Engenharia de recursos

March 27, 2025

[1]:
```
from IPython.display import Image
```

1 ENGENHARIA DE RECURSOS

2 Dimensionamento de Atributos

É uma técnica para **padronizar** os atributos independentes presentes nos dados para um intervalo fixo. Ela é realizada durante o pré-processamento dos dados.

2.1 Funcionamento:

Dado um conjunto de dados com as seguinte atributos: Idade, Salário e Apartamento BHK, com um tamanho de 5000 pessoas, cada uma apresentados esses atributos de forma independente.

Cada ponto de dados é rotulado como:

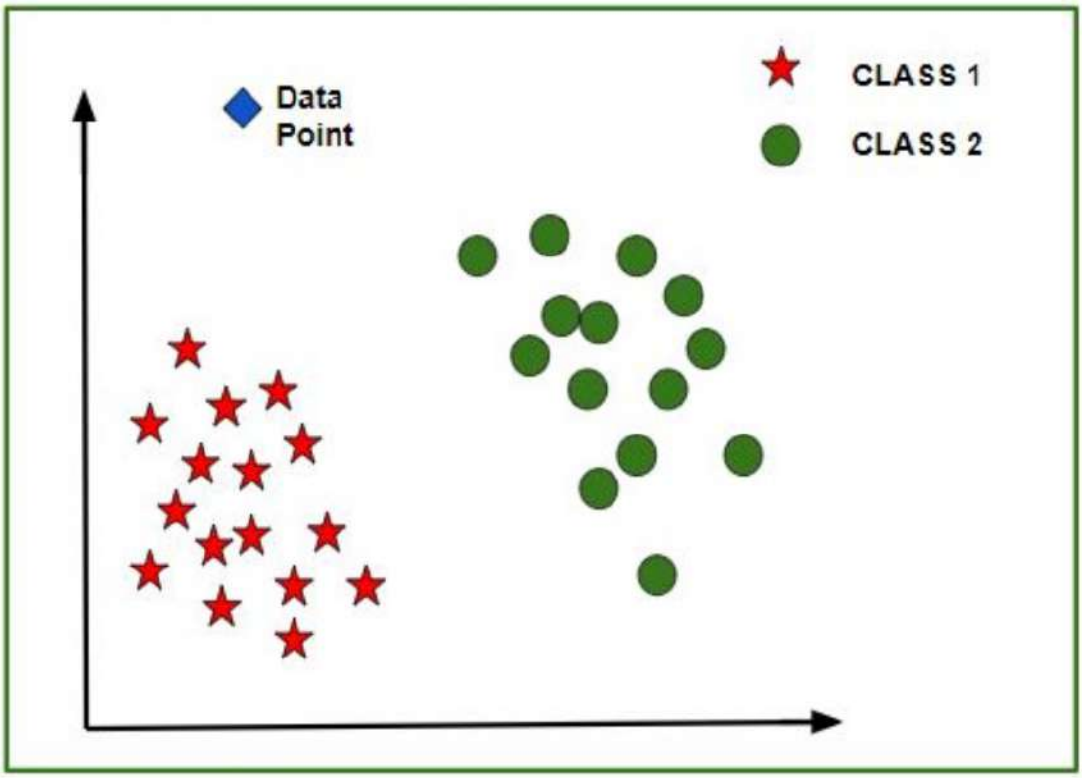
- **Classe 1** - SIM: Significa que, com a idade, salário e número de comodos, é possível comprar o imóvel.
- **Classe 2** - NÃO: Significa que, com a idade, salário e número de comodos, não é possível comprar o imóvel.

Usando esse conjunto de dados para treinar o modelo, pretende-se construir um modelo que possa prever se alguém pode comprar um imóvel ou não com determinados valores de características.

Uma vez que o modelo é treinado, pode-se criar um gráfico N-dimensional (onde N é o número de recursos presentes no conjunto de dados) com pontos de dados do conjunto fornecido. A figura abaixo é uma representação ideal do modelo.

[2]:
```
Image('Feature.png')
```

[2]:



Conforme mostrado na figura, os pontos de dados em forma de estrela pertencem à Classe 1 – SIM, e os círculos representam os rótulos da Classe 2 – NÃO.

O modelo é treinado usando esses pontos de dados. Agora, um novo ponto de dados (diamante, conforme mostrado na figura) é fornecido e tem diferentes valores independentes para os 3 recursos (Idade, Salário, número de comodos) mencionados acima.

