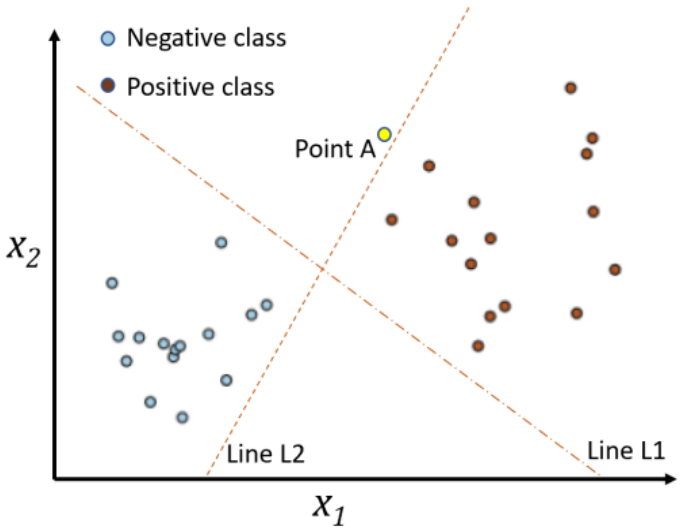
Máquinas de Vetores de Suporte: Da Classificação à Regressão

Introdução às Máquinas de Vetores de Suporte

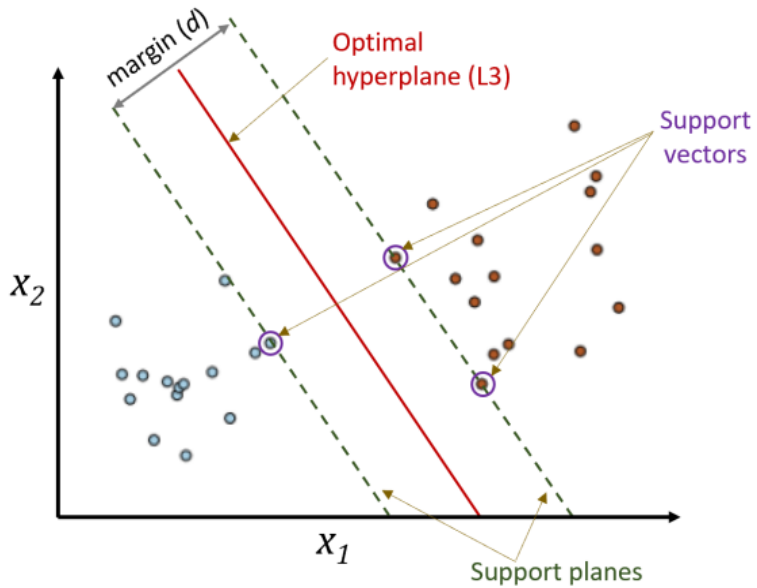
- Técnica de aprendizado supervisionado para classificação e regressão
- Originalmente desenvolvida para problemas de classificação binária
- Visa encontrar limites de separação ótimos entre classes
- Conhecida por excelentes capacidades de generalização
- Garante soluções únicas de ótimo global

SVM Clássico: Princípios Básicos

- Encontra limite linear para separar duas classes distintas
- Maximiza a margem entre planos de suporte
- Vetores de suporte: Pontos situados nos planos de suporte
- Margens grandes garantem previsões robustas
- Apenas vetores de suporte determinam o limite ótimo


Negative class
Positive class
Point A
Line L2
Line L1
x_2
x_1

(a)


margin (d)
Optimal
hyperplane (L3)
Support
vectors
Support planes
x_2
x_1

(b)

Fundamento Matemático do SVM

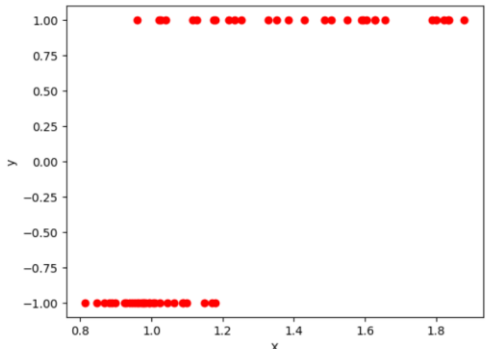
- Para N amostras de treinamento (x, y) onde:
  - x: Vetor de entrada no espaço m-dimensional
  - y: Rótulo de classe (±1)
- Hiperplano de separação ótimo: $\boldsymbol{w}^T\boldsymbol{x} + b = 0$
- Planos de suporte:

$$\begin{aligned} \boldsymbol{w}^T\boldsymbol{x}_i + b &\geq 1 \quad \text{for positive samples } (y_i = 1) \\ \boldsymbol{w}^T\boldsymbol{x}_i + b &\leq -1 \quad \text{for negative samples } (y_i = -1) \end{aligned}$$

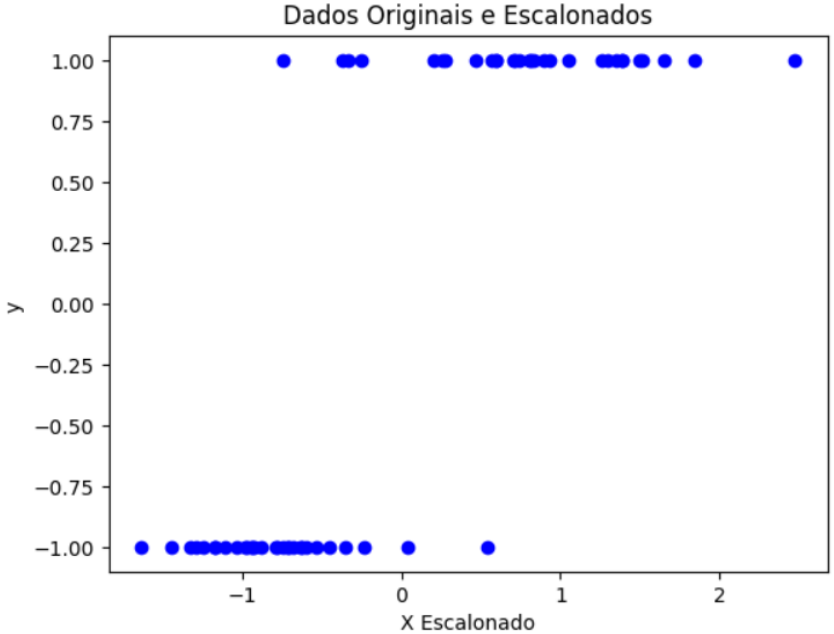
$$\begin{aligned} \min_{\boldsymbol{w},b} \quad & \frac{1}{2}\left|\left|\boldsymbol{w}\right|\right|^2 \qquad \text{eq. 1} \\ \text{s.t.} \quad & y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1, \qquad i = 1, \cdots N \end{aligned}$$

Ilustração de classificação linear simples

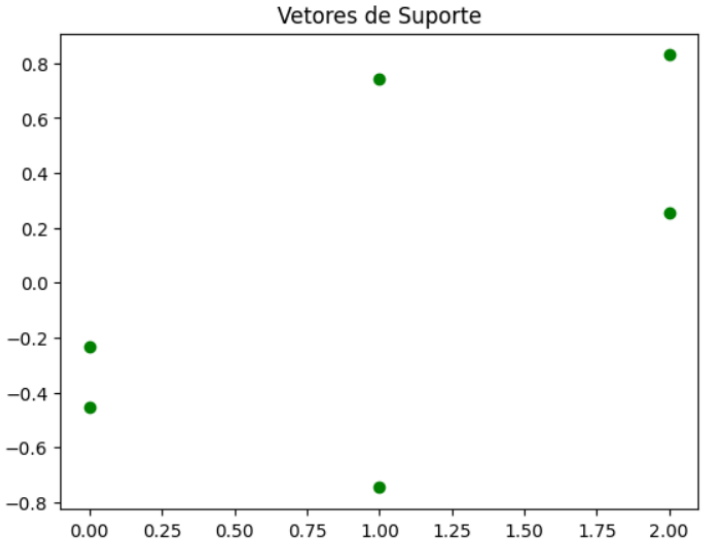
```
# Ler os dados de um arquivo CSV
import numpy as np
# Carrega os dados numéricos do arquivo, usando vírgula como separador
data = np.loadtxt('toyDataset.csv', delimiter=',')
# Separa as features (X) e a variável alvo (y):
# - X recebe as duas primeiras colunas (características)
# - y recebe a terceira coluna (rótulo/target)
X = data[:, [0, 1]]; y = data[:, 2]
```



```python
# Escalonamento das features para normalização
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()  # Cria um objeto para padronização (média=0, variância=1)
X_scaled = scaler.fit_transform(X)  # Aplica o escalonamento aos dados de entrada
```



```python
# Treinamento do modelo SVM (Máquina de Vetores de Suporte)
from sklearn.svm import SVC
# Configura o modelo:
# - kernel linear: para separação linear de classes
# - C=100: hiperparâmetro de regularização (C alto = menos regularização)
model = SVC(kernel='linear', C=100)
model.fit(X_scaled, y)  # Treina o modelo com dados escalonados
```




```
Coeficientes do modelo: [[1.9276981  0.59658647]]
Intercepto do modelo: [0.01725418]
Número de vetores de suporte por classe: [1 2]
Número total de vetores de suporte: 3
```

Classificação de Margem Rígida vs. Margem Suave

- Margem rígida: Todos os pontos devem ser corretamente classificados
- Margem suave: Permite algumas classificações incorretas
- Introduz variáveis de folga (ξ)
- Controla o equilíbrio entre precisão e tamanho da margem
- Hiperparâmetro C regula a penalidade para violações

Classificação "Hard Margin" (Margem Dura):

Imagine que você quer separar dois grupos de pontos com uma linha reta perfeita, sem nenhum erro. Isso só funciona se os dados forem perfeitamente separáveis (como no "toyDataset"). O SVM com margem dura exige que todos os pontos estejam do lado correto da linha, sem exceções.

Problema: Na vida real, os dados têm "sujeira"
(pontos fora do lugar, ruídos, sobreposição).

A margem dura falha porque
não permite flexibilidade.

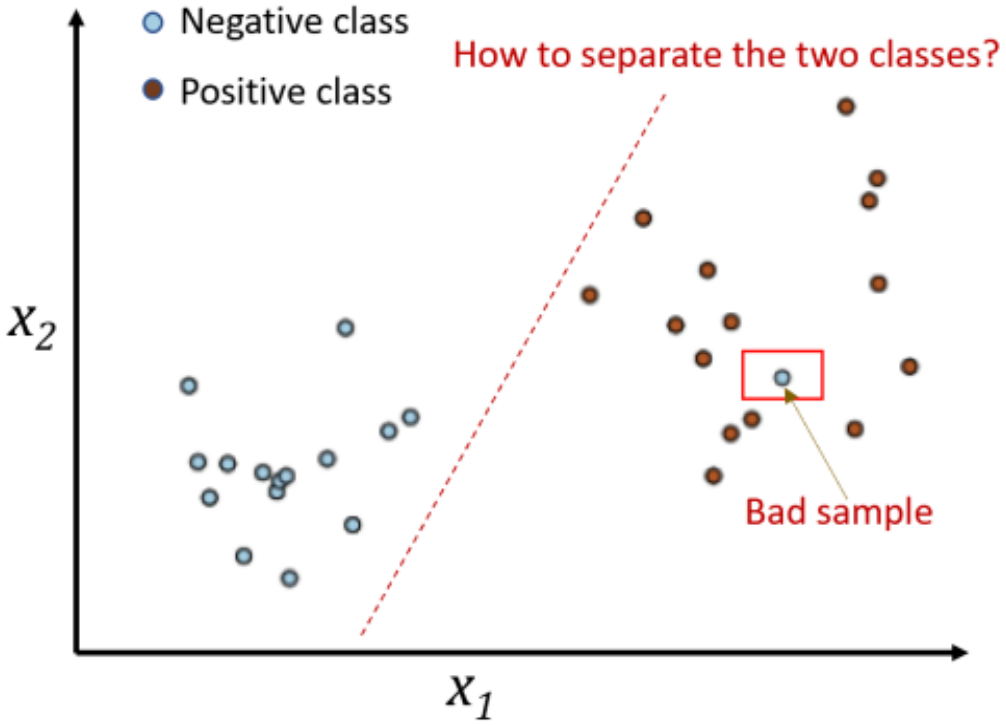


Figura 7.2: A presença da amostra ruim mostrada torna a separação linear perfeita inviável

Entendendo a Margem Flexível (Soft Margin) no SVM:

No SVM de margem flexível, permitimos que alguns pontos sejam classificados erradamente ou fiquem dentro da margem. Isso é feito usando variáveis de "folga" (ξ), que medem o "quanto" cada ponto errou.

$$\boldsymbol{w}^T\boldsymbol{x}_i + b \geq 1 - \xi_i \quad \text{for } y_i = 1$$
$$\boldsymbol{w}^T\boldsymbol{x}_i + b \leq -1 + \xi_i \quad \text{for } y_i = -1$$

Variaveis de folga

$$\min_{\boldsymbol{w},b,\xi} \quad \frac{1}{2}\left|\left|\boldsymbol{w}\right|\right|^2 + C\sum_{i=1}^{N}\xi_i \qquad \text{eq. 2}$$
$$\text{s.t.} \quad y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1 - \xi_i, \quad i = 1, \cdots N$$
$$\xi_i \geq 0$$

Como funciona?

