idade	salario	bonus	sócio
0	Diretor	45	24000.0	10000.0	sim
1	Analista	22	8000.0	2000.0	não
2	Programador	30	NaN	1000.0	não
3	Gerente	24	15100.0	NaN	não
4	Gerente	30	35000.0	6000.0	sim

Explicando um pouco melhor o nosso dataset, nós temos os dados de 13 funcionários de uma empresa e baseado no cargo, idade, salário e bônus o nosso modelo futuramente tentará nos dizer quem são os sócios desta empresa.

Agora que conhecemos melhor nossos dados a primeira coisa que devemos fazer é separar os dados em um conjunto de atributos que serão usados como variáveis de input: cargo, idade, salário e bônus e separar a variável resultante: sócio.

```
X = dataset.iloc[:, :-1].values # cargo, idade, salário e bônus  
Y = dataset.iloc[:, 4].values # sócio
```

Tratando dados faltantes

Show, vamos aos fatos interessantes agora. Como já podemos notar de cara existem alguns dados faltando, como por exemplo o salário e bônus de alguns funcionários. Este problema contém inúmeras formas de ser resolvido e a melhor solução depende muito do seu conjunto de dados e do seu problema, portanto analise a sua situação e escolha a melhor forma de tratar esse problema, talvez a melhor solução nem esteja aqui:

1. Deletar as colunas com dados faltantes: Essa solução ao meu ver é bem drástica e somente deverá ser utilizada quando a variável não exercer uma certa influência no resultado procurado. Talvez seja esta a solução **menos** recomendada! Mas caso queira utilizá-la basta usar a função `dropna` pandas, utilizando como parâmetros o `axis = 1` para dizer que queremos deletar a coluna e `inplace = True` para aplicarmos no dataset e não criarmos uma cópia deste:

```
dataset.dropna(axis=1, inplace=True)
```

2. Deletar os exemplos com dados faltantes: Uma solução bem melhor para o problema porém ainda não é a ideal para um dataset no qual você possui poucos exemplos, para aplicar basta utilizarmos a função `dropna` do pandas com o `axis = 0`:

```
dataset.dropna(axis=0, inplace=True)
```

3. Preencher os dados faltantes com a média dos valores do atributo: Essa é a minha solução favorita e que iremos utilizar nesse guia. Em nosso problema iremos fazer isso da forma mais simples, aplicando o valor da média das colunas salário e bônus nos exemplos que não possuem esse valor. Notem que se quiséssemos poderíamos ir um pouco mais afundo e calcular essas médias de acordo com o cargo para depois aplicarmos a média que mais faz sentido. Primeiro vamos voltar ao cabeçalho e importar a classe Imputer que irá nos auxiliar nessa tarefa:

```
from sklearn.preprocessing import Imputer
```

Em seguida vamos criar um objeto da classe Imputer:

```
imputer = Imputer(missing_values = np.nan, strategy = 'mean', axis = 0)
```

Agora vamos aplicar o método fit do imputer apenas nas colunas com dados faltantes:

```
imputer = imputer.fit(X[:, 2:4])
```

Por fim vamos aplicar o método transform do imputer apenas nas colunas que precisamos para calcularmos a média e salvarmos em nossa variável X:

```
X[:, 2:4] = imputer.transform(X[:, 1:3])
```

```
array([[ 'Diretor', 45, 24000.0, 10000.0],  
       [ 'Analista', 22, 8000.0, 2000.0],  
       [ 'Programador', 30, 16795.333333333332, 1000.0],  
       [ 'Gerente', 24, 15100.0, 7200.0],  
       [ 'Gerente', 30, 35000.0, 6000.0],  
       [ 'Programador', 22, 5300.0, 2000.0],  
       [ 'Analista', 20, 16795.333333333332, 1200.0],  
       [ 'Diretor', 50, 18000.0, 8000.0],  
       [ 'Fundador', 65, 38000.0, 28000.0],  
       [ 'Analista', 32, 7300.0, 4000.0],  
       [ 'Programador', 35, 2344.0, 7200.0],  
       [ 'Programador', 28, 4500.0, 2200.0],  
       [ 'Fundador', 28, 30000.0, 12000.0],  
       [ 'Programador', 30, 14000.0, 10000.0]], dtype=object)
```

4. Preencher os dados faltantes com o valor que você quiser: Com essa alternativa o céu é o limite e você poderá preencher os seus dados faltantes com o valor que melhor convier para o seu problema, para isto basta utilizar a função `fillna` do `pandas`, alguns exemplos para o nosso problema seriam:

```
dataset.fillna(0) #Preencher todos os valores faltantes por zero

#Preencher cada coluna com o valor que melhor satisfazer:
values = {'salario': valor, 'bonus': valor}
dataset.fillna(value=values)
```

Variáveis categóricas

Outro problema do nosso dataset são as variáveis categóricas que nesse caso se restringem apenas a coluna `cargo`, ou seja, uma variável categórica é uma variável nominal, sem escala, não numérica.

