

# **Moving2DS:**

## Parte 1 – Começando uma Jornada em **Data Science**

ALEX CARNEIRO<sup>1</sup>

1ª versão, 2020

<sup>1</sup><https://www.linkedin.com/in/alex-carneiro-789b9b33/>



Dedicado aos alunos que já tive, com os quais  
aprendi a ensinar.



# Sumário

<b>Lista de Figuras</b>	<b>vii</b>
<b>Lista de Tabelas</b>	<b>ix</b>
<b>1 Introdução</b>	<b>3</b>
1.1 Como saber para onde ir? . . . . .	3
1.2 E agora, para onde iremos? . . . . .	4
1.3 Como chegar lá? . . . . .	4
1.4 Por onde começar? . . . . .	5
<b>2 Conhecendo os Dados</b>	<b>7</b>
2.1 Qual o próximo passo? . . . . .	7
2.2 É para ler os dados? . . . . .	8
2.3 É para visualizar os dados? . . . . .	10
2.4 Entender os dados? Como assim? . . . . .	12
<b>3 Visão Geral</b>	<b>15</b>
3.1 Como pensar em tudo se temos nada? . . . . .	15
3.2 O que mostrar? . . . . .	16
3.3 Como juntar as peças? . . . . .	17
<b>4 Primeiro Teste</b>	<b>19</b>
4.1 Como saber se podemos avançar no nosso caminho? . . . . .	19
4.2 O que vamos servir? . . . . .	20
4.3 Como navegar localmente? . . . . .	20
4.4 Será que os dados chegaram? . . . . .	21
<b>5 Treinando e Avaliando Modelos</b>	<b>23</b>
5.1 O que precisamos aprender? . . . . .	23
5.2 Vamos com linhas retas? . . . . .	23
5.3 Como treinar nosso modelo? . . . . .	25
5.4 Será que o modelo treinou direito? . . . . .	25

<b>6</b>	<b>Teste Final</b>	<b>29</b>
6.1	Como o modelo vai para o sistema? . . . . .	29
6.2	Será que vai dar certo? . . . . .	30
<b>7</b>	<b>Fim da 1ª Parte</b>	<b>31</b>
7.1	Precisamos passar por tudo novamente? . . . . .	31
7.2	O que fazer para continuar minha jornada? . . . . .	31
	<b>Referências Bibliográficas</b>	<b>33</b>

# Lista de Figuras

1.1	Exemplos das 3 espécies de flores contidas no dataset Iris. . .	4
2.1	Interface do ambiente Jupyter. . . . .	7
2.2	Arquivo notebook <i>Data Analysis.ipynb</i> . . . . .	8
2.3	Gráfico de espalhamento. . . . .	11
2.4	Gráfico de espalhamento <b>petal_w</b> por <b>petal_l</b> com região de sobreposição em destaque. . . . .	12
3.1	Modelo genérico para um sistema de processamento de dados.	15
4.1	Página HTML com os campos de dados para o primeiro teste.	20
4.2	Página HTML com o resultado obtido para o primeiro teste. .	21
5.1	Exemplo de resultado para classificação obtido com 2 atributos após o treinamento do algoritmo de Regressão Logística. .	24





# Lista de Tabelas

1.1	Tabela de amostras de dados (medidas em cm). . . . .	5
5.1	Matriz de confusão para o dataset Iris com 30 amostras de teste. . . . .	26
6.1	Valores médios por atributo para cada espécie de flor. . . . .	30



# Prefácio

Este é um material gratuito desenvolvido pelo Prof. Alex Torquato S. Carneiro e utilizado no curso online *Moving to Data Science - Moving2DS*. Por se tratar de um material gratuito, o autor autoriza seu uso por terceiros, mas solicita que sejam feitas as devidas referências sempre que este material for utilizado.

Este material foi desenvolvido como referência para a 1ª parte do curso *Moving2DS*, em que o objetivo é auxiliar profissionais e estudantes que desejam entrar no mercado de ciência de dados. Neste texto, os conceitos de ciência de dados e machine learning são abordados de forma prática, assumindo que o leitor conhece apenas teoria básica de programação e matemática.

Estudaremos o conhecido dataset Iris e iremos utilizá-lo para desenvolver um sistema de classificação de flores com front-end web e uma API Web. Utilizaremos a linguagem Python para o desenvolvimento deste sistema e também iremos abordar as bibliotecas Pandas, Scikit-Learn, Matplotlib, Numpy e Flask, além do ambiente Jupyter Notebook ao longo dos capítulos.

## Sobre o autor

Alex Carneiro possui 15 anos de experiência acadêmica e profissional, tendo passado por grandes empresas: IBM Brasil e LG Electronics do Brasil, e startups brasileiras: CargoX, Idwall, Cognitivo.ai e Hoobox. Em sua passagem por essas empresas, atuou como engenheiro de software, desenvolvedor e cientista de dados, tendo desenvolvido/participado de projetos para:

- previsão de falhas em equipamentos;
- reconhecimento de objetos em ambientes externos;
- reconhecimento de gestos corporais;
- recuperação de imagens em bases de imagens;
- precificação de serviços de frete;
- otimização de alocação de entregas;
- OCR e reconhecimento de fraudes em documentos oficiais;
- reconhecimento de expressões faciais.

No meio acadêmico, tem exercido a função de professor no ensino superior por 10 anos, com passagem pela Universidade Ibirapuera e Faculdade Impacta de Tecnologia. Neste período, ministrou aulas em disciplinas de matemática para computação, programação e análise e processamento de dados. Orientou projetos de conclusão de curso e de iniciação científica, os quais foram publicados na revista da universidade e em conferências nacionais.

Antes de entrar no mercado de trabalho, fez graduação e mestrado em Engenharia de Teleinformática, tendo obtido o título de bacharel em dezembro de 2007 e de mestre em abril de 2010. Durante a graduação, participou de projetos de iniciação científica nas áreas de aprendizado de máquina (*machine learning*), sistemas embarcados e visão computacional. Em sua pesquisa de mestrado, propôs um sistema de reconhecimento de gestos da Língua Brasileira de Sinais - LIBRAS, o qual foi um dos primeiros propostos e avaliados especificamente para a LIBRAS.

## Agradecimentos

Agradeço aos mestres e alunos que já tive, com os quais aprendi a teoria, a prática e a didática. Agradeço também aos colegas de laboratório, de empresa, meus gestores e clientes com quem já trabalhei.

# 1

## Introdução

*“Se você não sabe para onde ir, qualquer lugar serve.”*

– Gato de Cheshire, *Alice no País das Maravilhas*

### 1.1 Como saber para onde ir?

O primeiro passo quando temos um desafio de ciência de dados é definir nosso objetivo. Definir o objetivo significa definir a pergunta que iremos responder através dos dados. Sempre que trabalhamos com dados, devemos ver os dados e as técnicas de machine learning como um meio para responder alguma pergunta. A pergunta que iremos responder dói em quem a faz, por exemplo: um médico faz a pergunta “**qual a doença do meu paciente?**” e enquanto esta pergunta não for respondida, o médico não irá descansar. Então, como podemos responder a pergunta deste médico aflito? Ou, de forma mais geral, como podemos acabar com a dor de alguém que faz uma pergunta?

