

INF 1514

Introdução à Machine Learning

Machine Learning – Support Vector Machine Classifier

[JUPYTER NOTEBOOK LINK](#)

Sergio Pimentel



Este curso é idealizado para dar uma ideia geral do modelo de Machine Learning SVM Classifier para previsão de variáveis.

INF 1514

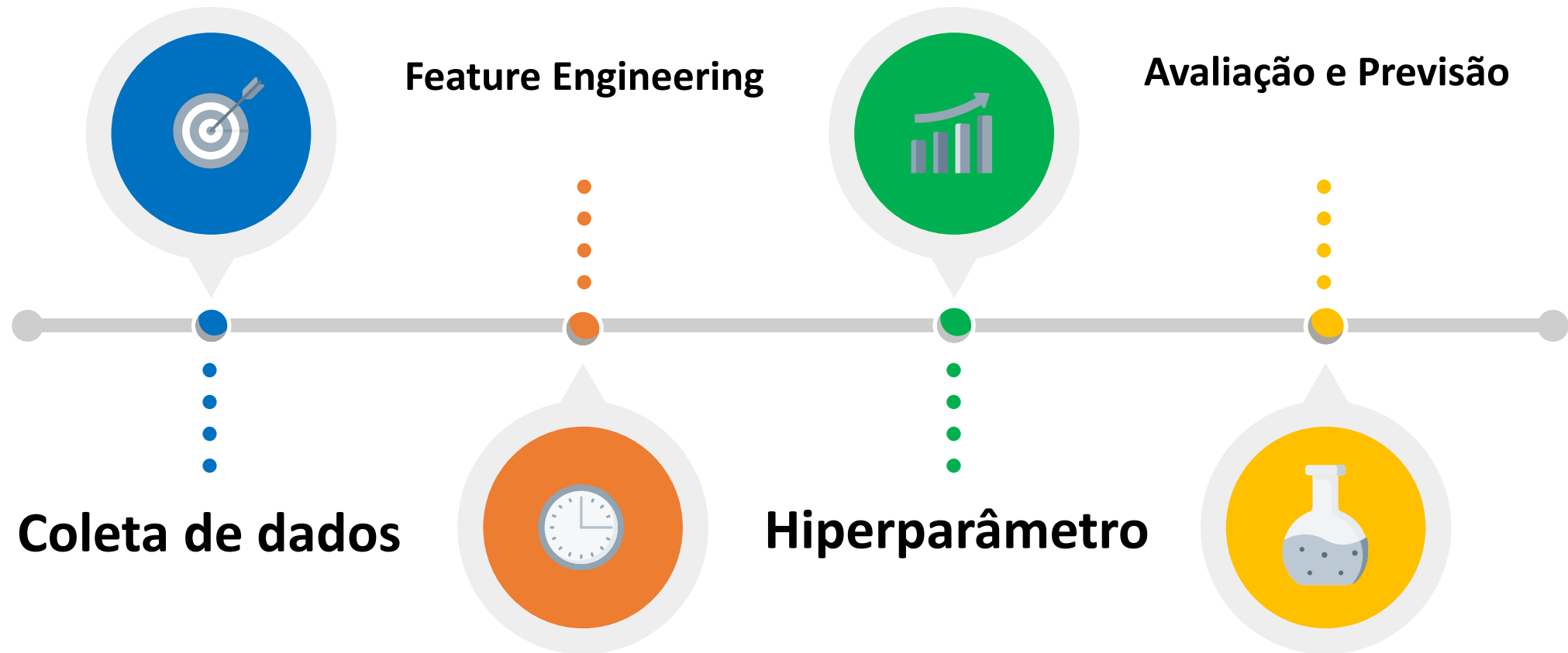


Machine Learning é uma técnica computacional para inferência de funções desconhecidas.



- Podemos modelar funções que mapeiam x à y a partir de bases teóricas e análise fundamentalista, porém, o que fazer quando esse tipo de modelagem não é possível? Pela complexidade do problema, por exemplo. O machine learning foi criado para esse tipo de problema, onde há uma alta dimensionalidade de variáveis e complexidades muito altas.
- Um exemplo simples da aplicação de modelos de Machine Learning pode ser a classificação de ativos no mercado de ações, uma ação irá subir ou cair no dia seguinte? Um e-mail é spam ou não?

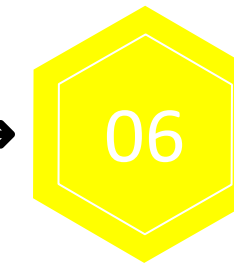
Etapas da criação de um modelo



Fases da construção de um Modelo

Definição dos requisitos de dados

Por que está fazendo essa análise? Quais dados irá utilizar?



Limpeza dos dados

Os dados podem conter registros duplicados, espaços em branco ou erros.

Treinamento e Teste do Modelo

Treinamento do Modelo com as features selecionadas.

Coleta dos dados

Coletar os dados com base nos requisitos, processá-los e organizá-los para análise.

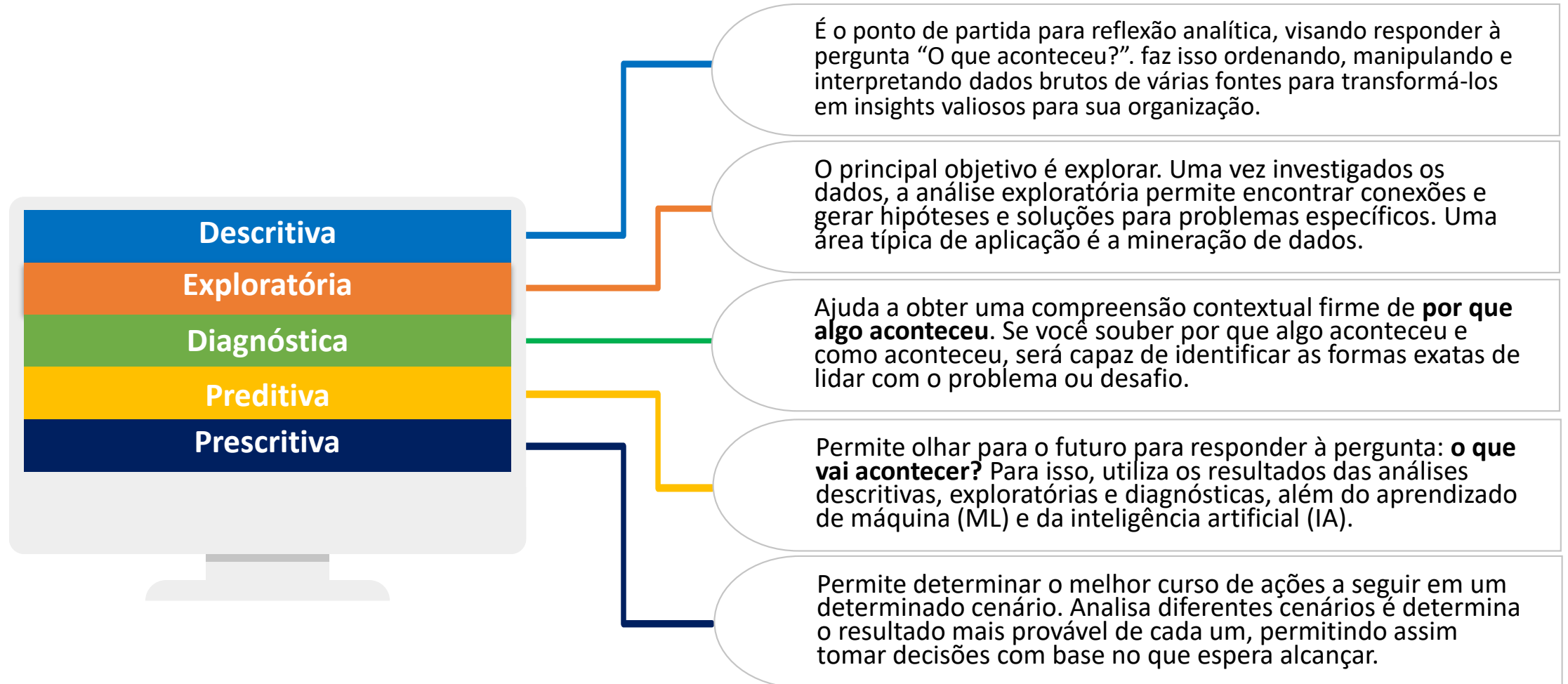
Feature

Manipulação e Criação de Features do Modelo

Teste do modelo

Testar o modelo já treinado em um dataset com novos dados

Análise de Dados



Métodos para Machine Learning

1

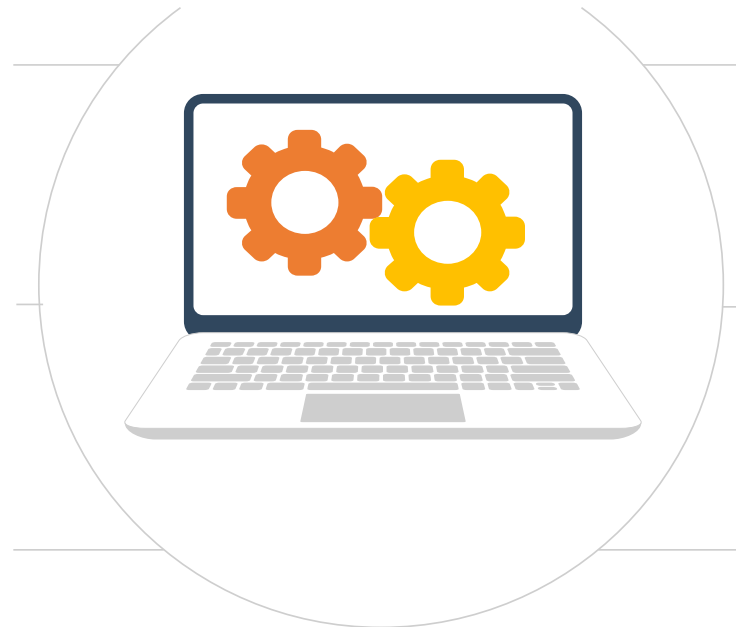
Análise de Cluster

2

Análise de Regressão

3

Redes Neurais



Análise de Classificação

4

Árvores de Decisão

5

Análise de Séries Temporais

6

Ferramentas para Machine Learning



Ajudam os usuários a processar, manipular e visualizar dados, analisar as relações e correlações entre conjuntos de dados e também a identificar padrões e tendências para interpretação. Além disso, oferecem diversos tipos de modelos de Machine Learning, Rede Neural e LLM, além de todas as ferramentas necessárias para Otimização, Treinamento e Previsão dos modelos além da visualização do funcionamento dos mesmos.

