



**INSTITUTO  
FEDERAL**  
Norte de Minas Gerais

# Introdução a Sistemas Inteligentes

Modelo KNN - parte 2  
A implementação do Modelo

**Prof<sup>a</sup>. Suzana Mota**





# KNN

# K-Nearest Neighbors

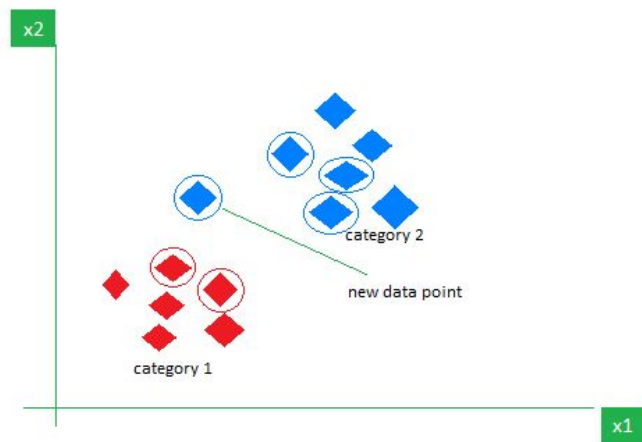
<https://scikit-learn.org/stable/>

<https://scikit-learn.org/dev/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>

# KNN Nearest Neighbors

KNN é um algoritmo de aprendizado supervisionado usado para classificação e regressão.

Baseia-se na ideia de que objetos semelhantes estão próximos uns dos outros no espaço de características.



# Dataset

**Pétala:** é a parte colorida e geralmente vistosa da flor, atraindo polinizadores como insetos e aves.

**Sépala:** é a parte da flor que protege a flor em desenvolvimento antes da floração. Em geral, as sépalas são as folhas verdes que estão na base da flor

**iris setosa**



petal

sepal

**iris versicolor**



petal

sepal

**iris virginica**



petal

sepal

# Dataset

**iris setosa**



petal

sepal

**iris versicolor**



petal

sepal

**iris virginica**







petal




sepal

	<b>Id</b>	<b>SepalLengthCm</b>	<b>SepalWidthCm</b>	<b>PetalLengthCm</b>	<b>PetalWidthCm</b>	<b>Species</b>
<b>0</b>	1	5.1	3.5	1.4	0.2	Iris-setosa
<b>50</b>	51	7.0	3.2	4.7	1.4	Iris-versicolor
<b>100</b>	101	6.3	3.3	6.0	2.5	Iris-virginica

# Dataset

 **suzanasvm** Add files via upload 718e309 · 24 minutes ago 15 Commits

	Delete aulas/a.txt	3 hours ago
	Add files via upload	3 hours ago
	Add files via upload	24 minutes ago
	Update README.md	3 hours ago

 **README**  

## Introdução a Sistemas Inteligentes

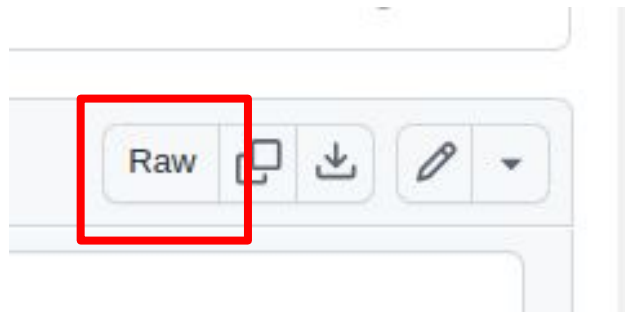
Este repositório concentra os materiais utilizados na disciplina **Introdução a Sistemas Inteligentes**, oferecida no IFNMG-Campus Almenara.

A disciplina visa fornecer uma base sólida em inteligência artificial (IA) e aprendizado de máquina (ML), com foco em conceitos fundamentais, algoritmos, técnicas e aplicações práticas.

