# AULA PRÁTICA DE NAIVE BAYES

- **Objetivo da aula**

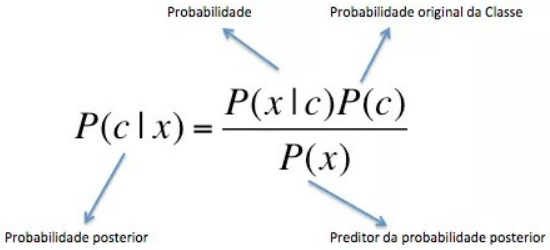
Assimilar os fundamentos da técnica de Naive Bayes (teoria) por meio da execução de exemplos baseado em linguagem Python.

- **Revisão teórica Naive Bayes**

É uma técnica de classificação baseado no teorema de Bayes com uma suposição de independência entre os preditores.
Em termos simples, um classificador Naive Bayes assume que a presença de uma característica particular em uma classe não está relacionada com a presença de qualquer outro recurso.

Por exemplo, um fruto pode ser considerado como uma maçã se é vermelho, redondo, e tiver cerca de 3 polegadas de diâmetro.
Mesmo que esses recursos dependam uns dos outros ou da existência de outras características, todas estas propriedades contribuem de forma independente para a probabilidade de que este fruto é uma maçã e é por isso que é conhecido como 'Naive' (ingênuo).

O modelo Naive Bayes é fácil de construir e particularmente útil para grandes volumes de dados. Além de simples, Naive Bayes é conhecido por ganhar de métodos de classificação altamente sofisticados.




$$P(c \mid X) = P(x_1 \mid c) x P(x_2 \mid c) x \ldots x P(x_n \mid c) x P(c)$$

P (c | x) é a probabilidade posterior da classe (c, alvo) dada preditor (x, atributos).
P (c) é a probabilidade original da classe.
P (x | c) é a probabilidade que representa a probabilidade de preditor dada a classe.

P (x) é a probabilidade original do preditor.

As principais aplicações do Algoritmo Naive Bayes:

- Previsões em tempo real;
- Previsões multi-classes;
- Classificação de textos/Filtragem de spam/Análise de sentimento; e
- Sistema de Recomendação.

- **METODOLOGIA da técnica Naive Bayes**

Passo 1 - Converter o conjunto de dados em uma tabela de frequência;

Passo 2 - Criar tabela de Probabilidade ao encontrar as probabilidades de tempo Nublado = 0,29 e probabilidade de jogar = 0,64; e

Passo 3 - Agora, use a equação Bayesiana Naive para calcular a probabilidade posterior para cada classe. A classe com maior probabilidade posterior é o resultado da previsão.

Exemplo de aplicação utilizando o *dataset* criado na atividade 1 do KNN:

| TEMPO | "PLAY" |
|---|---|
| Sol | Não |
| Nublado | Sim |
| Chuva | Sim |
| Sol | Sim |
| Sol | Sim |
| Nublado | Sim |
| Chuva | Não |
| Chuva | Não |
| Sol | Sim |
| Chuva | Sim |
| Sol | Não |
| Nublado | Sim |
| Nublado | Sim |
| Chuva | Não |

| Tabela de frequência | | |
|---|---|---|
| Clima | Não | Sim |
| Nublado | 0 | 4 |
| Sol | 3 | 2 |
| Chuva | 2 | 3 |
| Total | 5 | 9 |

| Tabela de probabilidade | | | | |
|---|---|---|---|---|
| Clima | Não | Sim | | |
| Nublado | 0 | 4 | =4/14 | 0,29 |
| Sol | 3 | 2 | =5/14 | 0,36 |
| Chuva | 2 | 3 | =5/14 | 0,36 |
| Total | 5 | 9 | | |
| | =5/14 | =9/14 | | |
| | 0,36 | 0,64 | | |

**Problema: Os jogadores irão jogar se o tempo está ensolarado. Esta afirmação está correta?**

**P (Sim |Ensolarado) = P (Ensolarado | Sim) * P (Sim) / P (Ensolarado)**

Aqui temos P (Ensolarado | Sim) = 3/9 = 0,33, P (Sim) = 9/14 = 0,64, P (Ensolarado) = 5/14 = 0,36.

Agora, P (Sim | Ensolarado) = 0,33 * 0,64 / 0,36 = 0,60, que tem maior probabilidade.