Saber Machine Learning é importante

1

Automatização

Automatização de tarefas complexas

2

Insight

Identificação de padrões e relações em grandes datasets



3

Decisão

Melhores tomadas de decisão por processar grandes quantidades de dados, impossível para seres humanos

4

Oportunidades

Abre portas para Boas oportunidades no mercado de trabalho

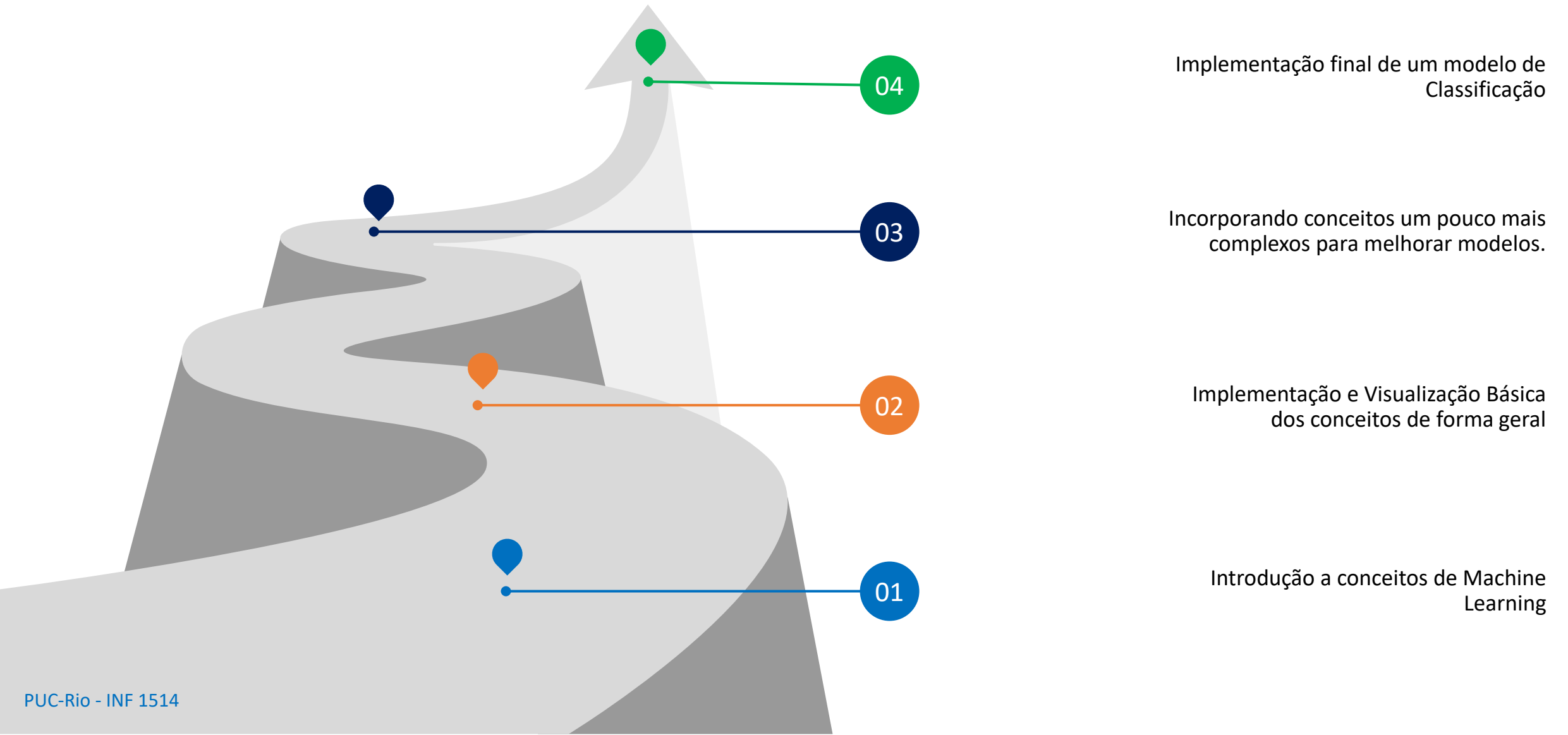
Onde estamos

Apresentação Geral do que é e como funciona um modelo de Machine Learning

Onde chegaremos

Ter a capacidade de implementar um modelo de Classificação em Python

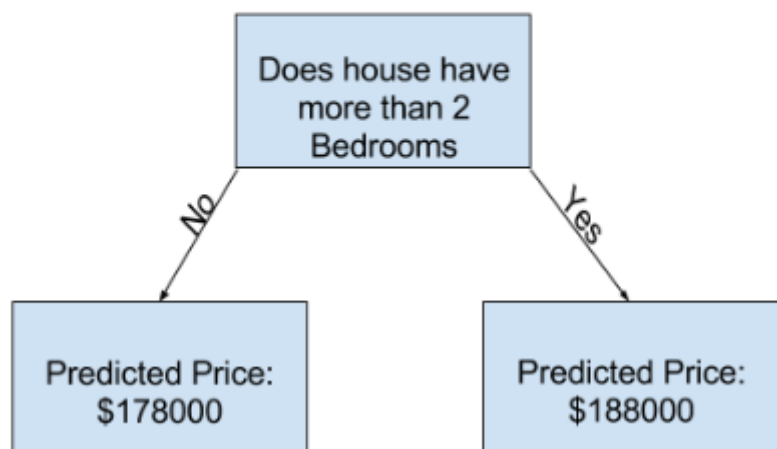
Nossa trilha



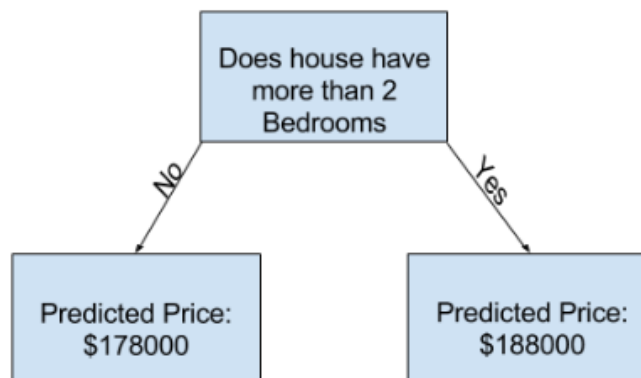
Introdução

Imagine o seguinte: Algum amigo seu têm um portfólio de ações que fez muito dinheiro e você o pergunta como ele fez pra prever seus valores e acertar com consistência, e ele te responde que essas previsões foram baseadas em intuição e identificação de padrões passados das respectivas ações avaliadas. Modelos de ML funcionam dessa mesma maneira.

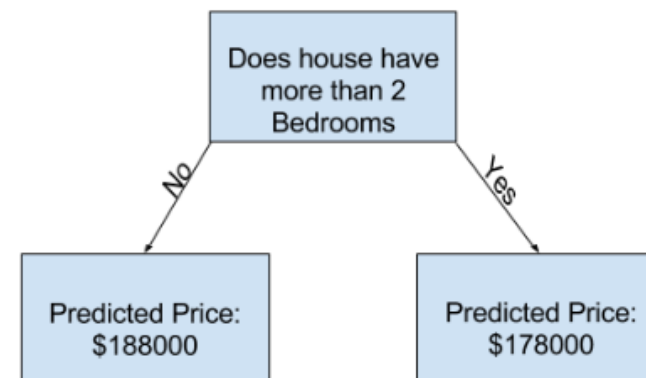
Sample Decision Tree



1st Decision Tree



2nd Decision Tree



Regressão x Classificação

Modelos de Regressão:

Definição: Modelos que prevêm valores contínuos

Exemplo: Dadas características como volatilidade e média de retornos passados, prever o preço de fechamento de uma ação no dia seguinte.

Modelos: Regressão Linear, Random Forest, SVM, Gradient Boosting Machine

Modelos de Classificação:

Definição: Modelos que **associam uma classe ou categoria** a um determinado ponto.

Exemplo: Dadas características como volatilidade e média de retornos passados, determinar se uma ação irá subir ou descer no fechamento do dia seguinte

Modelos: Regressão Logística, SVM, Gradient Boosting Machine, Random Forest, KNN, Naive Bayes, NN