1. **Objetivo do Modelo:**
   - **Minimizar dois termos:**
     - `(1/2)||w||²` **: Queremos uma linha de decisão "suave" (margem larga).**
     - `C * Σξi` **: Penalizamos os erros (ξi).**
   - **C é um hiperparâmetro que controla o peso dos erros:**
     - **C alto** **(ex: C=100): Foca em reduzir erros (pouca tolerância).**
     - **C baixo** **(ex: C=0.1): Prioriza margem larga (mais tolerância a erros).**
2. **Restrições:**
   - **Cada ponto deve obedecer:**

     $yi(wTxi + b) \geq 1 - \xi i$
   - **$\xi_i \geq 0$: A folga nunca é negativa (só mede o "erro para dentro").**

C como hiperparâmetro de regularização

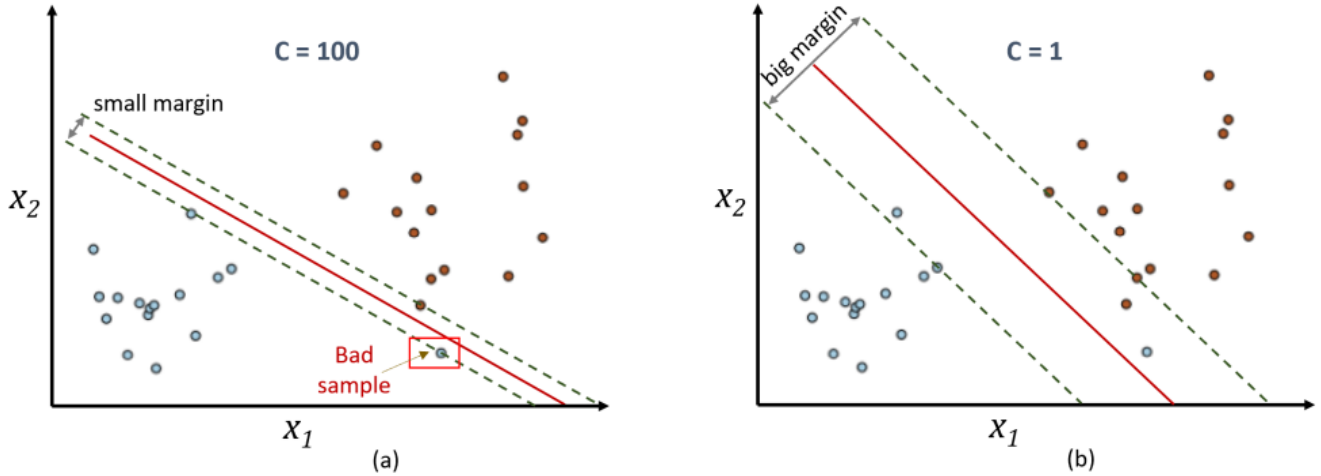


Figura 7.3: (a) Limite não regularizado superajustado (b) Ajuste regularizado

O Truque do Kernel

- Permite classificação não linear
- Mapeia dados para espaço de dimensão superior
- Separação linear no espaço transformado
- Sem computação explícita da transformação
- Usa funções kernel para mapeamento implícito

Embora a formulação de classificação de margem suave seja bastante flexível, ela não funcionará para problemas de classificação não linear onde limites curvos são garantidos.

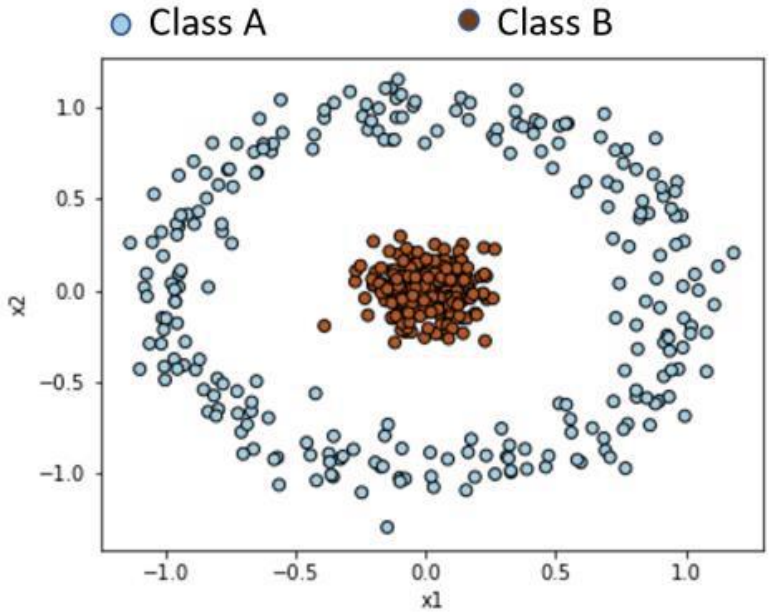


Figura 7.4: Amostras distribuídas não linearmente

Treinando SVM com Espaços de Características Complexos:

O SVM pode encontrar uma "linha de decisão" mais eficaz se os dados forem transformados para um **espaço de características mais complexo** (como um espaço 3D ou até com mais dimensões). Nesse novo espaço, dados que não eram separáveis por uma linha reta (em 2D) podem se tornar separáveis.

Exemplo Prático:

Imagine dados em formato de círculo (não separáveis por uma reta). Se transformarmos esses dados para um espaço 3D (por exemplo, usando uma função que cria uma "elevação" no centro do círculo), uma superfície plana (hiperplano) pode separá-los.

Para contornar esse problema é mapear as variáveis/características de entrada originais em um espaço dimensional mais alto

$$\varphi(\boldsymbol{x}) = \varphi(x_1, x_2) = [z_1, z_2, z_3] = [x_1, x_2, x_1^2 + x_2^2]$$

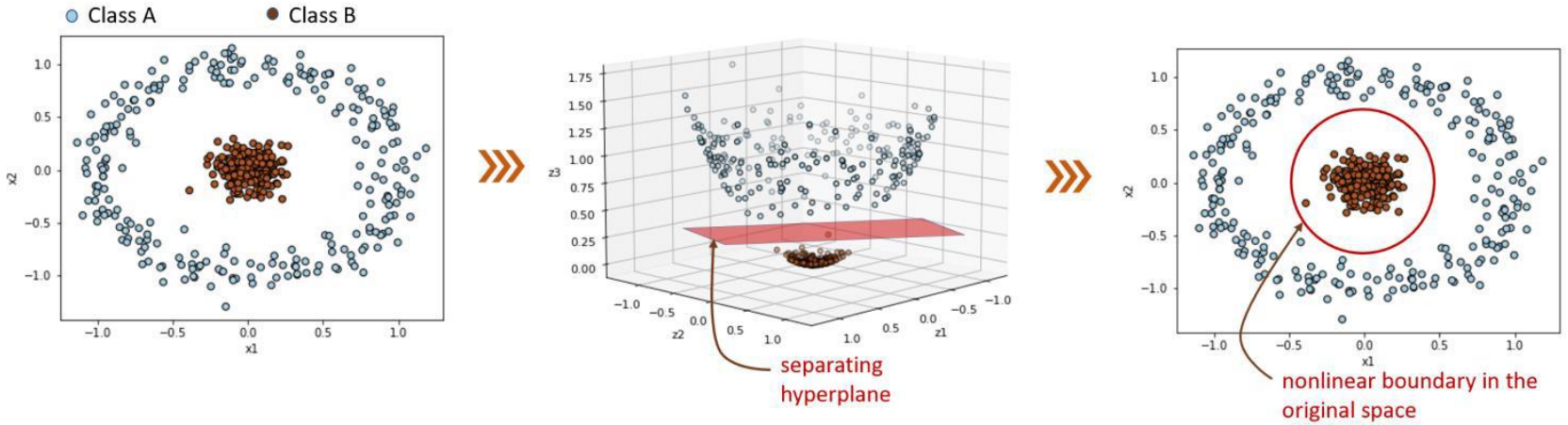


Figure 7.5: Nonlinear mapping to higher dimensional enabling linear segregation

A Solução Mágica: O "Kernel Trick"

O truque é: você não precisa calcular explicitamente a transformação φ! Em vez disso, usamos uma **função kernel** que calcula a "similaridade" entre pontos no espaço transformado, sem precisar passar por ele.

Como Funciona?

- **Kernel** é um atalho matemático que evita cálculos complexos.
- Exemplo: O kernel polinomial calcula interações entre features sem elevar manualmente os dados a uma potência.

Fundamentos matemáticos (revisitado)

$$\min_{\boldsymbol{w},b,\xi} \quad \frac{1}{2}\left|\left|\boldsymbol{w}\right|\right|^2 + C\sum_{i=1}^{N}\xi_i \qquad \text{eq. 2}$$

$$\text{s.t.} \quad y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1 - \xi_i, \quad i = 1, \cdots N$$

$$\xi_i \geq 0$$

O Segredo Matemático do SVM (Versão Simplificada):

O SVM resolve o problema de otimização de uma forma alternativa (chamada forma dual), que evita trabalhar diretamente com os parâmetros *w* e *b*. Em vez disso, usa α's (multiplicadores de Lagrange), que representam a "importância" de cada ponto no treinamento.

$$\min_{\boldsymbol{\alpha}} \quad \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_i \boldsymbol{x}_i^T\boldsymbol{x}_j - \sum_{i=1}^{N}\alpha_i \qquad \text{eq. 3}$$

$$\text{s.t.} \quad \sum_{i=1}^{N} y_i\alpha_i = 0$$

$$0 \leq \alpha_i \leq C \quad i = 1, \cdots N$$

$$\boldsymbol{w} = \sum_{i=1}^{N}\alpha_i y_i \boldsymbol{x}_i, \qquad \boldsymbol{b} = \frac{1}{N_s}\sum_{i\in\{SV\}} 1 - y_i\boldsymbol{w}^T\boldsymbol{x}_i$$

Fundamentos matemáticos (revisitado)

$$\min_{\boldsymbol{\alpha}} \quad \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_i \boldsymbol{x}_i^T \boldsymbol{x}_j \; - \; \sum_{i=1}^{N}\alpha_i \qquad \text{eq. 3}$$

$$\text{s.t.} \quad \sum_{i=1}^{N} y_i \alpha_i \; = \; 0$$

$$0 \; \leq \; \alpha_i \leq C \qquad i = 1, \cdots N$$

$$\boldsymbol{w} \; = \; \sum_{i=1}^{N} \alpha_i \, y_i \boldsymbol{x}_i, \qquad \boldsymbol{b} = \frac{1}{N_s} \sum_{i \in \{SV\}} 1 - y_i \boldsymbol{w}^T \boldsymbol{x}_i$$

Como funciona?

1. **Forma Dual (Equação 3):**
   - **O objetivo é encontrar os α's que:**
     - **Minimizem a combinação entre a "complexidade do modelo" (`ΣΣ αiαj...`) e a soma dos α's (`Σαi`).**
     - **Respeitem duas regras:**
       - **A soma dos α's multiplicados pelas suas classes (`Σ yiαi`) deve ser zero.**
       - **Cada α deve estar entre 0 e C (o hiperparâmetro de regularização).**

Fundamentos matemáticos (revisitado)

$$\min_{\boldsymbol{\alpha}} \quad \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_i \boldsymbol{x}_i^T \boldsymbol{x}_j \; - \; \sum_{i=1}^{N}\alpha_i \qquad \text{eq. 3}$$

$$\text{s.t.} \quad \sum_{i=1}^{N} y_i \alpha_i \; = \; 0$$

$$0 \; \leq \; \alpha_i \leq C \qquad i = 1, \cdots N$$

$$\boldsymbol{w} \; = \; \sum_{i=1}^{N} \alpha_i \, y_i \boldsymbol{x}_i, \qquad \boldsymbol{b} = \frac{1}{N_s} \sum_{i \in \{SV\}} 1 - y_i \boldsymbol{w}^T \boldsymbol{x}_i$$

2. **Resultado:**

- Os **α's não nulos** correspondem aos **vetores de suporte** (pontos críticos que definem a fronteira de decisão).
- **α = 0**: O ponto não influencia a fronteira.
- **0 < α < C**: O ponto está exatamente na margem.
- **α = C**: O ponto é um erro (está dentro da margem ou classificado errado).