### Estrutura do Repositório

- **Aulas/**  
Contém os slides e notas de aula em formato PDF.
- **Datasets/**  
Conjuntos de dados a serem realizados em nossos projetos.
- **Notebooks/**  
Exemplos e projetos práticos para cada tópico abordado ao longo da disciplina.

SistemasInteligentes / datasets / iris / iris.csv 



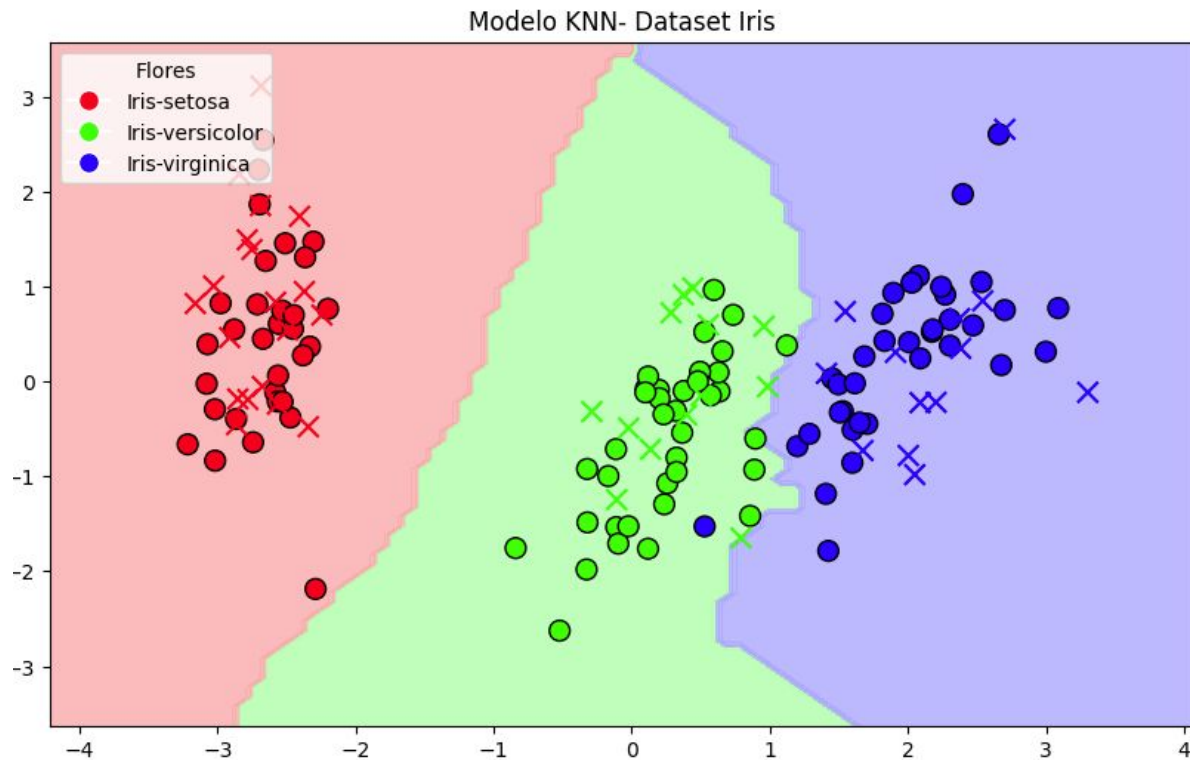
 raw.githubusercontent.com/suzanasvm/SistemasInteligentes/refs/heads/main/datasets/iris/iris.csv

**Copie o Endereço da URL do dataset, após clicar em RAW.**

<https://github.com/suzanasvm/SistemasInteligentes/tree/main>

# O que vamos fazer?

Gerar um modelo KNN, capaz de separar as diferentes tipos de flores iris



# Importando bibliotecas

```
#instalando lib para obter perfil de dados  
!pip install ydata_profiling
```

---

```
] import numpy as np  
import pandas as pd  
from sklearn.model_selection import train_test_split  
from sklearn.preprocessing import StandardScaler, LabelEncoder  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.metrics import accuracy_score  
from ydata_profiling import ProfileReport
```