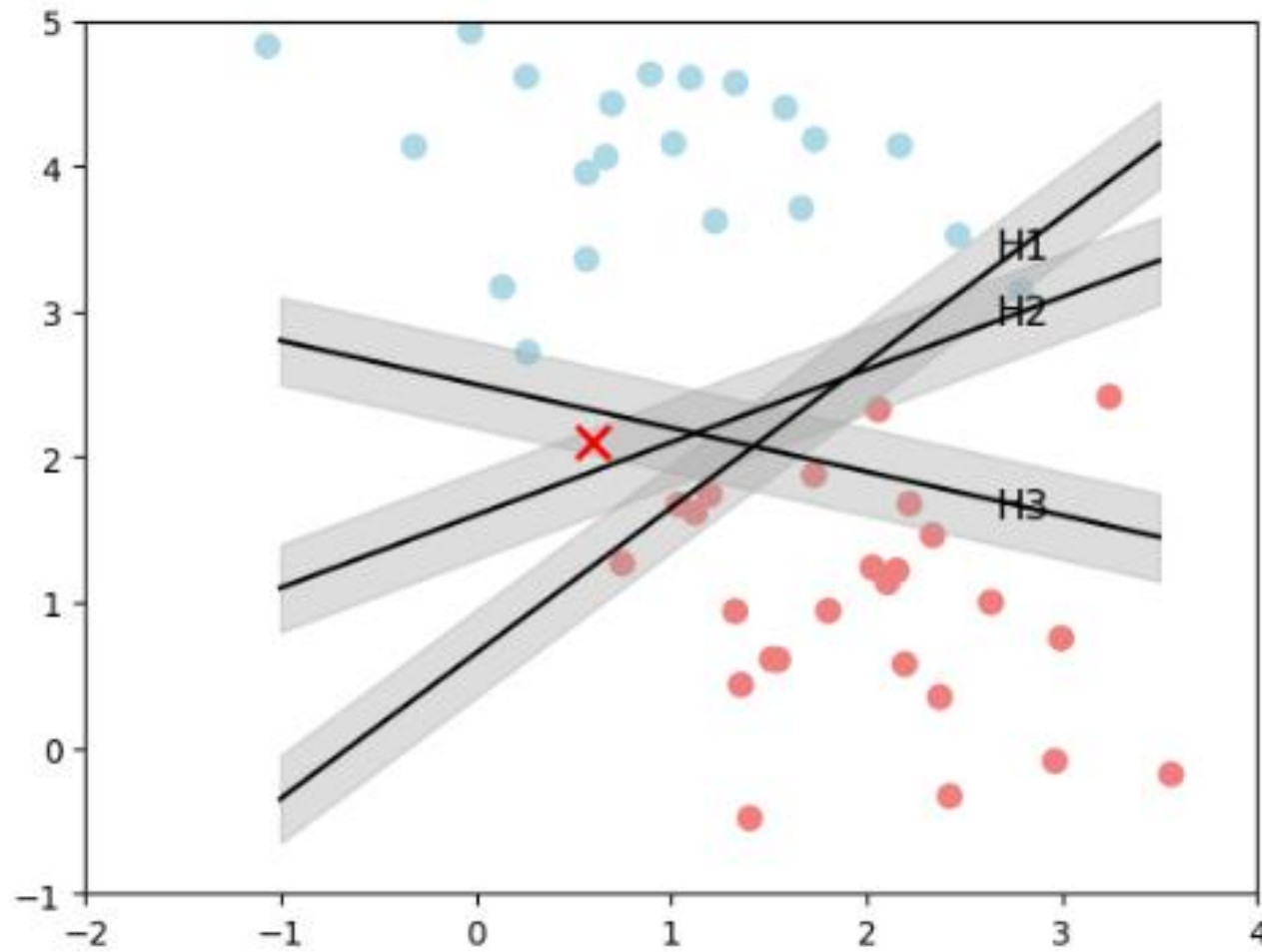
Escolha e Sugestão de Modelos

| Tipo de Modelo | Regressão | Classificação |
|----------------------------|---|--|
| Definição | Prevê valores contínuos | Associa uma classe ou categoria a um ponto de dados |
| Exemplo de Problema | Prever o preço de fechamento de uma ação no dia seguinte | Determinar se uma ação vai subir ou descer no fechamento do dia seguinte |
| Modelos Comuns | Regressão Linear, Random Forest Regressor, Gradient Boosting Regressor, | Regressão Logística, Random Forest Classifier, SVM, GBM, Naive Bayes, |
| | Decision Tree Regressor, Redes Neurais | Redes Neurais para Classificação |
| Exemplos de Uso | - Prever o preço de uma ação | - Classificar e-mails como "Spam" ou "Não Spam" |
| | - Estimar a renda mensal de uma pessoa | - Determinar se um cliente vai cancelar uma assinatura |
| | - Prever a temperatura de uma cidade no próximo dia | - Identificar a presença de uma doença com base em sintomas |
| | - Prever a quantidade de vendas de um produto | - Prever se um cliente irá comprar um produto |

O que é o SVM?

- O **Support Vector Machine (SVM)** é um algoritmo de aprendizado de máquina supervisionado usado para tarefas de **classificação** e **regressão**. Seu objetivo principal é encontrar a melhor forma de separar dados em diferentes categorias, por isso, sua principal aplicação é para problemas de Classificação.
- Imagine que você tem um conjunto de pontos em um gráfico, representando duas categorias diferentes, como maçãs e laranjas. O SVM procura o **hyperplano, linha** (em duas dimensões) ou o **plano** (em três dimensões), que melhor divide esses pontos em suas respectivas categorias. Mas não é qualquer linha, o SVM busca a linha que deixa a maior **margem** possível entre as duas categorias. Isso significa que está tentando fazer a separação de forma mais clara e definida possível.

No diagrama abaixo, H1, H2 e H3 são todos possíveis hyperplanos:



Vantagens e Desvantagem do SVM

| Vantagem | Descrição |
|---|---|
| Eficaz em Dados de Alta Dimensionalidade | O SVM funciona bem quando há muitos atributos ou características nos dados. Ex.: reconhecimento de texto. |
| Bom Desempenho com Poucos Dados | Funciona bem com poucos dados de treinamento. Útil em áreas com coleta limitada de dados. |
| Flexibilidade com Funções Kernel | Resolve problemas complexos com funções kernel que transformam os dados para facilitar a separação. |
| Robustez contra Overfitting | Generaliza bem para novos dados, evitando ajuste excessivo aos dados de treinamento. |
| Solução Ótima Global | O algoritmo SVM busca o máxima global. Pois a função objetivo é convexa |

| Desvantagem | Descrição |
|--|---|
| Escolha da Função Kernel Pode Ser Difícil | Selecionar a função kernel correta e ajustar seus parâmetros pode ser complexo. |
| Tempo de Treinamento com Grandes Conjuntos | O SVM pode ser lento para treinar quando há muitos exemplos, devido ao processamento necessário. |
| Dificuldade de Interpretação | O modelo resultante pode ser difícil de entender em termos de como cada característica afeta a decisão. |
| Estimativa de Probabilidades Requer Passos Adicionais | Calibração de Platt ou isotônica é necessária para obter probabilidades, aumentando a complexidade. |
| Sensível à Escala dos Dados | Características com escalas muito diferentes exigem normalização para que o SVM funcione bem. |

Aplicações do SVM

- **Reconhecimento de Imagens:**
 - Classificação de objetos em imagens, como identificar se há um carro ou uma pessoa
 - Reconhecimento de escrita à mão, como em leitura de cheques bancários
- **Bioinformática:**
 - Análise de sequências de DNA e proteínas
- **Detecção de Fraudes:**
 - Identificação de atividades suspeitas em transações financeiras
 - Proteção contra fraudes em cartões de crédito
- **Análise de Texto:**
 - Classificação de e-mails como spam ou não spam
- **Diagnóstico Médico:**
 - Auxílio na detecção de doenças com base em exames
 - Por exemplo, identificar tumores em imagens de ressonância magnética

Considerações Atuais sobre o SMV

- **Comparação com Redes Neurais Profundas:**

- As **redes neurais profundas** (deep learning) tornaram-se muito populares, especialmente para grandes conjuntos de dados e tarefas complexas, como reconhecimento de voz e imagens
- As redes neurais conseguem aprender diretamente a partir de dados brutos, enquanto o SVM geralmente requer que as features sejam extraídas ou selecionadas previamente

Quando usar o SMV:

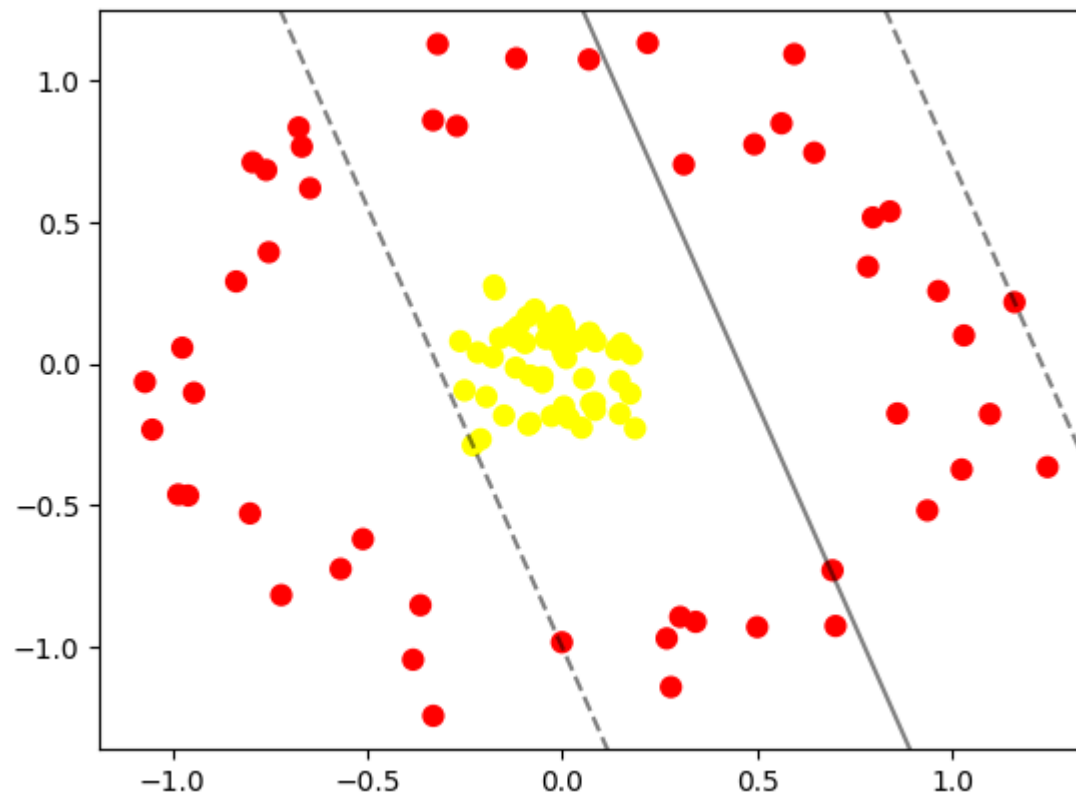
- O SVM ainda é muito útil quando você tem:
 - Conjuntos de dados médios
 - Dados bem estruturados
 - Necessidade de um modelo robusto com bom desempenho geral

Ferramentas Modernas Facilitam o Uso do SVM:

- Existem ferramentas e bibliotecas que tornam mais fácil ajustar os hiperparâmetros do SMV
- Isso ajuda a superar algumas das dificuldades na escolha da função kernel e outros fatores do modelo

Dados Não Linearmente Separáveis

Às vezes, não é possível separar os pontos de dados com uma linha reta. Nesses casos, o SVM usa uma técnica chamada **Kernel Trick** para transformar os dados em um espaço de maior dimensão, onde a separação linear é possível.



Função Kernel e Kernel Trick

O que é uma Função Kernel:

Uma técnica matemática que mapeia os dados de entrada (input data) para um espaço de maior dimensão, onde padrões podem ser mais facilmente identificados e classificados. Isso permite que os modelos capturem relações não lineares nos dados

Como Funciona o Kernel Trick?

