EABD-Preparacao-Dados

April 24, 2025

1 Aspectos de Preparação de Dados

1.1 Objetivos de Aprendizagem

- Entender o impacto da preparação de dados na análise.
- Identificar problemas em dados (faltantes, repetidos, discrepantes, padronização).
- Aplicar técnicas básicas para tratar esses problemas.
- Utilizar Pandas para limpar e transformar dados.

1.2 Introdução

A qualidade dos dados e a quantidade de informação útil que eles contêm são fatores-chave que determinam quão bem um algoritmo de aprendizado de máquina pode aprender. Portanto, é absolutamente crítico garantir que examinemos e pré-processemos um conjunto de dados antes de alimentá-lo a um algoritmo de aprendizado. Discutiremos as técnicas essenciais de pré-processamento de dados que nos ajudarão a construir bons modelos de aprendizado de máquina.

1.3 Lidando com dados ausentes

É comum em aplicações do mundo real que nossos exemplos de treinamento estejam faltando um ou mais valores por diversos motivos. Pode ter havido um erro no processo de coleta de dados, algumas medições podem não ser aplicáveis ou campos específicos simplesmente deixados em branco em uma pesquisa. Normalmente, vemos valores ausentes como espaços em branco na nossa tabela de dados ou como strings de marcador como NaN (que significa "não é um número") ou NULL (um indicador comumente usado de valores desconhecidos em bancos de dados relacionais). Infelizmente, a maioria das ferramentas computacionais não consegue lidar com esses valores ausentes ou produzirá resultados imprevisíveis se simplesmente os ignorarmos. Portanto, é crucial que cuidemos desses valores ausentes antes de prosseguirmos com análises adicionais.

Vamos trabalhar com várias técnicas práticas para lidar com valores ausentes, removendo entradas do nosso conjunto de dados ou imputando valores ausentes de outros exemplos de treinamento e recursos.

1.3.1 Identificando valores ausentes em dados tabulares

Antes de discutir várias técnicas para lidar com valores ausentes, vamos criar um dataframe simples a partir de um arquivo CSV (valores separados por vírgula) para ter uma melhor compreensão do problema. Além do Pandas, usamos o módulo IO, que fornece as principais facilidades do Python para lidar com vários tipos de I/O. Existem três tipos principais de I/O: I/O de texto, I/O binário e I/O bruto. O I/O de texto espera e produz objetos *str*; o módulo StringIO é um objeto semelhante

a arquivo na memória que pode ser usado como entrada ou saída para a maioria das funções que esperariam um objeto de arquivo padrão.

[ ]:
```
import pandas as pd
from io import StringIO
import sys

csv_data = \
'''A,B,C,D
1.0,2.0,3.0,4.0
5.0,6.0,,8.0
10.0,11.0,12.0,'''

# If you are using Python 2.7, you need
# to convert the string to unicode:

if (sys.version_info < (3, 0)):
    csv_data = unicode(csv_data)

df = pd.read_csv(StringIO(csv_data))
df
```

Lemos dados formatados em CSV para um pandas DataFrame usando a função `read_csv` e notamos que as duas células ausentes foram substituídas por NaN. A função `StringIO` no exemplo de código anterior foi usada apenas para fins ilustrativos. Ela nos permite ler a string atribuída a `csv_data` em um pandas DataFrame como se fosse um arquivo CSV regular em nosso disco.

Podemos usar o método `isnull` para retornar um DataFrame com valores booleanos que indicam se uma célula contém um valor numérico (False) ou se os dados estão faltando (True). Usando o método `sum`, podemos então retornar o número de valores ausentes por coluna como segue.