- **Vantagens e desvantagens da técnica Naive Bayes**

Pros:

- É fácil e rápido para prever o conjunto de dados da classe de teste. Também tem um bom desempenho na previsão de classes múltiplas;
- Quando a suposição de independência prevalece, um classificador Naive Bayes tem melhor desempenho em comparação com outros modelos como regressão logística, e você precisa de menos dados de treinamento; e
- O desempenho é bom em caso de variáveis categóricas de entrada comparada com a variáveis numéricas. Para variáveis numéricas, assume-se a distribuição normal (curva de sino, que é uma suposição forte).

Contras:

- Se a variável categórica tem uma categoria (no conjunto de dados de teste) que não foi observada no conjunto de dados de treinamento, então o modelo irá atribuir uma probabilidade de 0 (zero) e não será capaz de fazer uma previsão. Isso é muitas vezes conhecido como "*Zero Frequency*". Para resolver isso, podemos usar a técnica de alisamento. Uma das técnicas mais simples de alisamento é a chamada estimativa de Laplace;
- Por outro lado naive Bayes é também conhecido como um mau estimador, por isso, as probabilidades calculadas não devem ser levadas muito a sério; e
- Outra limitação do Naive Bayes é a suposição de preditores independentes. Na vida real, é quase impossível que ter um conjunto de indicadores que sejam completamente independentes.

# ATIVIDADE 1

Vamos dar uma olhada em como podemos classificar os dados usando o algoritmo Naive Bayes em Python.

Vamos usar o *dataset* criado na última aula sobre influência do clima e temperatura para opção de brincar.

```
# Criar um dataset
# Primeira Característica
clima=['Ensolarado', 'Ensolarado', 'Nublado', 'Chuvoso', 'Chuvoso', 'Chuvoso', 'Nublado',
'Ensolarado', 'Ensolarado', 'Chuvoso', 'Ensolarado', 'Nublado', 'Nublado', 'Chuvoso']

# Segunda Característica
temp=['Quente', 'Quente', 'Quente', 'Suave', 'Legal', 'Legal', 'Legal', 'Suave',
'Legal', 'Suave', 'Suave', 'Suave', 'Quente', 'Suave']

# Variavel alvo
brincar=['Não','Não','Sim','Sim','Sim','Não','Sim','Não','Sim','Sim','Sim','Sim','Sim','Não']

# Importando LabelEncoder
from sklearn import preprocessing
#criando labelEncoder
le = preprocessing.LabelEncoder()

# Convertendo string labels para numeros.
clima_encoded = le.fit_transform(clima)
temp_encoded = le.fit_transform(temp)
print(clima_encoded)
print(temp_encoded)

# convertendo string labels para numeros
alvo = le.fit_transform(brincar)
print(alvo)

# Combinando clima e temp em um unica lista de tuplas
carac=list(zip(clima_encoded,temp_encoded))
carac
```

Agora vamos importar a técnica Naive Bayes para o nosso projeto em Python e treinar o algoritmo com o *dataset* previamente criado. Mais uma vez, o scikit learn (biblioteca python) vai ajudar a construir um modelo em Python.

```
#Importa a biblioteca do modelo Naive Bayes Gaussiano
from sklearn.naive_bayes import GaussianNB
import numpy as np

#Cria um classificador Gaussiano
modelo = GaussianNB()

# Treinando o modelo usando os ajustes de treinamento.
modelo.fit(carac,alvo)

#Predito
predito = modelo.predict([[2,2]]) # 2:Nublado, 2:Suave
print(predito)
```

# ATIVIDADE 2

Existem três tipos de modelo Naive Bayes sob a biblioteca do scikit learn:

- Gaussian: É usado na classificação e assume uma distribuição normal;
- Multinomial: É usado para contagem discrete. Por exemplo, digamos que temos um problema de classificação de texto. Aqui podemos considerar tentativas de Bernoulli, que é um passo além e, em vez de “palavra que ocorre no documento”, temos “contar quantas vezes a palavra ocorre no documento”, você pode pensar nisso como “número de vezes que o número desfecho x_i é observado durante as n tentativas “; e
- Bernoulli: O modelo binomial é útil se os vetores são binários (ou seja zeros e uns). Uma aplicação seria de classificação de texto com um modelo de ‘saco de palavras’ onde os 1s e 0s são “palavra ocorre no documento” e “palavra não ocorre no documento”, respectivamente.