O Kernel Trick se utiliza de uma função Kernel para projetar os dados em uma maior dimensão sem precisar realizar cálculos pesados normalmente associados com esses espaços. Isso é possível porque o produto escalar entre os dados mapeados no espaço de alta dimensão pode ser expresso em termos do produto escalar entre os dados de entrada no espaço original. Dessa forma, calculamos as similaridades necessárias sem explicitamente converter os dados para o espaço de alta dimensão

Exemplo Número Simples do Kernel trick com Kernel Polinomial de Grau 2

Vetores de Entrada

Considere os dois vetores de entrada em um espaço de duas dimensões:

$$\mathbf{x}_i = [x_{i1}, x_{i2}] = [1, 2]$$

$$\mathbf{x}_j = [x_{j1}, x_{j2}] = [3, 4]$$

Mapear para um Espaço de Alta Dimensionalidade

Para um kernel polinomial de grau 2 sem constante, o mapeamento

$$\phi(\mathbf{x})$$

para o espaço de alta dimensionalidade inclui todos os termos quadráticos e cruzados dos componentes dos vetores originais.

O mapeamento

$$\phi(\mathbf{x})$$

para

$$\mathbf{x} = [x_1, x_2]$$

é dado por:

$$\phi(\mathbf{x}) = [x_1^2, \sqrt{2}x_1x_2, x_2^2]$$

Aplicando o mapeamento aos nossos vetores:

$$\mathbf{x}_i = [1, 2]$$

$$\phi(\mathbf{x}_i) = [(1)^2, \sqrt{2} \times 1 \times 2, (2)^2] = [1, 2\sqrt{2}, 4]$$

Para

$$\mathbf{x}_j = [3, 4]$$

:

$$\phi(\mathbf{x}_j) = [(3)^2, \sqrt{2} \times 3 \times 4, (4)^2] = [9, 12\sqrt{2}, 16]$$

Exemplo Número Simples do Kernel trick com Kernel Polinomial de Grau 2

Calcular o Produto Escalar no Espaço de Alta Dimensionalidade

O produto escalar entre

$$\phi(\mathbf{x}_i)$$

e

$$\phi(\mathbf{x}_j)$$

é dado por:

$$\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle = (1)(9) + (2\sqrt{2})(12\sqrt{2}) + (4)(16)$$

$$\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle = 9 + 48 + 64 = 121$$

Calcular o Kernel Diretamente no Espaço Original

O kernel polinomial de grau 2 sem constante é definido como:

$$K(\mathbf{x}_i, \mathbf{x}_j) = (\langle \mathbf{x}_i, \mathbf{x}_j \rangle)^2$$

Calculando o produto escalar no espaço original:

$$\langle \mathbf{x}_i, \mathbf{x}_j \rangle = (1)(3) + (2)(4) = 3 + 8 = 11$$

Então, o kernel é:

$$K(\mathbf{x}_i, \mathbf{x}_j) = (11)^2 = 121$$

Verificamos que são iguais

Confirmamos que:

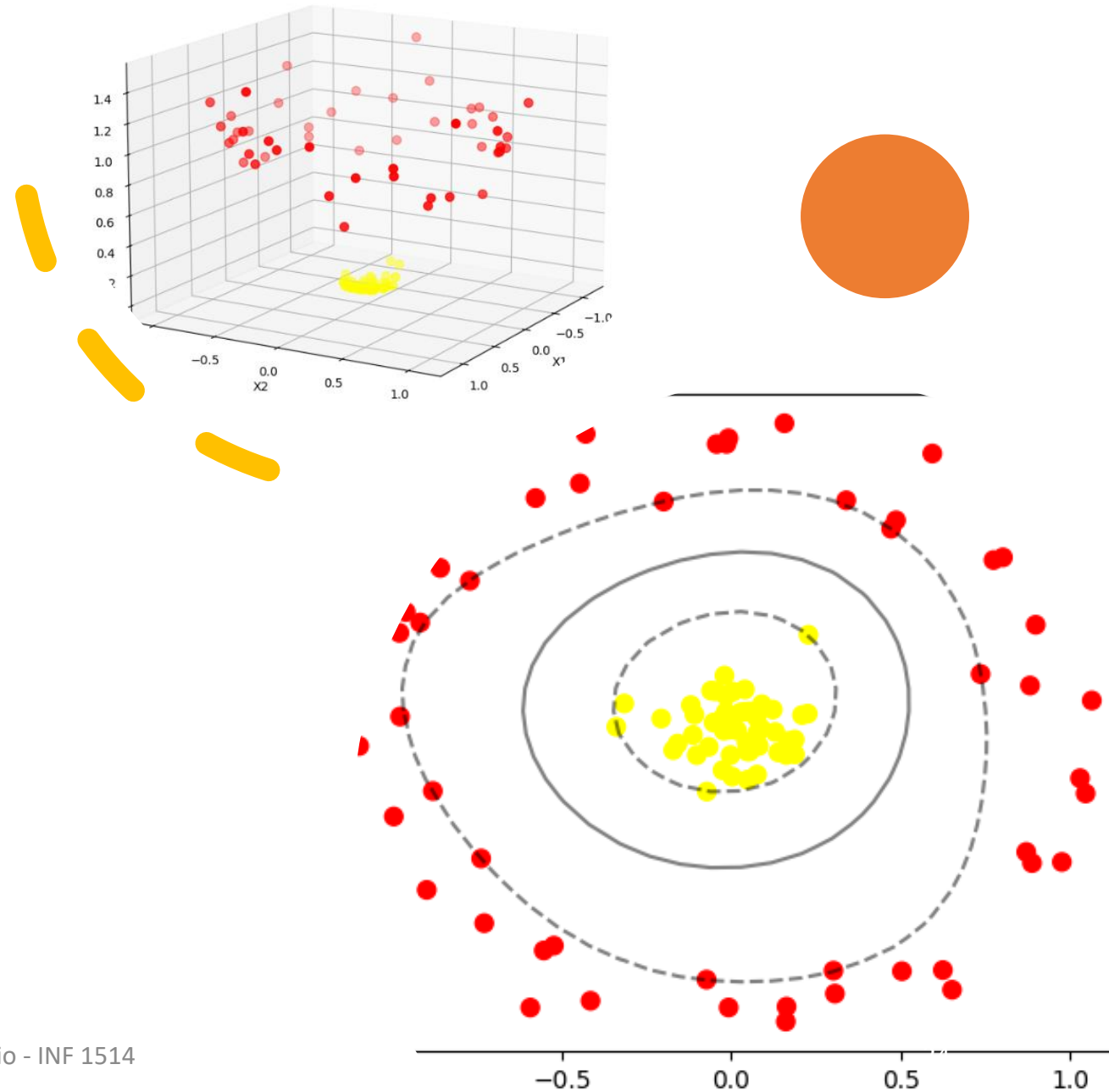
$$\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle = K(\mathbf{x}_i, \mathbf{x}_j) = 121$$

Isso demonstra que o produto escalar no espaço de alta dimensionalidade é igual ao valor do kernel calculado no espaço original.

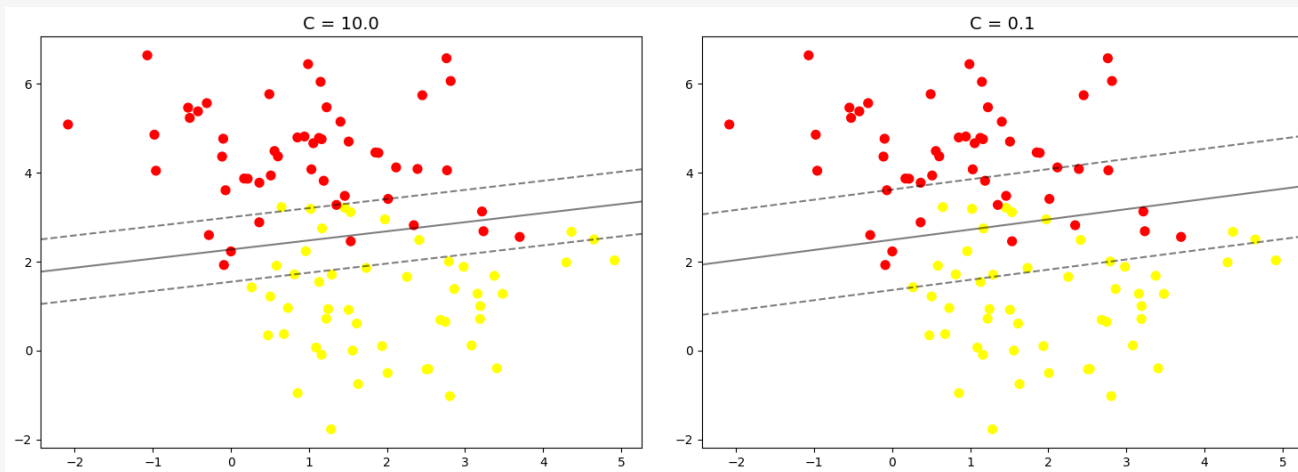
Resumindo: Transformamos uma função linear no espaço transformado em uma função quadrática no espaço original

Kernel Trick: Radial Basis Function (RBF)

- Vemos que usando uma transformação kernel, pela representação 3D ao lado, fica fácil encontrar um hiperplano que separa as duas classes, porém como vimos, fazer essa transformação direta para cada ponto de dados requer mais recursos computacionais, ainda mais quando usamos muitas features em datasets com muitas observações, o que é comum.
- Podemos, invés disso, usar funções de Kernel Trick já presentes no modelo SVM, como **RBF** (Radial Basis Function)



Pontos de Dados quase sobrepostos



- Nos exemplos passados, os pontos de dados podiam ser perfeitamente separados, porém, há casos em que isso não é possível e se faz necessário um “amaciação” da margem. Isso quer dizer que precisamos controlar a quantidade de pontos que iremos permitir estar dentro da margem se isso permitir um melhor treinamento do modelo.

- Essa maciez da margem é contralada por um dos hiperparâmetros do SVM: **C**

- C Grande:** “Margem mais dura”, os pontos de dados não podem estar dentro da margem
- C Pequeno:** “Margem mais macia”, os pontos de dados podem estar dentro da margem

Voltaremos falar sobre hiperparâmetros mais a frente

Target Label e Feature Engineering

A engenharia de feature é usada para criar variáveis explicativas que ajudam a prover mais informações ao modelo, para que possamos ter uma melhor previsão e melhorar na interpretação dos resultados.

Para um feature ser útil, é necessário que a mesma tenha uma relação com a variável **Target** que é justamente o que o modelo está tentando prever.