[ ]:
```
df.isnull().sum()
```

1.3.2 Eliminando exemplos de treinamento ou recursos com dados ausentes

Uma das maneiras mais fáceis de lidar com dados ausentes é simplesmente remover os recursos (colunas) ou exemplos de treinamento (linhas) correspondentes do conjunto de dados por completo; linhas com valores ausentes podem ser facilmente removidas usando o método `dropna`:

[ ]:
```
df.dropna(axis=0)
```

Da mesma forma, podemos remover colunas que tenham pelo menos um NaN em qualquer linha definindo o argumento `axis` como 1:

[ ]:
```
df.dropna(axis=1)
```

Podemos também definir um threshold para a remoção de linhas. No exemplo abaixo, as linhas com menos to que 4 valores preenchidos serão removidas:

[ ]:
```
df.dropna(thresh=4)
```

Embora a remoção de dados ausentes pareça uma abordagem conveniente, ela também tem desvantagens: podemos remover amostras demais, tornando uma análise confiável impossível.

Se removermos muitas colunas (ou as erradas), corremos o risco de perder informações valiosas.

Como alternativa, por exemplo, em um conjunto de dados de filmes com uma coluna para duração, se a duração de alguns filmes for desconhecida, podemos substituí-la pela média da duração dos filmes para os quais ela é conhecida.

1.3.3 Imputando valores ausentes

Em vez da remoção de linhas ou da eliminação de colunas inteiras, podemos usar diferentes técnicas de interpolação para estimar os valores ausentes a partir dos outros exemplos de treinamento em nosso conjunto de dados.

Uma das técnicas de interpolação mais comuns é a imputação por média, onde simplesmente substituímos o valor ausente pela média do valor da coluna de recursos inteira. Uma maneira conveniente de fazer isso é usando a classe `SimpleImputer` da scikit-learn, como mostrado no código a seguir:

[ ]:
```
from sklearn.impute import SimpleImputer
import numpy as np

imr = SimpleImputer(missing_values=np.nan, strategy='mean')
imr = imr.fit(df.values)
imputed_data = imr.transform(df.values)
imputed_data
```

Substituímos cada valor NaN pela média correspondente, que é calculada separadamente para cada coluna.

Outras opções para o parâmetro strategy são `median` ou `most_frequent`, onde o último substitui os valores ausentes pelos valores mais frequentes, respectivamente. Isso é útil para imputar valores de recursos categóricos, por exemplo, uma coluna de recursos que armazena uma codificação de nomes de cores, como vermelho, verde e azul.

Alternativamente, uma maneira ainda mais conveniente de imputar valores ausentes é usando o método `fillna` do pandas e fornecendo um método de imputação como argumento. Por exemplo, podemos alcançar a mesma imputação por média diretamente no objeto DataFrame através do seguinte comando:

[ ]:
```
df.fillna(df.mean())
```

2 Removendo Valores

Utilizaremos técnicas estatísticas para identificar valores anormais (i.e. raros ou errados); esses dados poderão ser removidos ou selecionados para correção (veremos imputação mais adiante)

2.1 IQR (interquartil-range)

Podemos usar as medidas de dispersão e distribuição para avaliar e encontrar amostras que estão muito distantes do esperado. Uma dar formas é o intervalo interquartil $IQR = (Q3 - Q1)$.

O interquartil separa metade dos dados que estão na parte central, i.e. de 25% à 75%. Podemos pensar que valores que se afastam em uma proporção dessa amplitude são raros ou ruído.

Considere o dataset abaixo:

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from io import StringIO
import sys

csv_data = \
'''0,24,21
1,30,27
2,26,24
3,24,21
4,27,22
5,24,23
6,25,23
7,26,23
8,23,21
9,29,21
10,23,21
11,29,26
12,27,24
13,21,19
14,22,25
15,25,22
16,21,20
17,25,22
18,20,18
19,24,21
20,18,17
21,22,20
22,17,16
23,20,19
0,29,28
1,31,30
2,31,30
3,24,22
4,25,20
5,30,28
6,23,21
7,23,21
8,25,24
9,24,21
10,23,22
11,26,25
```

```
12,23,20
13,28,26
14,26,10
15,26,24
16,50,21
17,27,26
18,23,22
19,18,16
20,22,20
21,18,17
22,19,17
23,21,20
0,32,31
1,31,28
2,26,42
3,30,28
4,13,29
5,28,25
6,28,25
7,23,21
8,25,22
9,22,21
10,22,21
11,22,21
12,25,21
13,29,25
14,23,21
15,27,24
16,19,17
17,10,22
18,18,17
19,25,2
20,18,15
21,20,19
22,20,17
23,19,17
0,27,25
1,25,22
2,25,22
3,27,25
4,24,55
5,29,26
6,80,24
7,27,24
8,25,12
9,26,7
10,23,20
```