# Lendo os dados

```
] url_dados = 'https://raw.githubusercontent.com/suzanasvm/SistemasInteligentes/refs/heads/main/datasets/iris/iris.csv'  
  
df = pd.read_csv(url_dados)  
df.set_index('Id', inplace=True)
```

```
df.set_index('Id', inplace=True)
```

Definir quem são seus identificadores é uma boa prática!

Para que o modelo não considere que eles são features (características) a serem observadas!

# Conhecendo os dados

Faça uma pequena análise exploratória para entender:

- Quantos dados temos?
- Eles estão balanceados (temos a mesma quantidade para todas as classes)?
- Existem dados faltantes?

```
profile = ProfileReport(df)
profile
```

# ATENÇÃO!

Neste dataset não será necessário realizar limpeza de dados como:

- Remover colunas
- Preencher Dados Faltantes

Frequentemente este passo é necessário, cada dataset deve ser observado com cuidado!

# Discretizando os dados

Os modelos como o KNN só sabem lidar com dados numéricos, para isso precisamos discretizar os dados, ou seja transformar palavras em números.

	<b>Id</b>	<b>SepalLengthCm</b>	<b>SepalWidthCm</b>	<b>PetalLengthCm</b>	<b>PetalWidthCm</b>	<b>Species</b>	
	0	1	5.1	3.5	1.4	0.2	Iris-setosa 0
	50	51	7.0	3.2	4.7	1.4	Iris-versicolor 1
	100	101	6.3	3.3	6.0	2.5	Iris-virginica 2

# Discretizando os dados

Existem algumas maneiras:

Você pode deixar explícito em um dicionário qual mapeamento deseja usar:

```
mapeamento = {  
    'vermelho': 0,  
    'amarelo': 1,  
    'verde': 2  
}  
  
# Aplicando o mapeamento  
df['Cor_codificada'] = df['Cor'].map(mapeamento)
```

Cor	Cor_codificada
vermelho	0
amarelo	1
verde	2

# Discretizando os dados

Existem algumas maneiras:

Utilizar o Label Encoder

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

# Aplicando o LabelEncoder
df['Cor_codificada'] = le.fit_transform(df['Cor'])

print(df)
```

Cor	Cor_codificada
vermelho	2
amarelo	0
verde	1

# Discretizando os dados

Existem algumas maneiras:

Utilizar o One-Hot Encoding, do `get_dummies`

```
df_dummies = pd.get_dummies(df, columns=['Cor'], prefix='Cor')  
print(df_dummies)
```

Cor_amarelo	Cor_verde	Cor_vermelho
False	False	True
True	False	False
False	True	False

# Discretizando os dados

	Mapeamento Manual	Label Encoder	Get Dummies (One Hot Encoding)
Saída	Produz uma coluna com inteiros gerados manualmente, associando o item a um número.	Produz uma coluna com inteiros gerados automaticamente.	Produz várias colunas com 0s e 1s, indicando a presença ou ausência daquela característica.
USO	Melhor para variáveis ordinais, ou seja, quando a ordem importa e você deseja ter mais controle sobre a associação	Melhor para variáveis ordinais, ou seja, quando a ordem importa.	Melhor para variáveis nominais, como classes.
Possíveis Problemas	Pode introduzir um significado de ordem onde não existe.	Pode introduzir um significado de ordem onde não existe.	Aumenta a dimensionalidade do DataFrame, especialmente se houver muitas categorias.



# Separando features e labels

Você precisa se perguntar: **Em quais colunas estão as características e qual coluna temos o nosso label classificador?**

	<b>Id</b>	<b>SepalLengthCm</b>	<b>SepalWidthCm</b>	<b>PetalLengthCm</b>	<b>PetalWidthCm</b>	<b>Species</b>
<b>0</b>	1	5.1	3.5	1.4	0.2	Iris-setosa
<b>50</b>	51	7.0	3.2	4.7	1.4	Iris-versicolor
<b>100</b>	101	6.3	3.3	6.0	2.5	Iris-virginica

# Separando features e labels

Você precisa se perguntar: **Em quais colunas estão as características e qual coluna temos o nosso label classificador?**

**X = features**

**y = label**

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
50	51	7.0	3.2	4.7	1.4	Iris-versicolor
100	101	6.3	3.3	6.0	2.5	Iris-virginica

# Separando features e labels

**X = features**

**y = label**

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
50	51	7.0	3.2	4.7	1.4	Iris-versicolor
100	101	6.3	3.3	6.0	2.5	Iris-virginica