Exemplo Conceitual de Feature Engineering

Previsão do Desempenho Escolar

Imagine que você está desenvolvendo um modelo para prever se alunos irão passar ao final do ano letivo (**Target Label**, 1 se irá passar e 0 se não for passar). Você possui dados como:

- **Horas de estudo diárias**
- **Frequência às aulas**
- **Participação em atividades extracurriculares**
- **Distância de casa até a escola**
- **Uso de tecnologia em sala de aula**
- **Ambiente familiar**

Para melhorar o desempenho do seu modelo, você pode criar novas features que capturam melhor os fatores que influenciam o desempenho acadêmico. Por exemplo:

- **Total de horas de estudo semanais:** Multiplicar as horas de estudo diárias por 7 para obter uma visão semanal.
- **Índice de envolvimento escolar:** Combinar a frequência às aulas com a participação em atividades extracurriculares para medir o nível de engajamento do aluno.
- **Tempo de deslocamento diário:** Calcular o tempo gasto no trajeto casa-escola-casa, considerando a distância e o meio de transporte, o que pode afetar a disposição do aluno.
- **Suporte educacional em casa:** Criar uma variável que reflete o ambiente familiar, como disponibilidade de um local adequado para estudar ou apoio dos pais.
- **Uso efetivo da tecnologia:** Avaliar não apenas o uso da tecnologia em sala de aula, mas também como ela é utilizada para fins educacionais em casa.

Essas novas features têm uma relação direta com o desempenho escolar (**Target**) e podem ajudar o modelo a identificar padrões mais profundos que influenciam os resultados acadêmicos dos alunos.

Caso Prático

Data Dictionary - Caso Prático

Variable Definition Key

- **survival**: 0 = No, 1 = Yes
- **pclass**: Ticket class 1 = 1st, 2 = 2nd, 3 = 3rd
- **sex**: Sex
- **Age**: Age in years
- **sibsp** # of siblings / spouses aboard the Titanic
- **parch** # of parents / children aboard the Titanic
- **ticket**: Ticket number
- **fare**: Passenger fare
- **cabin**: Cabin number
- **embarked**: Port of Embarkation C = Cherbourg, Q = Queenstown, S = Southampton

Variable Notes

- **pclass**: A proxy for socio-economic status (SES)
- **1st** = Upper
- **2nd** = Middle
- **3rd** = Lower
- **age**: Age is fractional if less than 1. If the age is estimated, is it in the form of xx.5
- **sibsp**: The dataset defines family relations in this way...
- **Sibling** = brother, sister, stepbrother, stepsister
- **Spouse** = husband, wife (mistresses and fiancés were ignored)
- **parch**: The dataset defines family relations in this way...
- **Parent** = mother, father
- **Child** = daughter, son, stepdaughter, stepson
- **Some** children travelled only with a nanny, therefore parch=0 for them.

Feature Engineering - Conceitual

Para melhorar o desempenho do modelo, podemos criar novas features ou transformar as existentes:

- **Title:** Extrair o título (Sr., Sra., Srta., etc.) a partir do nome para capturar informações sobre status social ou estado civil.
- **FamilySize:** Combinar as features SibSp e Parch para criar uma nova variável que representa o tamanho total da família a bordo.
- **IsAlone:** Criar uma variável binária que indica se o passageiro estava viajando sozinho.
- **AgeGroup:** Categorizar a idade em grupos (criança, adolescente, adulto, idoso) para capturar relações não lineares com a sobrevivência.
- **FareBand:** Categorizar a tarifa paga em faixas para reduzir o impacto de valores extremos e capturar padrões.
- **Deck:** Extrair a letra do deck a partir da cabine para possivelmente relacionar a localização no navio com a probabilidade de sobrevivência
- **Porto de Embarque:** Transformar a variável categórica Embarked em dummy (one-hot encoding).

Feature Engineering

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Inicializar o LabelEncoder

label_encoder = LabelEncoder()

def preprocess_features(df, dataset_type="train"):
    """

    # 1. Extrair o título do nome
    df['Name'] = df['Name'].fillna('Unknown')
    df['Title'] = df['Name'].str.extract(' ([A-Za-z]+)\.',
    expand=False)

    # Mapeamento dos títulos para categorias mais simples
    title_mapping = {
        'Miss': 'Miss',
        'Mlle': 'Miss',
        'Dona': 'Miss',
        'Mrs': 'Mrs',
        'Lady': 'Mrs',
        'the Countess': 'Mrs',
        'Mme': 'Mrs',
        'Ms': 'Ms',
        # Adicione outros títulos se necessário
    }
    df['Name'] = df['Name'].astype(str)
    df['Title'] = df['Title'].map(title_mapping).fillna('Other')

    # Codificar os títulos
    df['Title'] = label_encoder.fit_transform(df['Title'])
```

```
# 2. Calcular o tamanho da família
df['FamilySize'] = df['SibSp'] +
df['Parch'] + 1 # +1 para incluir o próprio
passageiro

# 3. Indicar se o passageiro está sozinho
df['IsAlone'] = 1 # Assume que está
sozinho inicialmente
df.loc[df['FamilySize'] > 1, 'IsAlone'] =
0 # Se família > 1, não está sozinho

# 4. Tratar a idade
df['Age'] =
df['Age'].fillna(df['Age'].median()) #
Preencher valores faltantes com a mediana
# Criar grupos etários
age_labels = ['Child', 'Young Adult',
'Adult', 'Middle-aged', 'Senior']
df['AgeGroup'] = pd.cut(df['Age'], bins=5,
labels=age_labels)

# Codificar os grupos etários
df = pd.get_dummies(df,
columns=["AgeGroup"], prefix='AgeGroup')

# 5. Codificar o sexo
df["Sex"] =
label_encoder.fit_transform(df["Sex"])
```

Feature Engineering

```
# 6. Tratar a tarifa
df["Fare"] =
df["Fare"].fillna(df["Fare"].median()) # Preencher
valores faltantes com a mediana
df['FareBand'] = pd.cut(df['Fare'], 3,
labels=['Low', 'Medium', 'High'])
df['FareBand_encoded'] =
label_encoder.fit_transform(df['FareBand'])

# 7. Tratar a localização no navio
df['Cabin'] = df['Cabin'].fillna('Unknown') #
Preencher valores faltantes
df['Deck'] = df['Cabin'].str[0] # Extrair a letra
do convés
df = pd.get_dummies(df, columns=['Deck'],
prefix='Deck')

# 8. Codificar o porto de embarque
df = pd.get_dummies(df, columns=['Embarked'],
prefix='Embarked')

# 9. Remover colunas que não serão usadas
cols_to_drop = ["Ticket", "Fare", "FareBand",
"Age", "Name", "Cabin"]
df.drop(columns=cols_to_drop, inplace=True)
```

```
# Definir a lista de features que serão
usadas no modelo
features = [
    'Pclass', 'Sex', 'SibSp', 'Parch',
    'Title',
    'FamilySize', 'IsAlone',
    'FareBand_encoded',
    'Deck_A', 'Deck_B', 'Deck_C',
    'Deck_D', 'Deck_E', 'Deck_F', 'Deck_G',
    'Deck_T',
    'Embarked_C', 'Embarked_Q',
    'Embarked_S',
    'AgeGroup_Child', 'AgeGroup_Young
Adult', 'AgeGroup_Adult',
    'AgeGroup_Middle-aged',
    'AgeGroup_Senior'
]

return df, features

# Exemplo de uso:
# Supondo que 'train' e 'test' sejam seus
DataFrames de treino e teste

# Processar dados de treino
X, features = preprocess_features(train,
"train")

# Processar dados de teste
y, features = preprocess_features(test,
"test")
```

Encoding de Features Categóricas

| | PassengerId | Survived | Pclass | Sex | SibSp | Parch | Title | FamilySize | IsAlone | FareBand_encoded | ... | Deck_C | Deck_D | Deck_E | Deck_F | Deck_G | Deck_T | Deck_U | Embarked_C | Embarked_Q | Embarked_S |
|---|-------------|----------|--------|-----|-------|-------|-------|------------|---------|------------------|-----|--------|--------|--------|--------|--------|--------|--------|------------|------------|------------|
| 0 | 1 | 0 | 3 | 1 | 1 | 0 | 3 | 2 | 0 | 1 | ... | False | False | False | False | False | False | True | False | False | True |
| 1 | 2 | 1 | 1 | 0 | 1 | 0 | 1 | 2 | 0 | 1 | ... | True | False | False | False | False | False | False | True | False | False |
| 2 | 3 | 1 | 3 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | ... | False | False | False | False | False | False | True | False | False | True |

Como podemos ver na função **Features** e nos dados tratados acima, algumas features foram transformadas através duma técnica chamada **Encoding**, que consiste na transformação de uma variável **categórica** numa variável **numérica** ou **booleana**. Isso é necessário pois a maioria dos modelos não consegue lidar diretamente com variáveis categóricas.

Alguns tipos de Encoding e Usos:

Label Encoding Cada categoria única é mapeada para um número inteiro. As categorias são ordenadas alfabeticamente ou de acordo com a ordem padrão do dataset, normalmente é usada quando há um ordenamento padrão nos dados, como "low", "medium" e "high" ou quando há poucas categorias ex: "male" e "female", como nos dados do Titanic da feature "Sex". Pode não ser recomendada para categorias que não possuem relação ordinal.

One-Hot Encoding: Cria colunas binárias (0 ou 1) para cada categoria, é recomendada quando as variáveis categóricas não têm ordem e quando o número de categorias é pequeno. Porém, não é recomendada quando a variáveis tem muitas categorias únicas como por exemplo, no dataset do Titanic, temos a variável **"Ticket"**, se fossemos usar esse tipo de encoding nessa feature, criaríamos muitas colunas e isso dificultaria o treinamento do modelo.

Frequency Encoding: Substitui cada categoria pela frequência com que ela aparece no Dataset, assim conseguimos capturar a importância de cada categoria com base na frequência relativa mas não conseguimos capturar as relações diretas entre a categoria e a Target.

Ordinal Encoding: Similar ao Label Encoding, mas permite especificar uma ordem explícita entre as categorias, boa para variáveis como níveis de escolaridade ou categorias de risco.

Data Splitting

A divisão de dados é uma técnica fundamental não só para avaliar o desempenho do modelo mas também como garantir uma melhor generalização para dados não vistos. A separação ocorre da seguinte forma:

1. **Conjunto de Treinamento (Training Set - Treinamento/fit):** Usado para treinar o modelo
2. **Conjunto de Teste (Test Set - Validação):** Usado para avaliar o desempenho do modelo em dados não vistos no conjunto de treino

```
# Target - Variável a qual queremos prever, comumente armazenada na variável "y"
y = X["Survived"]
# Input Data contendo as features sem Target para prevenir Data Leak, normalmente
# armazenada na variável X
X = X[features]

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)