Para responder uma pergunta, devemos saber qual é a pergunta. A afirmação parece óbvia, mas muitos profissionais cometem o erro de começar a trabalhar com dados e machine learning sem um objetivo, isto porque estes profissionais não têm uma pergunta para responder.

Ao longo deste e dos próximos capítulos, iremos trabalhar com o dataset<sup>1</sup> Iris. Este é o dataset mais conhecido e utilizado para introduzir estudantes no contexto da ciência de dados. Neste dataset, temos dados de flores do gênero Iris catalogadas por R.A. Fisher em 1936.

O dataset Iris está disponível em vários repositórios gratuitos, sendo o *UC Irvine Machine Learning Repository*<sup>2</sup> um dos mais conhecidos.

---

<sup>1</sup>Chamamos qualquer conjunto de dados de dataset, seja um arquivo de texto, uma tabela, uma planilha, etc.

<sup>2</sup>Disponível pela URL: <https://archive.ics.uci.edu/ml/datasets/Iris>.

## 1.2 E agora, para onde iremos?

Nosso objetivo com o dataset Iris é responder a seguinte pergunta:

“qual a espécie da minha flor?”

Devemos imaginar que esta pergunta é feita por um botânico ou por um jardineiro que tem uma flor do gênero Iris, mas não sabe a espécie da flor. Existem dezenas de espécies de flores do gênero Iris [1], porém nosso dataset contém amostras de apenas três espécies de Iris: Iris Setosa, Iris Versicolor e Iris Virgínica [2]. Neste caso, por uma limitação do dataset, somente poderemos identificar a espécie da flor dentro das três espécies contidas no dataset.

A Figura 1.1 apresenta fotos de cada uma das espécies de Iris contidas no dataset. Observando as fotos, é possível compreender a dúvida de quem faz a pergunta “qual a espécie da minha flor?”, pois as flores têm características parecidas como as cores, os formatos, a quantidade de pétalas, etc.



Figura 1.1: Exemplos das 3 espécies de flores contidas no dataset Iris.

Essas semelhanças entre as espécies justifica a dúvida de quem faz a pergunta, mas agora temos que dar o próximo passo: como responder a essa pergunta?

## 1.3 Como chegar lá?

Para responder uma pergunta através de dados, precisamos primeiro entender a pergunta e em seguida conhecer os dados que temos. Nós já temos a pergunta, então agora iremos conhecer nossos dados.

O dataset Iris é formado por uma tabela que contém 5 colunas, na qual as 4 primeiras correspondem a medidas de características físicas das flores e a última corresponde à espécie das flores. Cada linha da tabela corresponde aos dados coletados de uma flor, portanto, como temos 150 linhas na tabela, nosso dataset contém dados de 150 flores diferentes. As características físicas das flores que foram medidas para a elaboração do dataset são: largura da

Tabela 1.1: Tabela de amostras de dados (medidas em cm).

comprimento da sépala	largura da sépala	comprimento da pétala	largura da pétala	Espécie da flor
5,1	3,5	1,4	0,2	Iris-setosa
4,9	3,0	1,4	0,2	Iris-setosa
7,0	3,2	4,7	1,4	Iris-versicolor
6,4	3,2	4,5	1,5	Iris-versicolor
6,3	3,3	6,0	2,5	Iris-virginica
5,8	2,7	5,1	1,9	Iris-virginica

pétala, comprimento da pétala, largura da sépala e comprimento da sépala. A Tabela 1.1 apresenta algumas amostras de linhas contidas no dataset.

Já que estamos lidando com flores, podemos aproveitar para conhecer mais a respeito deste assunto [3]. As pétalas são folhas modificadas e coloridas com a função de atrair os polinizadores, o conjunto de pétalas é chamado de corola. As sépalas ficam localizadas abaixo das pétala e também são folhas modificadas, o conjunto de sépalas é chamado de cálice.

## 1.4 Por onde começar?

Vamos analisar nossa situação:

- temos uma pergunta: “qual a espécie da minha flor?”;
- temos um dataset que contém dados relacionados a flores;
- conhecemos um pouco do contexto de flores e as limitações impostas pelo dataset;
- acreditamos que os dados disponíveis são adequados para responder nossa pergunta.

Agora que já temos conhecimentos e hipóteses sobre o que devemos fazer, vamos começar a analisar os dados. Para nossas análises, precisaremos de ferramentas para:

- ler os dados de um arquivo;
- visualizar os dados através de gráficos;
- processar os dados com operações matemáticas;
- aplicar algoritmos de machine learning para responder nossa pergunta.

Iremos conduzir nossas análises no próximo capítulo, em que aprenderemos mais sobre programação em Python, o ambiente Jupyter Notebook e as bibliotecas: Pandas, Scikit-Learn, Matplotlib, Numpy e Flask.

Todos os dados, códigos e demais materiais abordados estão disponíveis no Github: <https://github.com/alex-carneiro/Moving2DS>. Faça o clone do repositório:

```
git clone https://github.com/alex-carneiro/Moving2DS.git
```



## 2

# Conhecendo os Dados

*“E quando temos de escolher, só esperamos que a nossa escolha seja a certa.”*

– Levi Rivaille, *Attack on Titan*

### 2.1 Qual o próximo passo?

Neste momento, iniciamos o contado com os dados do dataset Iris. Primeiramente, devemos ter acesso ao arquivo *iris.data* que contém os dados. Este arquivo pode ser encontrado no *UC Irvine Machine Learning Repository*, mas já está copiado no repositório Github do curso *Moving2DS*.

Depois de obter o arquivo, devemos escolher as ferramentas que usaremos em nossas análises. A escolha dessas ferramentas é importante para facilitar as análises e permitir que compreendamos os dados com o máximo de liberdade. Para isto, vamos usar o ambiente Jupyter Notebook para nossas análises. O ambiente Jupyter é uma aplicação Web que nos permite criar, executar e compartilhar arquivos, chamados notebooks, que contêm código, equações, gráficos e texto [4]. A Figura 2.1 apresenta a interface do ambiente Jupyter em um navegador Web.



Figura 2.1: Interface do ambiente Jupyter.

Para analisarmos os dados, vamos usar o arquivo *Data Analysis.ipynb*. A Figura 2.2 apresenta o arquivo notebook *Data Analysis.ipynb* aberto no ambiente Jupyter.

