Aula 3
Machine Learning com o Algoritmo Naive Bayes

Prof. Mauricio Duarte
FATEC - Pompeia

Naive Bayes - conceito

- O algoritmo “Naive Bayes” é um classificador probabilístico muito utilizado em machine learning.
- Ele recebe o nome de “naive” (ingênuo) porque desconsidera a correlação entre as variáveis (features). Ou seja, se determinada fruta é rotulada como “Limão”, caso ela também seja descrita como “Verde” e “Redonda”, o algoritmo não vai levar em consideração a correlação entre esses fatores. Isso porque trata cada um de forma independente.
- Um problema simples que exemplifica bem o teorema é o cálculo de probabilidades em cima de diagnóstico de doenças.

Exemplo

- Imagine que estamos trabalhando no diagnóstico de uma nova doença. Após realizar testes, coletas e análises com **100 pessoas** distintas, descobrimos que **20 pessoas possuíam a doença** (20%) e **80 pessoas estavam saudáveis** (80%).
- De todas as pessoas que possuíam a doença, 90% receberam Positivo no teste. Já 30% das pessoas que não possuíam a doença também receberam o teste positivo.

Exemplo

Vamos para a tabulação dos dados:

100 pessoas realizaram o teste.
20% das pessoas que realizaram o teste possuíam a doença.
90% das pessoas que possuíam a doença, receberam positivo no teste.
30% das pessoas que não possuíam a doença, receberam positivo.

A partir destes dados, surge o problema: se uma nova pessoa realizar o teste e receber um resultado positivo, qual a probabilidade dela realmente possuir a doença?

O algoritmo Naive Bayes faz os cálculos dessas probabilidades...

Programando em Python

Entrar no Google Colab...

Abrindo a base de dados para o exemplo prático...

```
import pandas as pd

base_credito = pd.read_csv("/content/risco_credito.csv", sep=';')

base_credito
```

Programando em Python

Criando as variáveis ...

```
x_risco_credito = base_credito.iloc[:,0:4].values
```

```
x_risco_credito
```

```
array([['ruim', 'alta', 'nenhuma', '0_15'],
       ['desconhecida', 'alta', 'nenhuma', '15_35'],
       ['desconhecida', 'baixa', 'nenhuma', '15_35'],
       ['desconhecida', 'baixa', 'nenhuma', 'acima_35'],
       ['desconhecida', 'baixa', 'nenhuma', 'acima_35'],
       ['desconhecida', 'baixa', 'adequada', 'acima_35'],
       ['ruim', 'baixa', 'nenhuma', '0_15'],
       ['ruim', 'baixa ', 'adequada', 'acima_35'],
       ['boa', 'baixa', 'nenhuma', 'acima_35'],
       ['boa', 'alta', 'adequada', 'acima_35'],
       ['boa', 'alta', 'nenhuma', '0_15'],
       ['boa', 'alta', 'nenhuma', '15_35'],
       ['boa', 'alta', 'nenhuma', 'acima_35'],
       ['ruim', 'alta', 'nenhuma', '15_35']], dtype=object)
```