Fundamentos matemáticos (revisitado)

$$\min_{\boldsymbol{\alpha}} \quad \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_i \boldsymbol{x}_i^T \boldsymbol{x}_j \; - \; \sum_{i=1}^{N}\alpha_i \qquad \text{eq. 3}$$

$$\text{s.t.} \quad \sum_{i=1}^{N} y_i \alpha_i \; = \; 0$$

$$0 \; \leq \; \alpha_i \leq C \qquad i = 1, \cdots N$$

$$\boldsymbol{w} \; = \; \sum_{i=1}^{N} \alpha_i \, y_i \boldsymbol{x}_i, \qquad \boldsymbol{b} = \frac{1}{N_s} \sum_{i \in \{SV\}} 1 - y_i \boldsymbol{w}^T \boldsymbol{x}_i$$

3. **Classificação de novos dados:**

   A previsão é feita com uma fórmula que depende dos α's, das classes (`yi`), e da similaridade entre o ponto de teste (`xt`) e os vetores de suporte (`xi`):

$$\widehat{y_t} \; = \; sign(\sum_{i=1}^{N} \alpha_i \, y_i \boldsymbol{x}_i^T \boldsymbol{x}_t + b)$$

Por que complicar a matemática do SVM?

Parece mais difícil usar a **forma dual** (com α's) em vez da forma original (com *w* e *b*), mas há uma razão poderosa: **os kernels**.
Isso permite que o SVM resolva problemas **não lineares** (como dados em formato de espiral) sem precisar calcular transformações complexas manualmente!

O Truque do Kernel Explicado:

1. **Problema Não Linear:**
   - Dados não separáveis por uma reta (em 2D) podem ser separados em um **espaço de alta dimensão** (3D, 4D, etc.) usando uma função $\varphi(x)$.
   - Exemplo: Transformar dados 2D em 3D para criar um "plano separador".
2. **Desafio:**
   - Calcular $\varphi(x)$ explicitamente é **lento e inviável** para dados grandes ou muitas dimensões.
3. **Solução Genial (Kernel Trick):**
   - **Kernel** é uma função que calcula a **similaridade entre dois pontos no espaço transformado** sem precisar conhecer $\varphi(x)$!
   - Exemplo de kernel:
     - $K(x_1, x_2) = (x_1 \cdot x_2 + 1)^2$ (kernel polinomial).
     - Equivale a calcular o produto escalar em um espaço 6D, mas sem fazer as contas manualmente!

Como Funciona na Prática?

1. **Treinamento (Equação 8):**
   - **O SVM otimiza os α's usando apenas $K(x_i, x_j)$ (não precisa de $\varphi(x)$).**
   - **C controla a tolerância a erros (como antes).**

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_i \varphi(\boldsymbol{x_i})^{\boldsymbol{T}}\varphi(\boldsymbol{x_j}) \; - \; \sum_{i=1}^{N}\alpha_i \\ \text{s.t.} \quad & \sum_{i=1}^{N} y_i\alpha_i \; = \; 0 \\ & 0 \; \leq \; \alpha_i \leq C \qquad i = 1, \cdots N \end{aligned}$$

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_i K\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) \; - \; \sum_{i=1}^{N}\alpha_i \\ \text{s.t.} \quad & \sum_{i=1}^{N} y_i\alpha_i \; = \; 0 \\ & 0 \; \leq \; \alpha_i \leq C \qquad i = 1, \cdots N \end{aligned}$$

2. **Previsão (Equação 9):**
   - **A classe de um novo ponto x_t depende da similaridade com os vetores de suporte:**

$$\hat{y}_t = sign(\sum_{i \in \{SV\}} \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{x}_t) + b)$$

   - **$K(x_i, x_t)$ substitui o produto escalar no espaço transformado.**

Por Que Isso é Revolucionário?

- **Eficiência:** Evita cálculos caros em alta dimensão.
- **Flexibilidade:** Escolha o kernel que melhor se adapta aos dados
  - **Linear:** Para dados separáveis por reta.
  - **RBF (Gaussiano):** Para padrões complexos (ex: círculos).
  - **Polinomial:** Para relações não lineares com grau definido.

Funções Kernel Comuns

- A tabela abaixo lista algumas funções de kernel comumente usadas. A primeira é simplesmente o produto escalar familiar de dois vetores, enquanto a segunda, RBF (função de base radial) ou kernel Gaussiano é o kernel mais popular (e geralmente a escolha padrão) para problemas não lineares.

Function	Equation	Hyperparameter
Linear	$K(\boldsymbol{x}, \boldsymbol{z}) = \boldsymbol{x}^T \boldsymbol{z}$	
Gaussian	$K(\boldsymbol{x}, \boldsymbol{z}) = exp(-\frolac{\|\boldsymbol{x} - \boldsymbol{z}\|^2}{2\sigma^2})$	σ
Polynomial	$K(\boldsymbol{x}, \boldsymbol{z}) = (\gamma \boldsymbol{x}^T \boldsymbol{z} + r)^d$	γ, r, d
Sigmoid	$K(\boldsymbol{x}, \boldsymbol{z}) = tanh(\gamma \boldsymbol{x}^T \boldsymbol{z} + r)$	γ, r

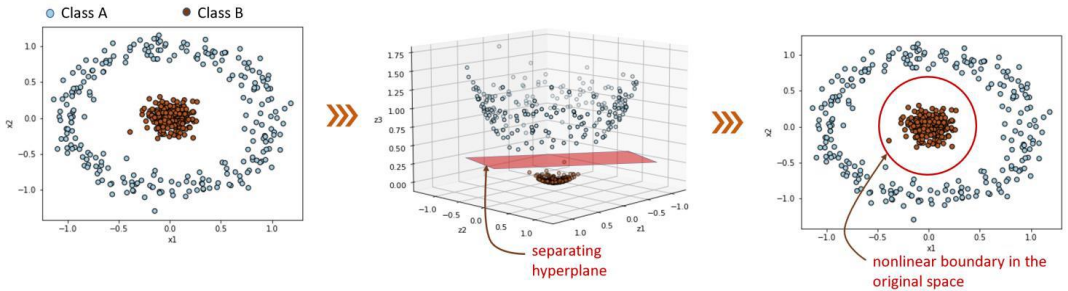
Resumo Visual:

- **Passo 1:** Dados não lineares em 2D → **Kernel** mapeia para espaço N-D "invisível".
- **Passo 2:** SVM encontra hiperplano nesse espaço.
- **Passo 3:** Classificação usa apenas **similaridades** (kernel), sem cálculos explícitos.

Exemplo Prático:

Imagine separar círculos concêntricos (dados 2D):

1. **Kernel RBF** transforma os dados em um espaço 3D onde um plano os separa.
2. O SVM usa $K(x_i, x_j)$ para calcular distâncias "ajustadas" no espaço original.



Implementação Sklearn: Kernel gaussiano

```
# generate data
from sklearn.datasets import make_circles
```

✓ 1.9s

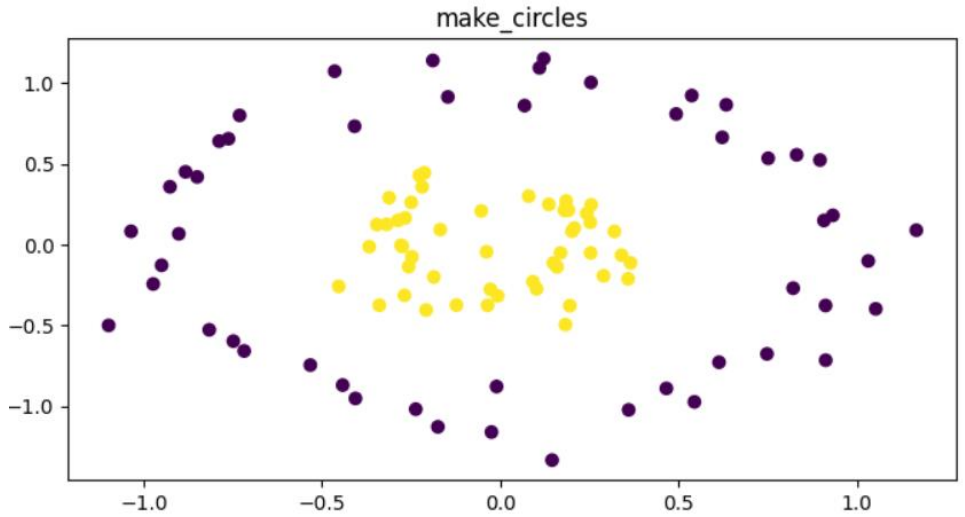
```
X, y = make_circles(500, factor=.08, noise=.1, random_state=1)
# note that y = 0,1 here and need not be ±1; SVM does internal transformation accordingly
```

✓ 0.0s

Implementação Sklearn: Kernel gaussiano

```
import matplotlib.pyplot as plt
fig, (ax1) = plt.subplots(nrows=1, ncols=1, figsize=(8, 4))

X, y = make_circles(noise=0.1, factor=0.3, random_state=0)
ax1.scatter(X[:, 0], X[:, 1], c=Y)
ax1.set_title("make_circles")
```

✓ 0.0s



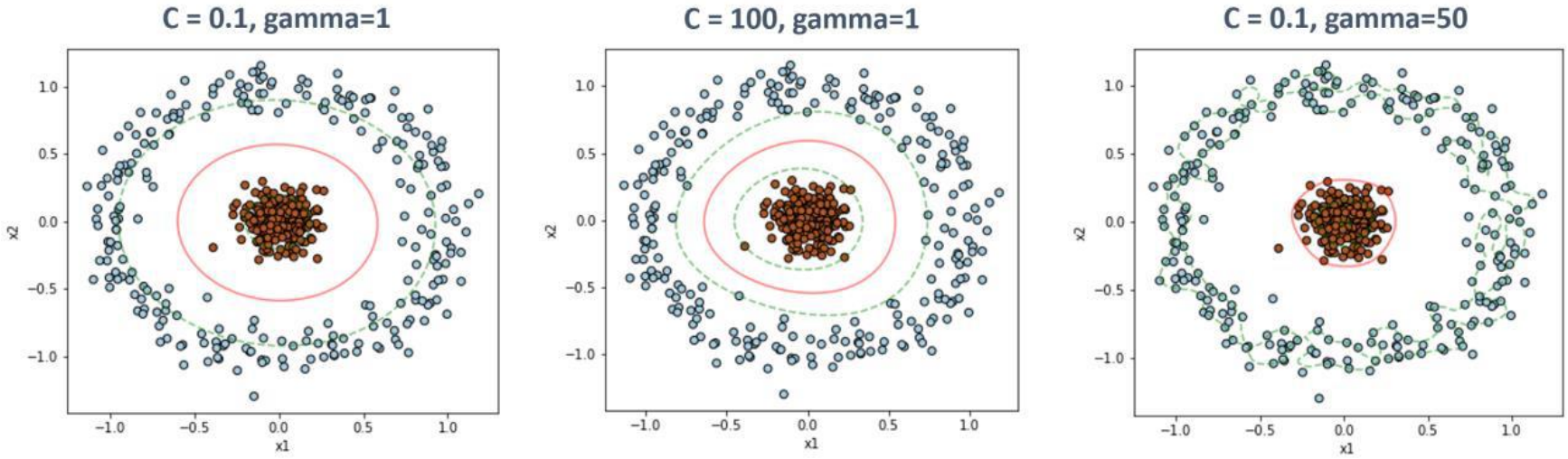
```
# find optimal hyperparameter via GridSearchCV
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
param_grid = {'C':[0.1, 1, 10, 100, 1000], 'gamma':[0.01, 0.1, 1, 10, 100]}
gs = GridSearchCV(SVC(), param_grid, cv=5).fit(X, y) # no scaling as inputs are already scaled
```

✓ 0.3s

```
gs.best_params_ # optimal hyperparameter
# find optimal hyperparameter via GridSearchCV
```

✓ 0.0s

```
{'C': 0.1, 'gamma': 1}
```



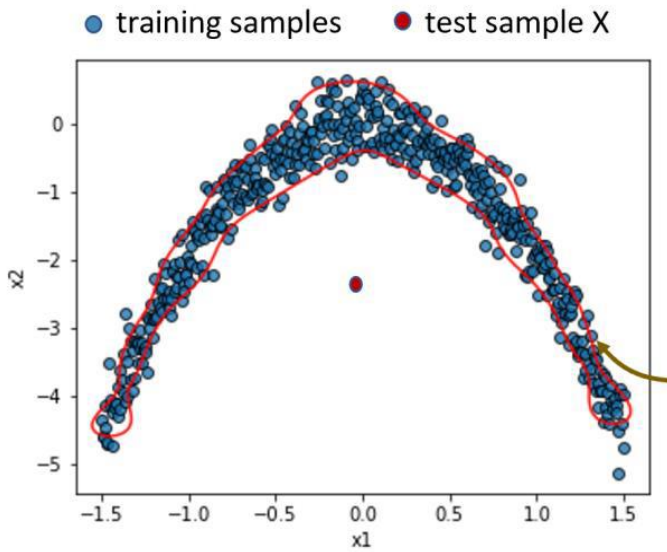
Descrição de Dados por Vetores de Suporte (SVDD)

- Variante não supervisionada do SVM
- Classificação de uma classe
- Envolve dados normais dentro de hiperesfera
- Identifica outliers/amostras anormais
- Usa funções kernel para limites não lineares

SVDD: Detectando Anomalias (Quando Só Temos Dados "Normais")

Problema: Imagine que você tem um conjunto de dados **apenas de operações normais** de uma fábrica (ex: temperatura, pressão). Como identificar se um novo dado é **anormal** (defeito, falha)?