Uma ideia para tratar esse problema é utilizar a classe **LabelEncoder** do `sklearn` para transformar os nomes em números (`diretor`: 0, `analista`: 1, `gerente`: 2, `programador`: 3 e `fundador`: 4) e por conseguinte transformar esse números em novas colunas do dataset com **OneHotEncoder** do `sklearn`, com objetivo de eliminar a hierarquia dos valores que não possuem muito significado para os cargos neste problema. Ou seja os cargos `diretor`, `analista`, `gerente`, `programador` e `fundador` seriam colunas do nossa dataset e cada funcionário receberá o valor 1 para o seu cargo na coluna e o valor 0 para os cargos que não ocupam.

Vamos começar importando o `LabelEncoder` e o `OneHotEncoder` no cabeçalho:

```
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
```

Agora vamos criar um objeto do `LabelEncoder` e fazer `fit_transform` apenas para a nossa coluna com valores categóricos:

```
labelencoder_X = LabelEncoder()
X[:, 0] = labelencoder_X.fit_transform(X[:, 0])
```

```
array([[1, 45, 24000.0, 10000.0],
       [0, 22, 8000.0, 2000.0],
       [4, 30, 16795.333333333332, 1000.0],
       [3, 24, 15100.0, 7200.0],
       [3, 30, 35000.0, 6000.0],
       [4, 22, 5300.0, 2000.0],
       [0, 20, 16795.333333333332, 1200.0],
       [1, 50, 18000.0, 8000.0],
       [2, 65, 38000.0, 28000.0],
       [0, 32, 7300.0, 4000.0],
       [4, 35, 2344.0, 7200.0],
       [4, 28, 4500.0, 2200.0],
       [2, 28, 30000.0, 12000.0],
       [4, 30, 14000.0, 10000.0]], dtype=object)
```

Pronto, já transformamos os nossos valores nominais em numéricos. Podemos notar que os cargos diretor, analista, programador, gerente e fundador são representados respectivamente por 1, 0, 4, 3 e 2. Agora basta transformar esses números em colunas com o OneHotEncoder:

```
onehotencoder = OneHotEncoder(categorical_features = [0])
X = onehotencoder.fit_transform(X).toarray()
```

```
array([[0.00000000e+00, 1.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 4.50000000e+01, 2.40000000e+04, 1.00000000e+04],
       [1.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 2.20000000e+01, 8.00000000e+03, 2.00000000e+03],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        1.00000000e+00, 3.00000000e+01, 1.67953333e+04, 1.00000000e+03],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 1.00000000e+00,
        0.00000000e+00, 2.40000000e+01, 1.51000000e+04, 7.20000000e+03],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 1.00000000e+00,
        0.00000000e+00, 3.00000000e+01, 3.50000000e+04, 6.00000000e+03],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        1.00000000e+00, 2.20000000e+01, 5.30000000e+03, 2.00000000e+03],
       [1.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 2.00000000e+01, 1.67953333e+04, 1.20000000e+03],
       [0.00000000e+00, 1.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 5.00000000e+01, 1.80000000e+04, 8.00000000e+03],
       [0.00000000e+00, 0.00000000e+00, 1.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 6.50000000e+01, 3.80000000e+04, 2.80000000e+04],
       [1.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 3.20000000e+01, 7.30000000e+03, 4.00000000e+03],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        1.00000000e+00, 3.50000000e+01, 2.34400000e+03, 7.20000000e+03],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        1.00000000e+00, 2.80000000e+01, 4.50000000e+03, 2.20000000e+03],
       [0.00000000e+00, 0.00000000e+00, 1.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 2.80000000e+01, 3.00000000e+04, 1.20000000e+04],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        1.00000000e+00, 3.00000000e+01, 1.40000000e+04, 1.00000000e+04]])
```

Continuando podemos deixar nossa variável X como um array ou transformar novamente em um dataframe:

```
X = pd.DataFrame(np.concatenate(X))
X = pd.DataFrame(X, columns=['cargo_diretor', 'cargo_analista',
'cargo_programador', 'cargo_gerente', 'cargo_fundador', 'idade',
'salario', 'bonus'])
```