O modelo precisa prever se esse ponto de dados pertence a SIM ou NÃO.

2.2 Previsão da Classe para novos pontos:

O modelo calcula a distância entre o novo ponto de dados e o centroide de cada grupo de classes.

Finalmente, esse ponto de dados será atribuído à classe cujo centroide estiver mais próximo.

A distância pode ser calculada entre o centroide e o ponto de dados usando os seguintes métodos:

Distância Euclidiana: É a raiz quadrada da soma dos quadrados das diferenças entre as coordenadas (valores das características – Idade, Salário, número de comodos) do ponto de dados e do centroide de cada classe. Esta fórmula é baseada no teorema de Pitágoras.

$$d(x,y) = \sqrt[2]{\sum_{k=1}^{n} (x_k - y_k)^2} \tag{1}$$

onde: - (x) é o valor do dos atributos, - (y) é o valor do Centroide e - (k) é o número de atributos.

Distância de Manhattan: É calculada como a soma das diferenças absolutas entre as coordenadas (valores das características) do ponto de dados e do centroide de cada classe.

$$d(x, y) = \sum_{k=1}^{n} \mid x_k - y_k \mid \tag{2}$$

2.2.1 Necessidade de dimensionamento de atributos

O conjunto de atributos tem os seguintes recursos: - Idade intervalo de 10 até 60 anos - Salário anual de 1 milhão até 40 milhões - Número de comodos de 1 até 5 comodos

Nota-se que o atributo salário dominará todos os outros recursos para previsão da classe, dados que todos os atributos são independentES. Ou seja o salário não depende da idade nem do número de comodos da casa. Assim se os dados continuarem como estão as previsões sempre serão erradas, visto que um atributo é dominante sobre os outros.

2.2.2 Exercicios:

Suponha que o centroide da classe 1 seja [40, 22000000, 3] e o ponto de dados a ser previsto seja [57, 33000000, 2]. Calcule a distância utilizando: - a distância Euclidiana - a distância Manhattan

2.3 Engenharia de Atributos: dimensionamento, normalização e padronização

Dimensionamento de atributos é uma técnica para padronizar os recursos independentes presentes nos dados em um intervalo fixo.

É realizada durante o pré-processamento de dados para lidar com magnitudes, valores ou unidades altamente variáveis.

Se o dimensionamento de recursos não for feito, um algoritmo de machine learning tende a pesar mais os valores maiores e considerar os valores menores como os mais baixos, independentemente da unidade dos valores.

2.3.1 Por que usar o Dimensionamento de atributos?

No aprendizado de máquina, o dimensionamento de atributos é empregado para diversas finalidades:

1. **Normalização de Recursos**: O dimensionamento garante que todos os recursos estejam em uma escala comparável e tenham intervalos comparáveis. Isso é significativo porque a magnitude dos recursos tem um impacto em muitas técnicas de aprendizado de máquina. Recursos de escala maior podem dominar o processo de aprendizado e ter um impacto excessivo nos resultados. Dimensionando os recursos, você pode evitar esse problema e garantir que cada recurso contribua igualmente para o processo de aprendizado.
2. **Melhoria do Desempenho do Algoritmo**: Quando os recursos são dimensionados, vários métodos de aprendizado de máquina, incluindo algoritmos baseados em descida de gradiente, algoritmos baseados em distância (como k-vizinhos mais próximos) e máquinas de vetor de suporte, têm melhor desempenho ou convergem mais rapidamente. O desempenho do algoritmo pode ser aprimorado dimensionando os recursos, o que pode acelerar a convergência do algoritmo para o resultado ideal.

3. **Prevenção de Instabilidade Numérica**: A instabilidade numérica pode ser prevenida evitando disparidades significativas de escala entre recursos. Exemplos incluem cálculos de distância ou operações de matriz, onde ter recursos com escalas radicalmente diferentes pode resultar em problemas de estouro ou subfluxo numérico.

4. Os atributos com mesmo peso garantem que cada recurso receba o mesmo peso durante o processo de aprendizado. Sem os pesos, recursos de escala maior podem dominar o aprendizado, produzindo resultados distorcidos. Esse viés é removido por meio do redimensionamento de atributos, o que também garante que cada recurso contribua de forma justa para as previsões do modelo.