# test_size = 0.2: Reserva 20% dos dados para teste
# random_state = 42: Garante reprodutibilidade dos resultados
# stratify = y: Mantém a proporção original das classes nos conjuntos de treinamento
# e teste
```

Normalização/Padronização dos dados – Scaling

Normalização (Scaling)

O Modelo SVM Classifier é **sensível** à escala dos dados, por isso se faz necessária a normalização ou padronização das features numéricas para que estejam na mesma escala. Isso acontece pois o SVM busca encontrar o hiperplano ótimo que separa as classes, maximizando a margem entre elas.

Se as features tiverem escalas diferentes, aquelas com valores maiores podem dominar o cálculo das distâncias e influenciar indevidamente o modelo.

Algumas das Normalizações mais usadas e conhecidas são a Padronização e o MinMax.

Standard Scaling:

$$x_{\text{scaled}} = \frac{x - \mu}{\sigma}$$

- Transforma os dados para que tenham média 0 e desvio padrão 1. Isso é feito subtraindo a média e dividindo pelo desvio padrão para cada observação de determinada feature.
- Útil quando a feature segue uma distribuição normal
- Reduz o impacto de outlier mas ainda os considera

MinMax:

$$x_{\text{scaled}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

- Reescala os dados para que fiquem dentro de um intervalo específico, geralmente entre 0 e 1, utilizando os valores máximo e mínimos de determinada feature.
- Preserva a distribuição original dos dados
- Pode ser fortemente afetado por outliers, pois utiliza valores máximo e mínimo

Normalização/Padronização dos dados – Scaling

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Apply scaling
scaler = MinMaxScaler()
numeric_columns = X_train.select_dtypes(include=['number']).columns
X_train[numeric_columns] = scaler.fit_transform(X_train[numeric_columns])
X_test[numeric_columns] = scaler.transform(X_test[numeric_columns])
```

| | Pclass | Sex | SibSp | Parch | Title | FamilySize | IsAlone | FareBand_encoded | Deck_A | Deck_B | ... | Deck_G | Deck_T | Embarked_C | Embarked_Q | Embarked_S | AgeGroup_Child | AgeGroup_Young Adult | AgeGroup_Adult | AgeGroup_Middle-aged | AgeGroup_Senior |
|-----|--------|-----|-------|----------|-------|------------|---------|------------------|--------|--------|-----|--------|--------|------------|------------|------------|----------------|----------------------|----------------|----------------------|-----------------|
| 860 | 1.0 | 1.0 | 0.250 | 0.000000 | 1.0 | 0.2 | 0.0 | 0.5 | False | False | ... | False | False | False | False | True | False | False | True | False | False |
| 435 | 0.0 | 0.0 | 0.125 | 0.333333 | 0.0 | 0.3 | 0.0 | 0.5 | False | True | ... | False | False | False | False | True | True | False | False | False | False |
| 102 | 0.0 | 1.0 | 0.000 | 0.166667 | 1.0 | 0.1 | 0.0 | 0.5 | False | False | ... | False | False | False | False | True | False | True | False | False | False |