Solução: O **SVDD** (Support Vector Data Description) é um algoritmo que "aprende" o padrão dos dados normais e cria uma **fronteira flexível** ao redor deles. Qualquer ponto fora dessa fronteira é considerado anormal.



- **Dados de Treino:** Pontos azuis agrupados (operações normais).
- **Ponto X:** Fora da bolha → classificado como **anomalia**.

Bounding boundary as
model of the training data?

Descrição de Dados por Vetores de Suporte (SVDD)

Aplicações Comuns:

- **Monitoramento Industrial:** Detectar falhas em equipamentos usando dados só de operação normal.
- **Controle de Qualidade:** Identificar produtos defeituosos em uma linha de produção.
- **Segurança Cibernética:** Encontrar atividades suspeitas em redes.

Descrição de Dados por Vetores de Suporte (SVDD)

Por Que Usar SVDD?

- **Não Supervisionado:** Não precisa de dados rotulados como "anormais" (raros ou inexistentes).
- **Adaptável:** A bolha se ajusta à forma complexa dos dados (usando **kernels**, como no SVM).

Comparação com SVM:

- **SVM:** Separa **duas classes** (ex: spam vs. não spam).
- **SVDD:** Define **uma única classe** (normal) e detecta outliers.

Descrição de Dados por Vetores de Suporte (SVDD)

Analogia:

Pense no SVDD como um "cão de guarda" que aprende o cheiro da casa (dados normais) e late sempre que sente um odor estranho (anomalia). 🐕🔍

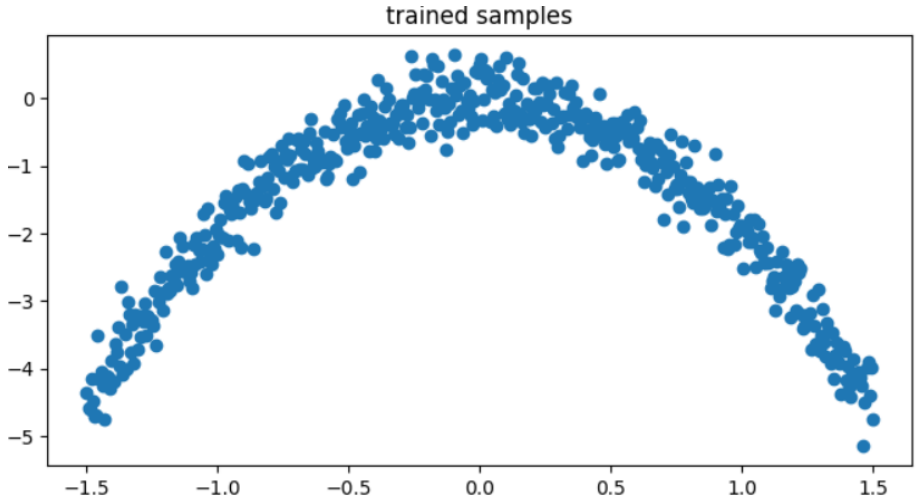
Resumo:

- **SVDD** é ideal para quando você só tem dados de **um grupo** e quer detectar desvios.
- Usado em indústrias, saúde, segurança e qualquer área onde "anormalidades" são críticas.

Descrição de Dados por Vetores de Suporte (SVDD)

A Matemática por Trás do SVDD (Detecção de Anomalias):

O objetivo do SVDD é criar uma "bolha" (esfera) em um espaço de alta dimensão que envolva a maioria dos dados normais. Pontos fora dessa bolha são considerados anomalias.



Descrição de Dados por Vetores de Suporte (SVDD)

$$\min_{R,a,\xi} \quad R^2 + C\sum_{i=1}^{N}\xi_i$$

$$\text{s.t.} \quad \left||\varphi(\boldsymbol{x_i}) - a\right||^2 \leq R^2 + \xi_i, \quad i = 1, \cdots N$$

$$\xi_i \geq 0$$

Elementos-chave da fórmula:

1. R^2: **Raio da bolha (queremos a menor bolha possível).**
2. a: **Centro da bolha.**
3. ξi **(variáveis de folga): Medem "quanto" cada ponto invade ou fica fora da bolha.**
4. C: **Hiperparâmetro que controla a tolerância a pontos fora da bolha.**

Descrição de Dados por Vetores de Suporte (SVDD)

$$\min_{R,a,\xi} \quad R^2 + C\sum_{i=1}^{N}\xi_i$$

$$\text{s.t.} \quad \left||\varphi(\boldsymbol{x_i}) - a\right||^2 \leq R^2 + \xi_i, \quad i = 1, \cdots N$$

$$\xi_i \geq 0$$

Elementos-chave da fórmula:

1. R^2: **Raio da bolha (queremos a menor bolha possível).**
2. a: **Centro da bolha.**
3. ξi **(variáveis de folga): Medem "quanto" cada ponto invade ou fica fora da bolha.**
4. C: **Hiperparâmetro que controla a tolerância a pontos fora da bolha.**

Descrição de Dados por Vetores de Suporte (SVDD)

$$\min_{R,a,\xi} \quad R^2 + C\sum_{i=1}^{N}\xi_i$$

$$\text{s.t.} \quad \|\varphi(\boldsymbol{x_i}) - a\|^2 \leq R^2 + \xi_i, \quad i = 1, \cdots N$$

$$\xi_i \geq 0$$

Como Funciona a Otimização?

- **Objetivo:** Minimizar duas coisas ao mesmo tempo:
  - O tamanho da bolha (R^2).
  - A soma das "folgas" (ξi), multiplicada por **C**.
- **Restrições:**
  - Cada ponto de dado $\varphi(xi)$ (transformado para alta dimensão) deve estar **dentro ou perto da bolha**:
    - Se $\xi i = 0$ → O ponto está dentro da bolha.
    - Se $\xi i > 0$ → O ponto está fora da bolha (contribui para o erro).

Descrição de Dados por Vetores de Suporte (SVDD)

$$\min_{R,a,\xi} \quad R^2 + C\sum_{i=1}^{N}\xi_i$$

$$\text{s.t.} \quad \left||\varphi(\boldsymbol{x_i}) - a\right||^2 \leq R^2 + \xi_i, \quad i = 1, \cdots N$$

$$\xi_i \geq 0$$

$$\min_{\boldsymbol{\alpha}} \quad \sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j K\left(\boldsymbol{x_i}, \boldsymbol{x_j}\right) - \sum_{i=1}^{N}\alpha_i K(\boldsymbol{x_i}, \boldsymbol{x_i})$$

$$\text{s.t.} \quad \sum_{i=1}^{N}\alpha_i = 1$$

$$0 \leq \alpha_i \leq C \quad i = 1, \cdots N$$

Descrição de Dados por Vetores de Suporte (SVDD)

$$\min_{\boldsymbol{\alpha}} \quad \sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j K\left(\boldsymbol{x_i}, \boldsymbol{x_j}\right) - \sum_{i=1}^{N}\alpha_i K(\boldsymbol{x_i}, \boldsymbol{x_i})$$

$$\text{s.t.} \quad \sum_{i=1}^{N}\alpha_i = 1$$

$$0 \leq \alpha_i \leq C \qquad i = 1, \cdots N$$

Partes da Fórmula:

1. **Termo de Minimização:**
   - **$\Sigma\Sigma\ \alpha_i\alpha_j\ K(x_i, x_j)$:**
     Mede a "similaridade combinada" entre todos os pares de pontos, ponderada pelos α's.
     - *Exemplo:* Se dois pontos são muito similares (K alto) e ambos têm α alto, contribuem muito para o custo.
   - **$-\ \Sigma\ \alpha_i\ K(x_i, x_i)$:**
     Subtrai a similaridade de cada ponto consigo mesmo. Isso ajuda a ajustar o tamanho da "bolha" que envolve os dados.

Descrição de Dados por Vetores de Suporte (SVDD)

$$\min_{\boldsymbol{\alpha}} \quad \sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j K(\boldsymbol{x_i}, \boldsymbol{x_j}) \; - \; \sum_{i=1}^{N}\alpha_i \, K(\boldsymbol{x_i}, \boldsymbol{x_i})$$

$$\text{s.t.} \quad \sum_{i=1}^{N}\alpha_i \; = \; 1$$

$$0 \; \leq \; \alpha_i \leq C \qquad i = 1, \cdots N$$

2. **Restrições:**
   - **$\Sigma \alpha_i = 1$: A soma de todos os α's deve ser 1 (como uma receita onde os ingredientes somam 100%).**
   - **$0 \leq \alpha_i \leq C$: Cada α deve estar entre 0 e C (um hiperparâmetro que controla a flexibilidade).**

Descrição de Dados por Vetores de Suporte (SVDD)

$$\min_{\boldsymbol{\alpha}} \quad \sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j K(\boldsymbol{x_i}, \boldsymbol{x_j}) \; - \; \sum_{i=1}^{N}\alpha_i \, K(\boldsymbol{x_i}, \boldsymbol{x_i})$$

$$\text{s.t.} \quad \sum_{i=1}^{N}\alpha_i \; = \; 1$$

$$0 \; \leq \; \alpha_i \leq C \qquad i = 1, \cdots N$$

Papel do Kernel (K):

- **$K(x_i, x_j)$ é uma função que mede a similaridade entre os pontos x_i e x_j em um espaço transformado (sem precisar calcular a transformação!).**
- ***Exemplo de Kernel:***
  - **$K(x, y) = (x \cdot y + 1)^2$ → Captura relações não lineares como círculos ou elipses.**

Hiperparâmetro C:

- **C alto** (ex: 100): Prioriza incluir mais pontos na bolha (menos outliers, bolha menor).
- **C baixo** (ex: 0.1): Permite mais pontos fora (bolha maior, mais flexível).

Descrição de Dados por Vetores de Suporte (SVDD)

$$\min_{\boldsymbol{\alpha}} \quad \sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j K(\boldsymbol{x_i}, \boldsymbol{x_j}) \; - \; \sum_{i=1}^{N}\alpha_i \, K(\boldsymbol{x_i}, \boldsymbol{x_i})$$

$$\text{s.t.} \quad \sum_{i=1}^{N}\alpha_i \; = \; 1$$

$$0 \; \leq \; \alpha_i \leq C \qquad i = 1, \cdots N$$

Como Funciona na Prática?

1. **Treinamento:**
   - **O modelo ajusta os α's para minimizar o custo, usando a similaridade entre os dados (K).**
   - **Pontos com α > 0 são os vetores de suporte que definem a fronteira.**
2. **Detecção de Anomalias:**
   - **Um novo ponto x_t é classificado como normal se estiver "perto o suficiente" dos vetores de suporte (calculado via K).**

Descrição de Dados por Vetores de Suporte (SVDD)

$$\min_{\boldsymbol{\alpha}} \quad \sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j K(\boldsymbol{x_i}, \boldsymbol{x_j}) \; - \; \sum_{i=1}^{N}\alpha_i \, K(\boldsymbol{x_i}, \boldsymbol{x_i})$$

$$\text{s.t.} \quad \sum_{i=1}^{N}\alpha_i \; = \; 1$$

$$0 \; \leq \; \alpha_i \leq C \qquad i = 1, \cdots N$$

Analogia:

Imagine que os α's são "pesos" que você atribui a diferentes amigos em uma foto de grupo:

- **Amigos com α alto são os que definem os limites da foto (vetores de suporte).**
- **Amigos com α = 0 estão no meio da foto e não afetam os limites.**
- **C é como o tamanho do quadro: um quadro pequeno (C alto) exclui alguns amigos; um quadro grande (C baixo) inclui mais.**

Descrição de Dados por Vetores de Suporte (SVDD)

$$\min_{\boldsymbol{\alpha}} \quad \sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j K(\boldsymbol{x_i}, \boldsymbol{x_j}) \; - \; \sum_{i=1}^{N}\alpha_i \, K(\boldsymbol{x_i}, \boldsymbol{x_i})$$

$$\text{s.t.} \quad \sum_{i=1}^{N}\alpha_i \; = \; 1$$

$$0 \; \leq \; \alpha_i \leq C \qquad i = 1, \cdots N$$

Resumo:

- **A fórmula ajusta uma "bolha" flexível ao redor dos dados normais usando α's e kernels.**
- **C controla o equilíbrio entre precisão e flexibilidade.**
- **Kernels permitem detectar padrões complexos sem cálculos complicados.**

Exemplo de Aplicação:

Em uma fábrica, o modelo usa dados normais (ex: vibração de máquinas) para criar uma "bolha".