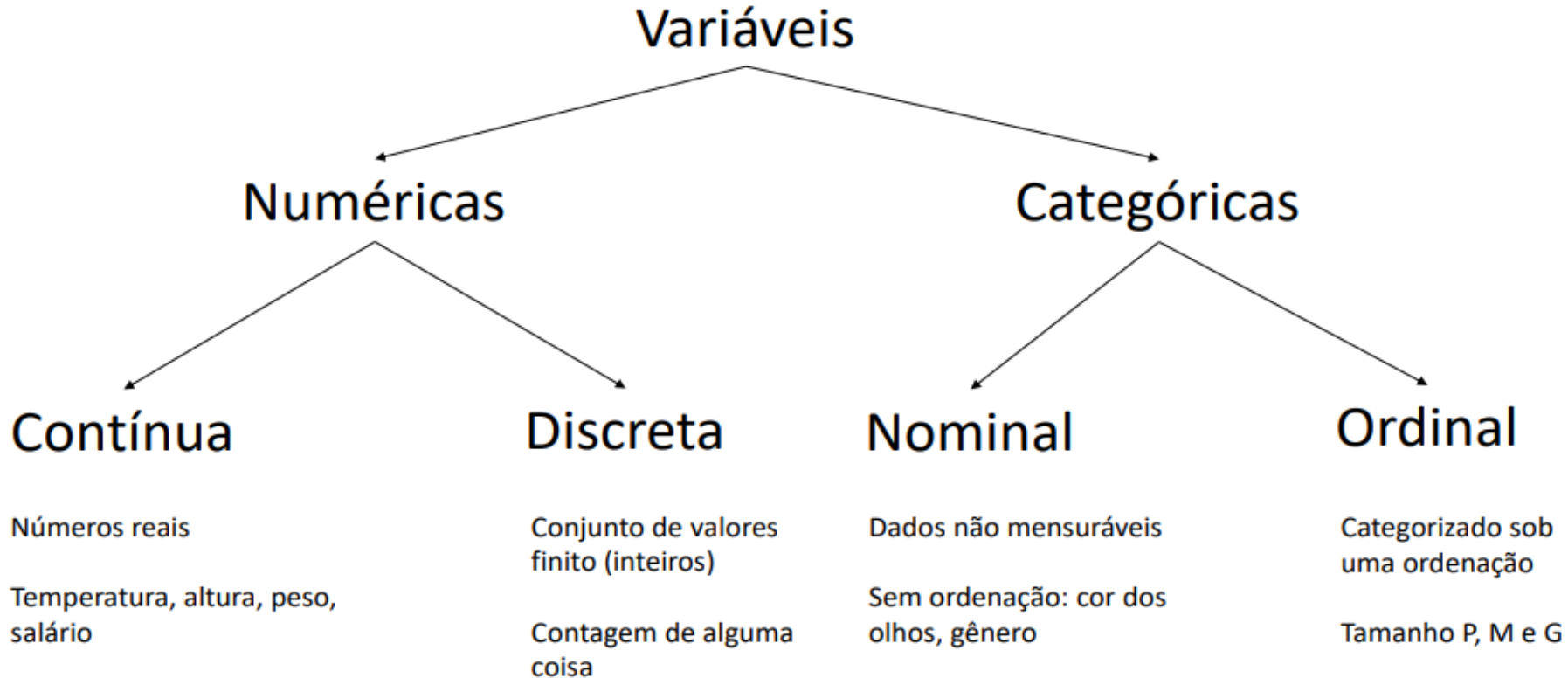
Programando em Python

Criando as variáveis ...

```
y_risco_credito = base_credito.iloc[:, 4].values
y_risco_credito
```

```
array(['alto', 'alto', 'moderado', 'alto', 'baixo',
'baixo', 'alto', 'moderado', 'moderado', 'baixo', 'alto',
'moderado', 'baixo', 'alto'], dtype=object)
```

Lembrando....


Variáveis
Numéricas
Categóricas
Contínua
Discreta
Nominal
Ordinal
Números reais
Temperatura, altura, peso, salário
Conjunto de valores finito (inteiros)
Contagem de alguma coisa
Dados não mensuráveis
Sem ordenação: cor dos olhos, gênero
Categorizado sob uma ordenação
Tamanho P, M e G

Programando em Python

Convertendo variáveis categóricas em numéricas...

```
#convertendo atributos categoricos em numéricos

from sklearn.preprocessing import LabelEncoder
label_encoder_historia = LabelEncoder()
label_encoder_divida = LabelEncoder()
label_encoder_garantia = LabelEncoder()
label_encoder_renda = LabelEncoder()
```