2.3.2 Escala Máxima Absoluta

Este método de dimensionamento requer duas etapas:

1. Selecione o valor absoluto máximo entre todas as entradas de uma medida específica.
2. Depois disso, dividimos cada entrada da coluna por esse valor máximo.

$$X_{scaled} = \frac{X_i - max(|X|)}{max(|X|)} \tag{3}$$

Após executar os dois passos acima mencionados, observaremos que cada entrada da coluna está no intervalo de -1 a 1. Mas esse método não é usado com tanta frequência, pois é muito **sensível aos outliers**. E ao lidar com dados do mundo real, a presença de outliers é algo muito comum.

Para fins de demonstração, usaremos um conjunto de dados que será disponibilizado. Este conjunto de dados é uma versão mais simples do conjunto de dados original de previsão de preço de casas, tendo apenas duas colunas do conjunto de dados original. As primeiras cinco linhas dos dados originais são mostradas abaixo:

[10]:
```
import pandas as pd
df = pd.read_csv('Amostra.csv')
print(df.head())
```

```
   LotArea  MSSubClass
0     8450          60
1     9600          20
2    11250          60
3     9550          70
4    14260          60
```

Agora vamos aplicar o primeiro método que é o da escala máxima absoluta. Para isso, primeiro, devemos avaliar os valores máximos absolutos das colunas.

[13]:
```
import numpy as np
max_vals = df.max()
max_vals
```

[13]:
```
LotArea       215245
MSSubClass       190
```

```
dtype: int64
```

Deve-se subtrair esses valores dos dados e depois dividir os resultados dos valores máximos também.

[14]:
```
print((df - max_vals) / max_vals)
```

```
       LotArea  MSSubClass
0    -0.960742   -0.684211
1    -0.955400   -0.894737
2    -0.947734   -0.684211
3    -0.955632   -0.631579
4    -0.933750   -0.684211
...        ...         ...
1455 -0.963219   -0.684211
1456 -0.938791   -0.894737
1457 -0.957992   -0.631579
1458 -0.954856   -0.894737
1459 -0.953834   -0.894737

[1460 rows x 2 columns]
```

2.3.3 Escala Min-Max

Este método de dimensionamento requer as duas etapas abaixo:

1. Encontrar o valor mínimo e o valor máximo da coluna.
2. Subtrair o valor mínimo da entrada e dividiremos o resultado pela diferença entre o valor máximo e o mínimo.

$$X_{scaled} = \frac{X_i - X_{min}}{X_{max} - X_{min}} \tag{4}$$

Como estamos usando o valor máximo e o mínimo, esse método também é propenso à sofre influência de outliers, mas o intervalo no qual os dados podem variar é entre 0 e 1.

[18]:
```
import numpy as np
max_vals = df.max()
max_vals
```

[18]:
```
LotArea       215245
MSSubClass       190
dtype: int64
```

[19]:
```
min_vals=df.min()
min_vals
```

[19]:
```
LotArea       1300
MSSubClass      20
dtype: int64
```

[22]:
```
print((df - min_vals) / (max_vals - min_vals))
```

```
       LotArea  MSSubClass
0     0.033420    0.235294
1     0.038795    0.000000
2     0.046507    0.235294
3     0.038561    0.294118
4     0.060576    0.235294
...        ...         ...
1455  0.030929    0.235294
1456  0.055505    0.000000
1457  0.036187    0.294118
1458  0.039342    0.000000
1459  0.040370    0.000000

[1460 rows x 2 columns]
```

[15]:
```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

scaled_data = scaler.fit_transform(df)

scaled_df = pd.DataFrame(scaled_data,
                         columns=df.columns)

scaled_df.head()
```

[15]:

	LotArea	MSSubClass
0	0.033420	0.235294
1	0.038795	0.000000
2	0.046507	0.235294
3	0.038561	0.294118
4	0.060576	0.235294

2.3.4 Normalização

Este método é bem semelhante ao método anterior, mas aqui, em vez do valor mínimo, subtraímos cada entrada pelo valor médio de todos os dados e então dividimos os resultados pela diferença entre o valor mínimo e o valor máximo.

$$X_{scaled} = \frac{X_i - X_{mdia}}{X_{max} - X_{min}} \tag{5}$$

[ ]:
```
mean_vals=df.mean()
mean_vals
```

[25]:
```
print((df - df.mean()) / (max_vals - min_vals))
```

```
       LotArea  MSSubClass
0    -0.009661    0.018251
1    -0.004285   -0.217043
2     0.003427    0.018251
3    -0.004519    0.077075
4     0.017496    0.018251
...        ...         ...
1455 -0.012152    0.018251
1456  0.012425   -0.217043
1457 -0.006893    0.077075
1458 -0.003738   -0.217043
1459 -0.002710   -0.217043

[1460 rows x 2 columns]
```

2.4 Padronização

Este método de dimensionamento é basicamente baseado na medida de tendência central e na variância dos dados.