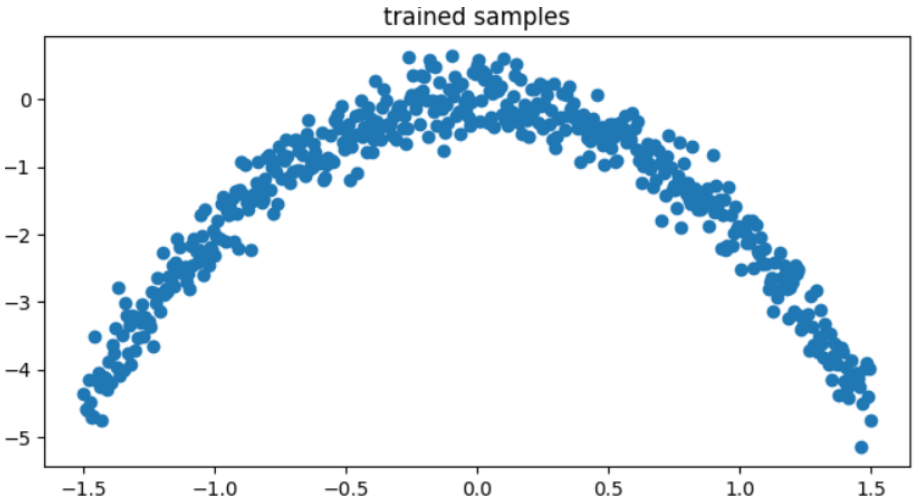
Se um novo dado (vibração anormal) ficar fora, um alerta é acionado! 🚨

```
# read data
import numpy as np
X = np.loadtxt('SVDD_toyDataset.csv', delimiter=',')
```

✓ 0.2s

```
import matplotlib.pyplot as plt
fig, (ax1) = plt.subplots(nrows=1, ncols=1, figsize=(8, 4))

ax1.scatter(X[:, 0], X[:, 1])
ax1.set_title("trained samples")
```

✓ 0.1s



```
# read data
import numpy as np
X = np.loadtxt('SVDD_toyDataset.csv', delimiter=',')
# compute bandwidth via modified mean criteria
import scipy

N = X.shape[0]
phi = 1/np.log(N-1)
delta = -0.14818008*np.power(phi,4) + 0.2846623624*np.power(phi,3) - 0.252853808*np.power(phi,2)
        + 0.159059498*phi - 0.001381145
D2 = np.sum(scipy.spatial.distance.pdist(X, 'sqeuclidean'))/(N*(N-1)/2)
sigma = np.sqrt(D2/np.log((N-1)/delta*delta))
gamma = 1/(2*sigma*sigma)

# SVM fit
from sklearn.svm import OneClassSVM
model = OneClassSVM(nu=0.01, gamma=gamma).fit(X) # nu corresponds to f
```

```
# plot SVM boundaries
import matplotlib.pyplot as plt

plt.figure()
plt.scatter(X[:, 0], X[:, 1], edgecolors='k', alpha=0.8)
plt.xlabel('x1')
plt.ylabel('x2')

# get axis limits
ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()

# create grid to evaluate model
xx = np.linspace(xlim[0], xlim[1], 100)
yy = np.linspace(ylim[0], ylim[1], 100)
YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T
Z = model.decision_function(xy).reshape(XX.shape)

# plot decision boundary and supporting planes
ax.contour(XX, YY, Z, levels=[0], alpha=0.9, linestyles=['-'], colors=['red'])
plt.show()
```

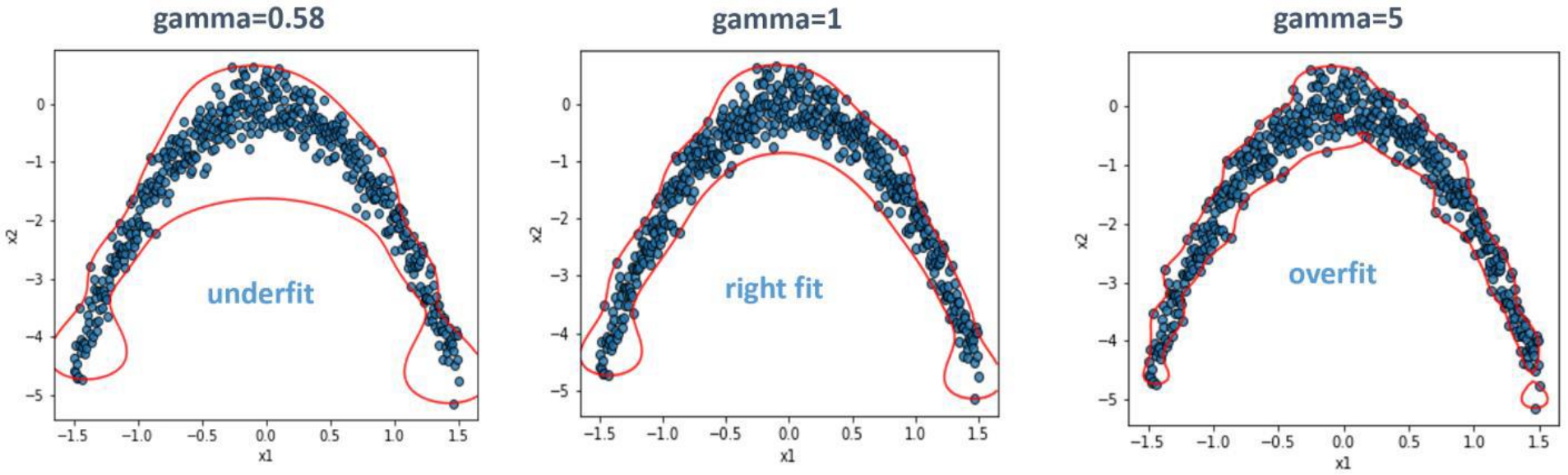
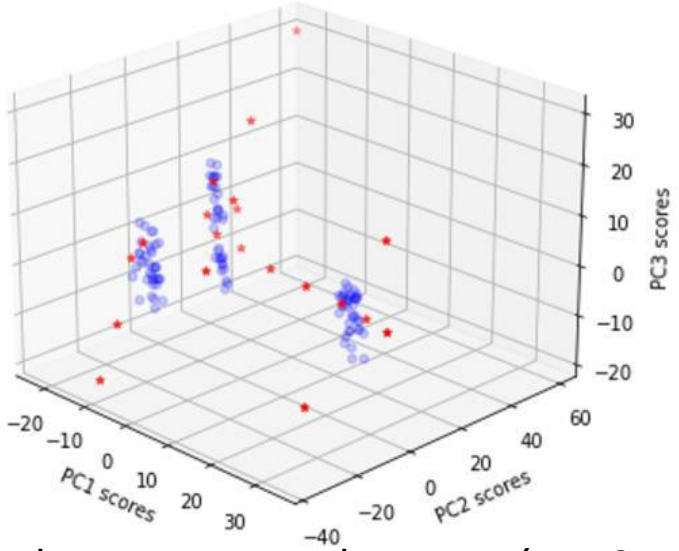
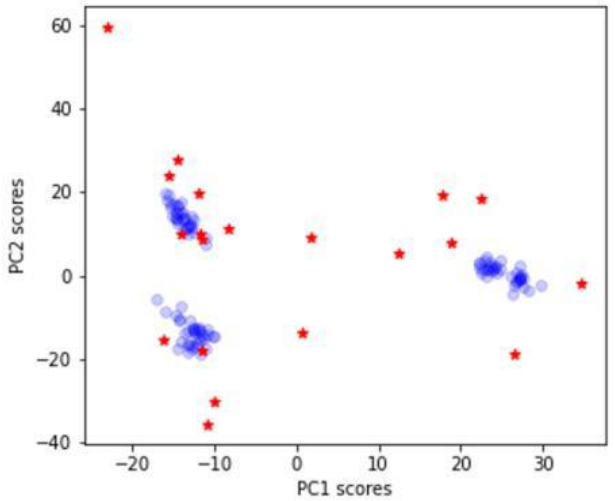


Figura 7.8: Aplicação SVDD para descrição de dados e impacto do hiperparâmetro do modelo.

Detecção de falhas de processo via SVDD - Processo de fabricação de semicondutores

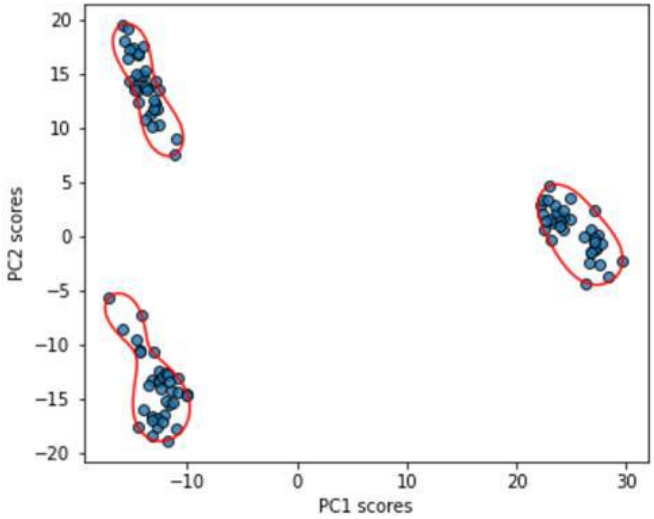




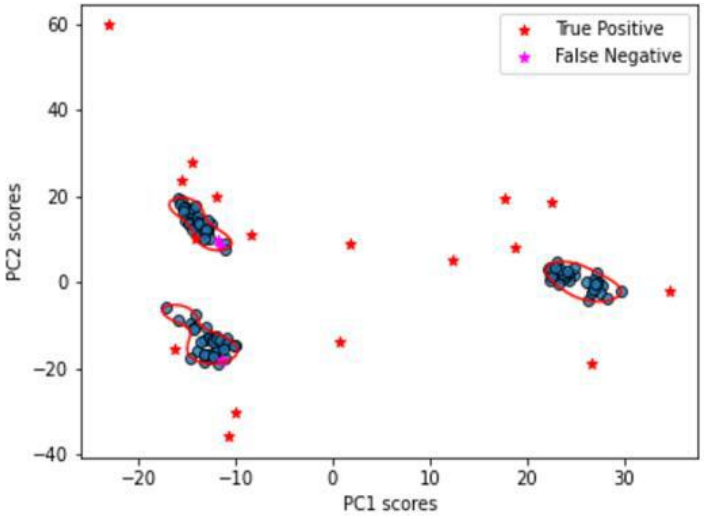
Dados de processo em lote contém 19 variáveis de processo medidas ao longo de 108 lotes normais e 21 lotes com defeito.