Programando em Python

Convertendo variáveis categóricas em numéricas...

```
x_risco_credito[:,0] = label_encoder_historia.fit_transform(x_risco_credito[:,0])
x_risco_credito[:,1] = label_encoder_divida.fit_transform(x_risco_credito[:,1])
x_risco_credito[:,2] = label_encoder_garantia.fit_transform(x_risco_credito[:,2])
x_risco_credito[:,3] = label_encoder_renda.fit_transform(x_risco_credito[:,3])
```

Programando em Python

- Visualizando...

```
x_risco_credito
```

```
array([[2, 0, 1, 0],
       [1, 0, 1, 1],
       [1, 1, 1, 1],
       [1, 1, 1, 2],
       [1, 1, 1, 2],
       [1, 1, 0, 2],
       [2, 1, 1, 0],
       [2, 2, 0, 2],
       [0, 1, 1, 2],
       [0, 0, 0, 2],
       [0, 0, 1, 0],
       [0, 0, 1, 1],
       [0, 0, 1, 2],
       [2, 0, 1, 1]], dtype=object)
```

Programando em Python - Treinamento

```
from sklearn.model_selection import train_test_split

x_treino, x_teste, y_treino, y_teste = train_test_sp
lit(x_risco, y_risco,test_size = 0.10, random_state=
0, )
```

Programando em Python – Classificador Naive

```
from sklearn.naive_bayes import GaussianNB

naive_risco = GaussianNB()

naive_risco.fit(x_treino, y_treino)
```

Programando em Python

Realizando previsões...

```
#previsões
#historia = Boa 0, Divida = alta 0, garantia = nenhuma 1, renda >= 35 2
#historia = ruim 2, divida = alta 0, garantia = adequada 0, renda

previsoes = naive_risco_credito.predict([[0,0,1,2], [2,0,0,0]])

previsoes

        array(['baixo', 'moderado'], dtype='<U8')
```

Programando em Python

Realizando previsões... Com os dados reservados da base

```
previsoes = naive_risco.predict(x_teste)
previsoes
```

```
array(['baixo', 'alto'], dtype='<U8')
```

Atividade: credit-data
fonte: UCI (Machine Learning Repository)

Objetivo: Prever possível aprovação de crédito agrícola.

Características do conjunto de dados: Multivalorados;

Atributos: 5. **Tipos:** contínuo e inteiro;

Tarefa associada: Classificação;

Valores Faltantes: Sim.

Coleta de Dados

```
import pandas as pd
base = pd.read_csv ("credit-data.csv")
```

1	clientid	income	age	loan	default
2	1	661.559.250.950.813	59.017.015.066.929.200	810.653.213.128.514	0
3	2	344.151.539.658.196	4.811.715.310.486.020	6.564.745.017.677.370	0
4	3	573.171.700.630.337	6.310.804.949.188.590	8.020.953.296.386.460	0
5	4	42.709.534.200.839.700	45.751.972.352.154.500	6.103.642.260.140.690	0
6	5	6.695.268.884.534.010	185.843.359.269.202	877.009.923.520.439	1
7	6	24.904.064.140.282.500	574.716.071.025.468	15.498.598.437.827.100	0
8	7	484.303.596.126.847	26.809.132.419.060.800	572.258.198.121.271	0
9	8	245.001.419.843.175	328.975.483.207.032	297.100.330.971.188	1
10	9	406.548.925.372.772	55.496.852.539.479.700	47.558.252.798.016	0
11	10	25.075.872.770.976.200	397.763.780.555.688	140.923.037.111.453	0
12	11	641.314.153.722.487	25.679.575.353.860.800	43.510.289.707.232.500	0
13	12	59.436.847.122.851.700	604.719.358.547.591	925.424.453.803.174	0
14	13	610.503.460.792.825	263.550.438.545.644	589.326.465.933.928	0
15	14	272.679.954.580.963	61.576.775.823.254.000	47.597.875.810.455	0
16	15	63.061.960.174.236.300	392.015.528.911.725	185.036.937.703.064	0
17	16	505.017.266.888.171	-28.218.361.321.371.000	39.772.874.324.738.300	0
18	17	435.486.547.113.396	395.745.303.500.444	39.355.444.533.423.400	0
19	18	433.781.751.943.752	6.084.831.793.932.230	32.777.375.531.263	0
20	19	20.542.365.072.760.700	616.905.707.078.986	31.574.422.899.706.600	0