```
] #A variavel X, fica com todas as  
X = df.drop('Species', axis=1)  
#A variavel y fica apenas com os  
y = df['Species']
```

X

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
Id				
1	5.1	3.5	1.4	0.2



# Dividir os dados em Treinamento e Teste

Dividir os dados em conjuntos de treino e teste é importante para garantir que um modelo de aprendizado de máquina possa generalizar bem para dados não vistos.

```
#Divisao de treino 70% e teste 30%, semente randomica 42
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Saída dados de Teste:

```
#Features de Teste
X_test.head()
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
Id				
74	6.1	2.8	4.7	1.2

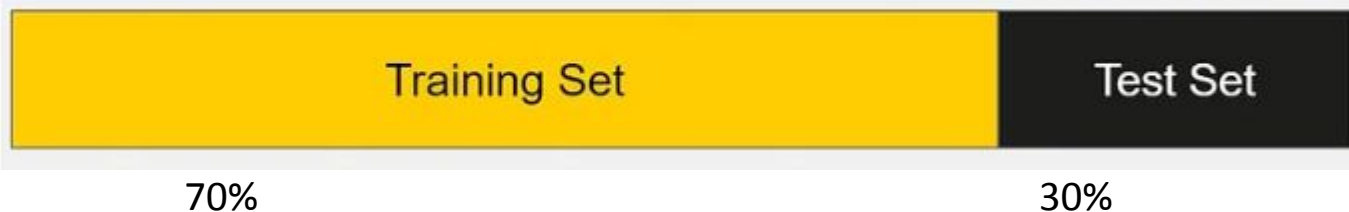
```
#Labels de Teste (já discretizadas)
y_test
```

```
array([1, 0, 2, 1, 1, 0, 1, 2, 1, 1,
```

# Dividir os dados em Treinamento e Teste

Dividir os dados em conjuntos de treino e teste é importante para garantir que um modelo de aprendizado de máquina possa generalizar bem para dados não vistos.

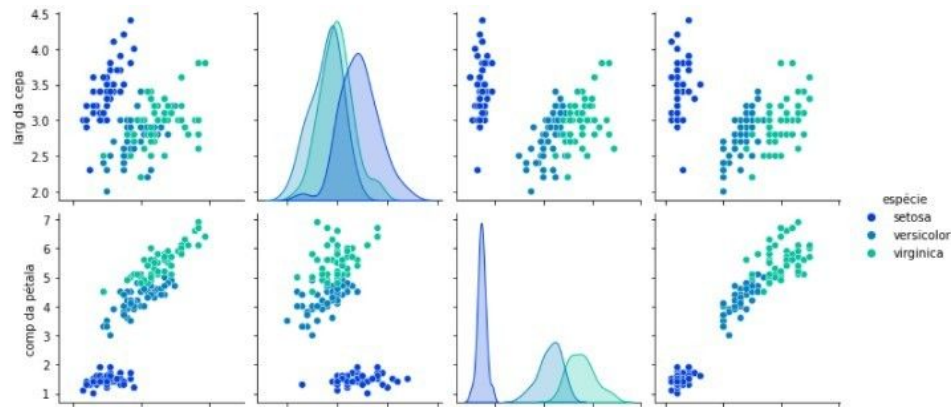
```
Tamanho do dataframe (150, 5)  
Tamanho dos dados de treino (105, 4)  
Tamanho dos dados de teste (45, 4)
```



# Normalizando os dados

Quando os dados possuem escalas diferentes, variáveis que apresentam valores mais altos podem dominar a análise, distorcendo os resultados.

Isso significa que as distâncias calculadas entre os pontos de dados podem não refletir a verdadeira similaridade entre eles.



# Normalizando os dados

Imagine uma corrida de Santa Catarina até a Bahia.  
Os competidores possuem os seguintes veículos.  
Precisamos calcular, quem chegará em menos tempo



15 km/h



20 milhas por  
hora



50 nós

A padronização dos dados consiste em trazer todos os dados do dataset para uma mesma escala.



# Padronizando os dados