```
11,48,24
12,21,18
13,28,27
14,28,24
15,24,22
16,22,19
17,22,19
18,27,25
19,23,20
20,25,24
21,17,15
22,21,20
23,24,23
0,30,26
1,24,20
2,31,29
3,35,27
4,30,26
5,30,26
6,30,26
7,28,24
8,26,24
9,26,22
10,25,22
11,28,25
12,26,18
13,29,27
14,20,19
15,20,17
16,20,17
17,21,20
18,24,22
19,24,22
20,19,18
21,20,23
22,21,40
23,25,24
0,40,25
1,29,26
2,28,26
3,31,23
4,24,23
5,30,26
6,23,19
7,26,23
8,27,24
9,23,21
```

```
10,28,25
11,29,26
12,29,27
13,21,19
14,23,21
15,28,25
16,19,17
17,22,18
18,22,16
19,23,21
20,21,18
21,26,25
22,23,20
23,19,18
0,30,29
1,33,31
2,29,26
3,25,21
4,29,27
5,24,21
6,29,28
7,23,22
8,27,25
9,30,27
10,23,21
11,29,28
12,25,24
13,27,25
14,27,26
15,29,28
16,27,26
17,24,21
18,25,22
19,24,23
20,27,26
21,21,20
22,19,16
23,22,21
0,24,22
1,28,27
2,31,28
3,26,24
4,30,27
5,31,28
6,31,28
7,30,29
8,23,19
```

```
9,26,23
10,29,26
11,29,26
12,28,26
13,22,18
14,22,19
15,25,23
16,22,20
17,27,23
18,22,21
19,18,15
20,24,22
21,21,18
22,20,18
23,26,25
0,32,31
1,31,30
2,32,29
3,30,27
4,27,24
5,25,23
6,29,28
7,24,22
8,27,25
9,30,29
10,24,23
11,22,18
12,30,28
13,26,25
14,21,19
15,27,25
16,21,18
17,25,21
18,21,19
19,26,25
20,26,25
21,23,22
22,21,17
23,26,23'''

# If you are using Python 2.7, you need
# to convert the string to unicode:

if (sys.version_info < (3, 0)):
    csv_data = unicode(csv_data)

vendas = pd.read_csv(StringIO(csv_data))
```

```
vendas.columns=['hora','prod1','prod2']
vendas[['prod1','prod2']].describe()
```

Por exemplo, vemos que para o produto 1, temos entre 10 e 80 vendas por hora; mas em 50% das horas amostradas, temos entre 22 e 28 vendas. É intuitivo pensar que se na metade das horas as vendas oscilam em 6 pontos (nosso IQR), valores que se distanciem em 6 pontos do Q1 e do Q3 sejam bem mais raros. Em aplicações reais tendemos a ser ainda mais permissivos na nossa margem, **multiplicamos o IQR por 1.5** e adicionamos ou subtraímos de Q3 e Q1 (respectivamente) para definir a nossa região de dados 'comuns'. Esse valor é tão usado que é pradrão em um plot estatístico, o **boxplot**

'> (Q3 + IQR*1.5) - Outlier

'< (Q1 - IQR*1.5) - Outlier

[ ]:
```
vendas.boxplot()
plt.show()
```

[ ]:
```
#Calculando IQR e margem 'válida'
IQR = vendas[['prod1','prod2']].quantile(0.75) - vendas[['prod1','prod2']].
 ↪quantile(0.25)
margemMin = vendas[['prod1','prod2']].quantile(0.25) - IQR*1.5
margemMax = vendas[['prod1','prod2']].quantile(0.75) + IQR*1.5