A duração dos lotes varia de 95 a 112 segundos (consulte o apêndice para mais detalhes).

```
# read data
import numpy as np
X_train = np.loadtxt('Metal_etch_2DPCA_trainingData.csv', delimiter=',')

# fit SVM
from sklearn.svm import OneClassSVM
model = OneClassSVM(nu=0.01, gamma=0.025).fit(X_train) # gamma from modified mean
criterion = 0.0025

# predict for test data
X_test = np.loadtxt('Metal_etch_2DPCA_testData.csv', delimiter=',')
y_test = model.predict(X_test) # y=-1 for outliers

print('Number of faults identified: ', np.sum(y_test == -1), ' out of ', len(y_test))
```


PC2 scores
PC1 scores
20
15
10
5
0
−5
−10
−15
−20
−10
0
10
20
30

(a)


True Positive
False Negative
PC2 scores
PC1 scores
60
40
20
0
−20
−40
−20
−10
0
10
20
30

(b)

Regressão por Vetores de Suporte (SVR)

- Variante de regressão do SVM
- Cria tubo-ε ao redor da curva de previsão
- Penaliza pontos fora do tubo
- Equilibra complexidade e precisão
- Usa truque do kernel para regressão não linear

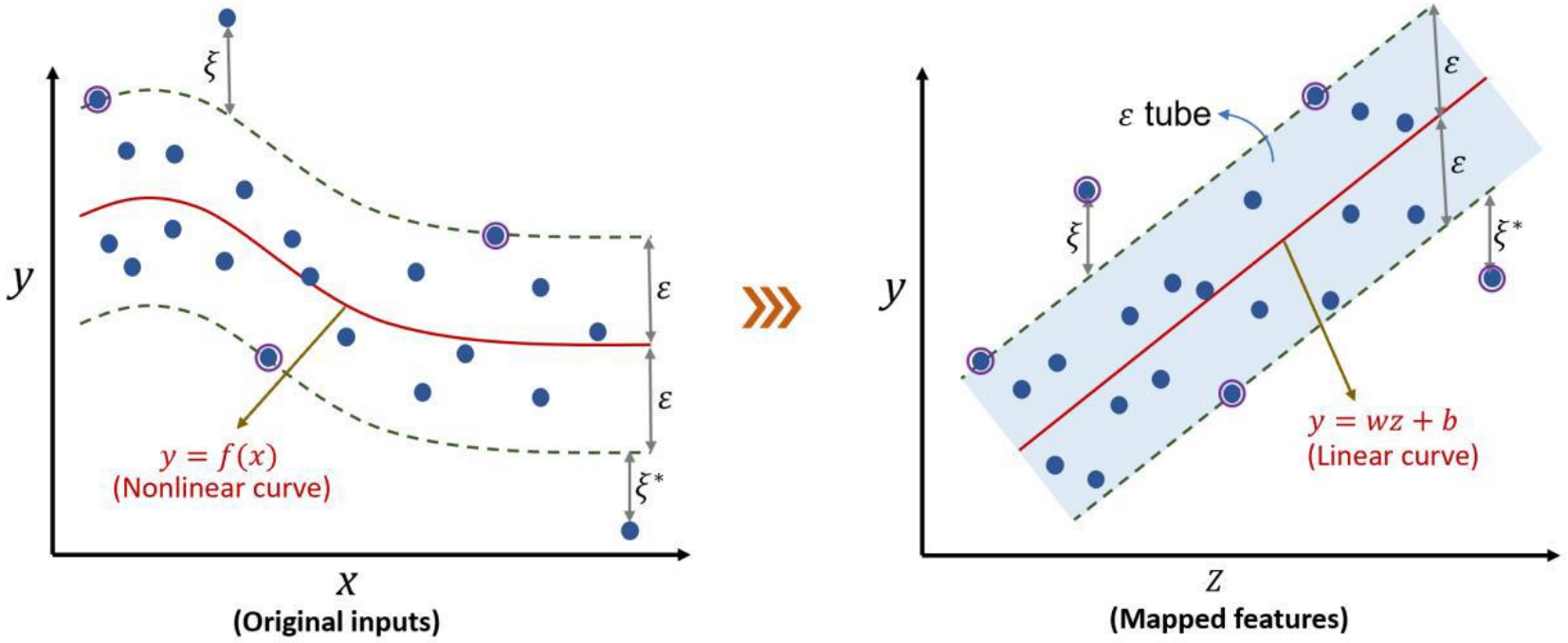


Figura 7.11: Ilustração SVR Kernelizada para ajuste de curva não linear

Estrutura Matemática do SVR

- Minimiza complexidade do modelo
- Usa função de perda ε-insensível
- Formulação dual com funções kernel
- Vetores de suporte determinam curva de regressão
- Previsões baseadas em avaliações kernel

Objetivo do Problema:

Encontrar os parâmetros w (vetor de pesos), b (viés) e as variáveis de folga ξ_i, ξ_i^* que minimizem a seguinte função:

$$\text{Minimizar: } \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N}(\xi_i + \xi_i^*)$$

Sujeito às restrições:

1. $y_i - \boldsymbol{w}^T\phi(\boldsymbol{x_i}) - b \leq \varepsilon + \xi_i$
2. $\boldsymbol{w}^T\phi(\boldsymbol{x_i}) + b - y_i \leq \varepsilon + \xi_i^*$
3. $\xi_i \geq 0, \xi_i^* \geq 0$, **para** $i = 1, \ldots, N$

1. **O que está sendo otimizado?**
   - **Primeiro termo ($\frac{1}{2}\|w\|^2$):** Controla a "complexidade" do modelo. Quanto menor $\|w\|$, mais "suave" é a função de predição (evita overfitting).
   - **Segundo termo ($C\sum(\xi_i + \xi_i^*)$):** Penaliza erros fora de uma margem ε. C define o quanto esses erros são tolerados (quanto maior C, menos erros são permitidos).

Objetivo do Problema:

Encontrar os parâmetros w (vetor de pesos), b (viés) e as variáveis de folga ξ_i, ξ_i^* que minimizem a seguinte função:

$$\text{Minimizar: } \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N}(\xi_i + \xi_i^*)$$

Sujeito às restrições:

1. $y_i - \boldsymbol{w}^T\phi(\boldsymbol{x_i}) - b \leq \varepsilon + \xi_i$
2. $\boldsymbol{w}^T\phi(\boldsymbol{x_i}) + b - y_i \leq \varepsilon + \xi_i^*$
3. $\xi_i \geq 0, \xi_i^* \geq 0$, **para** $i = 1, \ldots, N$

2. **Restrições:**
   - **As duas primeiras restrições definem um "tubo de tolerância" de largura 2ε em torno da predição. Se o valor real y_i estiver dentro desse tubo ($\pm\varepsilon$), não há penalização ($\xi_i = \xi_i^* = 0$).**
   - **Slack variables (ξ_i, ξ_i^*): Medem quanto um ponto ultrapassa a margem ε. São não-negativas e penalizadas na função objetivo.**

Objetivo do Problema:

Encontrar os parâmetros w (vetor de pesos), b (viés) e as variáveis de folga ξ_i, ξ_i^* que minimizem a seguinte função:

$$\text{Minimizar: } \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N}(\xi_i + \xi_i^*)$$

Sujeito às restrições:

1. $y_i - \boldsymbol{w}^T\phi(\boldsymbol{x_i}) - b \leq \varepsilon + \xi_i$
2. $\boldsymbol{w}^T\phi(\boldsymbol{x_i}) + b - y_i \leq \varepsilon + \xi_i^*$
3. $\xi_i \geq 0, \xi_i^* \geq 0$, **para** $i = 1, \ldots, N$

3. **Interpretação geométrica:**
   - **O modelo busca um hiperplano ($w^T\phi(x) + b$) que ajuste os dados com uma margem ε. Pontos dentro do tubo são ignorados; pontos fora geram "custos" proporcionais a ξ_i e ξ_i^*.**

Objetivo do Problema:

Encontrar os parâmetros w (vetor de pesos), b (viés) e as variáveis de folga ξ_i, ξ_i^* que minimizem a seguinte função:

$$\text{Minimizar: } \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N}(\xi_i + \xi_i^*)$$

Sujeito às restrições:

1. $y_i - \boldsymbol{w}^T\phi(\boldsymbol{x_i}) - b \leq \varepsilon + \xi_i$
2. $\boldsymbol{w}^T\phi(\boldsymbol{x_i}) + b - y_i \leq \varepsilon + \xi_i^*$
3. $\xi_i \geq 0, \xi_i^* \geq 0$, **para** $i = 1, \ldots, N$

4. **Parâmetros-chave:**
   - ε: **Largura do tubo de tolerância (erros dentro são aceitos).**
   - C: **Controla o equilíbrio entre a complexidade do modelo ($\|w\|$) e a permissividade com erros.**

Objetivo do Problema:

Encontrar os parâmetros w (vetor de pesos), b (viés) e as variáveis de folga ξ_i, ξ_i^* que minimizem a seguinte função:

$$\text{Minimizar: } \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N}(\xi_i + \xi_i^*)$$

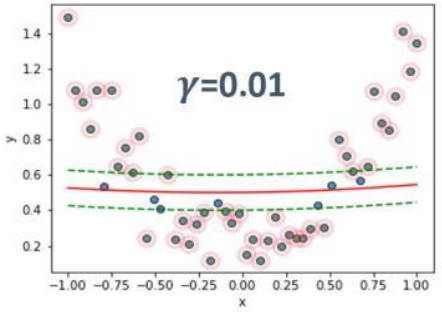
Sujeito às restrições:

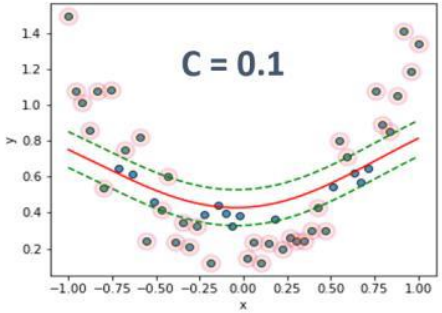
1. $y_i - \boldsymbol{w}^T\phi(\boldsymbol{x_i}) - b \leq \varepsilon + \xi_i$
2. $\boldsymbol{w}^T\phi(\boldsymbol{x_i}) + b - y_i \leq \varepsilon + \xi_i^*$
3. $\xi_i \geq 0, \xi_i^* \geq 0$, **para** $i = 1, \ldots, N$

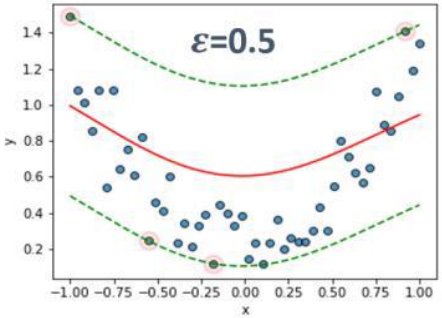
A função ϕ (x) permite modelar relações não lineares ao mapear os dados para um espaço de dimensão superior

Hiperparâmetros do SVR

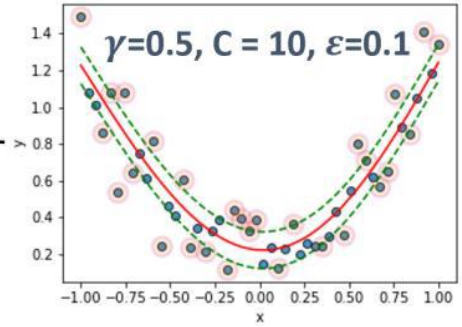
- C: Controla equilíbrio entre complexidade e ajuste
- ε: Define largura do tubo
- γ: Parâmetro do kernel (para RBF)
- C grande: Maior precisão de ajuste
- ε pequeno: Mais vetores de suporte



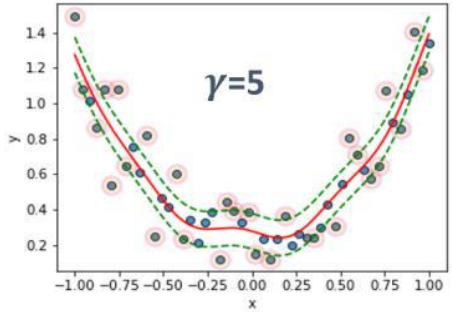


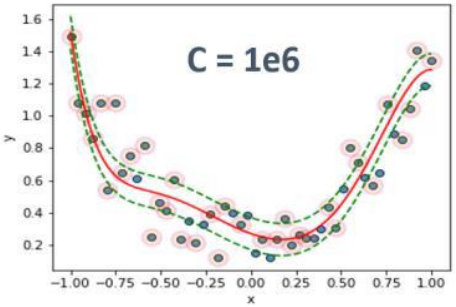


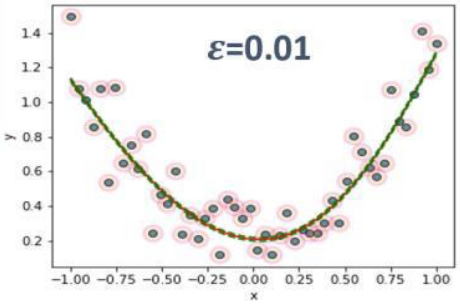
Underfitting



Overfitting



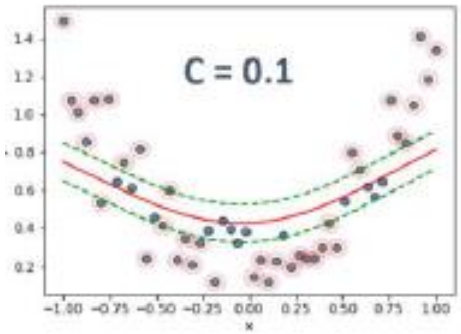


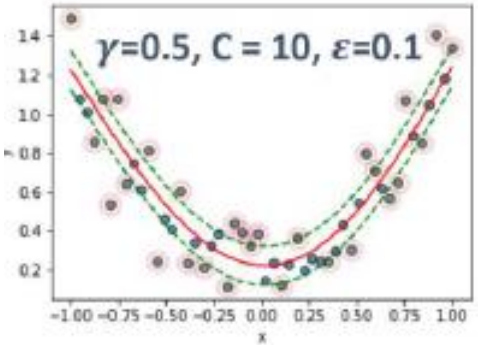


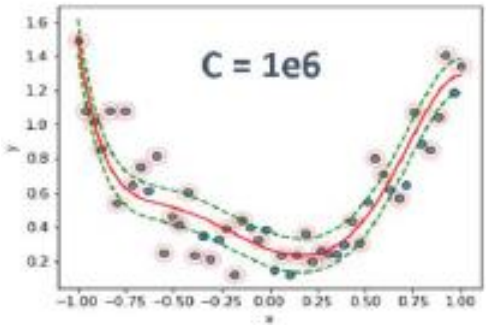
Impacto dos Hiperparâmetros no SVR (Kernel RBF):

1. C (Penalidade de Erro):

- **C alto:** Prioriza a redução de erros (pontos fora do tubo ε são fortemente penalizados).
  - **Resultado:** Modelo mais complexo, risco de *overfitting*.
- **C baixo:** Tolerância a erros maiores.
  - **Resultado:** Modelo mais simples, risco de *underfitting*.