```
#biblioteca para padronizar os dados
from sklearn.preprocessing import StandardScaler

# Cria uma instância do StandardScaler para
# padronização dos dados.
scaler = StandardScaler()
# Calcula a média e o desvio padrão de X_train,
#e aplica a padronização a esses dados.
X_train_normalized = scaler.fit_transform(X_train)
# Aplica a padronização a X_test usando a média |
#e o desvio padrão calculados a partir de X_train
X_test_normalized = scaler.transform(X_test)
```

# Treinando o Modelo

```
5] #biblioteca para criarmos o modelo de machine learning  
from sklearn.neighbors import KNeighborsClassifier
```

```
#instanciar o modelo (criamos o modelo) - por padrão são 5 vizinhos  
#se quisermos alterar o k, basta passar o parametro |  
#n_neighbors= NOVO NUMERO DE K vizinhos  
knn = KNeighborsClassifier()
```

```
#treinando o modelo com os dados de treino normalizados  
knn.fit(X_train_normalized, y_train)
```

---

# Treinando o Modelo

Fit = Método para chamar o treinamento do modelo

Fit = ajuste

```
#treinando o modelo com os dados de treino normalizados  
knn.fit(X_train_normalized, y_train)
```

Modelo  
Instanciado

Conjunto de Features que o  
modelo irá usar para detectar  
padrões

As classes

Um objeto  
da classe  
do Modelo

# Acurácia do Modelo

```
acuracia_treinamento = knn.score(X_train_normalized, y_train)
print(f"Acurácia do modelo nos dados de treinamento: {acuracia_treinamento:.2f}")
```

Acurácia do modelo nos dados de treinamento: 0.95

**acuracia\_treinamento**

VARIÁVEL PARA ARMAZENAR VALOR  
DE ACURÁCIA (TAXA DE ACERTO)

**= knn.score ( X\_train\_normalized,**

Método score  
do objeto do  
modelo

Features de Treinamento

**y\_train)**

Classes das  
flores

# Avaliando o Modelo

O meu modelo foi capaz de identificar padrões e está acertando 95% dos casos, dentro do conjunto de dados de treinamento.

E se eu apresentar para ele, dados completamente novos, quanto será que ele acerta?



# Avaliando o Modelo

```
] #testando o modelo com os dados de teste  
predito_knn = knn.predict(X_test_normalized)
```

```
#Predições do modelo  
predito_knn
```

```
array([1, 0, 2, 1, 1, 0, 1, 2, 1, 1, 2, 0, 0, 0, 0, 1, 2, 1, 1, 2, 0, 2,  
       0, 2, 2, 2, 2, 2, 0, 0, 0, 0, 1, 0, 0, 2, 1, 0, 0, 0, 2, 1, 1, 0,  
       0])
```

# Avaliando o Modelo