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Apply scaling
scaler = StandardScaler()
numeric_columns = X_train.select_dtypes(include=['number']).columns
X_train[numeric_columns] = scaler.fit_transform(X_train[numeric_columns])
X_test[numeric_columns] = scaler.transform(X_test[numeric_columns])
```

| | Pclass | Sex | SibSp | Parch | Title | FamilySize | IsAlone | FareBand_encoded | Deck_A | Deck_B | ... | Deck_G | Deck_T | Embarked_C | Embarked_Q | Embarked_S | AgeGroup_Child | AgeGroup_Young Adult | AgeGroup_Adult | AgeGroup_Middle-aged | AgeGroup_Senior |
|-----|-----------|-----------|-----------|-----------|-----------|------------|-----------|------------------|--------|--------|-----|--------|--------|------------|------------|------------|----------------|----------------------|----------------|----------------------|-----------------|
| 860 | 0.813034 | 0.724310 | 1.230569 | -0.479342 | 0.702416 | 0.634859 | -1.231219 | -0.100487 | False | False | ... | False | False | False | False | True | False | False | True | False | False |
| 435 | -1.614136 | -1.380624 | 0.379923 | 2.048742 | -1.671573 | 1.229621 | -1.231219 | -0.100487 | False | True | ... | False | False | False | False | True | True | False | False | False | False |
| 102 | -1.614136 | 0.724310 | -0.470722 | 0.784700 | 0.702416 | 0.040096 | -1.231219 | -0.100487 | False | False | ... | False | False | False | False | True | False | True | False | False | False |

Otimização de Hiperparâmetros

Otimização de Hiperparametros

Já abordados anteriormente, hiper parâmetros são importantes pois melhoram a performance do modelo ao otimizar o processo de treino. No entanto, hiper parâmetros mal otimizados podem resultar em modelos underfitted ou overfitted. Veremos uma maneira relativamente menos complexa de otimizá-los, mesmo que você não esteja muito familiarizado com o que cada um deles faz. Existem métodos de escolha automatizados que utilizaremos a partir de agora.

Conceito de Espaço de Hiperparâmetros

Primeiramente, vamos aprofundar esse conceito. O Espaço de Hiperparâmetros é o conjunto de possíveis combinações de hiperparâmetros que podem ser usados para treinar o modelo. É um espaço multidimensional, com cada dimensão representando um hiperparâmetro diferente. Por exemplo, no SVM Classifier temos C e Gamma, então o espaço teria 2 dimensões: uma para **C** e outra para **Gamma**. Além disso, cada hiper parâmetro está restrito a uma distribuição de valores que pode assumir, definindo assim seu range.

Otimização de Hiperparâmetros

Objetivo

Sendo assim, a otimização de hiperparâmetros nada mais é do que procurar a melhor combinação de valores para cada um deles que resulte no melhor resultado/treinamento do modelo, algumas das técnicas empregadas são:

Grid Search: Considerada uma abordagem de "força bruta". Basicamente treina o modelo com todas as combinações possíveis de hiperparâmetros e o escolhe o que teve a melhor performance. É computacionalmente cara.

Random Search: Como o nome sugere, procura combinações de hiperparâmetros de forma randômica, mas mesmo assim produz resultados relativamente parecidos com o Grid Search de forma mais rápida.

Bayesian Optimization: Apesar dos outros dois serem técnicas de Otimização de hiperparâmetros, o Bayesiana sim trata a questão como um problema probabilístico. Esse método considera os resultados da última combinação de hiperparâmetros e seleciona a próxima combinação com base numa função de probabilidade que irá, teoricamente, resultar numa performance melhor. Essa função pode, por exemplo, tentar minimizar o Erro Médio Quadrado nas iterações subsequentes no espaço de hiperparâmetros.

State of the Art: Optuna

Usaremos uma das melhores biblioteca atualmente disponível para otimização, **Optuna**. O Optuna é uma biblioteca de otimização de **hiperparâmetros** que utiliza técnicas avançadas baseado em **Bayesian Optimization** para encontrar as melhores combinações de forma eficiente e flexível.

Busca mais eficiente: Começa por uma alguns conjuntos amostrais de hiperparâmetros e avalia suas performances, então cria um modelo para prever quais hiperparametros performariam melhor baseado nos resultados anteriores. Isso permite ao modelo forcas nas áreas do ****Search Space**** mais promissora.

Tree-structure Parzen Estimator (TPE): Ao invés de usar um processo Gaussiano como métodos Bayesianos tradicionais usam, o TPE modela uma "Objective Function" usando duas funções de densidade de probabilidade, uma para o melhor conjunto de hiperparametro e outra para os piores que esse, a seguir, a distribuição dessas amostras é usada para fazer um novo conjuntos que tem uma probabilidade maior de performar melhor

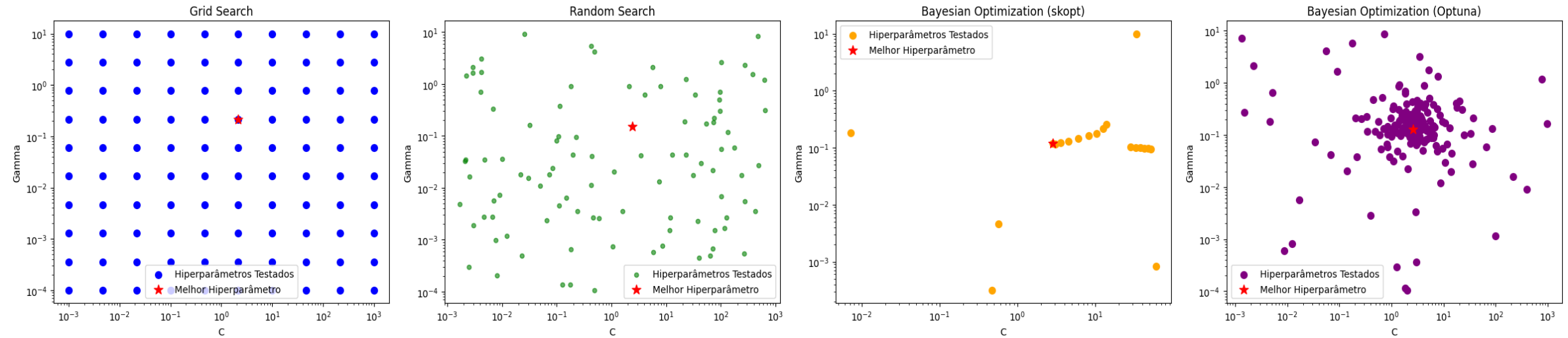
Multi-Objective Optimization: Permite uma otimização que equilibra mais de um objetivo, como por exemplo, balancear o Trade-off entre métricas de avaliação de modelo diferentes, como Precisão (**accuracy**) e **Recall** (Falso Positivos, Negativos). Isso é possível pois o algoritmo encontra uma gama de soluções que não são estritamente melhores que umas as outras em todos os objetivos especificados.

Resumo

Usando TPE, duas funções de densidade de probabilidade são construídas: $l(x)$ para bons resultados e $g(x)$ para resultados ruins, o algoritmo cria amostras de novos hiperparametros que maximizam a divisão $l(x)/g(x)$.

Através de outros cálculos matemáticos, ela garante que as novas amostras sejam retiradas das regiões do **Search Space** que há mais probabilidade de que bons conjuntos de hiperametros sejam encontrados.

State of the Art: Optuna



1. Grid Search:

$C = 2.1544$

$\Gamma = 0.2154$

Score (Objective): 0.2702

Tempo de Execução: 0.00 segundos

2. Random Search:

$C = 2.4375$

$\Gamma = 0.1501$

Score (Objective): 0.0227

Tempo de Execução: 0.00 segundos

3. Bayesian Optimization (scikit-optimize):

$C = 2.8428$

$\Gamma = 0.1182$

Score (Objective): 0.0204

Tempo de Execução: 4.19 segundos

4. Bayesian Optimization (Optuna):

$C = 2.5739$

$\Gamma = 0.1295$

Score (Objective): 0.0049

Tempo de Execução: 2.04 segundos

Métricas de Avaliação do Modelo

As métricas de avaliação de modelos diferem para modelos de **Regressão** e **Classificação**.

Regressão

Para modelos de regressão, que preveem valores contínuos, as métricas mais comuns são:

- **Mean Absolute Error (MAE):**

Mede a média das diferenças absolutas entre as previsões e os valores reais. É fácil de interpretar e robusto a outliers.

$$MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

- **Mean Squared Error (MSE):**

Calcula a média dos quadrados das diferenças entre as previsões e os valores reais. Penaliza erros maiores de forma mais severa que o MAE.

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

- **Root Mean Squared Error (RMSE):**

É a raiz quadrada do MSE, trazendo a métrica de volta à mesma unidade dos dados originais, facilitando a interpretação.

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$$

- **R-squared (R^2):**

Indica a proporção da variabilidade dos dados que é explicada pelo modelo. Valores próximos de 1 indicam um bom ajuste.

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$$

Métricas de Avaliação do Modelo

As métricas de avaliação de modelos diferem para modelos de **Regressão** e **Classificação**.

Classificação

Para modelos de classificação, que atribuem categorias a entradas, as métricas mais comuns são:

- **Accuracy:**

Mede a proporção de previsões corretas em relação ao total de previsões.

$$\text{Accuracy} = \frac{\text{Número de Previsões Corretas}}{\text{Total de Previsões}}$$

- **Precision:**

Indica a proporção de verdadeiros positivos em relação ao total de positivos previstos. É útil quando o custo de falsos positivos é alto.

$$\text{Precision} = \frac{TP}{TP + FP}$$

- **Recall (Sensitivity):**

Mede a proporção de verdadeiros positivos em relação ao total de positivos reais. É importante quando o custo de falsos negativos é alto.

$$\text{Recall} = \frac{TP}{TP + FN}$$

- **F1-Score:**

É a média harmônica entre precision e recall, proporcionando um equilíbrio entre ambas.

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- **Area Under the ROC Curve (AUC-ROC):**

Avalia a capacidade do modelo em distinguir entre classes, plotando a taxa de verdadeiros positivos contra a taxa de falsos positivos.

ROC Curve

A curva ROC exibida no gráfico avalia o desempenho de um modelo de classificação binária

O eixo X representa a Taxa de **Falsos Positivos (FPR - False Positive Rate)**. Quanto menor, melhor, pois significa que o modelo comete menos erros ao prever observações negativos como positivos.

O eixo Y representa a Taxa de Verdadeiros **Positivos (TPR - True Positive Rate)** ou **Recall**. Quanto maior, melhor, pois indica que o modelo consegue identificar corretamente observações positivas.

Curva ROC (linha azul):

A curva azul mostra a relação entre a **TPR** e a **FPR** para diferentes limiares de decisão. Quanto mais próximo essa curva está do canto superior esquerdo, melhor o modelo.

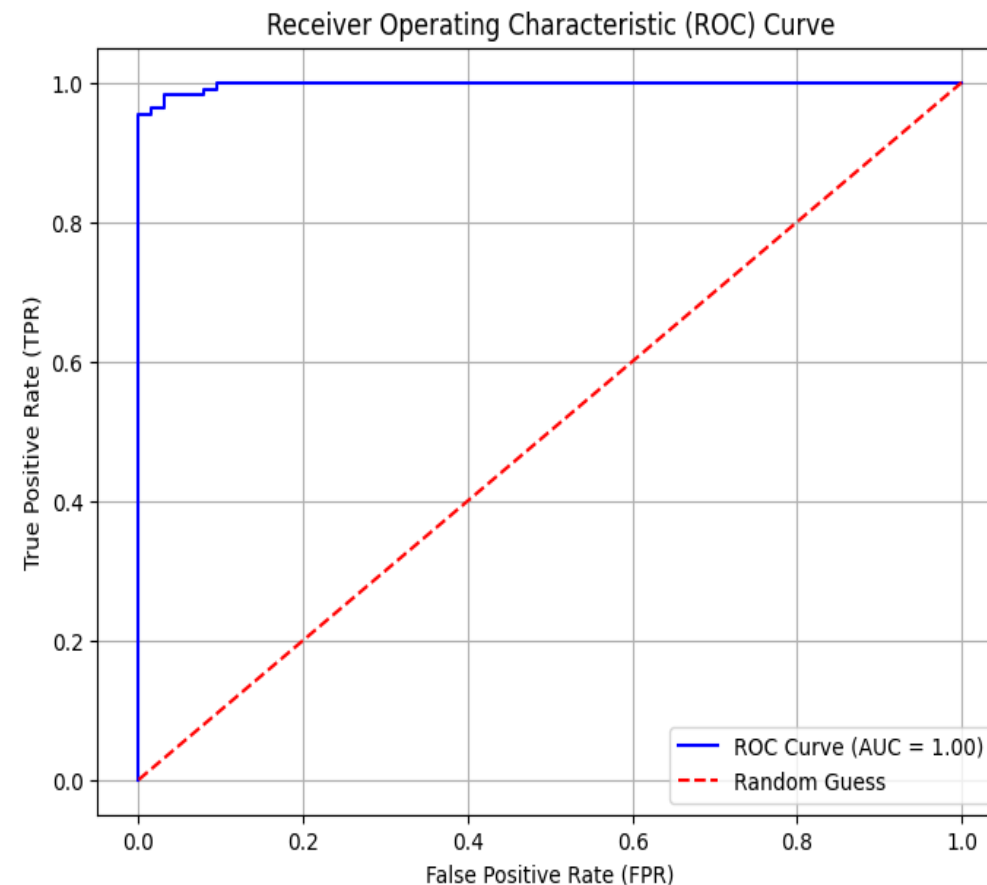
Linha diagonal (linha vermelha):

Essa linha representa uma adivinhação aleatória (Random Guess). Um modelo com desempenho equivalente a essa linha não seria útil, pois suas previsões seriam aleatórias.

AUC (Área Sob a Curva):

Desempenho excelente: Uma AUC igual a 1.00 indica que o modelo é capaz de prever corretamente todas as classes positivas e negativas sem cometer erros.

Sem falsos positivos ou falsos negativos: A curva azul atinge a TPR de 1 (ou 100%) antes mesmo de a FPR aumentar, indicando que o modelo não comete erros em nenhuma situação para este conjunto de dados.

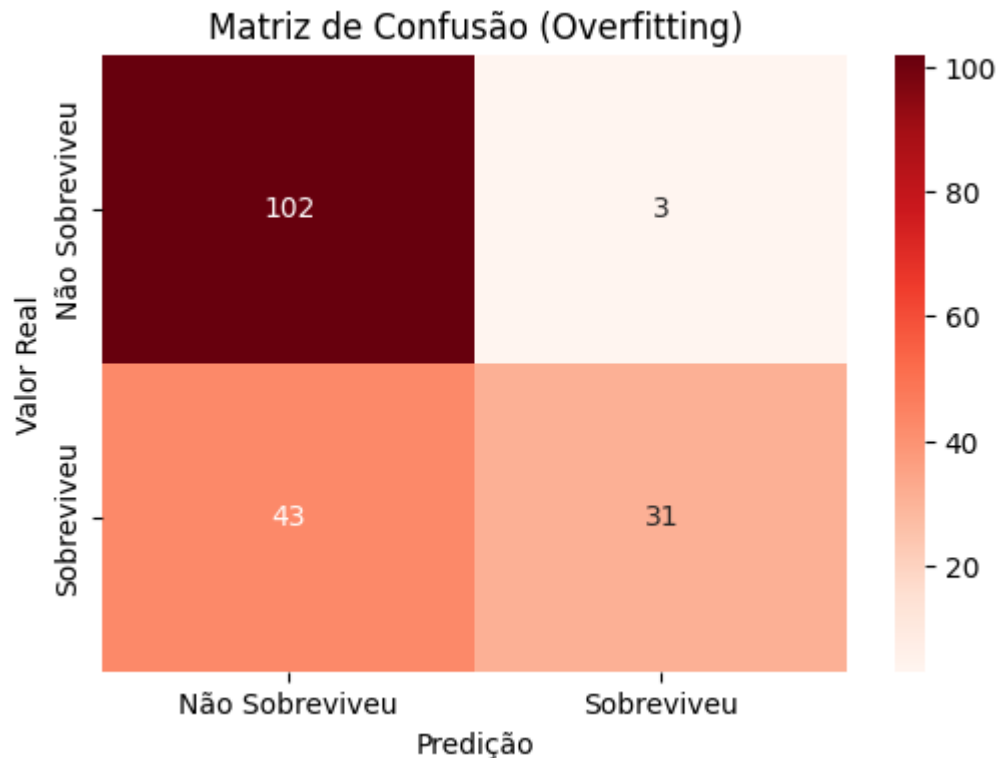


Matriz de Confusão (Confusion Matrix)

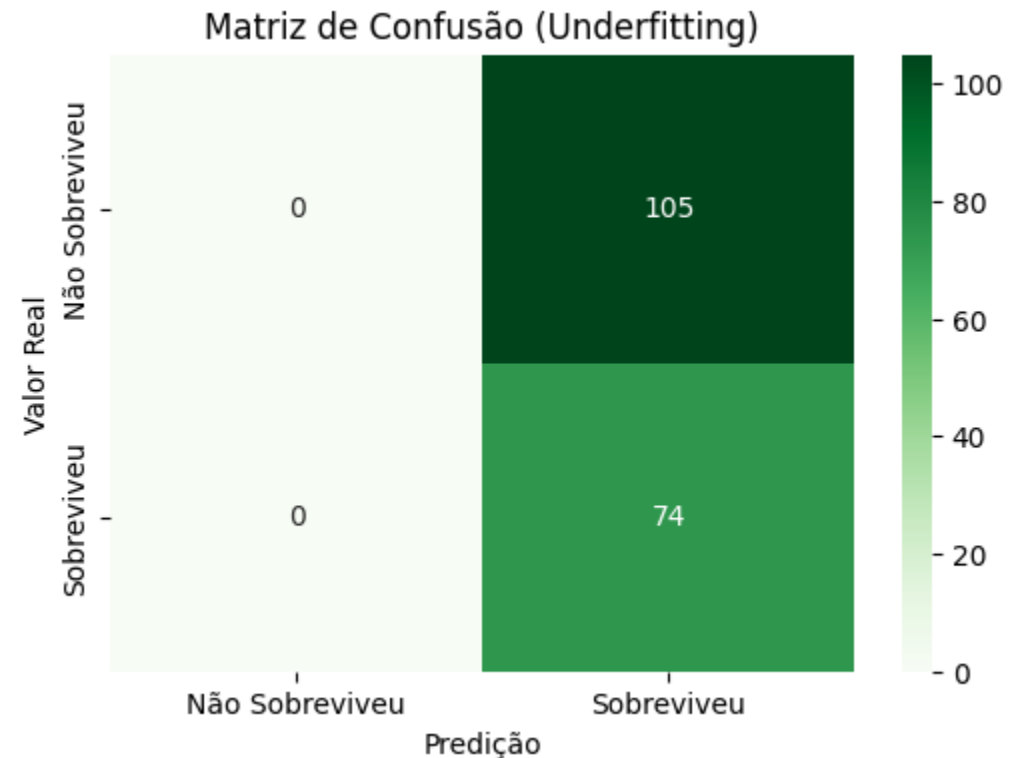
Apresenta uma tabela ou gráfico com os verdadeiros positivos, verdadeiros negativos, falsos positivos e falsos negativos, fornecendo uma visão detalhada do desempenho do modelo

| | Previsão Positivo | Previsão Negativo |
|---------------|-------------------|-------------------|
| Real Positivo | True Positive | False Negative |
| Real Negativo | False Positive | True Negative |

Acurácia no Conjunto de Treinamento (Overfitting): 0.82
Acurácia no Conjunto de Teste (Overfitting): 0.74



Acurácia no Conjunto de Treinamento (Underfitting): 0.38
Acurácia no Conjunto de Teste (Underfitting): 0.41



Overfitting

Como podemos ver pelos resultados no slide passado, o modelo **vermelho** apresenta um leve **overfitting**. Overfitting ocorre quando o modelo performa bem nos dados de treinamento, mas tem um desempenho bem inferior nos dados de teste. Isso acontece devido a dois fatores principais:

Alta complexidade na escolha dos Hiperparametro: Quanto mais específico o modelo for configurado através dos hiperparâmetros, maior a chance de ele se ajustar demais aos dados de treinamento. Isso resulta em maior erro e menos previsões corretas nos dados de teste, mesmo que o modelo consiga capturar bem os padrões dos dados de treinamento. As relações aprendidas podem não generalizar bem para novos dados, já que o modelo pode ter capturado ruídos específicos do conjunto de treinamento.

Muitas variáveis pouco explicativas: Quando o modelo inclui muitas variáveis que não contribuem significativamente para a previsão, ele pode criar relações empiricamente quase inexistentes entre essas features e o alvo. No conjunto de treinamento, isso pode elevar o score, pois o modelo se aproveita dessas relações que estão de fato presentes nessa amostra de dados. Contudo, como os dados de teste é outra amostra de dados, essas relações não se mantêm, o que resulta em queda na acurácia.

Observação Adicional: As vezes se faz necessário o Balanceamento de Classes, para evitar que o modelo favoreça uma em relação a outra. Por exemplo, se houver mais classe "1" do que "0", mas não abordaremos esse tema. Mas algumas das técnicas de ajuste de Balanceamento mais usadas são:

Oversampling: Duplica ou cria exemplos artificiais da classe minoritária.

Undersampling: Remove observações da classe majoritária.

Não iremos abordar esses conceitos pois assim como as Árvores de Decisão, o modelo SVM permite ajustar os pesos das classes para dar mais importância à classe minoritária através de um simples parâmetro:

class_weight

Underfitting

Já o Modelo acima **Verde** apresenta um claro e forte underfitting, que ocorre quando modelo não generaliza bem nem nos dados de treino e nem nos dados de teste, pelos seguintes fatores:

Escolha Inadequada do Kernel: Nos capítulos anterior vimos que o SVM utiliza diferentes tipos de kernels, se o kernel escolhido for muito simples (como o linear) para o complexidade dos dados, o modelo pode não capturar os padrões necessários.

Parâmetro de Regularização (C) Muito Alto: Como vimos, o Parâmetro C controla o trade-off entre maximizar a margem e minimizar o erro de classificação. Um valor muito alto pode fazer com que o modelo se concentre excessivamente em minimizar os erros de treinamento, mas na verdade, no underfitting, geralmente um **C muito baixo** faz o oposto: mais erros de classificação no treinamento para aumentar a margem, simplificando de mais o modelo.

Dados de Treinamento Insuficientes ou Não Explicativos: Por vezes o mais importante, mesmo com um SVM poderoso, se os dados de treinamento não forem suficientes ou não representarem bem a variabilidade dos dados reais, o modelo pode não aprender padrões relevantes, resultando em desempenho ruim tanto no treino quanto no teste.

Optuna Objective- Configuração