Tratamento de dados inconsistentes (pré-processamento – pandas)

Trate os dados inconsistentes

Age negativa ??????

Tente localizar na base as idades negativas...

Veja a figura ao lado...

client	income	age	loan	default
22	321.976.207.010.448	-52.423.279.919.661.500	4.244.057.136.123.400	0
16	505.017.266.888.171	-28.218.361.321.371.000	39.772.874.324.738.300	0
27	63.287.038.907.874.400	-364.969.755.136.408	959.528.628.892.989	0
1735	573.414.327.739.738	2.347.849.796.433	784.894.856.825.166	0
918	497.468.874.370.679	2.936.146.192.858	7.354.129.522.959.970	1
330	445.272.589.032.538	4.209.009.227.762	4.588.472.285.879.380	0
47	418.969.715.991.989	4.725.073.103.467	4.892.209.733.649.000	0
25	653.019.840.293.564	4.884.092.176.985	546.526.788.565.462	0
349	225.723.027.647.473	5.101.624.041.982	724.193.188.479.234	0
627	368.927.162.187.174	5.444.842.475.513	646.364.775.060.681	0
1637	479.235.755.093.396	5.721.906.913.165	6.931.716.435.305.070	0
809	274.277.894.455.008	5.970.451.917.639	719.946.664.588.216	0
1425	29.909.554.000.478.100	6.272.826.527.711	449.527.875.332.798	0
413	435.097.577.562.556	18.075.335.860.718	736.303.763.852.006	1
1605	332.616.460.206.918	18.229.629.389.476	5.866.510.961.630.650	0
454	425.225.757.574.663	18.326.122.155.413	503.625.528.047.363	0
903	4.899.185.367.859.900	18.621.307.099.238	745.326.426.764.463	1
1523	361.323.275.882.751	18.713.332.561.776	300.939.733.978.784	0
1361	525.296.987.663.023	18.812.804.243.993	9.808.190.939.503.140	1

Tratamento de dados inconsistentes (pré-processamento – pandas)

Após localizá-las, na coluna idade, faça a respectiva média considerando apenas as idades válidas (excluir as negativas). Verificar o método mean()

Esta média calculada deverá ser inserida nos locais destas idades negativas. Veja a figura:

client	income	age	loan	default
22	321.976.207.010.448	40.92	4.244.057.136.123.400	0
16	505.017.266.888.171	40.92	39.772.874.324.738.300	0
27	63.287.038.907.874.400	40.92	959.528.628.892.989	0

Tratamento de dados inconsistentes (pré-processamento – pandas)

Será que em outras colunas de nossa base também há inconsistências, como por exemplo letras ???

Verificar ... Ok.

Limpeza dos dados - Tratamento dos dados faltantes (pré-processamento sklearn)

Primeiramente, faremos duas variáveis

previsores (clientid, income, age e loan)

classe (default)