	cargo_diretor	cargo_analista	cargo_programador	cargo_gerente	cargo_fundador	idade	salario	bonus
0	0.0	1.0	0.0	0.0	0.0	45.0	24000.000000	10000.0
1	1.0	0.0	0.0	0.0	0.0	22.0	8000.000000	2000.0
2	0.0	0.0	0.0	0.0	1.0	30.0	16795.333333	1000.0
3	0.0	0.0	0.0	1.0	0.0	24.0	15100.000000	7200.0
4	0.0	0.0	0.0	1.0	0.0	30.0	35000.000000	6000.0
5	0.0	0.0	0.0	0.0	1.0	22.0	5300.000000	2000.0
6	1.0	0.0	0.0	0.0	0.0	20.0	16795.333333	1200.0
7	0.0	1.0	0.0	0.0	0.0	50.0	18000.000000	8000.0
8	0.0	0.0	1.0	0.0	0.0	65.0	38000.000000	28000.0
9	1.0	0.0	0.0	0.0	0.0	32.0	7300.000000	4000.0
10	0.0	0.0	0.0	0.0	1.0	35.0	2344.000000	7200.0
11	0.0	0.0	0.0	0.0	1.0	28.0	4500.000000	2200.0
12	0.0	0.0	1.0	0.0	0.0	28.0	30000.000000	12000.0
13	0.0	0.0	0.0	0.0	1.0	30.0	14000.000000	10000.0

Por fim vamos transformar nossa variável Y com valores categóricos sim e não em valores numéricos, onde teremos 1 para sim e 0 para não:

```
labelencoder_y = LabelEncoder()
Y = labelencoder_y.fit_transform(y)
```

```
array([1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1])
```

Outra forma de fazermos essa transformação é com o próprio pandas com o método, `get_dummies`:

Primeiro vamos criar um dataframe apenas com a coluna cargos:

```
X_cargo = pd.DataFrame({'cargo':X[:,0]})
```

	cargo
0	Diretor
1	Analista
2	Programador
3	Gerente
4	Gerente
5	Programador
6	Analista
7	Diretor
8	Fundador
9	Analista
10	Programador
11	Programador
12	Fundador
13	Programador

Agora vamos transformar o nosso dataframe com uma coluna de variáveis categóricas em colunas que representam cada cargo:

```
X_cargo = pd.get_dummies(X_cargo)
```

	cargo_Analista	cargo_Diretor	cargo_Fundador	cargo_Gerente	cargo_Programador
0	0	1	0	0	0
1	1	0	0	0	0
2	0	0	0	0	1
3	0	0	0	1	0
4	0	0	0	1	0

Agora vamos extrair da variável X apenas as colunas com variáveis numéricas e transformar em um dataframe

```
X = X[:, 1:]
X = pd.DataFrame({'idade':X[:,0], 'salario':X[:,1], 'bonus':X[:,2]})
```


	idade	salario	bonus
0	45	24000	10000
1	22	8000	2000
2	30	16795.3	1000
3	24	15100	7200
4	30	35000	6000

Por fim vamos unir as colunas dos cargos ao nosso dataframe X:

```
X = X.join(X_cargo)
```

	idade	salario	bonus	cargo_Analista	cargo_Diretor	cargo_Fundador	cargo_Gerente	cargo_Programador
0	45	24000	10000	0	1	0	0	0
1	22	8000	2000	1	0	0	0	0
2	30	16795.3	1000	0	0	0	0	1
3	24	15100	7200	0	0	0	1	0
4	30	35000	6000	0	0	0	1	0
5	22	5300	2000	0	0	0	0	1
6	20	16795.3	1200	1	0	0	0	0
7	50	18000	8000	0	1	0	0	0
8	65	38000	28000	0	0	1	0	0
9	32	7300	4000	1	0	0	0	0
10	35	2344	7200	0	0	0	0	1
11	28	4500	2200	0	0	0	0	1
12	28	30000	12000	0	0	1	0	0
13	30	14000	10000	0	0	0	0	1

Por conseguinte também vamos transformar a nossa variável Y (é sócio) em valores numéricos, onde sim será 1 e não será zero:

```
Y = pd.get_dummies(Y)
Y = Y['sim'].values
```

```
array([1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1], dtype=uint8)
```

Reescala dos dados

Agora temos um novo problema, você pode observar que os valores das colunas idade, salário e bônus estão em uma escala bem distinta, onde a idade varia entre 20 a 65, salário varia entre 2344 a 38000 e bônus varia entre 1200 a 28000, e isto pode causar um grande problema no treino do nosso modelo uma vez o salário por possuir uma escala muito maior que a idade terá uma influência consequentemente muito maior no resultado e isto não é que nós queremos! Além disso podemos notar que as bordas representam alguns possíveis outliers os quais queremos minimizar o impacto em nossa solução. Para solucionar esses problemas poderíamos usar diversas técnicas de estatística e algumas métricas como quartis, desvio padrão e variância ou podemos ser muito mais espertos e já usar inúmeras ferramentas do sklearn que já aplicam todas essas técnicas a fim de obter uma melhor solução para o nosso problema através da reescala dos dados.