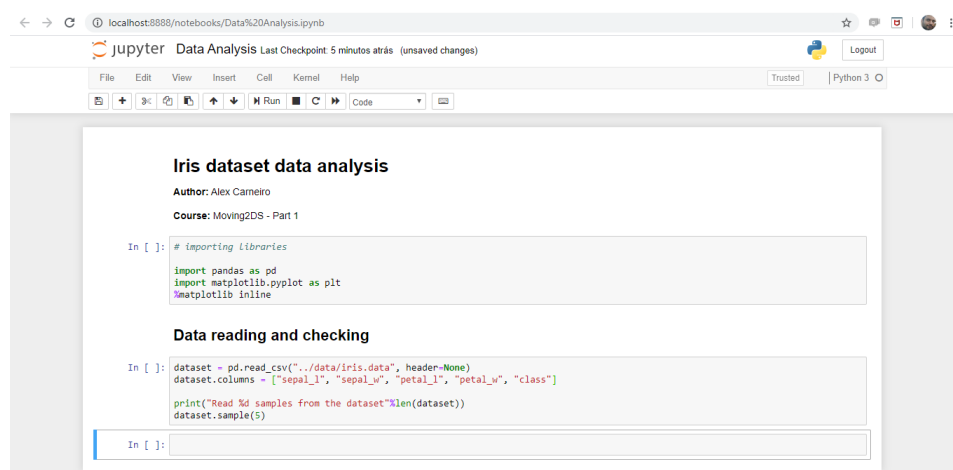


Figura 2.2: Arquivo notebook *Data Analysis.ipynb*.

Com o arquivo *Data Analysis.ipynb* aberto, vamos efetivamente começar a analisar os dados.

## 2.2 É para ler os dados?

Para analisarmos os dados, primeiramente devemos ler os dados dentro do nosso arquivo notebook. Ler os dados significa abrir o arquivo e copiar as informações contidas no arquivo para variáveis do nosso código. Para ler os dados, vamos usar a biblioteca Pandas, que dispõe de várias funções e estruturas de dados que facilitam nossas análises [5]. Então, inserimos a seguinte linha de código para carregar a biblioteca Pandas no nosso notebook:

```
import pandas as pd
```

Após carregar a biblioteca, podemos criar uma variável **dataset** que irá conter os dados lidos do arquivo *iris.data*. Em Python, criamos variáveis inicializando-as com o valor que desejamos que tenham. No caso da leitura do arquivo *iris.data*, desejamos que a variável **dataset** contenha os valores armazenados como uma tabela. O trecho de código a seguir mostra como lemos os dados do dataset com a função **read\_csv()**:

```
dataset = pd.read_csv("../data/iris.data", header=None)
dataset.columns = ["sepal_l", "sepal_w", "petal_l", "petal_w",
                  "class"]
```

A função **read\_csv()** retorna um objeto da classe **DataFrame** que permite a manipulação de dados como uma tabela de forma bastante simples.

O arquivo *iris.data* apresenta uma estrutura semelhante a um arquivo CSV, portanto podemos usar a função `read_csv()`.

Devemos tomar o cuidado de verificar se existe um cabeçalho no arquivo lido, pois não há cabeçalho no caso do *iris.data*. Assim, definimos o parâmetro `header=None` na chamada da função e, na linha seguinte, atribuímos os nomes às colunas do nosso DataFrame.

Uma vez que os dados estão armazenados na variável `dataset`, precisamos verificar os dados:

```
# imprime a quantidade de linhas do dataset
print("Read %d samples from the dataset"%len(dataset))

# imprime uma breve análise sobre a coluna 'class'
print(dataset['class'].describe())

# contabiliza e imprime a quantidade de linhas por classe
# com o nome de cada classe
for c in dataset['class'].unique():
    print("Class", c,
          "has", (dataset['class'] == c).sum(), "samples")
```

Uma verificação muito importante consiste em identificar os tipos de dados de cada coluna e se há colunas com valores nulos:

```
# nomes das colunas no dataset
columns = ["sepal_l", "sepal_w", "petal_l", "petal_w",
           "class"]

# imprime o tipo de dados de cada coluna
for c in columns:
    print("Data type of column", c, "is", dataset[c].dtype)

any_null = [] # lista que identifica qualquer valor nulo

# verificacao se temos algum valor nulo por coluna
for col in columns:
    any_null.append(any(dataset[col].isnull()))

# imprime as colunas com valores nulos
# ou uma mensagem de nenhum valor nulo
if any(any_null):
    print("Those columns have NULL values:",
          [c for c, n in zip(columns, any_null) if n is True])
else:
    print("There are no NULL values")
```

Após executarmos os códigos apresentados, concluímos que nosso dataset tem 150 colunas, 3 classes, exatamente 50 linhas para cada classe, as colunas com medidas contém dados `float64` e não há valores nulos. Agora estamos prontos para a próxima etapa da análise dos dados.

## 2.3 É para visualizar os dados?

Vamos partir agora para a visualização gráfica dos dados, para isto, usaremos a biblioteca Matplotlib [6]. Como estamos usando o ambiente Jupyter, é interessante indicar como desejamos que nossos gráficos sejam apresentados no notebook. Definimos a forma **inline** de exibição dos gráficos, pois ela gera gráficos estáticos e que ocupam menos espaço na interface do Jupyter. De forma mais específica, vamos usar o pacote **pyplot** e a classe **Line2D** que estão disponíveis na biblioteca Matplotlib. Vamos importar os recursos da biblioteca Matplotlib e definir a forma de exibição dos gráficos com o seguinte trecho de código:

```
from matplotlib.lines import Line2D
import matplotlib.pyplot as plt
%matplotlib inline
```

Nossa primeira análise gráfica será feita através de gráficos de espalhamento, em que iremos selecionar duas colunas do dataset para os eixos dos gráficos e usaremos uma cor para cada espécie de flor. Para facilitar a geração dos gráficos, podemos separar cada coluna do objeto **dataset** em uma variável. Um detalhe que merece atenção é que precisamos mapear as strings com os nomes das flores para valores inteiros:

- '*Iris - setosa*'  $\rightarrow 0$ ,
- '*Iris - versicolor*'  $\rightarrow 1$ , e
- '*Iris - virginica*'  $\rightarrow 2$ .

O trecho de código a seguir faz a separação dos dados das colunas do dataset e também mapeia os nomes das espécies das flores em números:

```
# nomes ordenados das classes como lista de strings
cls = sorted(list(dataset['class'].unique()))
# funcao lambda que mapeia as strings para valores int
convert = lambda i: cls.index(i)

# separa cada coluna do dataset em uma variavel
data_sepal_l = dataset['sepal_l'].values
data_sepal_w = dataset['sepal_w'].values
data_petal_l = dataset['petal_l'].values
data_petal_w = dataset['petal_w'].values
data_class = dataset['class'].values
data_class_int = dataset['class'].map(convert).values
```

Vamos usar as novas variáveis para gerar seis gráficos de espalhamento, os quais consistem em todas as combinações de duas colunas. Para cada gráfico, vamos inserir os nomes das colunas nos respectivos eixos, criar o gráfico com os pontos que correspondem às flores medidas para a elaboração

do dataset e inserir a legenda com os nomes das espécies com suas respectivas cores.