Verificar nos previsores, como inserir a média nas colunas que possuem NaN

	previsores			classe
clientid	income	age	loan	default
1	661.559.250.950.813	59.017.015.066.929.200	810.653.213.128.514	0
2	344.151.539.658.196	4.811.715.310.486.020	6.564.745.017.677.370	0
3	573.171.700.630.337	6.310.804.949.188.590	8.020.953.296.386.460	0
4	42.709.534.200.839.700	45.751.972.352.154.500	6.103.642.260.140.690	0
5	6.695.268.884.534.010	185.843.359.269.202	877.009.923.520.439	1
6	24.904.064.140.282.500	574.716.071.025.468	15.498.598.437.827.100	0
7	484.303.596.126.847	26.809.132.419.060.800	572.258.198.121.271	0
8	245.001.419.843.175	328.975.483.207.032	297.100.330.971.188	1
9	406.548.925.372.772	55.496.852.539.479.700	47.558.252.798.016	0
10	25.075.872.770.976.200	397.763.780.555.688	140.923.037.111.453	0
11	641.314.153.722.487	25.679.575.353.860.800	43.510.289.707.232.500	0
12	59.436.847.122.851.700	604.719.358.547.591	925.424.453.803.174	0
13	610.503.460.792.825	263.550.438.545.644	589.326.465.933.928	0
14	272.679.954.580.963	61.576.775.823.254.000	47.597.875.810.455	0
15	63.061.960.174.236.300	392.015.528.911.725	185.036.937.703.064	0
16	505.017.266.888.171	-28.218.361.321.371.000	39.772.874.324.738.300	0
17	435.486.547.113.396	395.745.303.500.444	39.355.444.533.423.400	0
18	433.781.751.943.752	6.084.831.793.932.230	32.777.375.531.263	0
19	20.542.365.072.760.700	616.905.707.078.986	31.574.422.899.706.600	0
20	588.873.575.488.105	26.076.093.018.055	496.551.606.626.419	0
21	230.007.840.017.788	31.761.354.170.739.600	114.811.805.715.996	0

Outra forma de tratar NaN
Classe SimpleImputer

SimpleImputer é uma classe **scikit** -learn que é útil para lidar com os dados ausentes no conjunto de dados do modelo preditivo.
Ele substitui os valores NaN por um espaço reservado especificado.

Parâmetros:
missing_values : o placeholder missing_values que deve ser imputado.
Por padrão, é a **estratégia** *NaN* : os dados que substituirão os valores NaN do conjunto de dados.
O argumento de estratégia pode assumir os valores - 'média' (padrão), 'mediana', 'mais_frequente' e 'constante'. **fill_value** : o valor constante a ser dado aos dados NaN usando a estratégia constante.

Classe SimpleImputer - Exemplo

```
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(missing_values = np.nan,
                        strategy ='mean')

data = [[12, np.nan, 34], [10, 32, np.nan],
        [np.nan, 11, 20]]

print("Original Data : \n", data)
# Fitting the data to the imputer object
imputer = imputer.fit(data)

# Imputing the data
data = imputer.transform(data)

print("Imputed Data : \n", data)
```

```
Original Data :

[[12, nan, 34]
[10, 32, nan]
[nan, 11, 20]]


Imputed Data :

[[12, 21.5, 34]
[10, 32, 27]
[11, 11, 20]]
```

Limpeza dos dados em nossa base

Seguindo o exemplo anterior, faça, na variável previsores os tratamentos dos dados faltantes....

Padronização
(pré-processamento sklearn)

A transformação dos seus dados, que já estão tratados, é uma prática para evitar que seu algoritmo fique enviesado para as variáveis com maior ordem de grandeza. A padronização irá **transformar** todas as variáveis na mesma ordem de grandeza.

Padronização (Standardisation)

$$x = \frac{x - média(x)}{desvio\ padrão(x)}$$

Para reescalar os dados utilize a biblioteca:
from sklearn.preprocessing import StandardScaler

Pesquisar métodos fit_transform

Classificação
(pré-processamento sklearn)

Para fazer o treinamento e teste utilize a biblioteca (pesquise-a...):

from sklearn.model_selection import train_test_split

Use como classificador o Algoritmo Naive Bayes

→ classificador = GaussianNB()

Para analisar o desempenho do classificador utilize a biblioteca:

from sklearn.metrics import confusion_matrix, accuracy_score

Fazer a pesquisa de suas funcionalidades e de como usá-las para prover o resultado.