Normalizer

Vamos começar com o Normalizer que talvez seja a solução mais diferente, uma vez que o Normalizer age reescalando os dados por exemplos/linhas e não por colunas, ou seja, o Normalizer levará em contas os atributos idade, salário e bonus e reescalar os valores com base nesses três valores. O Normalizer é uma boa escolha quando você sabe que a distribuição dos seus dados não é normal/gaussiana ou quando você não sabe qual é o tipo de distribuição dos seus dados.

```
from sklearn.preprocessing import Normalizer
```

```
X_normalize = X.copy()
X_normalize[['idade', 'salario', 'bonus']] =
Normalizer().fit_transform(X[['idade', 'salario', 'bonus']])
```

	idade	salario	bonus	cargo_Analista	cargo_Diretor	cargo_Fundador	cargo_Gerente	cargo_Programador
0	0.001731	0.923076	0.384615	0	1	0	0	0
1	0.002668	0.970139	0.242535	1	0	0	0	0
2	0.001783	0.998231	0.059435	0	0	0	0	1
3	0.001435	0.902638	0.430397	0	0	0	1	0
4	0.000845	0.985622	0.168964	0	0	0	1	0

MinMaxScaler

O MinMaxScaler é uma outra alternativa a reescala de dados, seu diferencial se dá uma vez que este age sobre sobre a coluna, ou seja, o cálculo da reescala é feito de forma independente entre cada coluna, de tal forma que a nova escala se dará entre 0 e 1 (ou -1 e 1 se houver valores negativos no dataset). Importante ressaltar que essa técnica funciona melhor se a distribuição dos dados não for normal e se o desvio padrão for pequeno, além disso o MinMaxScaler não reduz de forma eficaz o impacto de outliers e também preserva a distribuição original. De forma simples o MinMaxScaler subtrai o valor em questão pelo menor valor da coluna e então divide pela diferença entre o valor máximo e mínimo:

$$\text{valor} = (\text{valor} - \text{Coluna.min}) / (\text{Coluna.max} - \text{Coluna.min})$$

```
from sklearn.preprocessing import MinMaxScaler

X_minMax = X.copy()
X_minMax[['idade', 'salario', 'bonus']] =
MinMaxScaler().fit_transform(X[['idade', 'salario', 'bonus']])
```

	idade	salario	bonus	cargo_Analista	cargo_Diretor	cargo_Fundador	cargo_Gerente	cargo_Programador
0	0.555556	0.607359	0.333333	0	1	0	0	0
1	0.044444	0.158627	0.037037	1	0	0	0	0
2	0.222222	0.405299	0.000000	0	0	0	0	1
3	0.088889	0.357752	0.229630	0	0	0	1	0
4	0.222222	0.915863	0.185185	0	0	0	1	0

StandardScaler

Assim como o MinMaxScaler o StandardScaler age sobre as colunas, porém seu método é diferente uma vez que este subtrai do valor em questão a média da coluna e divide o resultado pelo desvio padrão. No final temos uma distribuição de dados com desvio padrão igual a 1 e variância de 1 também. Esse método trabalha melhor em dados com distribuição normal porém vale a tentativa para outros tipos de distribuições, além disso podemos deixar como dica que esse método resulta em ótimos frutos quando usado em conjunto com algoritmos como Linear Regression e Logistic Regression.

$$\text{valor} = (\text{valor} - \text{média}) / \text{desvioPadão}$$

```

from sklearn.preprocessing import StandardScaler

X_standard = X.copy()
X_standard[['idade', 'salario', 'bonus']] =
StandardScaler().fit_transform(X[['idade', 'salario', 'bonus']])

```

	idade	salario	bonus	cargo_Analista	cargo_Diretor	cargo_Fundador	cargo_Gerente	cargo_Programador
0	1.002248	0.657862	0.415034	0	1	0	0	0
1	-0.907361	-0.803107	-0.770778	1	0	0	0	0
2	-0.243149	0.000000	-0.919005	0	0	0	0	1
3	-0.741308	-0.154802	0.000000	0	0	0	1	0
4	-0.243149	1.662279	-0.177872	0	0	0	1	0

RobustScaler

Também atua sobre as colunas e o diferencial deste método é a combinação com o uso de quartis o que nos garante um bom tratamento dos outliers. Em seu método o RobustScaler subtrai a média do valor em questão e então divide o resultado pelo segundo quartil. Importante notar que os outliers ainda estão presentes porém estão representados dentro de uma escala em que o seu impacto negativo é reduzido.

```

from sklearn.preprocessing import RobustScaler

X_robust = X.copy()
X_robust[['idade', 'salario', 'bonus']] =
RobustScaler().fit_transform(X[['idade', 'salario', 'bonus']])