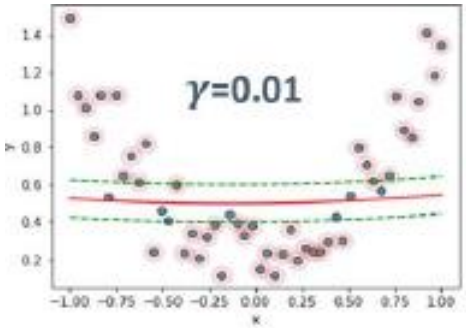


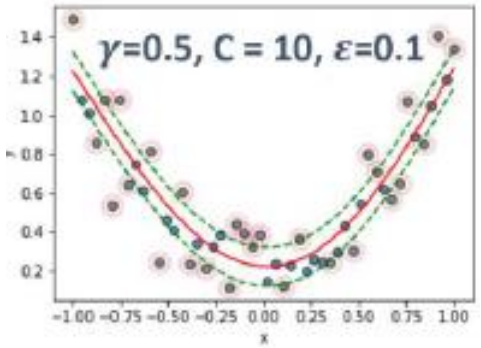


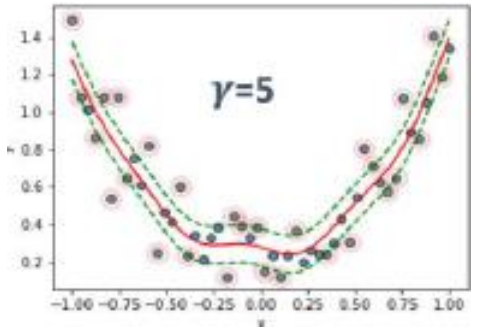


2. **γ (Gamma - Kernel RBF):**

- **γ alto:** O kernel RBF tem alcance curto (ex.: cada ponto influencia apenas regiões próximas).
  - **Resultado:** Modelo "pontiagudo", ajusta-se a detalhes (*overfitting*).
- **γ baixo:** O kernel RBF tem alcance amplo (ex.: pontos influenciam regiões maiores).
  - **Resultado:** Modelo mais suave, generaliza melhor (*underfitting*).

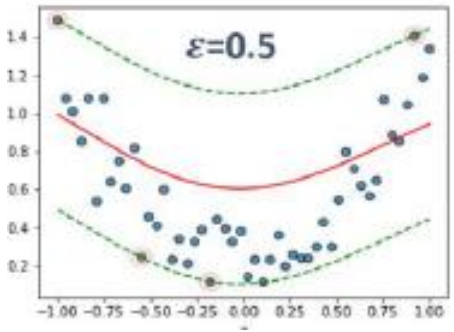


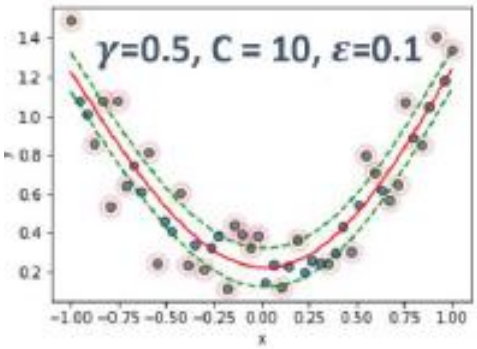


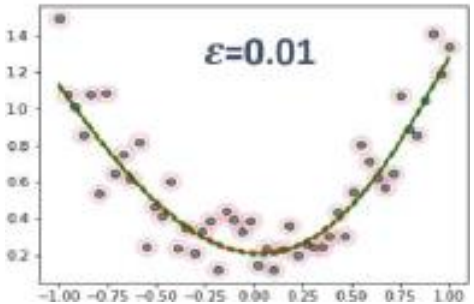


3. ε (Epsilon - Largura do Tubo):

- ε **grande:** Tubo de tolerância amplo.
  - **Resultado:** Modelo plano, ignora variações pequenas (*underfitting*).
- ε **pequeno:** Tubo estreito.
  - **Resultado:** Modelo sensível a detalhes, mais vetores de suporte (*overfitting* se não controlado).







SVR vs Redes Neurais

- Ambos lidam com relações não lineares
- SVR: Determinação automática da estrutura
- SVR: Menos hiperparâmetros
- SVR: Melhor para conjuntos pequenos/médios
- SVR: Identificação explícita de pontos importantes

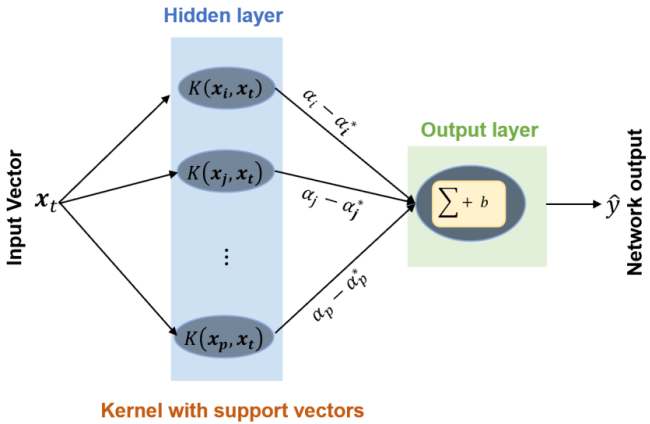


Fig: Network representation of SVR prediction

Detecção suave via SVR em uma planta de processamento de polímeros

- O conjunto de dados consiste em 61 amostras de 14 variáveis (10 preditores e 4 saídas da planta).
- Os preditores correspondem a temperaturas, taxas de alimentação
- Os dados é particularmente adequado para testar a robustez de métodos de modelagem não linear para dados irregularmente esparsos.

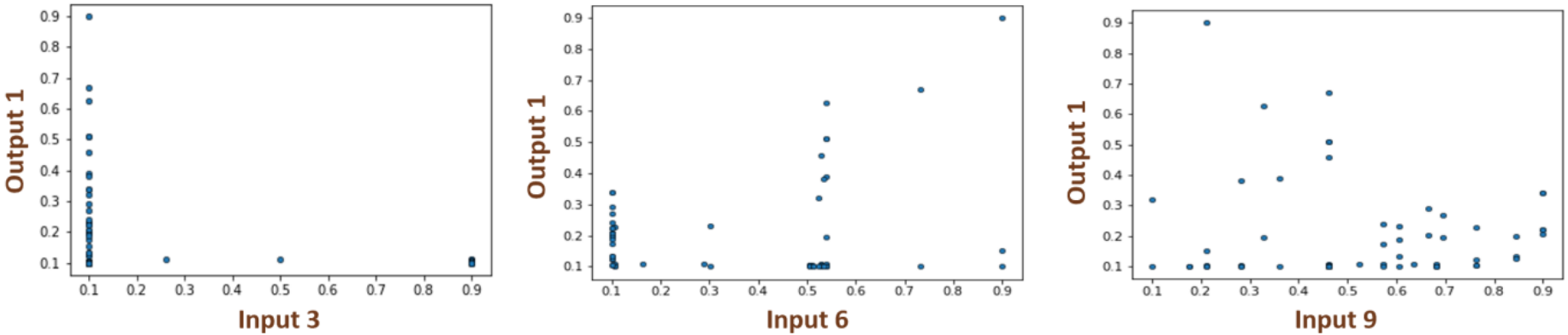


Figura 7.13: Distribuição das entradas e saídas do processo de polímero 1

```
# read data
import numpy as np
data = np.loadtxt('polymer.dat')

X, Y = data[:, 0:10], data[:, 10:]
y = Y[:, 0] # modeling the 1st output

# fit SVR model using grid-search
from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV

model = SVR(epsilon=0.01) # default epsilon = 0.1
param_grid = [{'gamma': np.linspace(0.1e-05,5,100), 'C': np.linspace(0.01,5000,100)}]
gs = GridSearchCV(model, param_grid, scoring='neg_mean_squared_error', cv=10)
```

```
gs.fit(X, y)
print('Optimal hyperparameter:', gs.best_params_)

>>> Optimal hyperparameter: {'C': 50.51, 'gamma': 0.1}

# predict using the best model
y_predicted_SVR = gs.predict(X)

# plots of raw and predicted data
import matplotlib.pyplot as plt

plt.figure()
plt.plot(y, y_predicted_SVR, '.', markeredgecolor='k', markeredgewidth=0.5, ms=9)
plt.plot(y, y, '-r', linewidth=0.5)
plt.xlabel('measured data'), plt.ylabel('predicted data ')
```