Para a criação das legendas, precisamos de um mapa de cores (colormap) e as strings associadas a cada mapa. O trecho de código a seguir nos fornece esses recursos:

```
cmap = plt.cm.get_cmap('Set1')
legend_elements = [Line2D([0], [0],
                           marker='o',
                           color=cmap(i/(len(cls)-1)),
                           label=c) for i, c in enumerate(cls)]
```

Por fim, vamos gerar os gráficos, para isto usaremos o seguinte trecho de código para cada gráfico:

```
plt.xlabel("sepal_l")
plt.ylabel("sepal_w")
plt.scatter(data_sepal_l, data_sepal_w,
            c=data_class_int, cmap=cmap)
plt.legend(handles=legend_elements, loc=1)
```

O código anterior cria o gráfico com as colunas **sepal\_l** e **sepal\_w**, então devemos trocar os nomes dos eixos e das variáveis da função **scatter()** para criarmos os demais gráficos. O resultado final que devemos obter está apresentado na Figura 2.3.

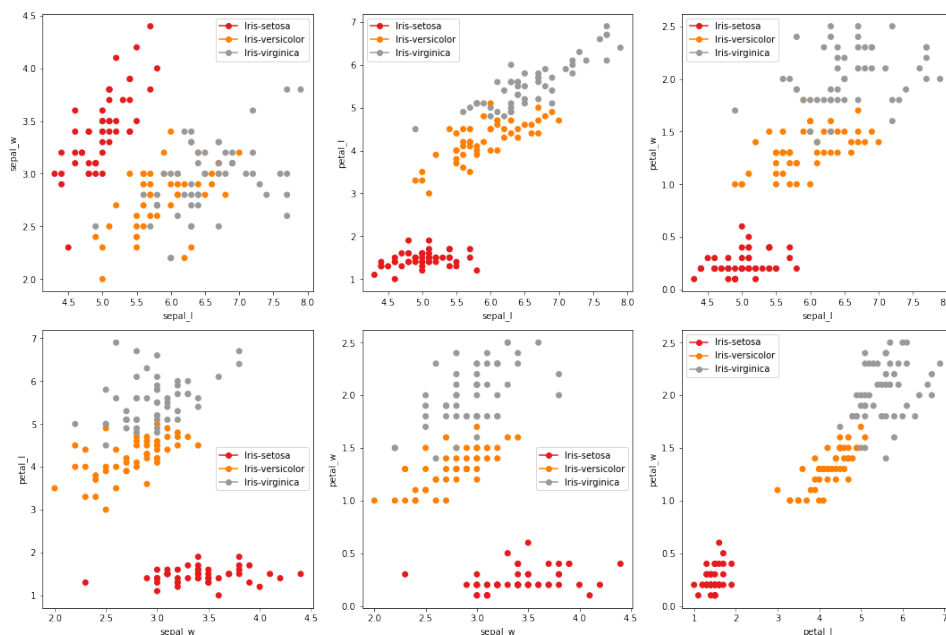


Figura 2.3: Gráfico de espalhamento.

Com esses gráficos, podemos explorar as características dos dados e assim escolher como automatizar o reconhecimento da espécie da flor com base nas

medidas de largura e comprimento da pétala e da sépala.

## 2.4 Entender os dados? Como assim?

A primeira observação que fazemos é que a classe **‘Iris-setosa’** está totalmente isolada das demais, isso fica bastante evidente nos gráficos, mas de forma mais acentuada no gráfico de **petal\_w** por **petal\_l** em que os pontos que correspondem a flores **‘Iris-setosa’** estão muito agrupados.

Outra observação é que as classes **‘Iris-versicolor’** e **‘Iris-virginica’** apresentam um pouco de sobreposição nos pontos. Esta região está destacada na Figura 2.4.

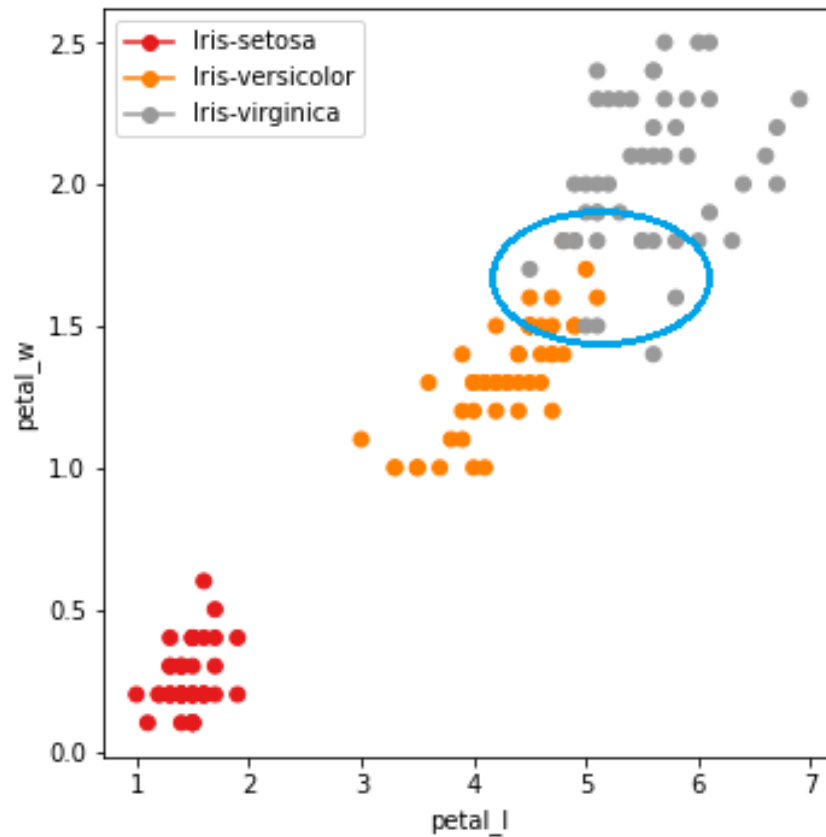


Figura 2.4: Gráfico de espalhamento **petal\_w** por **petal\_l** com região de sobreposição em destaque.

Além desta análise visual dos dados, que fizemos através da observação dos gráficos, também podemos fazer algumas análises estatísticas, por exemplo com os valores de média e variância das medidas para cada espécie de flor. O trecho de código a seguir usa a biblioteca Numpy para calcular a

média e a variância dos dados de cada coluna do dataset, após separar cada uma das classes:

```
# separando as classes de especies de flores
data_setosa = dataset[dataset['class']== 'Iris-setosa']
data_versicolor = dataset[dataset['class']== 'Iris-versicolor']
data_virginica = dataset[dataset['class']== 'Iris-virginica']

columns = ["sepal_l", "sepal_w", "petal_l", "petal_w"]

print("Mean and variance for class 'Iris-setosa'")
for c in columns:
    mean = np.mean(dataset_setosa[c])
    var = np.var(dataset_setosa[c], ddof=1)

    print("Column:", c)
    print("Mean value =", mean)
    print("Variance value =", var)
```

No trecho de código anterior, analisamos apenas a classe **'Iris-setosa'**, mas o mesmo deve ser feito para as demais classes. Os resultados da média e variância mostram que as pétalas das flores **'Iris-setosa'** são bem menores que das outras espécies, o que já era esperado pela observação dos gráficos. Também percebemos que as medidas das sépalas apresentam alguma variação entre as classes, sendo assim, todas as colunas têm algum potencial de discriminar a espécie das flores.