```

	idade	salario	bonus	cargo_Analista	cargo_Diretor	cargo_Fundador	cargo_Gerente	cargo_Programador
0	1.621622	0.535929	0.456376	0	1	0	0	0
1	-0.864865	-0.528963	-0.617450	1	0	0	0	0
2	0.000000	0.056417	-0.751678	0	0	0	0	1
3	-0.648649	-0.056417	0.080537	0	0	0	1	0
4	0.000000	1.268042	-0.080537	0	0	0	1	0

QuantileTransformer

Assim como o RobustScaler atua sobre as colunas e também trata os outliers com uso de quartis. Este método transforma os valores de tal forma que a distribuição tende a se aproximar de uma distribuição normal. Uma observação importante é que essa

transformação pode distorcer as correlações lineares entre as colunas. Neste método todos os valores serão reescalados em um intervalo de 0 a 1 de tal forma que os outliers não poderão mais ser distinguidos logo ao contrário do RobustScaler o impacto da ação em cima dos outliers será grande.

```
from sklearn.preprocessing import QuantileTransformer

X_quantile = X.copy()
X_quantile[['idade', 'salario', 'bonus']] =
QuantileTransformer().fit_transform(X[['idade', 'salario',
'bonus']])
```

	idade	salario	bonus	cargo_Analista	cargo_Diretor	cargo_Fundador	cargo_Gerente	cargo_Programador
0	0.846317	0.769231	8.078078e-01	0	1	0	0	0
1	0.115115	0.307375	1.921922e-01	1	0	0	0	0
2	0.538539	0.576577	1.000000e-07	0	0	0	0	1
3	0.230599	0.461513	5.765766e-01	0	0	0	1	0
4	0.538539	0.923156	4.615835e-01	0	0	0	1	0

PowerTransformer

Atua sobre as colunas e assim como o Quantile procura transformar os valores em uma distribuição mais normal, sendo indicado em situações onde uma distribuição normal é desejada para os dados, além disso esse método ainda suporta os métodos de transformação

Box-Cox (dataset com dados positivos) e Yeo-Johnson (dataset com dados positivos e negativos).

```
from sklearn.preprocessing import PowerTransformer

X_power = X.copy()
X_power[['idade', 'salario', 'bonus']] =
PowerTransformer().fit_transform(X[['idade', 'salario', 'bonus']])
```

	idade	salario	bonus	cargo_Analista	cargo_Diretor	cargo_Fundador	cargo_Gerente	cargo_Programador
0	1.143230	0.768685	0.775384	0	1	0	0	0
1	-1.075613	-0.711676	-0.943097	1	0	0	0	0
2	-0.120103	0.222110	-1.639884	0	0	0	0	1

3	-0.808699	0.072326	0.412848	0	0	0	1	0
4	-0.120103	1.428193	0.214291	0	0	0	1	0

Resumindo as opções de reescala

	Método	Quando usar	Observações
1	Normalizer	Quando a distribuição dos seus dados não é normal ou quando você não sabe qual é o tipo de distribuição dos seus dados.	Atua sobre as linhas/exemplos e não sobre as colunas/atributos
2	MinMaxScaler	Quando a distribuição dos dados não for normal e se o desvio padrão for pequeno	Não reduz de forma eficaz o impacto de outliers e também preserva a distribuição original. valor = (valor – Coluna.min) /
3	StandardScaler	Quando os dados estão com distribuição normal ou quando é necessário transformar os valores em uma distribuição mais normal	É uma boa combinação com algoritmos como Linear Regression e Logistic Regression
4	RobustScaler	Quando queremos reduzir o impacto de outliers	
5	QuantileTransformer	Quando queremos reduzir o impacto de outliers	Trata os outliers de uma forma mais agressiva do que o RobustScaler
6	PowerTransformer	Quando é necessário transformar os valores em uma distribuição mais normal	

	Método	Dados em distribuição normal	Dados não estão em distribuição normal	É desejado que os dados estejam em distribuição normal	É desejado eliminar a influência dos outliers
1	Normalizer	×	✓	×	×
2	MinMaxScaler	×	✓	×	×
3	StandardScaler	✓	✓	✓	×
4	RobustScaler				✓
5	QuantileTransformer				✓
6	PowerTransformer				

Observação sobre reescala: Não existe uma receita de bolo clara para a escolha da melhor forma de se reescalar os dados, portanto encorajamos que durante a solução do seu problema todas as alternativas acima sejam levadas em consideração e escolhida a forma que apresentar melhores resultados no seu contexto. Importante ressaltar que não se deve aplicar mais de um tipo de transformação ao mesmo tempo, uma vez que isso pode causar em perda total das suas informações originais!