```
from sklearn.model_selection import StratifiedKFold, TimeSeriesSplit, KFold, ShuffleSplit
from sklearn.metrics import accuracy_score, roc_auc_score, f1_score
def objective(trial):
    # Define o espaço de busca dos hiperparâmetros
    C = trial.suggest_float('C', 1e-3, 1e3, log=True)
    kernel = trial.suggest_categorical('kernel', ['linear', 'poly', 'rbf', 'sigmoid'])

    if kernel == 'poly':
        degree = trial.suggest_int('degree', 2, 12)
    else:
        degree = 3 # Valor padrão quando se é usado o Kernel "Poly"

    gamma = trial.suggest_categorical('gamma', ['scale', 'auto'])
    coef0 = trial.suggest_float('coef0', -1.5, 1.5) if kernel in ['poly', 'sigmoid'] else 0.0
    shrinking = trial.suggest_categorical('shrinking', [True, False])
    tol = trial.suggest_float('tol', 1e-6, 1e-2, log=True)
    cache_size = trial.suggest_int('cache_size', 100, 10000)
    class_weight = trial.suggest_categorical('class_weight', ['balanced', None])
    max_iter = trial.suggest_int('max_iter', 1000, 30000)
    random_state = 42
    decision_function_shape = trial.suggest_categorical('decision_function_shape', ['ovr', 'ovo'])
    break_ties = trial.suggest_categorical('break_ties', [False])
```

Optuna Objective - Configuração

```
# Criando um pipeline para o modelo
pipeline = Pipeline([
    # ('scaler', StandardScaler()), # Comentado pois os dados já estão escalados,
    #                               # mas escalalos no pipeline é boa prática
    ('svc', SVC(
        C=C,
        kernel=kernel,
        degree=degree,
        gamma=gamma,
        coef0=coef0,
        shrinking=shrinking,
        probability=False, # Se TRUE é retornado qual a probabilidade de o ponto estar em
determinada classe, invés da classe
        tol=tol,
        cache_size=cache_size,
        class_weight=class_weight,
        max_iter=max_iter,
        random_state=random_state,
        decision_function_shape=decision_function_shape,
        break_ties=break_ties
    ))
])
```

Optuna Objective - Configuração

```
scores = []

# Avaliação cruzada (Cross-
Validation)
cv = ShuffleSplit(n_splits=2, test_size = 0.2,
random_state=42)

for train_idx, test_idx in cv.split(X_train, y_train):
    X_train_cv, X_test_cv = X_train.iloc[train_idx], X_train.iloc[test_idx]
    y_train_cv, y_test_cv = y_train.iloc[train_idx], y_train.iloc[test_idx]

    # Treinando o modelo
    pipeline.fit(X_train_cv, y_train_cv)

    # Previsão e avaliação
    y_pred = pipeline.predict(X_test_cv)
    score = f1_score(y_test_cv, y_pred, average='weighted')

    scores.append(score)

# Retorna a média do F1 score entre todos os splits
return np.mean(scores)
```

Optuna Study

```
# Study
study = optuna.create_study(
    direction='maximize',
    sampler=TPESampler(seed=42))

# Otimizando a função Objetivo
study.optimize(objective, n_trials=1000, n_jobs=-1)
```

N_Jobs = -1 Paraleliza o estudo por multithreading (Ver Jupyter para multiprocessing)

Melhores Parâmetros

```
best_params = study.best_params.copy()
# Generate features with the best window sizes
# Create the pipeline with the best SVC parameters
final_pipeline = Pipeline([
    ('svc', SVC(
        C=best_params['C'],
        kernel=best_params['kernel'],
        degree=best_params.get('degree', 3),
        gamma=best_params['gamma'],
        coef0=best_params.get('coef0', 0.0),
        shrinking=best_params['shrinking'],
        probability=False, # Binary predictions
        tol=best_params['tol'],
        cache_size=best_params['cache_size'],
        class_weight=best_params['class_weight'],
        max_iter=best_params['max_iter'],
        random_state=42,
    ))
])

# Fit the pipeline on the training data
final_pipeline.fit(X_train, y_train)
```

Previsões

```
# Fazer predições no conjunto de teste -> Novo modelo (AUC ROC)
decision_scores = final_pipeline.decision_function(X_test) # Para ROC AUC

# Calcular ROC AUC no conjunto de teste
test_auc_roc = roc_auc_score(y_test, decision_scores)
print(f"\nTest Set ROC AUC Score: {test_auc_roc:.4f}")

# Mantém F1-score como métrica complementar (se necessário)
predictions = final_pipeline.predict(X_test)
test_f1_score = f1_score(y_test, predictions, average='weighted')
print(f"\nTest Set F1 Score (Complementary): {test_f1_score:.4f}")

# Relatório de Classificação
print("\nClassification Report:")
print(classification_report(y_test, predictions))

# Matriz de Confusão
conf_matrix = confusion_matrix(y_test, predictions)
print("\nMatriz de Confusão:")

plt.figure(figsize=(6,4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Não Sobreviveu', 'Sobreviveu'],
            yticklabels=['Não Sobreviveu', 'Sobreviveu'])
plt.ylabel('Valor Real')
plt.xlabel('Predição')
plt.title('Matriz de Confusão')
plt.show()
```

Avaliação do Modelo

Test Set F1 Score: 0.8049

Test Set ROC AUC Score Complementary): 0.8286

Classification Report:

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.84 | 0.83 | 0.84 | 105 |
| 1 | 0.76 | 0.78 | 0.77 | 74 |
| accuracy | | | 0.81 | 179 |
| macro avg | 0.80 | 0.81 | 0.80 | 179 |
| weighted avg | 0.81 | 0.81 | 0.81 | 179 |