Agora que conhecemos o dataset, vamos elaborar um sistema de classificação das flores.





## 3

# Visão Geral

*“Às vezes a gente tem que se distanciar do papel pra conseguir enxergar o desenho todo com mais clareza.”*

– Edgar, 2 Coelhos

### 3.1 Como pensar em tudo se temos nada?

O desenvolvimento de um sistema computacional, independente de envolver ciência de dados/machine learning ou não, envolve um exercício mental por parte da equipe de desenvolvimento no sentido de pensar em tudo o que o sistema deve realizar. No caso de sistemas de classificação, podemos começar nosso exercício mental com o fluxo dos dados.

As variáveis de entrada do nosso sistema (*input data*), isto é, os dados que iremos fornecer ao sistema, são as medidas de comprimento e largura da pétala e da sépala de uma flor. A variável de saída do sistema (*output data*), isto é, a resposta que vamos receber do sistema, é a espécie da flor.

Podemos usar um modelo bastante simples de entrada e saída para montarmos a estrutura do nosso sistema, como apresentado na Figura 3.1. Neste modelo, precisamos de dois elementos para entrada / saída dos dados e um terceiro elemento de processamento.



Figura 3.1: Modelo genérico para um sistema de processamento de dados.

Para os elementos de entrada e saída, vamos criar páginas HTML simples que contêm as informações que desejamos receber ou fornecer ao usuário do nosso sistema.

### 3.2 O que mostrar?

Vamos criar um arquivo HTML para as variáveis de entrada entrada e outro para a variável de saída. O arquivo de entrada deve conter uma orientação sobre quais dados nosso sistema espera receber e os campos correspondentes aos dados. Segue um exemplo de como podemos elaborar o arquivo HTML para as variáveis de entrada de forma simples:

```
<!DOCTYPE html>
<html>
<body>

<h1>Classificacao de flores</h1>
<p>Digite as medidas da flor.</p>
<p>As medidas ser feitas em centimetros.</p>
<p>Exemplos: 1.2, 2.5, 3.2, 4.9</p>

<form action="/predict">
  Comprimento da sepala: <input type="text" name="sepal_l"><br>
  Largura da sepala: <input type="text" name="sepal_w"><br>
  Comprimento da petala: <input type="text" name="petal_l"><br>
  Largura da petala: <input type="text" name="petal_w"><br>
  <input type="submit" value="Enviar">
</form>

</body>
</html>
```

Para o arquivo HTML que irá conter as informações fornecidas ao usuário, precisamos apresentar os dados recebidos e o resultado obtido. É interessante apresentar os dados recebidos para que o usuário tenha certeza que o sistema não fez confusão com seus dados e forneceu um resultado baseado em valores errados. Segue um exemplo de como podemos elaborar o arquivo HTML com o resultado de forma simples:

```
<!DOCTYPE html>
<html>
<body>

<h1>Classificacao de flores</h1>
<p>A flor com <b>comprimento da sepala = {{sepal_l}}cm</b>,
      <b>largura da sepala = {{sepal_w}}cm</b>,
      <b>comprimento da petala = {{petal_l}}cm</b> e
      <b>largura da petala={{petal_w}}cm</b>
      pertence a especie <b>{{result}}</b>.</p>

</body>
</html>
```

Caso deseje melhorar as páginas propostas, podemos inserir elementos interativos, gráficos, etc. Porém, estas páginas contêm o suficiente para nosso sistema sem utilizar recursos mais avançados, sendo portanto mais fácil de se entender por todos.

### 3.3 Como juntar as peças?

Os arquivos HTML para as variáveis de entrada e de saída foram salvos como *index.html* e *result.html*, respectivamente. Os arquivos encontram-se no diretório *templates/* e serão acessados pelo nosso código servidor, que corresponde ao elemento de processamento do nosso modelo genérico apresentado na Figura 3.1.

Para nosso servidor, usaremos a biblioteca Flask que contém vários recursos Web e não exige grandes esforços no desenvolvimento de aplicações. Nosso servidor deve ser capaz de realizar as seguintes ações: receber a requisição do usuário, apresentar a página HTML, receber os dados da flor, processar esses dados, estimar a espécie da flor e enviar a página HTML com o resultado da espécie da flor. O código a seguir é um exemplo de como podemos implementar nosso servidor com os requisitos que definimos:

```
from flask import Flask, render_template, request

app = Flask(__name__)

# funcao que recebe a requisicao do usuario e
# envia o arquivo 'index.html'
@app.route('/')
def home():
    return render_template('index.html')

# funcao que recebe os dados da flor, estima sua especie e
# envia o arquivo 'result.html' com o resultado
@app.route('/predict')
def predict():
    sepal_l = request.args.get('sepal_l')
    sepal_w = request.args.get('sepal_w')
    petal_l = request.args.get('petal_l')
    petal_w = request.args.get('petal_w')
    result = 'Iris-setosa'
    # este resultado deve ser obtido com os dados:
    # [sepal_l, sepal_w, petal_l, petal_w]

    return render_template('result.html',
                           sepal_l=sepal_l, sepal_w=sepal_w,
                           petal_l=petal_l, petal_w=petal_w,
                           result=result)

if __name__ == '__main__':
    app.run() # execucao do servidor
```

Neste momento, temos apenas um servidor que sempre apresenta o mesmo resultado para todas as flores. Esta solução não é útil para nosso usuário, mas servirá com base para o sistema completo que desenvolveremos. Por hora, vamos testar nosso servidor para depois continuarmos com a ciência de dados.



## 4

# Primeiro Teste

*“Para conseguir o que quer, você deve olhar além do que você vê.”*

– Rafiki, *O Rei Leão 3*

### 4.1 Como saber se podemos avançar no nosso caminho?

Antes de levantar as paredes de uma casa, primeiro devemos construir seus fundamentos. Esta é a ideia que devemos ter em mente para testar nosso servidor. Mesmo que ainda não exista um modelo de classificação das flores, precisamos verificar se temos a base de nosso sistema funcionando.

Para verificarmos a base do sistema, vamos fazer nosso primeiro teste. Os objetivos deste teste são:

- executar o servidor localmente;
- acessar o servidor pelo navegador Web;
- enviar os dados de uma flor para o servidor; e
- receber o resultado da espécie da flor<sup>1</sup>.

Para nosso teste, devemos garantir que nosso computador tenha o interpretador Python<sup>2</sup> instalado e acesso ao terminal de comandos: CMD para Windows ou Bash para sistemas operacionais Unix-like, como Linux e Mac.

Com isso pronto, a primeira etapa do nosso teste é executar o servidor.