Selecionando os melhores atributos

Agora que temos nossos dados reescalados em alguns datasets podemos contar com muitas colunas/atributos para nos auxiliar na tarefa de predição, porém nem sempre todos esses atributos possuem informações relevantes e muita das vezes podem levar o modelo a ter um resultado inferior do que se tivéssemos usados apenas poucos atributos. Portanto um trabalho importante é selecionar os atributos que mais fazem sentido e agregam valor em nossa solução. Para isso podemos contar com o **SelectKBest** do sklearn, onde o **K** representa o número máximo de atributos que desejamos ter em nosso dataset a ser “inputado” em nossa etapa de treino, dado o **K** o **SelectKBest** trata de encontrar os **K** melhores atributos a serem usados. Importante ressaltar que aqui não existe uma melhor maneira de se escolher o valor para **K** a solução é tentar com diferentes valores e compararmos os resultados. Uma observação importante é que o **SelectKBest** não suporta dados valores negativos, portanto todos os métodos de reescala que transformam os valores em um intervalo que possui negativos devem ser descartados caso você queira aplica-lo. Em nosso exemplo irei experimentar **K = 6** pois possuímos 8 atributos e irei utilizar o método de reescala **MinMaxScaler**, sinta-se à vontade para experimentar qualquer valor de **K** e qualquer método de reescala! O método **fit_transform** já nos retorna os valores dos atributos mais importantes por tanto podemos passar a usar o retorno como o nosso novo **X**

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
```

```
X_new = SelectKBest(chi2, k=6).fit_transform(X_minMax, Y)
```

```
array([[0.55555556, 0.60735921, 0.33333333, 0.        , 1.        ,
        0.        ],
       [0.04444444, 0.15625, 0.02702703, 1.        , 0.        ,
```



```
[0.04444444, 0.15862688, 0.03703704, 1.      , 0.      ,
 0.      ],
[0.22222222, 0.40529878, 0.      , 0.      , 0.      ,
 0.      ],
[0.08888889, 0.35775185, 0.22962963, 0.      , 0.      ,
 0.      ],
[0.22222222, 0.91586269, 0.18518519, 0.      , 0.      ,
 0.      ],
[0.04444444, 0.0829033 , 0.03703704, 0.      , 0.      ,
 0.      ],
[0.      , 0.40529878, 0.00740741, 1.      , 0.      ,
 0.      ],
_
```

Outra forma de escolhermos os melhores atributos é utilizando o **SelectFromModel** também do sklearn, o diferencial do SelectFromModel se dá na forma como ele escolhe os melhores atributos, uma vez que esta escolha está atrelada a importância de um atributo para um dado modelo, além disso podemos definir um valor limite(threshold) a partir do qual passamos a considerar um atributo como importante. Em nosso exemplo irei atrelar SelectFromModel com o classificador ExtraTreesClassifier. Ainda sobre o valor limite(threshold), podemos passar alguns valores no SelectFromModel ou não(None), caso não seja passado nada o limite utilizado por padrão será 1e-5, também podemos passar uma string “median” onde a média da importância dos atributos será o limite ou ainda podemos passar qualquer valor que desejamos. Aqui irei optar por passar “median”.

```
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.feature_selection import SelectFromModel

clf = ExtraTreesClassifier()
clf = clf.fit(X, y)
clf.feature_importances_ #Mostra a importância de cada atributo

model = SelectFromModel(clf, prefit=True, threshold="median")
X_new = model.transform(X_minMax)
```

```
array([[0.55555556, 0.60735921, 0.33333333, 1.      ],
       [0.04444444, 0.15862688, 0.03703704, 0.      ],
       [0.22222222, 0.40529878, 0.      , 0.      ],
       [0.08888889, 0.35775185, 0.22962963, 0.      ],
       [0.22222222, 0.91586269, 0.18518519, 0.      ],
       [0.04444444, 0.0829033 , 0.03703704, 0.      ],
       [0.      , 0.40529878, 0.00740741, 0.      ],
       [0.66666667, 0.43908459, 0.25925926, 1.      ],
       [1.      , 1.      , 1.      , 0.      ],
       [0.26666667, 0.13899484, 0.11111111, 0.      ],
       [0.33333333, 0.      , 0.22962963, 0.      ],
       [0.17777778, 0.06046668, 0.04444444, 0.      ],
       [0.17777778, 0.77563383, 0.40740741, 0.      ],
       [0.22222222, 0.3269015 , 0.33333333, 0.      ]])
```


Outra opção muito similar ao SelectKBest é o **SelectPercentile** a diferença se dá que no SelectPercentile iremos passar um atributo chamado *percentile* que irá selecionar a porcentagem indicada dos melhores atributos, por padrão esse valor é 10. Uma dica importante é que utilizamos o SelectKBest quando temos uma noção da quantidade de atributos que temos e a quantidade de atributos que queremos, já o SelectPercentile é mais utilizado quando não temos uma noção muito clara dos números de atributos.

PCA

Por fim podemos aplicar a técnica PCA nos atributos escolhidos, de forma grossa o PCA nada mais é que uma técnica a qual transforma atributos com uma certa correlação em um único atributo. O PCA deve ser aplicado apenas em casos em que o seu dataset possui muitas colunas, realmente um número muito grande e o treino do seu modelo acaba por ser muito demorado ou inviável devido ao alto número de colunas, uma vez que o PCA é uma técnica na qual sempre haverá perda de informações. Em nosso exemplo o PCA será aplicado de forma meramente ilustrativa uma vez que não possuímos muitas colunas. O atributo *n_components* representa quantos atributos queremos deixar, outra ideia é utilizar o parâmetro *n_components='mle'* dado isto Minka's MLE será utilizado para escolher a melhor dimensão a ser mantida no seu caso