```
accuracy = accuracy_score(y_test, predito_knn)  
print(f"Acurácia do modelo KNN: {accuracy * 100:.2f}%")
```

• Acurácia do modelo KNN: 100.00%



# Avaliando o Modelo

Como interpretar esses dados?



Acurácia Treinamento	Acurácia Testes	O que interpretar?
ALTA	ALTA	O modelo está generalizando bem e se ajustando aos dados corretamente.
ALTA	BAIXA	O modelo está com overfitting; aprendeu os dados de treinamento, mas não consegue generalizar.
BAIXA	BAIXA	O modelo não está aprendendo corretamente e pode ser necessário ajustar ou reavaliar as features.
BAIXA	ALTA	O modelo provavelmente está sendo avaliado em um conjunto de teste que é muito diferente do treinamento.



# Avaliando o Modelo



**Acurácia de Treinamento: 95%**

**Acurácia de Teste: 98%**

## **Interpretação:**

O modelo aprendeu bem os padrões nos dados de treinamento e é capaz de aplicá-los corretamente em dados desconhecidos. É um sinal positivo de que o modelo generaliza bem.

# Avaliando o Modelo

**Acurácia de Treinamento: 95%**  
**Acurácia de Teste: 60%**



## Interpretação:

O modelo se ajustou muito bem aos dados de treinamento, mas não consegue aplicar esse conhecimento a novos dados. Isso indica um problema de overfitting, onde o modelo memoriza os dados de treinamento em vez de aprender a generalizar.