#Filtrando valores anormais/outliers
idx=[]
for col in ['prod1','prod2']:
    filtered = vendas[col][(vendas[col] < margemMin[col]) | (vendas[col] >␣
 ↪margemMax[col])]
    idx.extend(filtered.index)
vendas.iloc[idx]
```

Alta distância para a média em termos de desvios padrões Ainda sob essa perspectiva de usar alguma medida de amplitude para medir valores anormais. Podemos utilizar o desvio padrão. Sabemos que em amostras simétricas temos 68% das amostras dentro de 1 desvio padrão da média, e quando aumentamos essa margem para 3 desvios padrões temos cerca de 99.7%. Iremos filtrar fora todas as amostras que estão a 3 desvios padrões da média, para mais ou para menos

[ ]:
```
#Calculando IQR e margem 'válida'
deviation = vendas[['prod1','prod2']].std()
margemMin = vendas[['prod1','prod2']].mean() - 3*deviation
margemMax = vendas[['prod1','prod2']].mean() + 3*deviation

#Filtrando valores anormais/outliers
idx=[]
for col in ['prod1','prod2']:
    filtered = vendas[col][(vendas[col] < margemMin[col]) | (vendas[col] >␣
 ↪margemMax[col])]
```

```
    idx.extend(filtered.index)
vendas.iloc[idx]
```

[ ]:
```
#vendas.info() #nossa base original possui 216 amostras
vendas.iloc[idx].info() #dessas 216, 8 estão muito fora da média ~ 3,7%
```

Dicussões e reflexões: - Notamos que cada abordagem se comporta de uma maneira diferente. A IQR se preocupa com a amplitude, independente de assimetria ou desvio padrão; enquanto que a baseada em 3 desvios padrões tem uma interpretação muito boa para distribuições normais - Sempre serão usados esse limiares (i.e. 3 desvios e 1.5 vezes o IQR)?

2.2 Trabalhando com Dados Categóricos

Até agora, trabalhamos apenas com valores numéricos. No entanto, é comum que conjuntos de dados do mundo real contenham uma ou mais colunas de valores categóricos.

Exemplos de dados categóricos ou variáveis categóricas:

- Informações demográficas de uma população: gênero, status da doença.
- O tipo sanguíneo de uma pessoa: A, B, AB ou O.

Quando falamos sobre dados categóricos, precisamos distinguir entre **ordinais** e **nominais**. Dados ordinais podem ser entendidos como valores categóricos que podem ser ordenados. Por exemplo, o tamanho da camiseta seria um dado ordinal, porque podemos definir uma ordem: XL > L > M. Em contraste, os dados nominais não implicam nenhuma ordem e, continuando com o exemplo anterior, poderíamos pensar na cor da camiseta como um recurso nominal.

[ ]:
```
import pandas as pd

df = pd.DataFrame([['green', 'M', 10.1, 'class2'],
                   ['red', 'L', 13.5, 'class1'],
                   ['blue', 'XL', 15.3, 'class2']])

df.columns = ['color', 'size', 'price', 'classlabel']
df
```

Como podemos ver, o novo DataFrame contém um dado nominal (cor), um dado ordinal (tamanho) e uma coluna numérica (preço). As etiquetas (labels) de classe (assumindo que criamos um conjunto de dados para uma tarefa de aprendizado supervisionado) são armazenadas na última coluna.

2.2.1 Mapeamento de dados ordinais

Para garantir que um modelo interprete corretamente os dados ordinais, precisamos converter os valores categóricos da string em inteiros.

No seguinte exemplo simples, vamos supor que sabemos a diferença numérica entre os recursos, por exemplo, XL = L + 1 = M + 2:

[ ]:
```
size_mapping = {'XL': 3,
                'L': 2,
                'M': 1}
```

```
df['size'] = df['size'].map(size_mapping)
df
```

2.2.2 Codificação das etiquetas de classe

Para codificar as etiquetas (labels) de classe, podemos usar uma abordagem semelhante ao mapeamento de recursos ordinais discutido anteriormente. Precisamos lembrar que as etiquetas de classe não são ordinais e não importa qual número inteiro atribuímos a uma determinada etiqueta de string. Assim, podemos simplesmente enumerar as etiquetas de classe, começando em 0:

[ ]:
```
import numpy as np