```
from sklearn.decomposition import PCA
```

```
pca = PCA(n_components='mle')
X_new = pca.fit_transform(X_minMax)
```

```
array([[ 0.53866608, -0.13267348, -0.3357473 ,  0.79791559,  0.08466453,
        -0.05714549,  0.07021413],
       [-0.1368266 ,  0.93283828, -0.21270972, -0.16271694, -0.0028759 ,
         0.04533437,  0.01861439],
       [-0.60363558, -0.37644469,  0.01636196, -0.04850587,  0.07254725,
        -0.23334703, -0.09461641],
       [ 0.20186796,  0.18208778,  0.90457672,  0.15443852, -0.06257746,
         0.26362392,  0.03431419],
       [ 0.4695611 ,  0.08126605,  0.95831876,  0.12838664, -0.03621429,
        -0.22627592, -0.0374659 ],
       [-0.78274982, -0.29609104,  0.01031809, -0.04706371,  0.11946656,
         0.05719386,  0.01365582],
       [-0.0501463 ,  0.91612012, -0.16114226, -0.18467334,  0.08231014,
        -0.16703018,  0.03887834],
       [ 0.47351528, -0.11871599, -0.38245359,  0.83764906,  0.01517078,
         0.06756885, -0.06232372],
       [ 1.21379905, -0.62653624, -0.30107854, -0.5921759 , -0.31002738,
        -0.02640987, -0.0082365 ],
       [-0.0615711 ,  0.86322067, -0.28071454, -0.1471028 , -0.18805305,
```

```
[ 0.06181369, -0.0542862 ],
[-0.6810227 , -0.40144957, -0.09634357, -0.03739642, -0.17453742,
 0.14340386, -0.01148692],
[-0.7534116 , -0.32754108, -0.02838242, -0.03074252,  0.02326011,
 0.07001957, -0.05915848],
[ 0.70951022, -0.25128851, -0.06224468, -0.57218556,  0.46819143,
 0.11267141, -0.00293375],
[-0.53755599, -0.4447923 , -0.02875891, -0.09582675, -0.09132531,
 -0.11142105,  0.15483102]])
```

Separando os seus dados em um conjunto de treino e de teste

Uma das boas práticas no desenvolvimento de modelos é separar o seu dataset um conjunto de treino para treinar o seu modelo e outro de teste para validar os resultados do seu modelo. Essa separação é crucial a fim de identificar se o seu modelo não está “decorando” os resultados ao invés de aprender, o famoso overfitting.

Vamos começar com o método Train Test Split do sklearn

Neste exemplo irei continuar utilizando os atributos da variável **X** reescalados com o **MinMaxScaler**. Aqui iremos separar em 4 subconjuntos, **X_train** com as variáveis de input para treino do modelo e **Y_train** com os resultados de predição para treino, ambos serão utilizados mais tarde no método fit do seu modelo, **X_test** com as variáveis de input para test e **Y_test** com os resultados da predição para testes, o **X_test** será utilizado no método predict do seu modelo e o **Y_test** será utilizado para comparar com resultado retornado do seu método predict. O parâmetro **test_size** serve para indicar que o nosso conjunto de teste terá um tamanho de 20% em relação ao tamanho do dataset e o parâmetro **random_state** nos permite garantir que a separação dos conjuntos de teste e treino será sempre o mesmo, com a “semente” de aleatoriedade “semeada” sempre no valor indicado.

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X_minMax, Y,
test_size = 0.2, random_state = 0)
```

In [90]: X_train

Out[90]:

	idade	salario	bonus	cargo_Analista	cargo_Diretor	cargo_Fundador	cargo_Gerente	cargo_Programador
11	0.177778	0.060467	0.044444	0	0	0	0	1
2	0.222222	0.405299	0.000000	0	0	0	0	1
13	0.222222	0.326902	0.333333	0	0	0	0	1
9	0.266667	0.138995	0.111111	1	0	0	0	0