Para testar cada classificador Naive Bayes, vamos utilizar novamente o *dataset* IRIS e o scikit learn (biblioteca python) vai ajudar a construir um modelo Naive Bayes em Python com cada um dos três classificadores.

```
# bibliotecas
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# dataset
iris = load_iris()

# caracteristicas e alvo
X = iris.data
y = iris.target

# treino e teste
(X_train, X_test, y_train, y_test) = train_test_split(X,y)


# Modelo 1 classificador Gaussiano
from sklearn.naive_bayes import GaussianNB

# instanciando o modelo
modelo1 = GaussianNB()
# treinando o modelo utilizando o conjunto de treino
modelo1.fit(X_train,y_train)
# validando o modelo utilizando o conjunto de teste
precisao1 = str(round(modelo1.score(X_test,y_test) * 100, 2))+"%"
# imprimindo o resultado
print("A acurácia do modelo 1 foi",precisao1)


# Modelo 2 classificador Multinomial
from sklearn.naive_bayes import MultinomialNB

# instanciando o modelo
modelo2 = MultinomialNB()
```

```
# treinando o modelo utilizando o conjunto de treino
modelo2.fit(X_train,y_train)
# validando o modelo utilizando o conjunto de teste
precisao2 = str(round(modelo2.score(X_test,y_test) * 100, 2))+"%"
# imprimindo o resultado
print("A acurácia do modelo 2 foi",precisao2)


# Modelo 3 classificador Bernoulli
from sklearn.naive_bayes import BernoulliNB

# instanciando o modelo
modelo3 = BernoulliNB()
# treinando o modelo utilizando o conjunto de treino
modelo3.fit(X_train,y_train)
# validando o modelo utilizando o conjunto de teste
precisao3 = str(round(modelo3.score(X_test,y_test) * 100, 2))+"%"
# imprimindo o resultado
print("A acurácia do modelo 3 foi",precisao3)
```

# ATIVIDADE 3

Para este exemplo, usaremos o conjunto de dados Wine do módulo sklearn.datasets e realizaremos um procedimentos de comparação de acurácia obtida pela técnica de K-NN e pela técnica de Naive Bayes.

```
# bibliotecas
import numpy as np
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split

# remover warnings
import warnings
warnings.filterwarnings("ignore")
```

```
# dataset
vinho = load_wine()

# caracteristicas e alvo
X = vinho.data
y = vinho.target

# treino e teste
(X_train, X_test, y_train, y_test) = train_test_split(X,y)

#Importa a biblioteca do modelo KNN

from sklearn.neighbors import KNeighborsClassifier
# instanciando o modelo
modeloKNN = KNeighborsClassifier()
# treinando o modelo utilizando o conjunto de treino
modeloKNN.fit(X_train,y_train)
# validando o modelo utilizando o conjunto de teste
precisaoKNN = str(round(modeloKNN.score(X_test,y_test) * 100, 2))+"%"
# imprimindo o resultado
print("A acurácia do modelo k-NN foi",precisaoKNN)
# predizendo o teste
y_pred = modeloKNN.predict(X_test)
# comparando predição com o real
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))


#Importa a biblioteca do modelo Naive Bayes Gaussiano

from sklearn.naive_bayes import GaussianNB
#Cria um classificador Gaussiano
modeloNB = GaussianNB()
# Treinando o modelo usando os ajustes de treinamento.
modeloNB.fit(X_train,y_train)
# validando o modelo utilizando o conjunto de teste
```

```
precisaoNB = str(round(modeloNB.score(X_test,y_test) * 100, 2))+"%"
# imprimindo o resultado
print("A acurácia do modelo Naive Bayes foi",precisaoNB)
# predizendo o teste
y_pred = modeloNB.predict(X_test)
# comparando predição com o real
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))


# normalizando
from sklearn.preprocessing import Normalizer
scaler = Normalizer()
scaler.fit(X)
X = scaler.transform(X)

scoresKNN = []
scoresNB = []
for i in range(2000):
    X_train, X_test, y_train, y_test = train_test_split(X,y)
    modeloKNN = KNeighborsClassifier()
    modeloKNN.fit(X_train,y_train)
    precisaoKNN = modeloNB.score(X_test,y_test)
    scoresKNN.append(precisaoKNN)
    modeloNB = GaussianNB()
    modeloNB.fit(X_train,y_train)
    precisaoNB = modeloNB.score(X_test,y_test)
    scoresNB.append(precisaoNB)

print("Média do KNN: {:.2f}%".format(np.mean(scoresKNN)*100))
print("Desvio padrão do KNN: {:.2f}%".format(np.std(scoresKNN)*100))

print("Média do Naive Bayes: {:.2f}%".format(np.mean(scoresNB)*100))
print("Desvio padrão do Naive Bayes: {:.2f}%".format(np.std(scoresNB)*100))

import matplotlib.pyplot as plt
import seaborn as sns
```

```
sns.distplot(scoresKNN)
plt.yticks([])
plt.title("Acurácias do KNN")
plt.show()

sns.distplot(scoresNB)
plt.yticks([])
plt.title("Acurácias do Naive Bayers")
plt.show()
```