---

<sup>1</sup>Lembrando que não estamos avaliando a classificação da flor.

<sup>2</sup>Recomendamos versão 3.6 ou superior.

## 4.2 O que vamos servir?

Para executarmos nosso servidor, precisamos conhecer um pouco de como a biblioteca Flask funciona. Ao criarmos um servidor baseado em Flask, este pode ser executado localmente e irá criar uma aplicação que atende requisições HTTP na porta 5000 do *localhost*.

A biblioteca Flask não é padrão do Python, então devemos instalá-la isoladamente:

```
# uso do pip3 para instalar a biblioteca para o Python3
pip3 install Flask
```

O arquivo do servidor está disponível no repositório Github do curso *Moving2DS*, então podemos executá-lo da seguinte forma:

```
alexc:~/Moving2DS$ cd Parte1/
alexc:~/Moving2DS/Parte1$ python3 server.py
```

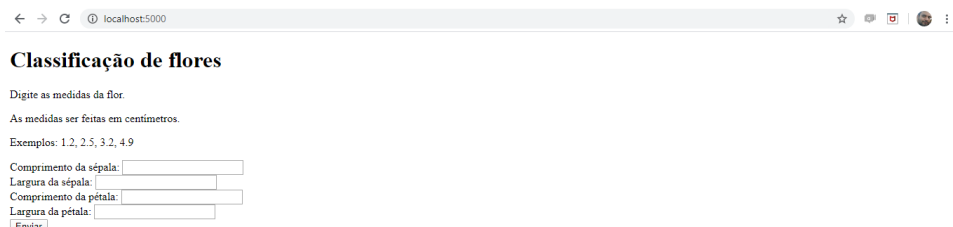
Após executar o servidor, devemos ver as seguintes mensagens:

```
* Serving Flask app "server" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a
  production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Com a verificação destas mensagens, podemos partir para o próximo objetivo do primeiro teste: acessar o servidor pelo navegador Web.

## 4.3 Como navegar localmente?

Com o servidor em execução, podemos acessá-lo através do navegador Web com a URL: <http://localhost:5000/>. Com esta URL, acessamos nosso próprio servidor através da porta 5000. A Figura 4.1 apresenta como devemos ver a página HTML que desenvolvemos para receber os dados fornecidos pelo usuário.



The screenshot shows a web browser window with the address bar set to [localhost:5000](http://localhost:5000/). The page title is "Classificação de flores". The content includes instructions to enter flower measurements in centimeters, with examples: 1.2, 2.5, 3.2, 4.9. There are four input fields: "Comprimento da sépala:", "Largura da sépala:", "Comprimento da pétala:", and "Largura da pétala:". An "Enviar" button is at the bottom left.

Figura 4.1: Página HTML com os campos de dados para o primeiro teste.

Como estamos no primeiro teste, podemos inserir quaisquer valores nos campos da página. Para exemplificar, vamos considerar que todas as medidas sejam 1.2 cm, assim podemos verificar o terceiro objetivo do primeiro teste: enviar os dados de uma flor para o servidor. Podemos colocar quaisquer valores nos campos, o valor 1.2 foi escolhido apenas como exemplo. Após inserir os dados nos campos, podemos clicar em **Enviar**.

## 4.4 Será que os dados chegaram?

Como a nossa aplicação devolve os dados inseridos na página com o resultado, poderemos verificar se os dados foram enviados ao servidor conferindo a página com o resultado. A Figura 4.2 apresenta como devemos ver a página HTML com o resultado da classificação da flor e os dados fornecidos pelo usuário.



Figura 4.2: Página HTML com o resultado obtido para o primeiro teste.

Nosso primeiro teste é encerrado com sucesso se verificarmos a página com o resultado de classificação **‘Iris-setosa’**, o qual foi definido no código do servidor, e os valores inseridos.

Como este servidor consiste na base da nossa aplicação, podemos seguir para a elaboração do modelo de machine learning para classificação das flores.





## 5

# Treinando e Avaliando Modelos

*“Hoje é um bom dia para tentar.”*

– Quasimodo, *O Corcunda de Notre Dame*

### 5.1 O que precisamos aprender?

Modelos de machine learning são algoritmos que resolvem problemas genéricos, como classificação ou agrupamento, cujos parâmetros foram ajustados para se adequar a um conjunto de dados específico. O processo de ajustar os parâmetros do algoritmo é chamado treinamento do modelo. O conjunto de dados específico que usaremos é o dataset Iris.

Nosso objetivo agora é treinar um modelo de machine learning com o dataset Iris para obter um modelo capaz de classificar flores com base nos dados que estão contidos no dataset. Então, usaremos algoritmos de machine learning da biblioteca Scikit-Learn para nosso modelo [8].

Existem vários algoritmos de classificação diferentes, por exemplo: Regressão Logística, Redes Neurais, Máquinas de Vetor Suporte, Árvores de Decisão e K Vizinhos Mais Próximos. Cada um desses algoritmos tem suas particularidades, as quais impactam no tempo de resposta do modelo depois de treinado e também na forma como a classificação é feita.

### 5.2 Vamos com linhas retas?

Dentre os vários algoritmos de classificação disponíveis na biblioteca Scikit-Learn, vamos utilizar a Regressão Logística. Escolhemos este algoritmo devido à facilidade de compreendermos seu funcionamento após o treinamento: a classificação dos dados é feita separando o espaço dos atributos (colunas do dataset) através de retas.

Conforme vimos na análise dos dados do dataset Iris, os pontos que correspondem às flores ficam agrupados no espaço de atributos de acordo com suas classes, onde há pouca sobreposição entre duas das classes. Porém, para separarmos cada classe das demais, precisamos de retas diagonais, o que é possível de obtermos com a Regressão Logística. A Figura 5.1 apresenta um exemplo de como a Regressão Logística pode separar o espaço de dois atributos com retas de modo a minimizar o erro de classificação.

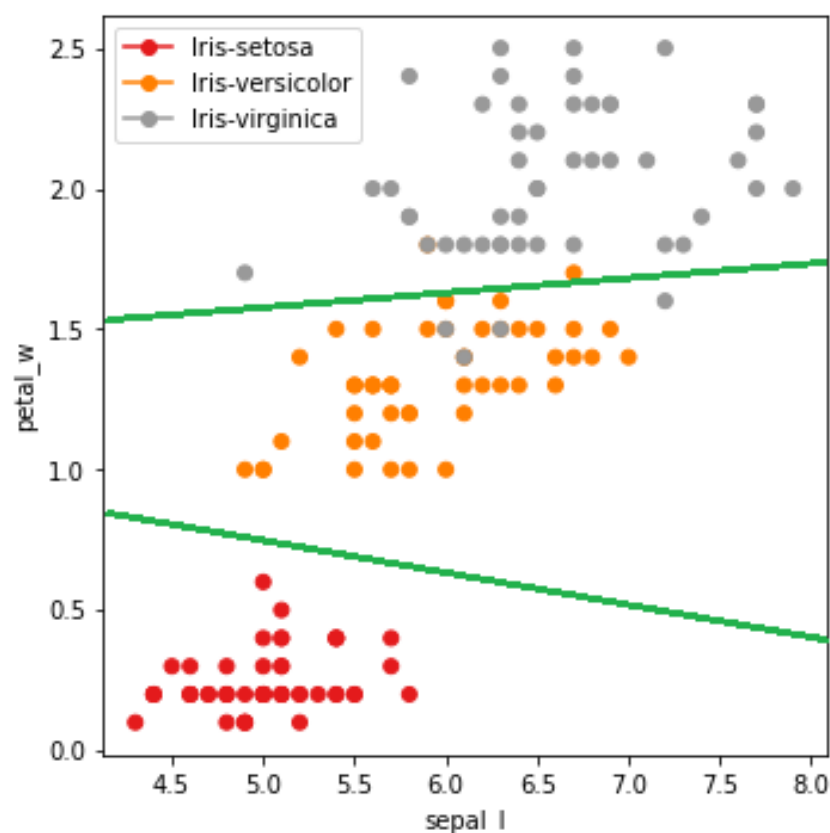


Figura 5.1: Exemplo de resultado para classificação obtido com 2 atributos após o treinamento do algoritmo de Regressão Logística.

No exemplo da Figura 5.1, temos todos os dados do dataset sendo separados em três classes por duas retas. As retas foram posicionadas manualmente para ilustrar um possível resultado. No contexto de algoritmos de classificação, chamamos as retas que separam as classes de curvas de decisão, pois são através delas que decidimos qual a classe de cada nova amostra de dados que ainda não foi classificada.

## 5.3 Como treinar nosso modelo?

Agora que já escolhemos um algoritmo para treinar, vamos abrir o arquivo notebook *Model training and evaluation.ipynb* no qual faremos o treinamento do nosso modelo. Este arquivo está no diretório *notebooks/*.

Mas, antes de treinar o modelo devemos separar nossos dados em dois conjuntos: treinamento e teste. Esta separação servirá para simularmos o uso do nosso modelo com dados diferentes dos que foram usados no treinamento. Assim, avaliaremos a capacidade de generalização do modelo após o treinamento. Se usarmos os mesmos dados para treinamento e teste, não identificaremos se nosso modelo for capaz de classificar corretamente apenas os dados de treinamento. A biblioteca Scikit-Learn tem a função **train\_test\_split()** que faz a separação dos dados nos conjuntos de treinamento e teste:

```
full_data_input = dataset[["sepal_l", "sepal_w",
                           "petal_l", "petal_w"]].values
full_data_output = dataset["class"].values

train_input, test_input, \
train_output, test_output = train_test_split(full_data_input,
                                             full_data_output,
                                             test_size=.2)
```

No código apresentado, separamos 20% dos dados do dataset para teste e 80% para treinamento. Os dados de treinamento e teste são selecionados aleatoriamente no dataset completo.

Agora sim, faremos o treinamento do nosso modelo. As classes que contêm os algoritmos de classificação na biblioteca Scikit-Learn utilizam o método **fit()** para o treinamento. Segue o treinamento do nosso modelo:

```
model = LogisticRegression()
model.fit(train_input, train_output)
```

Caso a chamada do método **fit()** não tenha erros, veremos uma mensagem semelhante a essa:

```
LogisticRegression(C=1.0, class_weight=None, dual=False,
                    fit_intercept=True, intercept_scaling=1,
                    max_iter=100, multi_class='warn',
                    n_jobs=None, penalty='l2',
                    random_state=None, solver='warn',
                    tol=0.0001, verbose=0, warm_start=False)
```

Nesta mensagem, temos os parâmetros default para inicialização da classe **LogisticRegression**.

## 5.4 Será que o modelo treinou direito?

Depois que nosso modelo está treinado, devemos sempre levantar a questão: “Posso confiar no modelo que treinei?” Apenas depois de termos evidências

de que nosso modelo é confiável poderemos integrá-lo ao nosso sistema.

Para gerar essas evidências, usaremos os dados do conjunto de teste para obter duas informações: a acurácia e a matriz de confusão do modelo. A acurácia de um modelo corresponde à taxa esperada de acerto do modelo considerando todas as classes. Podemos obter essa medida através do método `score()`:

```
acc = model.score(test_input, test_output)
print("Model accuracy = %.1f%%"%(100 * acc))
```

Nosso modelo fornece uma acurácia de 96,7%, o que é um interessante para apresentarmos ao usuário do nosso sistema. Porém, apenas a acurácia não é informação suficiente, pois não sabemos quais os erros mais prováveis.

Para podermos identificar esses erros mais prováveis, podemos obter a matriz de confusão do modelo treinado a partir dos dados de teste. Para obtermos essa matriz, iremos fazer as predições do modelo com as variáveis de entrada dos dados de teste e usar a função `confusion_matrix()` disponível na biblioteca Scikit-Learn:

```
predictions = model.predict(test_input)
matrix = confusion_matrix(predictions, test_output)
print(matrix)
```

Obtivemos o seguinte resultado como matriz de confusão:

```
[[12  0  0]
 [ 0  9  0]
 [ 0  1  8]]
```

Para formatar a matriz de confusão de forma mais compreensível, podemos usar o seguinte trecho de código:

```
classes = sorted(np.unique(test_output))
pd.DataFrame(dict([(cl, 1) for cl, 1 in zip(classes,
                                           matrix.T)]),
              index=classes)
```

Usando esta formatação, a matriz obtida será semelhante à matriz apresentada na Tabela 5.1.

Tabela 5.1: Matriz de confusão para o dataset Iris com 30 amostras de teste.

	Iris-setosa	Iris-versicolor	Iris-virginica
Iris-setosa	12	0	0
Iris-versicolor	0	9	0
Iris-virginica	0	1	8

Nesta matriz, temos os valores reais das classes dos dados de teste nas linhas e os valores fornecidos pelo modelo nas colunas. Concluímos que nosso modelo cometeu um erro ao confundir uma flor ‘**Iris-virginica**’ classificando-a como ‘**Iris-versicolor**’.

Cada vez que fizermos a separação dos dados em treinamento e teste, os resultados de acurácia e a matriz de confusão serão diferentes, então uma forma de obtermos acurácia e matriz de confusão confiáveis é repetir o processo de treinamento e avaliação do modelo várias vezes e calcular as médias para a acurácia e para os elementos da matriz de confusão.

Com o modelo treinado e avaliado, podemos agora integrá-lo ao sistema.



## 6

# Teste Final

*“A minha melhor regra é esperarmos o melhor e praparamo-nos para o pior.”*

– Ezra Kramer, *Ultimato*

### 6.1 Como o modelo vai para o sistema?

Para utilizar o modelo gerado no notebook em outros códigos, precisamos salvar o modelo treinado em um arquivo. Este processo de salvar o modelo treinado para uso posterior é denominado persistência do modelo.

Podemos salvar um modelo treinado da biblioteca Scikit Learn com a função **dump()** da biblioteca joblib. A chamada da função **dump()** é feita assim:

```
# salva o modelo treinado no diretorio 'models/'
joblib.dump(model, '../models/model.pkl');
```

Depois de salvar o modelo como o arquivo *model.pkl*, podemos verificar o arquivo no diretório. Caso o arquivo *model.pkl* esteja lá, vamos agora para o arquivo *server.py* para inserir o carregamento do modelo treinado.

Para carregar o modelo, usamos a função **load()** da biblioteca joblib. A chamada da função **load()** é feita assim:

```
# carrega o modelo treinado do diretorio 'models/'
model = joblib.load('models/model.pkl');
```

O objeto **model** está definido como uma variável fora da função **predict()** do nosso servidor para que o carregamento ocorra apenas quando o servidor é iniciado.

Dentro do arquivo *server.py*, devemos montar a entrada do modelo com os dados enviados pelo usuário. Essa entrada consiste em um objeto array, da biblioteca Numpy. É importante lembrar que os dados são enviados ao servidor como strings, então devemos convertê-los para float ao montarmos o array de entrada:

```
# dados de entrada convertidos para float
# e estruturados como um array
input_data = np.array([[float(sepal_l),
                        float(sepal_w),
                        float(petal_l),
                        float(petal_w)]])
```

Agora, iremos usar nosso modelo para estimar a espécie mais provável para a flor:

```
# 'result' contém o nome da espécie fornecida pelo modelo
result = model.predict(input_data)[0]
```

Com isso, concluímos nosso sistema de classificação de flores.

## 6.2 Será que vai dar certo?

Agora vamos executar o servidor e fazer nosso teste final. Primeiro, para verificar se o servidor continua funcionando, vamos novamente acessar a URL <http://localhost:5000/>. Caso a página HTML para entrada dos dados seja exibida, podemos continuar com nosso teste.

Nosso objetivo neste teste final é verificar se o sistema faz classificações de forma coerente com os dados do dataset Iris. Sendo assim, vamos tomar medidas que correspondam a padrões seguramente adequados às espécies de flores. Para obtermos estas medidas, podemos voltar à análise dos dados e identificar valores adequados pelos gráficos ou pelas médias de cada atributo por classe. Os valores médios por atributo para cada classe estão apresentados na Tabela 6.1.

Tabela 6.1: Valores médios por atributo para cada espécie de flor.

Espécie	Comprimento da sépala	Largura da sépala	Comprimento da pétala	Largura da pétala
Iris-setosa	5.0	3.4	1.4	0.2
Iris-versicolor	5.9	2.7	4.2	1.3
Iris-virginica	6.5	2.9	5.5	2.0

Ao inserir os dados no nosso sistema, observamos que este fornece os resultados esperados, portanto nosso teste final foi bem sucedido. Para melhorar ainda mais nossas classificações, podemos usar todo o dataset para treinamento do modelo, o que deve torná-lo mais robusto. Quanto ao sistema, podemos incrementá-lo para aceitar separações tanto por vírgula como por ponto nos dados inseridos.

Mas, o maior legado que podemos obter com nosso sistema é o conhecimento que adquirimos ao desenvolvê-lo. E, não menos importante, a possibilidade de usá-lo como base para outros sistemas.



## 7

# Fim da 1ª Parte

*“Para a pessoa errada você nunca terá valor nenhum, mas para a pessoa certa você será tudo.”*

– Reverendo Dave, *Deus não está morto*

### 7.1 Precisamos passar por tudo novamente?

O caminho que percorremos para desenvolver este sistema até que foi breve, mas foi intenso. Porém, é preciso saber que entrar na área de ciência de dados é uma aventura sem fim. Novos algoritmos, técnicas de treinamento, problemas e aplicações são apresentados constantemente.

Tudo o que há em nossa volta pode ser medido, transformado em dados e analisado. O sistema que desenvolvemos pode parecer simples, mas será útil para aprendermos e começarmos a estudar ciência de dados na prática.

Mas, a cada novo aprendizado, percebemos como tudo faz um pouco mais de sentido. Cada vez que entendemos como um algoritmo funciona, será mais fácil entender como os próximos algoritmos estudados funcionam. Cada vez que aprendemos sobre um novo problema, conseguimos amadurecer nossa forma de pensar e isso faz com que encontremos perguntas cada vez mais certas.

### 7.2 O que fazer para continuar minha jornada?

Os nossos próximos passos serão: estudar, codificar e perguntar. Nas próximas partes do curso *Moving2DS*, iremos aprender a lidar com problemas mais complexos, com dezenas de atributos. Aprenderemos a trabalhar com textos, com imagens e com vídeos.

Ainda temos muito o que aprender e nunca aprenderemos tudo, mas poderemos ajudar outras pessoas através dos resultados de nossos esforços. Procure por dados disponíveis em suas áreas de interesse, sejam quais forem:

esportes, culinária, viagens, astronomia, biologia, etc. Use esses dados para elaborar suas perguntas, tente respondê-las e estude o que precisar para chegar aos seus objetivos.

Se desejar, podemos manter contato pelo grupo no Linked In criado para conectar quem desejar trocar experiências sobre Data Science:

<https://www.linkedin.com/groups/8919447/>.

Até a próxima!

Paz e bem!

# Referências Bibliográficas

- [1] Wikipedia (2019) *Iris (género)*, Disponível Online; acessado em 21 de março de 2020. [https://pt.wikipedia.org/w/index.php?title=Iris\\_\(g%C3%A9nero\)&action=history](https://pt.wikipedia.org/w/index.php?title=Iris_(g%C3%A9nero)&action=history)
- [2] R.A. Fisher (1936) *Iris Plants Database*, Disponível Online; acessado em 21 de março de 2020. <https://archive.ics.uci.edu/ml/datasets/Iris>
- [3] Lana Magalhães (2018) *Tipos de flores e suas funções*, Disponível Online; acessado em 21 de março de 2020. <https://www.todamateria.com.br/tipos-de-flores-e-suas-funcoes/>
- [4] Jupyter (2020) *Jupyter*, Disponível Online; acessado em 21 de março de 2020. <https://jupyter.org/>
- [5] Pandas (2020) *pandas - Python Data Analysis Library*, Disponível Online; acessado em 21 de março de 2020. <https://pandas.pydata.org>
- [6] Matplotlib (2020) *Matplotlib: Visualization with Python*, Disponível Online; acessado em 21 de março de 2020. <https://matplotlib.org/>
- [7] NumPy (2020) *NumPy*, Disponível Online; acessado em 21 de março de 2020. <https://numpy.org/>
- [8] SKLearn (2020) *scikit-learn: Machine Learning in Python*, Disponível Online; acessado em 21 de março de 2020. <https://scikit-learn.org/>
- [9] Joblib (2020) *Joblib: running Python functions as pipeline jobs*, Disponível Online; acessado em 21 de março de 2020. <https://joblib.readthedocs.io/>