1	0.044444	0.158627	0.037037	1	0	0	0	0
7	0.666667	0.439085	0.259259	0	1	0	0	0
10	0.333333	0.000000	0.229630	0	0	0	0	1
3	0.088889	0.357752	0.229630	0	0	0	1	0
0	0.555556	0.607359	0.333333	0	1	0	0	0
5	0.044444	0.082903	0.037037	0	0	0	0	1
12	0.177778	0.775634	0.407407	0	0	1	0	0

In [91]: X_test

Out[91]:

	idade	salario	bonus	cargo_Analista	cargo_Diretor	cargo_Fundador	cargo_Gerente	cargo_Programador
8	1.000000	1.000000	1.000000	0	0	1	0	0
6	0.000000	0.405299	0.007407	1	0	0	0	0
4	0.222222	0.915863	0.185185	0	0	0	1	0

Outra forma de fazemos essa separação é com o KFold do sklearn

Esta é a minha forma favorita de separar o dataset, pois o KFold facilita que possamos separar o nosso conjunto de dados em conjunto de treinos e teste K vezes (indicado no parâmetro *n_splits*) e portanto fazer o treino e validar os resultados para 3 conjuntos diferentes de treino e teste. Antes precisamos colocar a nossa variável X em um formato de array.

```
X_new = X_minMax.values

from sklearn.model_selection import KFold

kf = KFold(n_splits=2)

time = 1
for train_index, test_index in kf.split(X_new):
    print("K = " + str(time) + " - TRAIN:", train_index, "TEST:",
          test_index)
    X_train, X_test = X_new[train_index], X_new[test_index]
    y_train, y_test = Y[train_index], Y[test_index]
    time += 1
```

```
K = 1 - TRAIN: [ 7  8  9 10 11 12 13] TEST: [0 1 2 3 4 5 6]
K = 2 - TRAIN: [0 1 2 3 4 5 6] TEST: [ 7  8  9 10 11 12 13]
```

Para finalizarmos é importante termos em mente que é de extrema importância que saibamos explorar os parâmetros disponíveis de cada método e classe utilizados no

processo de pré-processamento utilizando a documentação do Sklearn que é incrível! Alguns ajustes de parâmetros podem trazer grandes ganhos para a sua solução, porém apenas com testes será possível encontrar os melhores parâmetros pois cada dataset tem suas peculiaridades, assim como cada problema e sua solução!

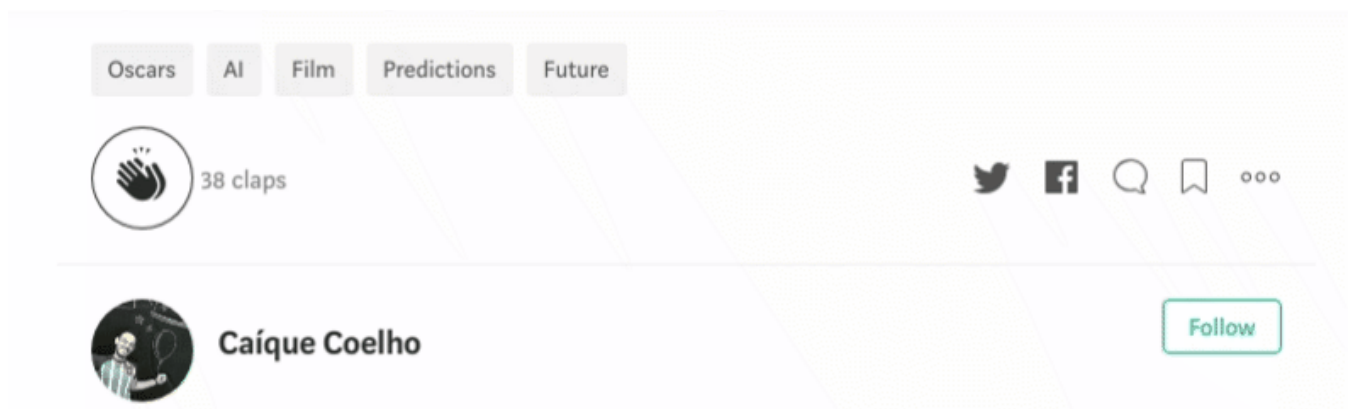
Para mais detalhes acesso o projeto no github e explore o arquivo python e o jupyter notebook:

CaiqueCoelho/Preprocessing-Dataset-Template A template to preprocessing your golden dataset before to put your data in your best model ... github.com	
--	--

. . .

Referências


<https://towardsdatascience.com/the-complete-beginners-guide-to-data-cleaning-and-preprocessing-2070b7d4c6d?gi=c5101f25cd16>




AI Machine Learning Data Preprocessing Classification Data Science

About Help Legal

Get the Medium app

 A button that says 'Download on the App Store', and if clicked it will lead you to the iOS App store

 A button that says 'Get it on, Google Play', and if clicked it will lead you to the Google Play store