1. Calcular a média e o desvio padrão dos dados que gostaríamos de normalizar.
2. subtrair o valor médio de cada entrada e então dividir o resultado pelo desvio padrão.

Isso nos ajuda a alcançar uma **distribuição normal** dos dados com uma média igual a zero e um desvio padrão igual a 1.

$$X_{scaled} = \frac{X_i - X_{mdia}}{\sigma} \tag{6}$$

[36]:
```
std_vals = df.std()
```

[37]:
```
print((df - df.mean()) / (std_vals))
```

```
       LotArea  MSSubClass
0    -0.207071    0.073350
1    -0.091855   -0.872264
2     0.073455    0.073350
3    -0.096864    0.309753
4     0.375020    0.073350
...        ...         ...
1455 -0.260471    0.073350
1456  0.266316   -0.872264
1457 -0.147760    0.309753
1458 -0.080133   -0.872264
1459 -0.058092   -0.872264

[1460 rows x 2 columns]
```

[38]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaled_data = scaler.fit_transform(df)
scaled_df = pd.DataFrame(scaled_data,
                         columns=df.columns)
print(scaled_df.head())
```

```
    LotArea  MSSubClass
0 -0.207142    0.073375
1 -0.091886   -0.872563
2  0.073480    0.073375
3 -0.096897    0.309859
4  0.375148    0.073375
```

2.4.1 Robust scaler

Neste médodo de dimensionamento, usamos duas medidas estatísticas principais dos dados.

- Mediana
- Intervalo Interquartil

Depois de calcular esses dois valores, devemos subtrair a mediana de cada entrada e então dividir o resultado pelo intervalo interquartil, onde $IRQ = Q_3 - Q_1$.

$$X_{scaled} = \frac{X_i - X_{mediana}}{IRQ} \tag{7}$$

[56]:
```
print( (df-df.median())/(df.quantile(.75)-df.quantile(.25)))
```

```
       LotArea  MSSubClass
0    -0.254076         0.2
1     0.030015        -0.6
2     0.437624         0.2
3     0.017663         0.4
4     1.181201         0.2
...        ...         ...
1455 -0.385746         0.2
1456  0.913167        -0.6
1457 -0.107831         0.4
1458  0.058918        -0.6
1459  0.113266        -0.6

[1460 rows x 2 columns]
```

[57]:
```
import numpy as np
from sklearn.preprocessing import RobustScaler

scaler = RobustScaler(quantile_range=(25.0, 75.0),with_centering=True)
```

```
df_new = scaler.fit_transform(df)
print(df_new)
```

```
[[-0.25407609  0.2       ]
 [ 0.03001482 -0.6       ]
 [ 0.43762352  0.2       ]
 …
 [-0.10783103  0.4       ]
 [ 0.05891798 -0.6       ]
 [ 0.11326581 -0.6       ]]
```

A normalização de dados é uma etapa vital no pipeline de pré-processamento de qualquer projeto de aprendizado de máquina.

Usando o scikit-learn, é possível aplicar diferentes técnicas de normalização, como Min-Max Scaling, Standardization e Robust Scaling.

Escolher o método de normalização certo pode impactar significativamente o desempenho dos seus modelos de aprendizado de máquina.

Ao incorporar essas técnicas de normalização, você pode garantir que seus dados estejam bem preparados para modelagem, resultando em previsões mais **precisas** e **confiáveis**.

Qual é a diferença entre normalização e padronização? A normalização dimensiona os dados para um intervalo específico ou valor unitário, enquanto a padronização dimensiona os dados para ter uma média de 0 e um desvio padrão de 1.

A normalização é útil para garantir contribuição igual dos atributos, enquanto a padronização é frequentemente usada para algoritmos como regressão linear.

[ ]: