MARCELO MIKY MINE

MACHINE LEARNING ENGINEERING

APRENDIZADO SUPERVISIONADO

AULA 03

PDF exclusivo para Marcio Anastacio da Luz Junior - rm354541
marcioluzj@gmail.com

SUMÁRIO

O QUE VEM POR AÍ? 3
HANDS ON 4
SAIBA MAIS 5
MERCADO, CASES E TENDÊNCIAS 27
O QUE VOCÊ VIU NESTA AULA? 28
REFERÊNCIAS 29

FIAP

PDF exclusivo para Marcio Anastacio da Luz Junior - rm354541
marcioluzj@gmail.com

O QUE VEM POR AÍ?

No universo do aprendizado supervisionado, a tarefa de classificação se destaca como um pilar fundamental. Nela, modelos computacionais são treinados para identificar a qual categoria um determinado dado pertence, seja um e-mail como spam ou não, uma imagem como um gato ou um cachorro ou até mesmo um cliente com alto potencial de compra ou não.

Para dominar esta técnica, diversos modelos de aprendizado supervisionado foram desenvolvidos e aprimorados ao longo dos anos. Cada um com suas características, vantagens e desvantagens, esses modelos se tornaram ferramentas essenciais para empresas dos mais diversos setores.

FIAP

PDF exclusivo para Marcio Anastacio da Luz Junior - rm354541
marcioluzj@gmail.com

HANDS ON

Nesta aula iremos finalmente aplicar um algoritmo de aprendizado supervisionado. Após realizarmos todos os tratamentos, os dados estarão prontos para servir de entrada para o algoritmo. Dentre eles temos a Regressão Logística, K Vizinhos Mais Próximos (KNN), Árvore de Decisão, Support Vector Machines (SVM) e Redes Neurais Artificiais.

FIAP

PDF exclusivo para Marcio Anastacio da Luz Junior - rm354541
marcioluzj@gmail.com

SAIBA MAIS

A Regressão Logística é amplamente usada para tarefas de classificação binária, em que o objetivo é prever a probabilidade de um dado pertencer a uma das duas classes possíveis. O algoritmo K Vizinhos Mais Próximos (KNN) é um método de aprendizado supervisionado no qual a classe de um novo dado é determinada pela classe da maioria de seus K vizinhos mais próximos no espaço de características. Iremos discutir de que forma obter o melhor valor para K.

Já a Árvore de Decisão constrói uma estrutura hierárquica de regras que dividem os dados em subconjuntos com base em características específicas, resultando em um modelo fácil de interpretar e visualizar. As Support Vector Machines (SVMs) constroem um hiperplano que separa as classes no espaço de features, maximizando a margem entre o hiperplano e os pontos de dados mais próximos (vetores de suporte).

As arquiteturas mais simples de uma Rede Neural Artificial são o Perceptron e o Multilayer Perceptron (MLP). O Perceptron é um modelo simples que aprende a separar duas classes em um espaço bidimensional através de uma reta, ajustando os pesos de suas conexões para minimizar o erro entre suas previsões e os dados reais. O Multilayer Perceptron (MLP) é uma rede neural artificial com múltiplas camadas de neurônios interconectados. Ele é capaz de aprender funções complexas e não lineares, tornando-o mais poderoso que o Perceptron para lidar com diversos tipos de problemas.

Regressão Logística

Também conhecida como Classificador Logístico, é um modelo clássico e robusto que se baseia tanto na probabilidade para classificar dados binários (duas classes) como pode ser estendida para lidar com mais de duas classes. Através de uma função matemática, ele estima a probabilidade de um dado pertencer a uma das classes.

Vantagens:

- **Simplicidade:** fácil de implementar e interpretar, ideal para iniciantes.
- **Robustez:** desempenho consistente em diversos datasets.

PDF exclusivo para Marcio Anastacio da Luz Junior - rm354541
marcioluzj@gmail.com

- **Eficiência:** treinamento rápido e com baixo custo computacional.

Desvantagens:

- **Linearidade:** assume uma relação linear entre as variáveis e a classe alvo, o que pode limitar seu desempenho em casos com relações complexas.
- **Alta Dimensionalidade:** pode ter dificuldade em lidar com datasets com muitas variáveis.

Aplicações:

- **Diagnóstico médico:** prever a probabilidade de um paciente ter uma doença.
- **Análise de risco de crédito:** avaliar a probabilidade de inadimplência de um cliente.
- **Detecção de fraude:** identificar transações fraudulentas em cartões de crédito.

Neste exemplo, exploraremos a aplicação da Regressão Logística com a biblioteca **scikit-learn**, uma das ferramentas mais populares para aprendizado de máquina em Python. Através de um passo a passo detalhado, construiremos um modelo para classificar e-mails como spam ou não spam, utilizando o conjunto de dados Íris que utilizamos em aulas anteriores. Este é um dataset público e está disponível online.

Precisamos, para isso, importar as bibliotecas necessárias para este aprendizado (código-fonte 1).

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score
```

Código-fonte 1 – Importação das bibliotecas
Fonte: Elaborado pelo autor (2024)

Após isso, carregar o conjunto de dados e separar as features e a classe (código-fonte 2) em DataFrame **x** e Series **y**, respectivamente. Denominamos **Series** no Pandas quando a lista possui apenas uma dimensão. Algumas literaturas chamam a coluna de classe como variável alvo.

PDF exclusivo para Marcio Anastacio da Luz Junior - rm354541
marcioluzj@gmail.com

```
# Carregando o conjunto de dados CSV em um DataFrame
df = pd.read_csv('iris.data', header=None)

df.columns = ["sepal_length", "sepal_width", "petal_length",
"petal_width", "species"]

# Separando as features (X) e a classe (y)
x = df[["sepal_length",
        "sepal_width",
        "petal_length",
        "petal_width"]]
y = df["species"]
```

Código-fonte 2 – Carregando o conjunto de dados com a biblioteca Pandas
Fonte: Elaborado pelo autor (2024)

Aqui, diferentemente das aulas anteriores, irei utilizar o nome original das features e classe para vocês também praticarem o Inglês. Uma tabela com a tradução se encontra a seguir (tabela 1).

Feature em Inglês	Feature em Português
sepal_length	comprimento_sépala
sepal_width	largura_sépala
petal_length	comprimento_pétala
petal_width	largura_pétala

Tabela 1 – Carregando o conjunto de dados
Fonte: Elaborado pelo autor (2024)

Após a separação dos dados em x e y, precisamos dividir o conjunto de dados em treinamento e teste (código-fonte 3). Relembre o tópico "Etapas do Aprendizado de Máquina" de aulas anteriores! Aqui usamos a função do scikit-learn que faz a divisão para nós. Nesta função precisamos identificar a variável com as features (`x`) e a classe (`y`); além disso o tamanho do conjunto de treino (`test_size`), que aqui fizemos para 30% dos dados, e um valor que controla o embaralhamento aplicado nesta divisão dos dados (`random_state`).

É importante ter salvo este número para **replicar** os experimentos de forma coerente, obter os mesmos resultados no futuro - mediante os mesmos critérios adotados ao longo do fluxo - e poder utilizar sempre o mesmo conjunto de dados para **comparar modelos**.

PDF exclusivo para Marcio Anastacio da Luz Junior - rm354541
marcioluzj@gmail.com

```
# Dividindo o conjunto de dados em 70% para treinamento e 30% para teste

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)
```

Código-fonte 3 – Divisão do conjunto de dados com a biblioteca scikit-learn
Fonte: Elaborado pelo autor (2024)

Aqui temos 4 conjunto distintos de dados:

- `x_train` são as features que serão usadas para treinar o modelo.
- `y_train` são as classes que o modelo olhará para ajustar sua função de aprendizado.
- `x_test` são as features que serão mostradas para o modelo após ser treinado com a `x_train`.
- `y_test` são as classes que iremos comparar.

Com os dados divididos, chegamos ao momento de treinar o modelo. Precisamos carregar o módulo da Regressão Logística e aplicar o treinamento (código-fonte 4).

```
from sklearn.linear_model import LogisticRegression

# Criando e treinando o modelo de Regressão Logística
modelo = LogisticRegression()
modelo.fit(x_train, y_train)
```

Código-fonte 4 – Carregando o módulo da Regressão Logística e treinamento do modelo
Fonte: Elaborado pelo autor (2024)

Com o modelo treinado, iremos pedir para que ele faça previsões (código-fonte 5) do que ele acredita ser a classe correta mediante cada exemplo do conjunto de teste (`x_test`). Exemplificando, a primeira linha do `x_test` é a de índice 73, com determinados valores para as features (figura 1).

Estas features desta linha possuem a classe "Iris-versicolor" e iremos ver se o modelo consegue prever, mediante seu aprendizado com o conjunto de treino (x_train) e a respectiva classe (y_train), qual é a classe que ele sugere.

	sepal_length	sepal_width	petal_length	petal_width
73	6.1	2.8	4.7	1.2

Figura 1 – Valores, em centímetros, de cada feature do conjunto de teste, para o índice 73
Fonte: Elaborado pelo autor (2024)

PDF exclusivo para Marcio Anastacio da Luz Junior - rm354541
marcioluzj@gmail.com

```
# Fazendo previsões sobre o conjunto de dados de teste
y_pred = modelo.predict(x_test)
```

Código-fonte 5 – Modelo realizando previsões para as features do conjunto de teste
Fonte: Elaborado pelo autor (2024)

Para o índice 73, o modelo fez a seguinte previsão, que é o primeiro elemento de `y_test` (figura 2).

Figura 2 – Classe predita pelo modelo
Fonte: Elaborado pelo autor (2024)

Vejamos somente as 5 primeiras previsões do modelo e a classe real (código-fonte 6).

```
# 5 primeiras previsões das classes pelo modelo
print(y_pred[:5])

['Iris-versicolor'   'Iris-setosa'   'Iris-virginica'   'Iris-
versicolor' 'Iris-versicolor']

# 5 primeiras classes reais
print(y_test[:5])

73     Iris-versicolor
18        Iris-setosa
118    Iris-virginica
78     Iris-versicolor
76     Iris-versicolor
Name: species, dtype: object
```

Código-fonte 6 – Comparação das 5 primeiras classes preditas pelo modelo e as classes reais
Fonte: Elaborado pelo autor (2024)

Podemos ver que, até o momento, o modelo acertou todas. Vamos calcular para todo o conjunto de teste (código-fonte 7).

```
# Calculando métricas de desempenho (acurácia, precisão,
revocação e F1-Score)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='micro')
recall = recall_score(y_test, y_pred, average='micro')
f1 = f1_score(y_test, y_pred, average='micro')
```

PDF exclusivo para Marcio Anastacio da Luz Junior - rm354541
marcioluzj@gmail.com

```
print("Acurácia:", accuracy)
print("Precisão:", precision)
print("Revocação:", recall)
print("F1-Score:", f1)

Acurácia: 1.0
Precisão: 1.0
Revocação: 1.0
F1-Score: 1.0
```

Código-fonte 7 – Cálculo das métricas de desempenho
Fonte: Elaborado pelo autor (2024)

Com este exemplo você começa a entender o motivo pelo qual o dataset Íris é utilizado. Além de serem dados "bem-comportados", com a mesma quantidade de exemplos para cada uma das três classes, são dados com relativa facilidade para serem classificados. A mensagem importante aqui é a seguinte: com conjunto de dados **reais**, conseguir uma precisão de 100% é normalmente um mal sinal, no qual muito provavelmente seu modelo overffitou. Este conceito será explicado no futuro.

K-Nearest Neighbors (KNN)

O KNN, ou K Vizinhos Mais Próximos, se baseia em um princípio simples: um dado é classificado de acordo com a classe mais frequente entre seus K vizinhos mais próximos no espaço de características. Essa proximidade é medida por distâncias, como a distância Euclidiana ou Manhattan. Seguem algumas vantagens e desvantagens deste modelo.

Vantagens:

- **Simplicidade:** fácil de implementar e interpretar.
- **Não paramétrico:** não faz suposições sobre a distribuição dos dados.
- **Flexibilidade:** pode lidar com datasets com diferentes tipos de variáveis.

Desvantagens:

- **Armazenamento de dados:** pode ser computacionalmente custoso para grandes datasets.
- **Sensibilidade à ruído:** outliers podem afetar significativamente o desempenho do modelo.
- **Escala:** encontrar os K vizinhos mais próximos pode ser lento para datasets muito grandes.

PDF exclusivo para Marcio Anastacio da Luz Junior - rm354541
marcioluzj@gmail.com

Este modelo pode ser aplicado em segmentos como a de recomendação de produtos ao sugerir produtos para um cliente com base em suas compras anteriores e nas compras de outros clientes semelhantes, reconhecimento de padrões para identificar padrões em dados não estruturados, como imagens ou textos, ou análise de mercado para segmentar clientes com base em suas características e comportamentos.

O algoritmo KNN:

1. Inicialize K com o número escolhido de vizinhos.
2. Para cada exemplo nos dados:
   a) Calcule a distância entre o exemplo de consulta e o exemplo atual a partir dos dados.
   b) Adicione a distância e o índice do exemplo a uma coleção ordenada.
3. Classifique a coleção ordenada de distâncias e índices do menor para o maior (em ordem crescente) pelas distâncias.
4. Escolha as primeiras K entradas da coleção classificada.
5. Obtenha os rótulos das K entradas selecionadas.
6. Se for regressão, retorne a média dos K rótulos.
7. Se for classificação, retorne a moda dos rótulos K.

O livro "Introduction to machine learning with Python: a guide for data scientists" (2016) traz um exemplo gráfico bem didático. Dado um conjunto de dados com duas features e duas classes, bolas azuis, classe 0; e triângulos vermelhos, classe 1, é elaborado um gráfico com estas features e as classes para todos os exemplos (Figura 3).

PDF exclusivo para Marcio Anastacio da Luz Junior - rm354541
marcioluzj@gmail.com

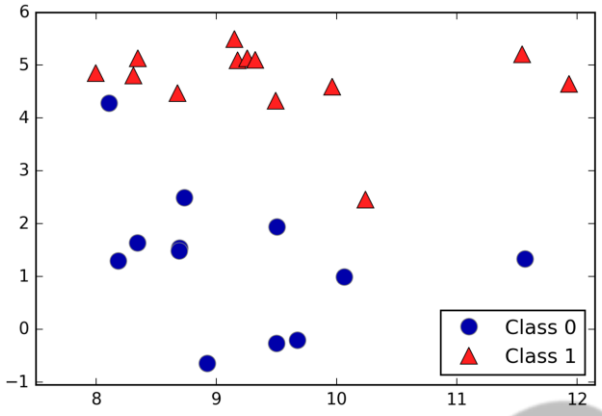


Figura 3 – Distribuição dos exemplos de um conjunto de dados com 2 features e 2 classes, gráfico do tipo scatter plot
Fonte: Guido, Mueller (2016)

Em sua versão mais simples, o KNN considera somente um vizinho mais próximo, ou seja, o ponto de treino mais próximo, para fazer a previsão. As estrelas são as features do conjunto de teste e queremos classificar por seu vizinho mais próximo (figura 4).

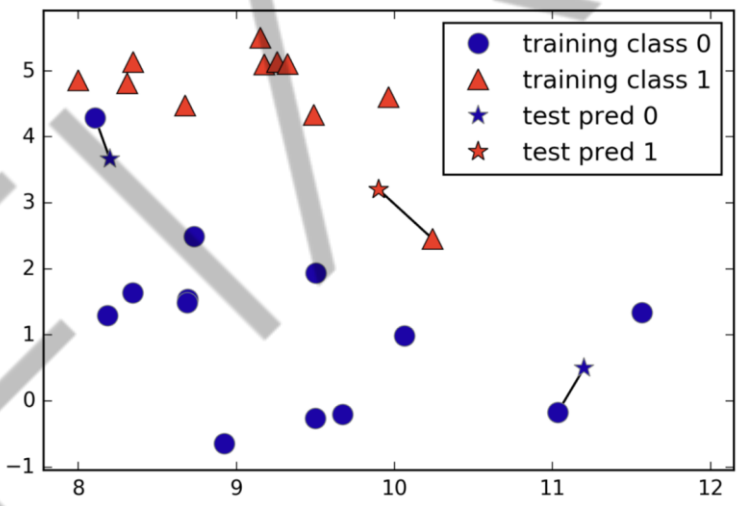


Figura 4 – Classificação de exemplos de teste usando KNN com K = 1
Fonte: Guido, Mueller (2016)

Agora, vamos fazer para 3 vizinhos (K = 3). Da mesma forma que para 1 vizinho, olharemos agora para os 3 vizinhos mais próximos e usamos a votação para atribuir um rótulo. Isso significa que para cada ponto de teste contamos quantos vizinhos pertencem à classe 0 e quantos vizinhos pertencem à classe 1. Em seguida, atribuímos a classe que é mais frequente: em outras palavras, a classe majoritária entre os 3 vizinhos mais próximos (figura 5)

PDF exclusivo para Marcio Anastacio da Luz Junior - rm354541
marcioluzj@gmail.com

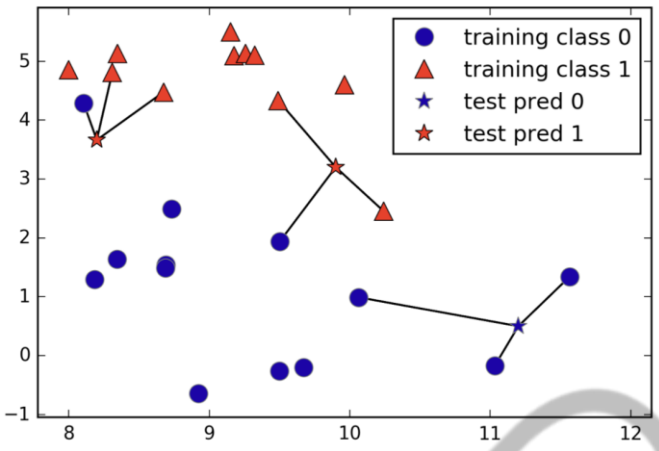


Figura 5 – Classificação de exemplos de teste usando KNN com K = 3
Fonte: Guido, Mueller (2016)

Voltaremos ao dataset Íris e aplicamos o KNN nele agora. Iremos partir do código-fonte 3, com os dados já divididos.

```
from sklearn.neighbors import KNeighborsClassifier

# Criando e treinando o modelo KNN com K=3 vizinhos
modelo = KNeighborsClassifier(n_neighbors=3)
modelo.fit(x_train, y_train)

# Fazendo previsões sobre o conjunto de dados de teste
y_pred = modelo.predict(x_test)

# Calculando a acurácia do modelo
accuracy = accuracy_score(y_test, y_pred)
print(f"Acurácia: {accuracy}")
Acurácia: 0.977
```

Código-fonte 8 – Importação com Scikit-Learn, treinamento e cálculo do desempenho do modelo de KNN
Fonte: Elaborado pelo autor (2024)

Chamamos o modelo `KNeighborsClassifier`, criamos para 3 vizinhos (`n_neighbors=3`) e treinamos o modelo com os dados de treino. Na sequência, utilizamos este treinamento para as previsões do modelo dadas as features do conjunto de teste.

Um experimento interessante aqui é ver a relação da acurácia conforme alteramos a quantidade de vizinhos. No código-fonte 9 é realizado um for loop para calcular a acurácia dada uma quantidade de vizinhos, variando de 1 até 105 (quantidade máxima, pelo tamanho do conjunto de treino) e armazenando em um

PDF exclusivo para Marcio Anastacio da Luz Junior - rm354541
marcioluzj@gmail.com

dicionário `accuracy_dict`, com a chave sendo o número de vizinhos e o valor com o valor da acurácia.

```
# Relação da acurácia com a quantidade de vizinhos
number_neighbors = range(1, 106)
accuracy_dict = dict()

for i in number_neighbors:
    knn_model=KNeighborsClassifier(n_neighbors=i)
    knn_model.fit(x_train, y_train)
    y_pred = knn_model.predict(x_test)
    accuracy_dict[i] = accuracy_score(y_test, y_pred)
```

Código-fonte 9 – Código Python iterando a quantidade de vizinhos e o respectivo valor da acurácia
Fonte: Elaborado pelo autor (2024)

Alguns dos valores armazenados neste dicionário:

```
print(accuracy_dict)
{1: 0.9777777777777777,
 2: 0.9777777777777777,
 3: 0.9777777777777777,
 4: 0.9777777777777777,
 5: 1.0,
 6: 0.9777777777777777,
 7: 1.0,
 ...
 21: 0.9777777777777777,
 22: 0.9555555555555556,
 23: 0.9777777777777777,
 ...
 73: 0.4444444444444444,
 74: 0.4444444444444444,
 75: 0.4444444444444444,
 ...
 103: 0.4222222222222222,
 104: 0.4222222222222222,
 105: 0.28888888888888886}
```

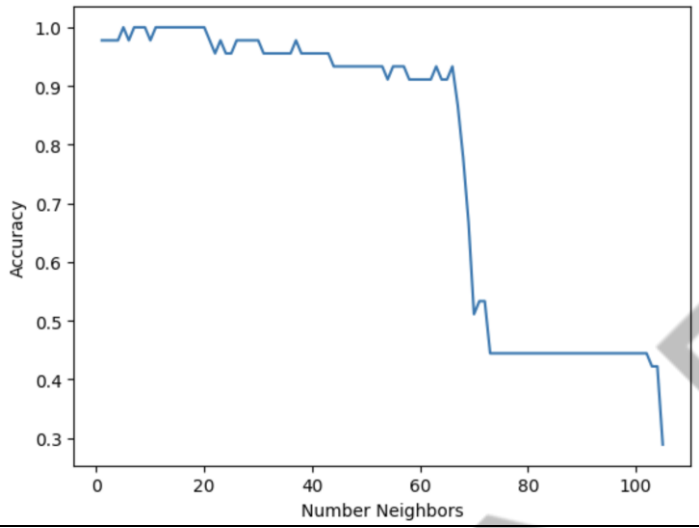
Código-fonte 10 – Visualizando alguns valores armazenados no dicionário accuracy_dict.
Fonte: Elaborado pelo autor (2024)

Para se ter uma melhor noção, veja o gráfico da quantidade de vizinhos pela acurácia (código-fonte 11).

PDF exclusivo para Marcio Anastacio da Luz Junior - rm354541
marcioluzj@gmail.com

```
import matplotlib.pyplot as plt

plt.plot(number_neighbors,accuracy_dict.values())
plt.xlabel("Number Neighbors")
plt.ylabel("Accuracy")
plt.show()
```



Código-fonte 11 – Código que exibe um gráfico com a relação de vizinhos com a acurácia
Fonte: Elaborado pelo autor (2024)

Por que será que há este comportamento? Pense no que acontece quando aumentamos muito a quantidade de vizinhos.

Conforme vamos aumentando a quantidade de vizinhos, estamos aumentando a quantidade de votos que cada exemplo de teste precisa olhar para decidir qual classe será a sua. No caso extremo, com a quantidade máxima de vizinhos possíveis, cada exemplo de teste possui a mesma quantidade de vizinhos, que são todos os exemplos de treinamento. Em uma situação de dados desbalanceados (uma classe é mais frequente que outra no conjunto de treinamento), é um viés que afeta muito o desempenho do modelo.

Desta forma, podemos ver que é necessário achar um bom número de vizinhos para o modelo. Apenas um vizinho é ter uma visão única e ter muitos faz com que não "se tenha personalidade". O valor ideal de K pode ser encontrado utilizando técnicas como validação cruzada ou otimização de hiperparâmetros, que veremos em aulas adiante.

Árvore de Decisão

A árvore de decisão é um dos modelos mais famosos para a tarefa de aprendizado supervisionado, incluindo para tarefas de classificação e regressão. Elas

PDF exclusivo para Marcio Anastacio da Luz Junior - rm354541
marcioluzj@gmail.com

constroem uma estrutura hierárquica de regras que dividem os dados em subconjuntos com base em características específicas, resultando em um modelo fácil de interpretar e visualizar.

Imagine uma árvore frondosa, com galhos que se ramificam em diversos caminhos. Cada um desses caminhos representa uma decisão, levando você a um destino final. Essa é a essência de uma **árvore de decisão**. Cada estrutura possui um nome específico:

- **Nó Raiz:** o ponto de partida da jornada, onde se inicia o modelo. Diferentemente de uma árvore real, esta raiz normalmente é visualizada no topo com a árvore crescendo para baixo.
- **Nó Interno:** ponto que divide o caminho do ramo da árvore mediante uma decisão, dividindo o galho em duas ou mais possibilidades.
- **Nó Externo:** o destino final, em que você encontra a resposta que busca.
- **Ramo:** a ponte que liga os nós, conectando as decisões e guiando você pela árvore.

Um conjunto de dados famoso que é utilizado em Árvore de Decisão é o da pessoa decidir se irá jogar tênis, com base em dados de tempo. As features são relacionadas a:

- **Panorama do tempo**: ensolarado, nublado ou chuvoso.
- **Temperatura**: quente, amena, fria.
- **Umidade**: alta, normal.
- **Vento**: fraco, forte.

A classe é a decisão de ir jogar ou não tênis.

FIAP

PDF exclusivo para Marcio Anastacio da Luz Junior - rm354541
marcioluzj@gmail.com

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Figura 6 – Conjunto de Dados PlayTennis
Fonte: Mitchell (2011)

O Processo Decisivo:

1. **Início:** você apresenta seus dados ao **nó**.
2. **Pergunta Crucial:** o nó faz uma pergunta importante, baseada em um teste lógico baseado em if/else.
3. **Ramificação:** com base na resposta à pergunta, você segue para um dos nós.
4. **Repetição:** esse processo se repete até que você chegue a um nó externo, o destino final com a resposta que busca.

O critério para se definir cada nó é baseado em Entropia, uma medida de aleatoriedade (impureza) de uma variável. Este conceito vem da Teoria da Informação que define a medida da "falta de informação", mais especificamente o número de bits necessários, em média, para representar a informação em falta. Recomenda-se a leitura do "A mathematical theory of communication" de Shannon (1948) para entender melhor este conceito e o processo de criação de decisão de cada nó, além da documentação do Scikit-Learn para a implementação da Árvore de Decisão.

Uma vez a árvore pronta (figura 7), fica fácil de tomar decisões olhando para ela. Por exemplo, a pessoa sempre irá jogar quando:

- O tempo está nublado (overcast).

PDF exclusivo para Marcio Anastacio da Luz Junior - rm354541
marcioluzj@gmail.com

- O tempo está ensolarado e a umidade normal.
- O tempo está chuvoso e o vento está fraco.

E não irá jogar quando

- Está ensolarado e a umidade alta.
- Está chuvoso e a vento forte.

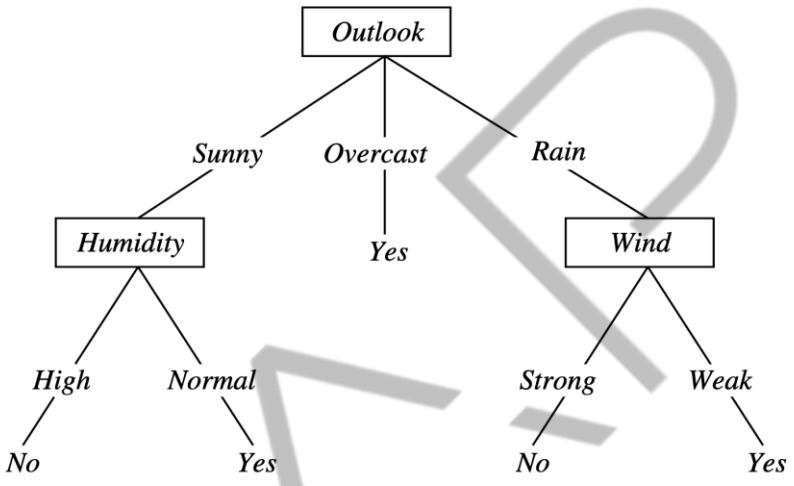


Figura 7 – Árvore de Decisão do dataset PlayTennis
Fonte: CMU EDU [s.d.]

Vamos ver uma utilização da árvore de decisão (código-fonte 12).

```
from sklearn.tree import DecisionTreeClassifier

# Criando e treinando o modelo de Árvore de Decisão
modelo = DecisionTreeClassifier()
modelo.fit(x_train, y_train)

# Fazendo previsões sobre o conjunto de dados de teste
y_pred = modelo.predict(x_test)

# Calculando a acurácia do modelo
accuracy = accuracy_score(y_test, y_pred)
print(f"Acurácia: {accuracy}")

Acurácia: 0.977
```

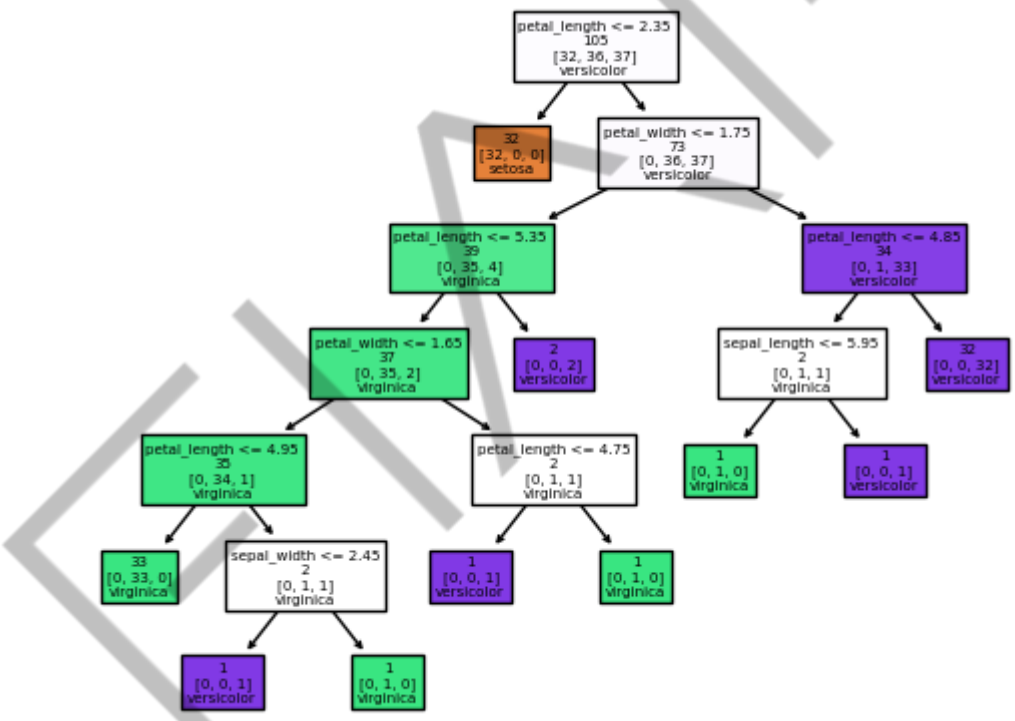
Código-fonte 12 – Aplicação do modelo de Árvore de Decisão com a biblioteca Scikit-Learn
Fonte: Elaborado pelo autor (2024)

PDF exclusivo para Marcio Anastacio da Luz Junior - rm354541
marcioluzj@gmail.com

É possível também observar a estrutura dessa árvore de decisão (código-fonte 13).

```
from sklearn import tree

tree.plot_tree(modelo,
               feature_names = ['sepal_length',
                                'sepal_width',
                                'petal_length',
                                'petal_width'],
               class_names = ['setosa',
                              'virginica',
                              'versicolor'],
               impurity=False,
               label='none',
               filled=True)
```



Código-fonte 13 – Plot da árvore de decisão
Fonte: Elaborado pelo autor (2024)

Máquinas de Vetores de Suporte (Support Vector Machines)

As Support Vector Machines (SVMs) são um método de aprendizado supervisionado poderoso para tarefas de classificação e regressão. Elas constroem um hiperplano que separa as classes no espaço de características, maximizando a margem entre o hiperplano e os pontos de dados mais próximos (os vetores de suporte).

PDF exclusivo para Marcio Anastacio da Luz Junior - rm354541
marcioluzj@gmail.com

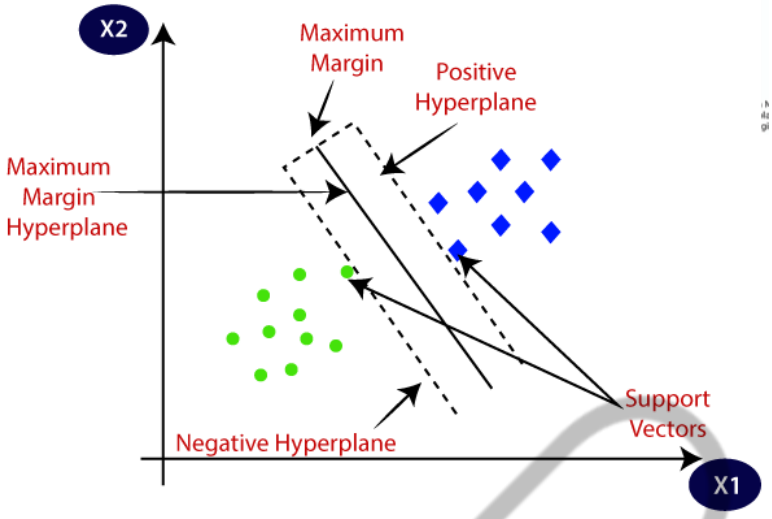


Figura 8 - Representação do SVM com o hiperplano de separação, as margens e os vetores de suporte
Fonte: Saini (2024)

Uma grande vantagem deste modelo é poder mudar a geometria da distribuição dos dados ao aplicar o truque do Kernel. Com isso, é possível adaptar dados não lineares, simulando cenários lineares em demais dimensões. Qualquer dado pode ser separado em uma dimensão maior que a dimensão atual.

Imagine este exemplo da figura 9. De que forma é possível separar, com apenas uma linha, os quadrados vermelhos dos círculos verdes?

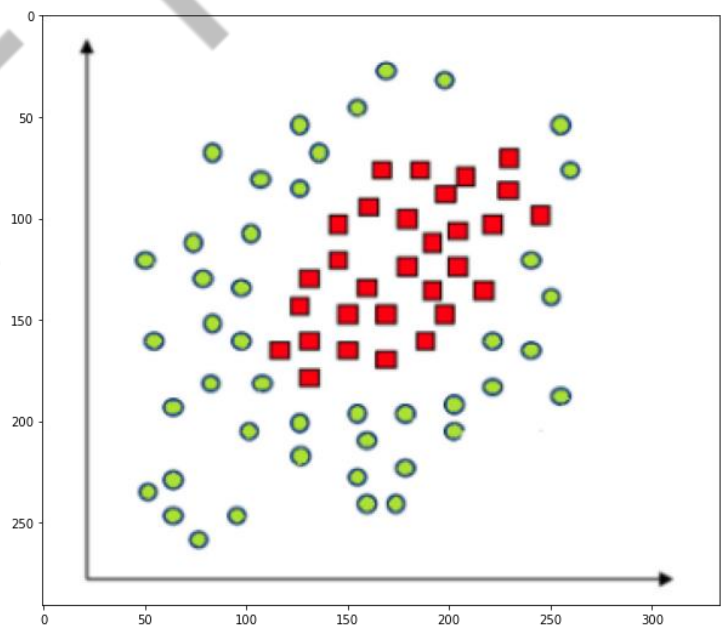


Figura 9 - Dados com 2 classes representados em um plano
Fonte: Zhang (2018)

PDF exclusivo para Marcio Anastacio da Luz Junior - rm354541
marcioluzj@gmail.com

Com esta distribuição desta forma, ao passar uma linha ou plano não é possível separar sem errar algumas classificações.

Com o truque do Kernel, é possível transformar os dados de 2 para 3 dimensões, ficando mais fácil separar os dados com apenas um plano. Desta forma, a regra do aprendizado para este exemplo fica a seguinte: o que está para cima do plano é considerado como da classe quadrados vermelhos, o que está para baixo do plano é da classe bolas verdes (figura 10).

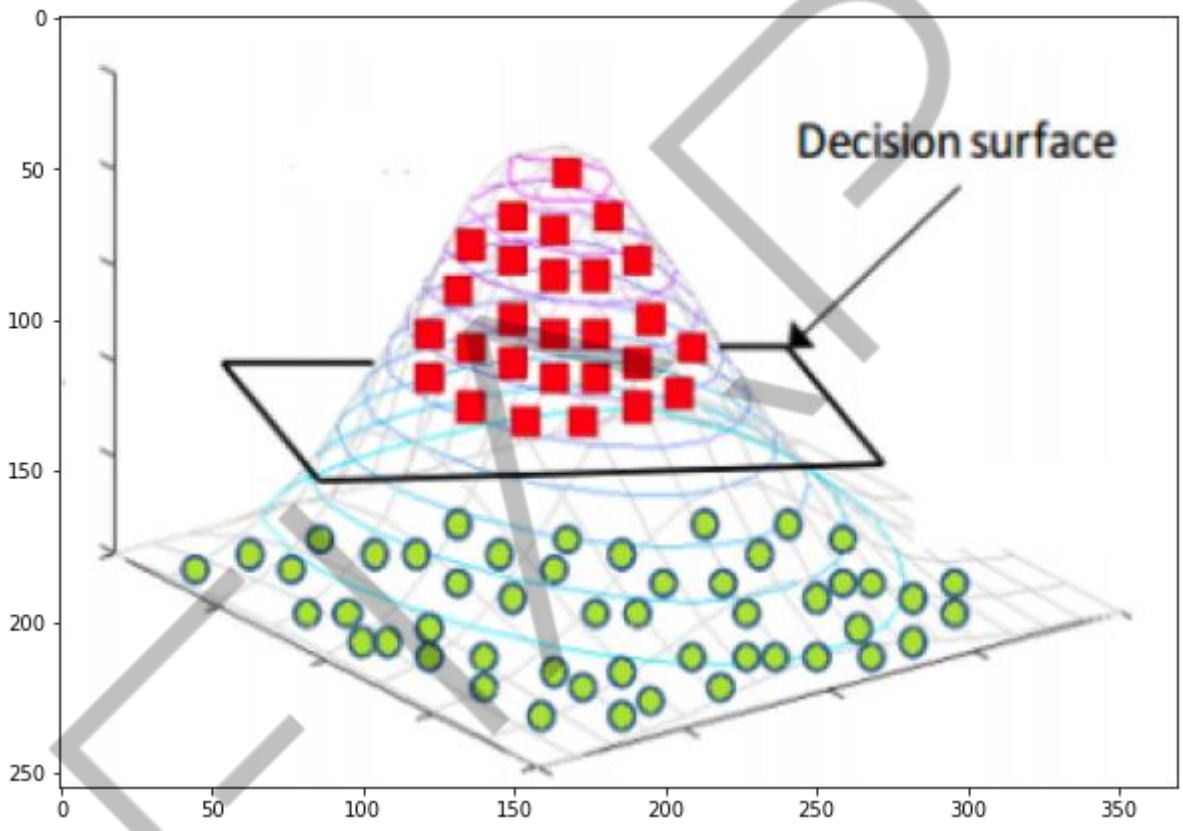


Figura 10 - Aplicação do Kernel nos dados e classificação com SVM
Fonte: Zhang (2018)

Outro exemplo de uso do Kernel pode ser visto neste vídeo do Youtube.

Voltando ao nosso exemplo com os dados do Íris, vamos seguir o mesmo pipeline dos outros modelos: aplicar o modelo, treinar, fazer previsões e medir o desempenho (código-fonte 14).

PDF exclusivo para Marcio Anastacio da Luz Junior - rm354541
marcioluzj@gmail.com

```
from sklearn.svm import SVC

# Criando e treinando o modelo SVM
modelo = SVC()
modelo.fit(x_train, y_train)

# Fazendo previsões sobre o conjunto de dados de teste
y_pred = modelo.predict(x_test)

# Calculando a acurácia do modelo
accuracy = accuracy_score(y_test, y_pred)
print(f"Acurácia: {accuracy}")
Acurácia: 1.0
```

Código-fonte 14 – Implementação do modelo de Support Vector Machines
Fonte: Elaborado pelo autor (2024)

Para um estudo mais aprofundado no tema, recomenda-se a leitura do livro "An Introduction to Support Vector Machines and Other Kernel-based Learning Methods" (2013), de Nello Cristianini e John Shawe-Taylor.

Redes Neurais Artificiais

Chegamos ao último tópico dos modelos de classificação, as redes neurais artificiais (RNA). Muitos modelos utilizados hoje em dia são baseados em RNA, dada a sua robustez e a grande quantidade de dado disponível para seu treinamento.

Você já deve ter ouvido falar de Deep Learning, o aprendizado profundo. Este é um tópico ainda em alta e muitas empresas demandam profissionais especializados(as) na área. Para entender de Deep Learning, comece pelo básico de redes neurais antes de se aprofundar no tema.

As RNA são modelos computacionais que imitam a estrutura e o funcionamento do cérebro humano para resolver problemas complexos. As RNA são compostas por unidades básicas chamadas **neurônios artificiais**, que se conectam entre si como uma teia neural complexa. Cada neurônio artificial recebe inputs (sinais de entrada), processa-os e gera um output (sinal de saída).

Assim como o cérebro humano aprende com a experiência, as RNAs também são treinadas através de um processo chamado **aprendizado de máquina**. Através da exposição a grandes conjuntos de dados, as RNAs ajustam as conexões entre seus neurônios artificiais, buscando padrões e regras que lhes permitam fazer previsões ou tomar decisões.

PDF exclusivo para Marcio Anastacio da Luz Junior - rm354541
marcioluzj@gmail.com

Vantagens das Redes Neurais na Classificação:

- **Alta Precisão:** as RNAs podem alcançar alta precisão na classificação, superando inclusive métodos tradicionais em alguns casos.
- **Flexibilidade:** as RNAs podem ser aplicadas a uma ampla gama de tarefas de classificação, desde simples até as mais complexas.
- **Aprendizado Contínuo:** as RNAs podem ser continuamente aprimoradas com novos dados, adaptando-se a mudanças no ambiente.

Desafios das Redes Neurais na Classificação:

- **Complexidade:** as RNAs podem ser complexas de implementar e treinar, exigindo alto poder computacional e expertise em machine learning.
- **Interpretabilidade:** o processo de decisão das RNAs pode ser difícil de interpretar, limitando a compreensão dos resultados por humanos.
- **Dados:** as RNAs necessitam de grandes volumes de dados de alta qualidade para serem treinadas com eficácia.

Perceptron

O Perceptron é a rede neural mais simples, que recebe várias entradas e produz uma única saída binária (0 ou 1), ou seja, aprende a separar duas classes em um espaço bidimensional. Ele funciona ajustando os pesos de cada uma das suas conexões para minimizar o erro entre suas previsões e os dados reais.

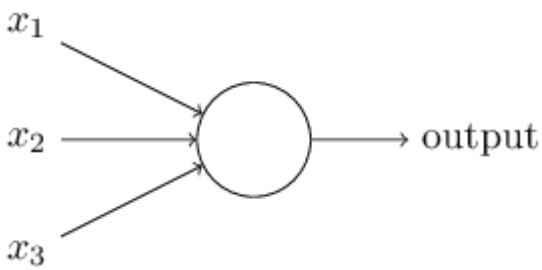


Figura 11 - Desenho da arquitetura da rede neural artificial Perceptron
Fonte: Deep Learning Book (2024)

Para exemplificar o uso do Perceptron e como ele lida com a tarefa de classificar para apenas duas classes, vamos escolher apenas duas classes do Íris e seguir com o mesmo pipeline dos modelos anteriores: carregar o modelo, treinar, fazer previsões e avaliar o desempenho (código-fonte 15).

PDF exclusivo para Marcio Anastacio da Luz Junior - rm354541
marcioluzj@gmail.com

```
from sklearn.linear_model import Perceptron

X = df[["sepal_length", "sepal_width"]]
y = df["species"]

# Dividindo o conjunto de dados em treinamento e teste
x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.3, random_state=42)

# Treinando o modelo Perceptron
modelo = Perceptron()
modelo.fit(x_train, y_train)

# Fazendo previsões com o modelo treinado
y_pred = modelo.predict(x_test)

# Avaliando o desempenho do modelo
accuracy = accuracy_score(y_test, y_pred)

print(f"Acurácia: {accuracy}")
Acurácia: 0.8
```

Código-fonte 15 – Pipeline do Perceptron com a biblioteca do Scikit-Learn
Fonte: Elaborado pelo autor (2024)

Observações:

- Este é um exemplo básico do Perceptron. Hiperparâmetros como a taxa de aprendizado e o número máximo de iterações podem ser ajustados para melhorar o desempenho.
- O Perceptron pode ter dificuldade em lidar com dados não linearmente separáveis.

Multilayer Perceptron (MLP)

O Multilayer Perceptron (MLP) é uma rede neural artificial com múltiplas camadas de neurônios interconectados. Ele é capaz de aprender funções complexas e não lineares, tornando-o mais poderoso que o Perceptron para lidar com diversos tipos de problemas. O MLP possui uma camada a mais entre os dados de entrada e saída, denominada camada intermediária ou camada escondida (figura 12). É possível também ter outras arquiteturas com diversos neurônios na camada intermediária e, além disso, diversas outras camadas intermediárias; neste caso temos as Deep Learning.

PDF exclusivo para Marcio Anastacio da Luz Junior - rm354541
marcioluzj@gmail.com

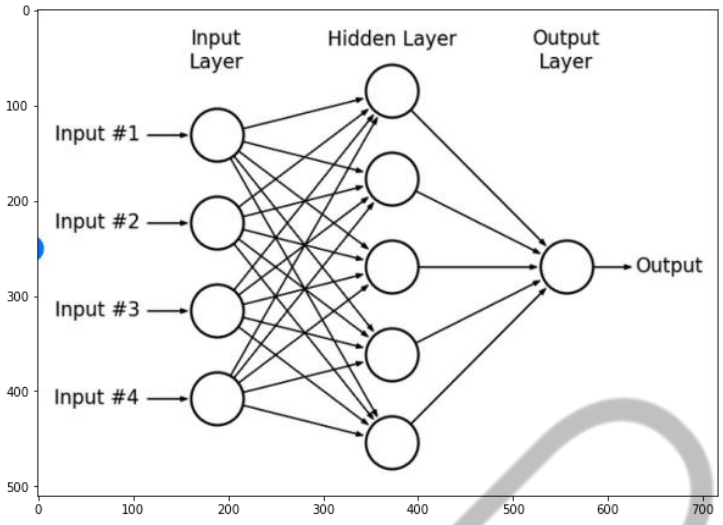


Figura 12 - Desenho de um exemplo de arquitetura da rede neural artificial Multilayer Perceptron
Fonte: Zahran (2024)

O problema mais simples que é possível resolver com o MLP é o do XOR lógico (figura 13). Repare que com Perceptron não é possível fazer a separação das duas classes, pois com uma reta algum ponto fica de fora da separação. Vemos que são necessárias duas retas para fazer esta separação.

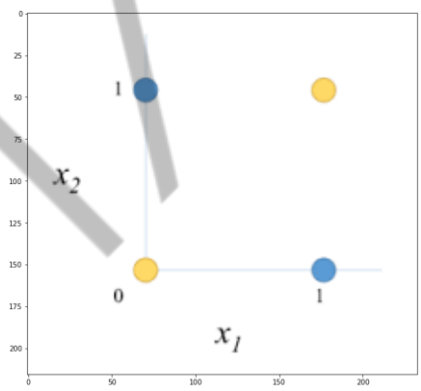


Figura 13 - Representação gráfica do XOR lógico
Fonte: Oman (2015)

Para se ter duas retas nesta tarefa de classificação precisamos de dois neurônios na camada intermediária. Repare que com duas retas conseguimos fazer a separação das duas classes no XOR, aplicando uma regra do que estiver "dentro" das duas retas é da classe 1 e o que estiver fora é da classe 0 (figura 14).

PDF exclusivo para Marcio Anastacio da Luz Junior - rm354541
marcioluzj@gmail.com

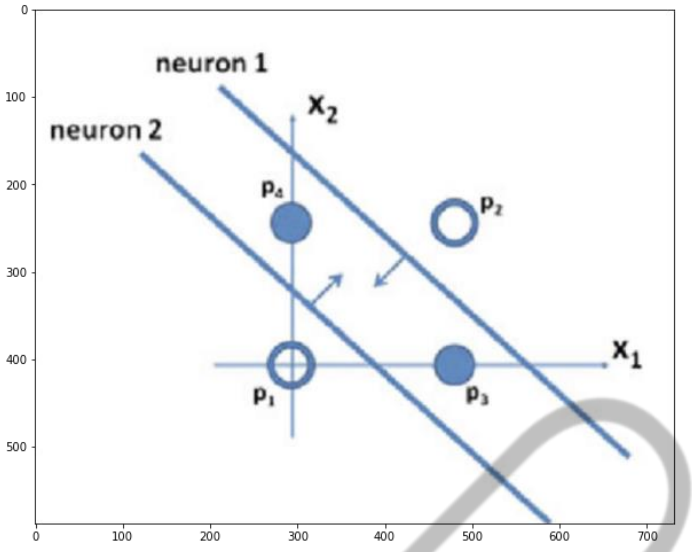


Figura 14 - Resolvendo o problema do XOR com MLP
Fonte: Modares

Vamos criar o pipeline do MLP com as mesmas etapas que os outros modelos. Veja na documentação oficial da biblioteca que há alguns parâmetros já definidos por padrão, como 100 neurônios na camada escondida e 200 iterações (código-fonte 16).

```
from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier()

# Treinando o modelo Multilayer Perceptron
mlp.fit(x_train, y_train)

# Fazendo previsões sobre o conjunto de dados de teste
y_pred = mlp.predict(x_test)

# Calculando a acurácia do modelo
accuracy = accuracy_score(y_test, y_pred)
print(f"Acurácia: {accuracy}")
Acurácia: 1.0
```

Código-fonte 16 – Pipeline do MLP com a biblioteca Scikit-Learn
Fonte: Elaborado pelo autor (2024)

PDF exclusivo para Marcio Anastacio da Luz Junior - rm354541
marcioluzj@gmail.com

MERCADO, CASES E TENDÊNCIAS

Um ChatGPT para música é a proposta da startup Suno AI, que pode criar um blues com violão e voz em apenas 15 segundos, incluindo a letra e até o título da música. Veja mais aqui.

Segundo fontes, há a possibilidade de a Apple integrar o Gemini da Google em seus iPhones. Leia sobre aqui.

FIAP

PDF exclusivo para Marcio Anastacio da Luz Junior - rm354541
marcioluzj@gmail.com

O QUE VOCÊ VIU NESTA AULA?

Nesta aula vimos alguns modelos de classificação, alguns mais simples como regressão logística, KNN e árvore de decisão. Outros um pouco mais complexos como o SVM e as redes neurais.

Cada um deles possui suas características e cientistas de dados devem treinar suas habilidades nestes modelos e entender qual deles é o melhor a ser usado mediante um conjunto de dados.

FIAP

PDF exclusivo para Marcio Anastacio da Luz Junior - rm354541
marcioluzj@gmail.com

REFERÊNCIAS

CMU EDU. **Decision Tree Learning**. [s.d.] Disponível em: <www.cs.cmu.edu/afs/cs/project/theo-20/www/mlbook/ch3.pdf>. Acesso em: 04 jul. 2024.

CRISTIANINI, N.; SHAWE-TAYLOR, J. **An introduction to support vector machines and other kernel-based learning methods**. Cambridge: Cambridge University Press, 2013.

DEEP LEARNING BOOK. **Deep Learning Book**. 2024. Disponível em: <https://www.deeplearningbook.com.br/o-perceptron-parte-1/>. Acesso em: 04 jul. 2024.

GUIDO, S.; MUELLER, A. C. **Introduction to machine learning with python:** a guide for data scientists. Sebastopol: O'Reilly Media, 2016.

HAYKIN, S. **Redes neurais:** princípios e práticas. Porto Alegre: Bookman, 2000.

MITCHELL, T. **Machine Learning**. Nova lorque: McGraw-Hill, 2011.

MODARES, H. **Fig 4**. 2024. Disponível em: <https://www.researchgate.net/figure/Solving-XOR-problem-using-3-conventional-neurons-as-a-2-2-1-MLP-network_fig1_220283506>. Acesso em: 04 jul. 2024.

OMAN, S. **A Simple Neural Network In Octave – Part 1**. 2015. Disponível em: <https://aimatters.wordpress.com/2015/12/19/a-simple-neural-network-in-octave-part-1/>. Acesso em: 04 jul. 2024.SAINI, A. **Guide on Support Vector Machine (SVM) Algorithm**. 2024. Disponível em: <https://www.analyticsvidhya.com/blog/2021/10/support-vector-machinessvm-a-complete-guide-for-beginners/>. Acesso em: 04 jul. 2024.

SCIKIT LEARN. **Decision Trees**. 2024. Disponível em: <https://scikit-learn.org/stable/modules/tree.html>. Acesso em: 04 jul. 2024.

SCIKIT LEARN. **Sklearn.Linear_model.LogisticRegression**. 2024. Disponível em: <https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html>. Acesso em: 04 jul. 2024.

SCIKIT LEARN. **Sklearn.Model_selection.Train_test_split**. 2024.Disponível em: <https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html>. Acesso em: 04 jul. 2024.

SCIKIT LEARN. **Support vector machines**. 2024. Disponível em: <https://scikit-learn.org/stable/modules/svm.html>. Acesso em: 04 jul. 2024.

SHANNON, C. E. A mathematical theory of communication. **The Bell System technical journal**, v. 27, n. 3, p. 379–423, 1948.

PDF exclusivo para Marcio Anastacio da Luz Junior - rm354541
marcioluzj@gmail.com

ZAHRAN, M. **Figure 4**. 2024. Disponível em: <https://www.researchgate.net/figure/A-hypothetical-example-of-Multilayer-Perceptron-Network_fig4_303875065>. Acesso em: 04 jul. 2024.

ZHANG, G. **What is the kernel trick? Why is it important?** 2018. Disponível em: <https://medium.com/@zxr.nju/what-is-the-kernel-trick-why-is-it-important-98a98db0961d>. Acesso em: 04 jul. 2024.

FIAP

PDF exclusivo para Marcio Anastacio da Luz Junior - rm354541
marcioluzj@gmail.com

PALAVRAS-CHAVE

Palavras-chave: Modelos de Classificação. Regressão Logística. K-Vizinhos mais próximos. Árvore de decisão. Máquinas de vetor de suporte. Rede Neural artificial. Perceptron. Multilayer Perceptron. Pandas. Numpy. Scikit-Learn. Python.

FIAP

PDF exclusivo para Marcio Anastacio da Luz Junior - rm354541
marcioluzj@gmail.com

PDF exclusivo para Marcio Anastacio da Luz Junior - rm354541
marcioluzj@gmail.com