Se tivéssemos um modelo com 2 classes, o modelo seria um pouco melhor do que jogar uma moeda ( o que é um resultado muito ruim).

# Avaliando o Modelo

**Acurácia de Treinamento: 60%**

**Acurácia de Teste: 50%**



## **Interpretação:**

O modelo não conseguiu aprender os padrões nem nos dados de treinamento, resultando em desempenho fraco em ambos os conjuntos. Isso pode sugerir que o modelo não está conseguindo generalizar, os dados podem ser poucos ou as features podem não ser representativas.

# Avaliando o Modelo

**Acurácia de Treinamento: 60%**

**Acurácia de Teste: 95%**



## **Interpretação:**

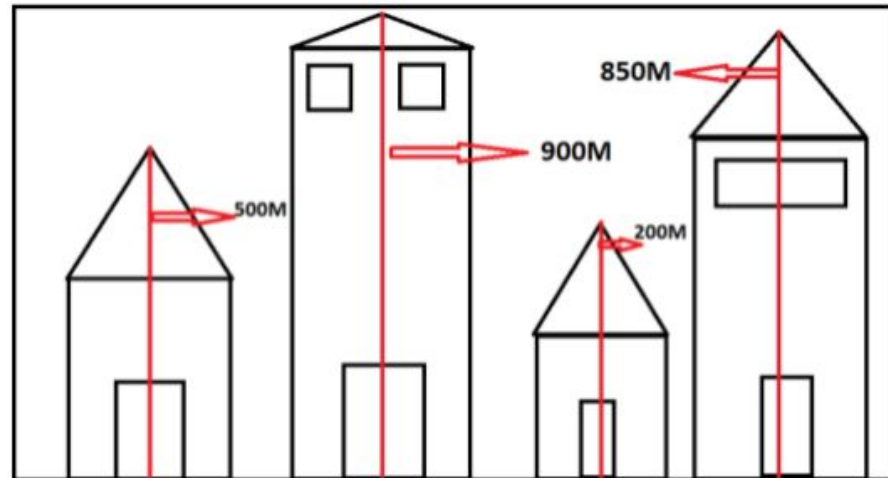
O modelo não aprendeu os dados de treinamento corretamente, mas se saiu bem em testes. Isso pode ocorrer se os dados de teste forem muito diferentes dos dados de treinamento ou se houver algum tipo de viés nas amostras.

# Variância

Observar a variância das features e sua importância é crucial para entender como os dados se separam e quais variáveis influenciam essa separação.

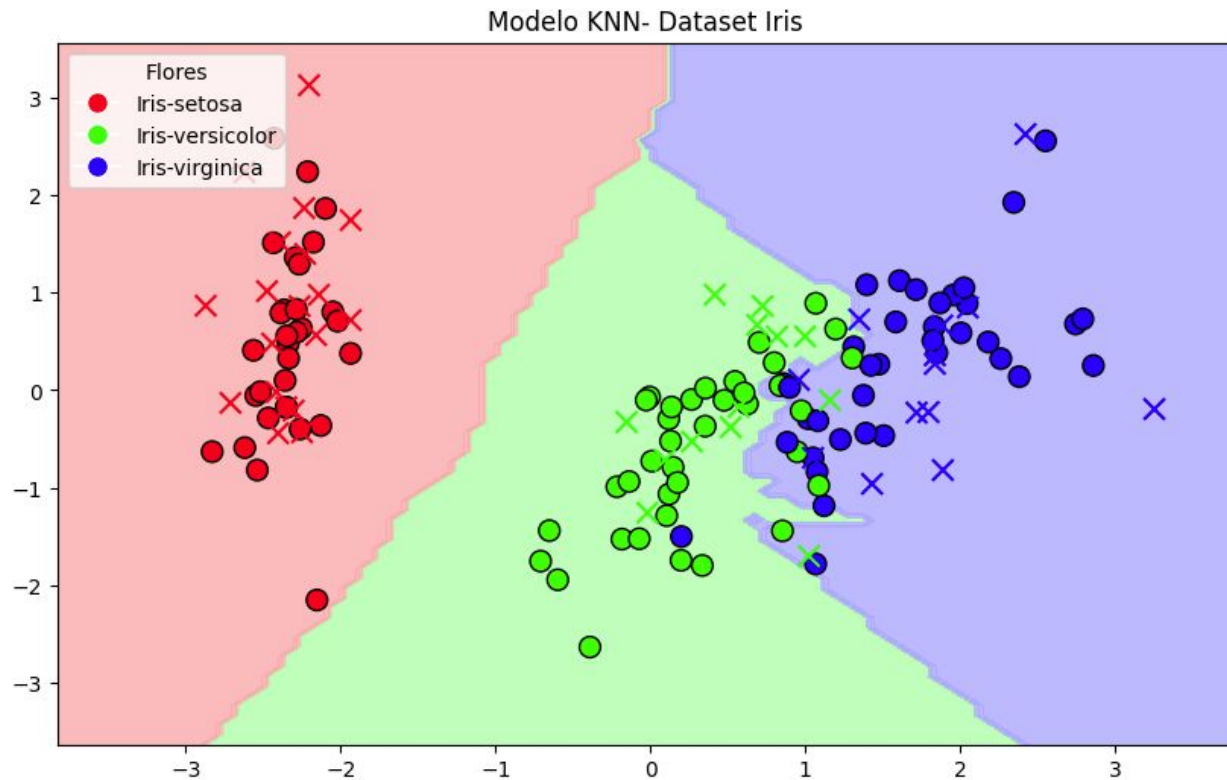
A variância indica a quantidade de informação que uma feature traz para o modelo; quanto maior a variância, mais relevante ela pode ser para capturar padrões nos dados.