# cria um dicionário de mapeamento
# para converter as etiquetas de classe de strings para inteiros
class_mapping = {label: idx for idx, label in enumerate(np.
 ↪unique(df['classlabel']))}
# para converter as etiquetas de classe de strings para inteiros
df['classlabel'] = df['classlabel'].map(class_mapping)
df
```

2.2.3 One-Hot Encoding: Transformando Categorias em Números

Em muitos problemas de aprendizado de máquina, os dados precisam ser preparados adequadamente. Quando você tem variáveis categóricas (como cores, tipos de produtos ou regiões), o algoritmo precisa entender essas categorias como números. O one-hot encoding é uma técnica que resolve isso.

Basicamente, ele transforma cada categoria em uma nova coluna binária (0 ou 1). Uma linha representa um exemplo e, para cada categoria, a coluna correspondente terá um '1' se o exemplo pertencer àquela categoria e '0' caso contrário.

[ ]:
```
# one-hot encoding via pandas
pd.get_dummies(df[['price', 'color', 'size']])
```

3 Feature Scaling: Normalizando Seus Dados para o Sucesso

Em muitos algoritmos de aprendizado de máquina, as características (features) dos seus dados podem ter escalas muito diferentes. Por exemplo, a idade de uma pessoa pode variar de 20 a 80 anos, enquanto o número de filhos pode variar de 0 a 10. Essa disparidade de escalas pode prejudicar o desempenho de alguns algoritmos, especialmente aqueles baseados em distância (como k-NN ou regressão linear).

Considere o dataset abaixo:

[ ]:
```
import pandas as pd
import numpy as np

# Criando o DataFrame
```

```
data = {'Idade': [25, 30, 45, 60, 38],
        'Renda': [50000, 75000, 120000, 90000, 62000]}

df = pd.DataFrame(data)
df
```

Feature Scaling é uma técnica que visa transformar todas as características para terem uma escala semelhante. Existem duas abordagens principais:

- **Min-Max Scaling:** Escala os valores para um intervalo entre 0 e 1.
- **Standardization (Z-Score):** Transforma os valores para ter média 0 e desvio padrão 1.

Ao aplicar o feature scaling, você garante que nenhuma característica domine as outras durante o treinamento do modelo, melhorando a convergência e precisão dos resultados.

Em resumo, o Feature Scaling é um passo importante na preparação de dados para aprendizado de máquina, garantindo que todos os recursos contribuam igualmente para o processo de modelagem.

Para "escalar" nossos dados, podemos simplesmente aplicar a escala Min-Max a cada coluna de recurso, onde o novo valor $x^{(i)}_{norm}$ do exemplo $x^{(i)}$ pode ser calculado da seguinte forma:

$$x^{(i)}_{norm} = \frac{x^{(i)} - x_{min}}{x_{max} - x_{min}}$$

Aqui, x_{min} é o menor valor na coluna de recurso e x_{max} é o maior.

[ ]:
```
from sklearn.preprocessing import MinMaxScaler

mms = MinMaxScaler()
df_scaled = mms.fit_transform(df)
df_scaled
```

Já o Z-Score transforma os valores de uma característica, centralizando-os em torno de zero e escalando-os com base no desvio padrão. A fórmula é:

$$x^{(i)}_{scaled} = \frac{x^{(i)} - \mu}{\sigma}$$

Onde:

- $x^{(i)}_{scaled}$ é o valor normalizado da característica i.
- $x^{(i)}$ é o valor original da característica i.
- (mu) é a média (média aritmética) de todos os valores da característica.
- (sigma) é o desvio padrão de todos os valores da característica.

Em resumo, subtraímos a média da característica e dividimos pelo seu desvio padrão, resultando em um valor que indica quantos desvios padrão um ponto está da média. Isso ajuda a remover o impacto de diferentes escalas e permite que algoritmos mais sensíveis à escala funcionem corretamente.

[ ]:
```
from sklearn.preprocessing import StandardScaler
```

```
std = StandardScaler()
df_scaled = std.fit_transform(df)
df_scaled
```