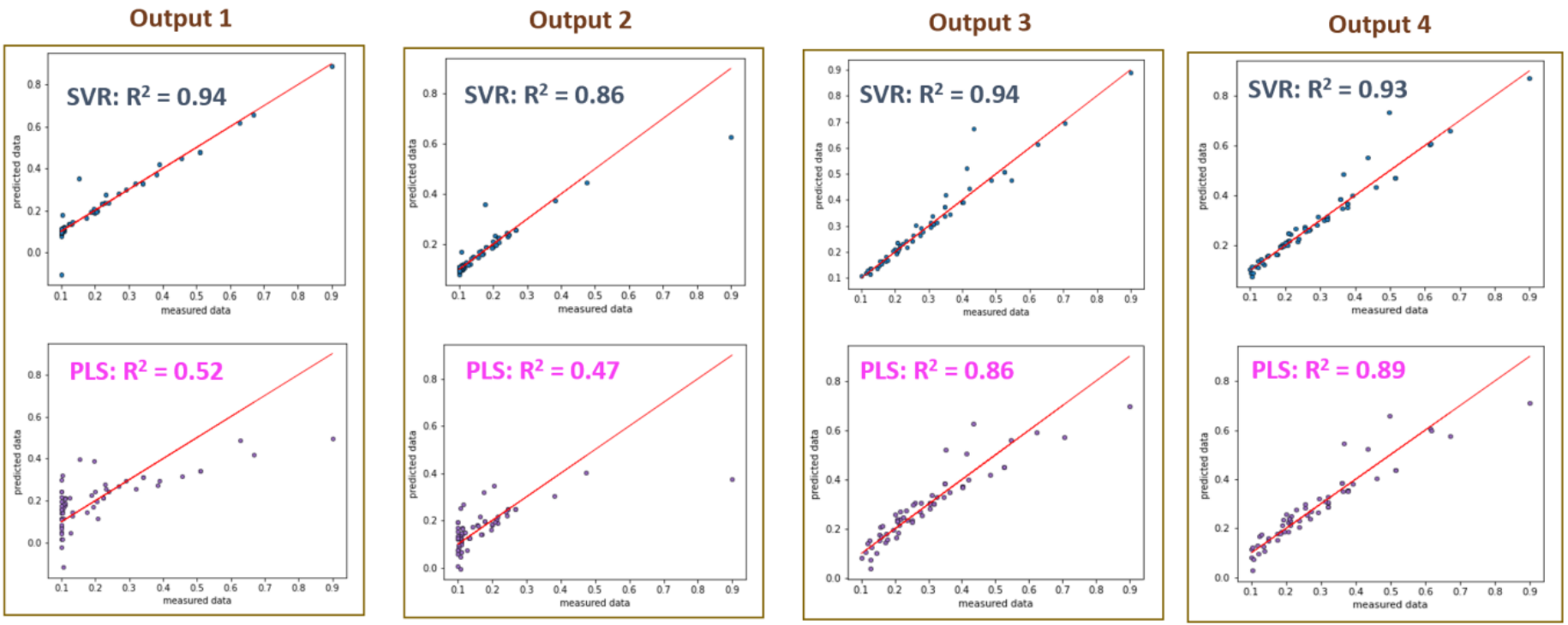
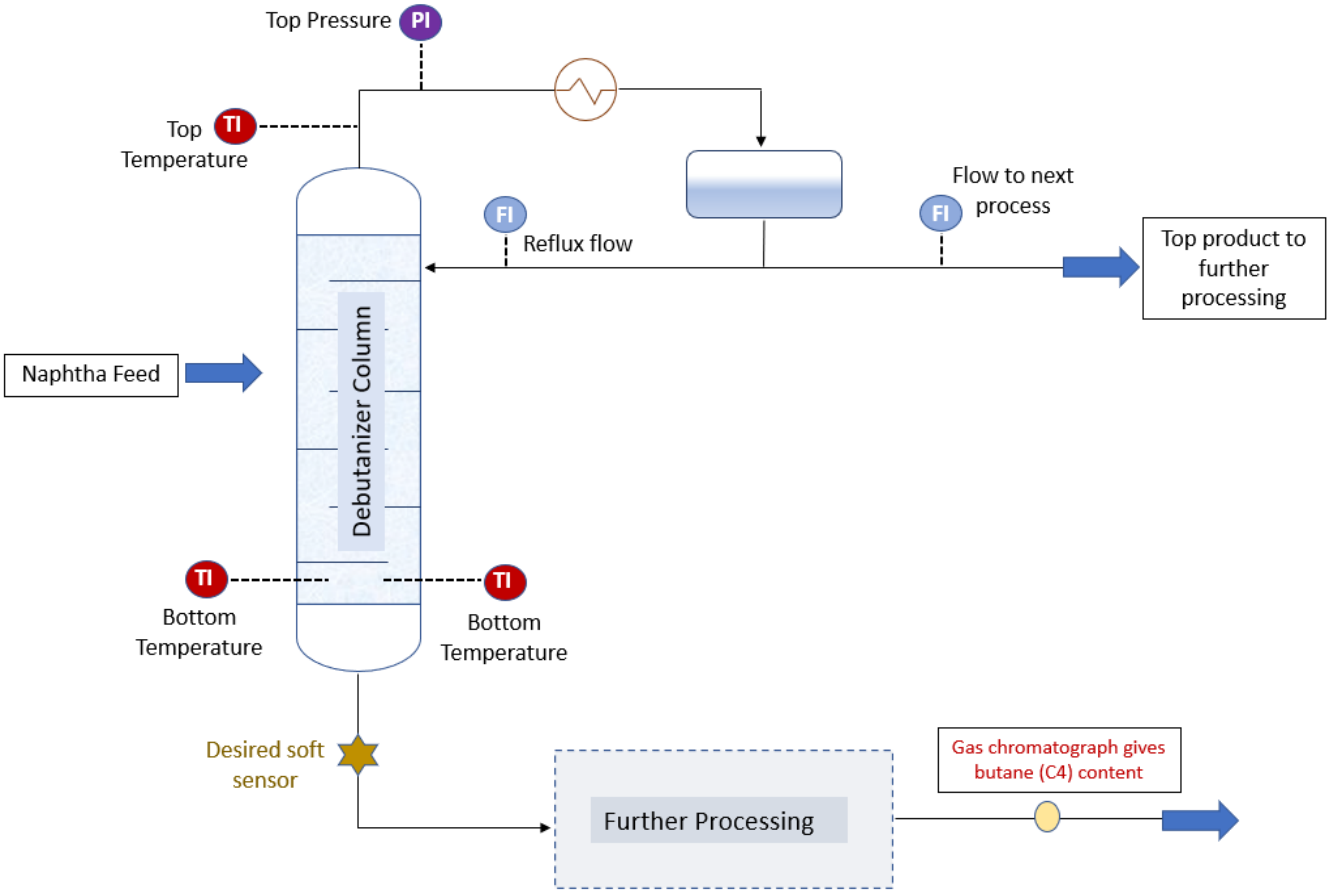


Figure 7.14: SVR and PLS predictions for polymer plant dataset. The red line denotes the ideal $y_{prediction} = y_{measured}$ reference.

Soft Sensing via SVR para coluna de debutanizadora em uma refinaria de petróleo



Soft Sensing via SVR para coluna de debutanizadora em uma refinaria de petróleo

- Dados de tamanho médio que vem de uma operação de coluna de debutanizador em uma refinaria de petróleo.
- O conteúdo de butano (C4) no produto de fundo de gasolina da coluna de debutanizador não está disponível em tempo real.
- Precisa ser previsto usando outros dados de processo ao redor da coluna.
- O conjunto de dados contém 2394 amostras de valores de processo de entrada-saída.
- Sete variáveis de processo (pressões, temperaturas, fluxos ao redor da coluna) são usadas como preditores.

Soft Sensing via SVR para coluna de debutanizadora em uma refinaria de petróleo

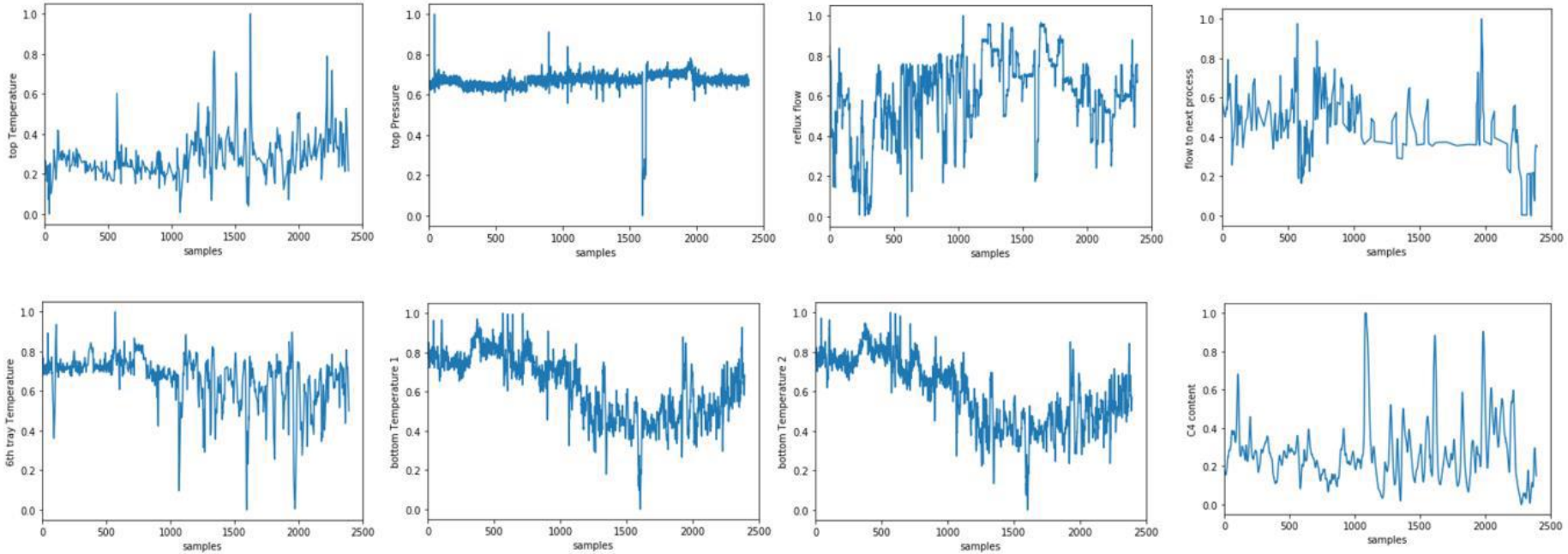
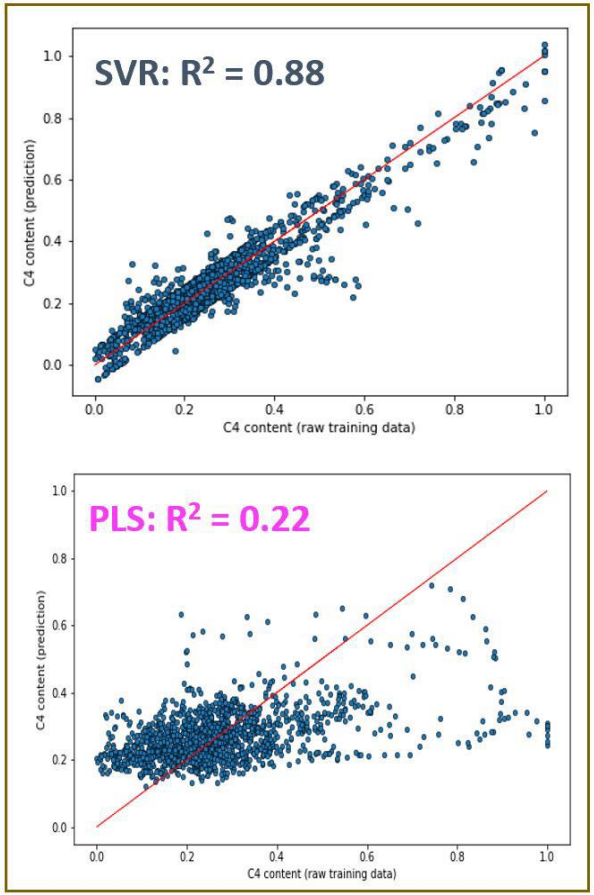


Figura 7.15: Gráficos de variáveis de entrada e saída (último gráfico) para o conjunto de dados da coluna debutanizer

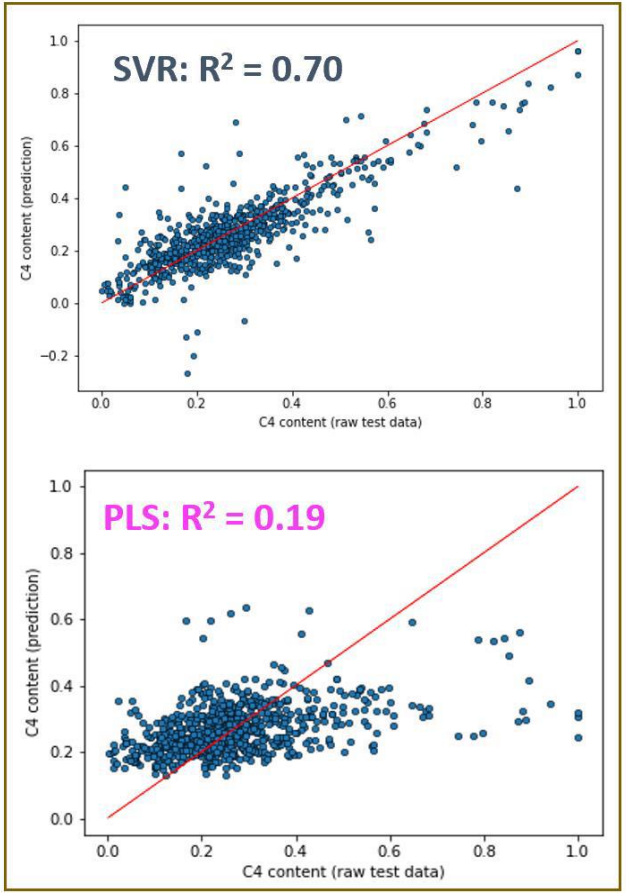
```
# read data
import numpy as np
data = np.loadtxt('debutanizer_data.txt', skiprows=5)

# separate train and test data
from sklearn.model_selection import train_test_split
X, y = data[:,0:-1], data[:,-1]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33, random_state = 100)

# fit SVR model via grid-search
from sklearn.svm import SVR
```

```
from sklearn.model_selection import GridSearchCV

model = SVR(epsilon=0.05)
param_grid = [{'gamma': np.linspace(1,10,10), 'C': np.linspace(0.01,500,10)}]
gs = GridSearchCV(model, param_grid, scoring='neg_mean_squared_error', cv=10)

gs.fit(X_train, y_train)
```

Training data



Test data



Aplicações Industriais: Classificação

- Monitoramento de processos
- Detecção de falhas
- Controle de qualidade
- Reconhecimento de padrões
- Detecção de anomalias

Aplicações Industriais: Regressão

- Sensoriamento virtual
- Previsão de parâmetros de processo
- Estimação de variáveis de qualidade
- Monitoramento em tempo real
- Otimização de processos

Vantagens do SVM

- Ótimo global garantido
- Efetivo em altas dimensões
- Eficiente em memória
- Funções kernel versáteis
- Generalização robusta

Limitações e Considerações

- Seleção do kernel pode ser desafiadora
- Computacionalmente intensivo para grandes conjuntos de dados
- Sensível à seleção de hiperparâmetros
- Natureza caixa-preta do mapeamento kernel
- Requer dados de treinamento de qualidade