Imóveis	
Qtd quartos	2 em todos
Qtd banheiros	1 em todos
Metros Quadrados	



# Predições do Modelo

$K = 3$

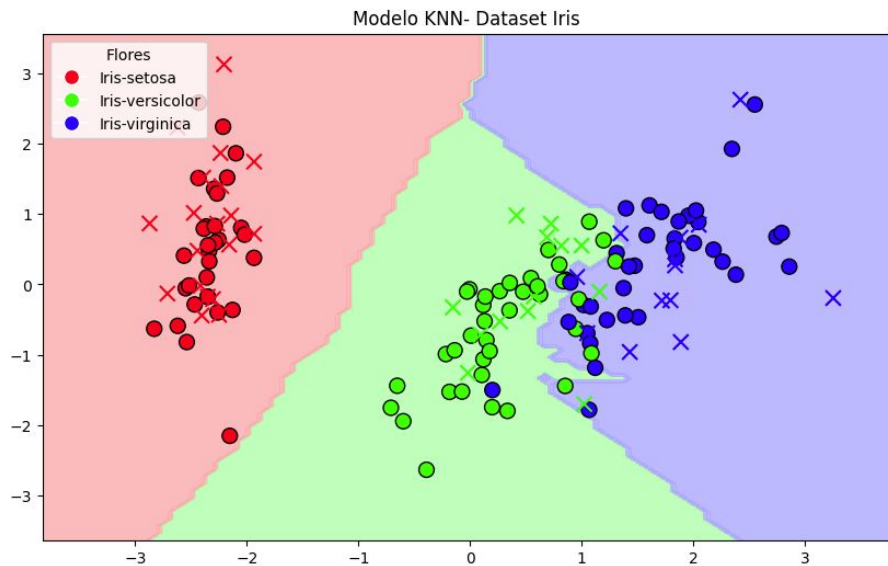


Importância das Features em Porcentagem:

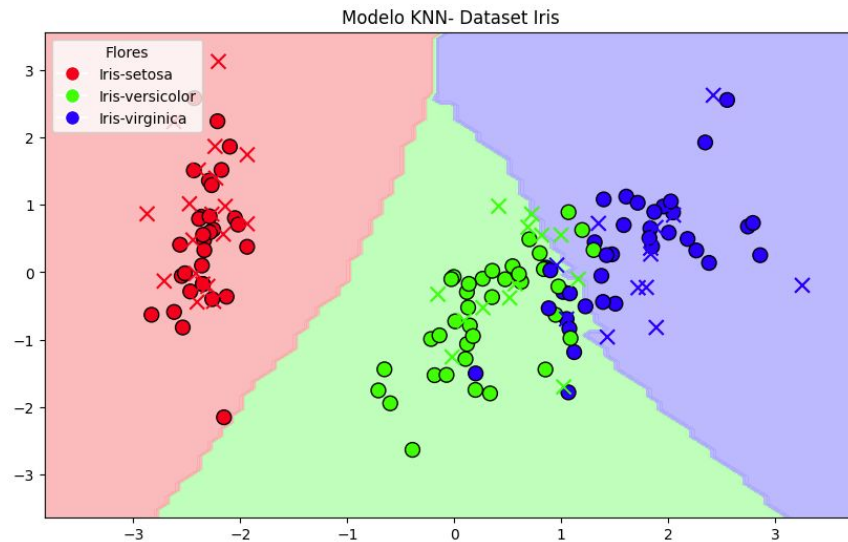
	Feature	Importance (%)
1	SepalWidthCm	35.541888
0	SepalLengthCm	26.594793
3	PetalWidthCm	18.969898
2	PetalLengthCm	18.893421

# Predições do Modelo

K = 3



K = 30



# AGORA É COM VOCÊ!

Experimente alterar alguns parâmetros do modelo, como:

## **Número de Vizinhos (n\_neighbors)**

```
knn = KNeighborsClassifier(n_neighbors=5)
```

## **Método de Distância (metric)**

```
knn = KNeighborsClassifier(metric='manhattan')
```

## **Peso dos Vizinhos (weights)**

Este parâmetro define como os vizinhos são ponderados na decisão final. Se uniform (padrão), todos os vizinhos têm o mesmo peso. Se distance, vizinhos mais próximos têm maior influência.

```
knn = KNeighborsClassifier(weights='distance')
```

## **Tamanho do Conjunto de Treinamento (test\_size)**

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
```

## **Semente Aleatória (random\_state)**

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
```