Máquina de Vetores de Suporte (SVM)

Classificação Linear e Não Linear

Slides extraídos do livro "*Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*" de Aurélien Géron

Sumário

1. Classificação SVM Linear
2. Classificação SVM Não-Linear

Sumário

1 Classificação SVM Linear

2 Classificação SVM Não-Linear

Classificação SVM Linear

- Uma **Máquina de Vetores de Suporte (SVM)** é um modelo de Aprendizado de Máquina poderoso e versátil.
- É capaz de realizar classificação linear ou não linear, regressão e até mesmo detecção de novidades.

Ideia Fundamental: Classificação de Margem Larga

O limite de decisão de um classificador SVM não apenas separa as duas classes, mas também se mantém o mais distante possível dos exemplos de treinamento mais próximos. Isso pode ser visualizado como o ajuste da "rua" mais larga possível entre as classes.

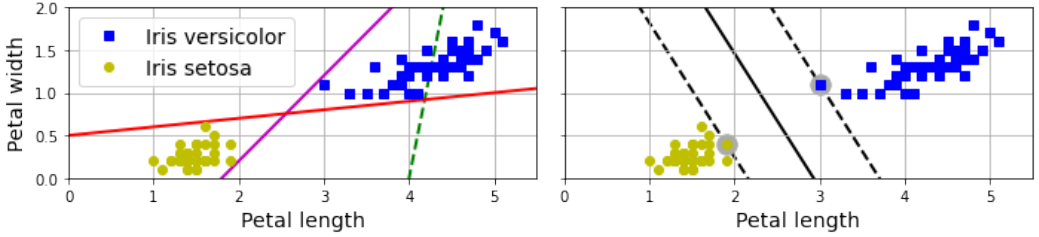


Figure: Classificação de margem larga.

As instâncias localizadas na borda da rua são chamadas de **vetores de suporte** (circuladas na figura).

Aviso

As SVMs são sensíveis às escalas das features.

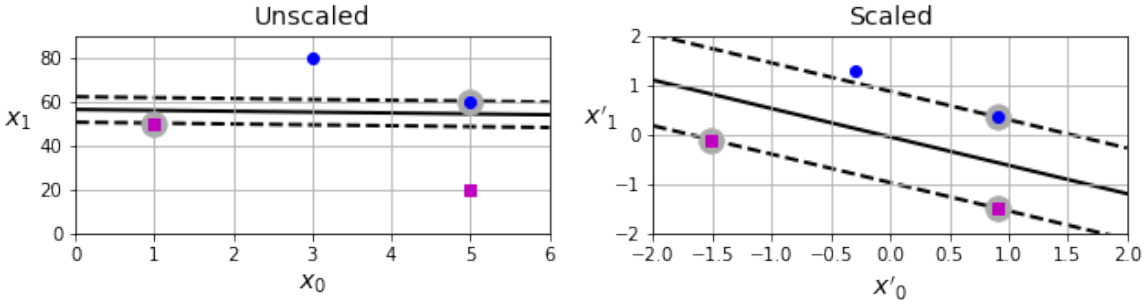


Figure: Sensibilidade às escalas das features.

- Esquerda: escala vertical é muito maior que a horizontal, então a rua mais larga possível é quase horizontal.
- Direita: após o `StandardScaler`, o limite de decisão parece muito melhor.

Classificação de Margem Rígida (Hard Margin)

Se impormos estritamente que todas as instâncias devem estar fora da rua e do lado correto, isso é chamado de **classificação de margem rígida**.
Existem duas questões principais com a classificação de margem rígida:

1. Só funciona se os dados forem linearmente separáveis.
2. É sensível a outliers.

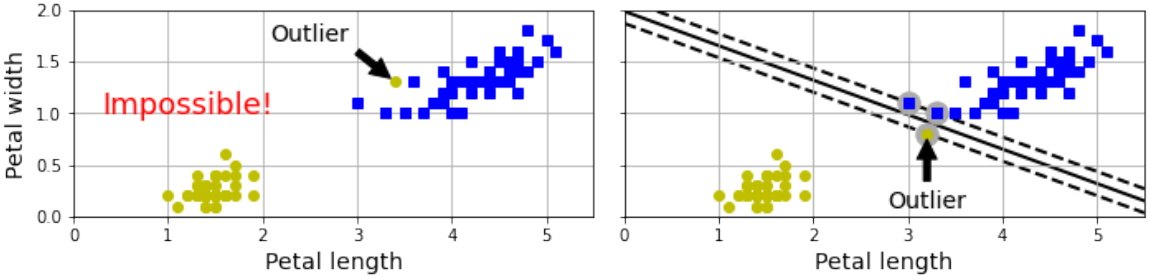


Figure: Sensibilidade da margem rígida a outliers.

No gráfico da esquerda, é impossível encontrar uma margem rígida.
No da direita, o limite de decisão é muito diferente e provavelmente não generalizará tão bem.

Classificação de Margem Suave (Soft Margin)

Para evitar os problemas da margem rígida, usamos um modelo mais flexível.

Objetivo

Encontrar um bom equilíbrio entre manter a rua o mais larga possível e limitar as **violações de margem** (instâncias que acabam no meio da rua ou mesmo do lado errado). Chamado de **classificação de margem suave**.

- Ao criar um modelo SVM, você pode especificar o hiperparâmetro de regularização, chamado `C`.
- Um valor baixo de `C` torna a rua mais larga, mas leva a mais violações de margem.
- Um valor alto de `C` torna a rua mais estreita, com menos violações.

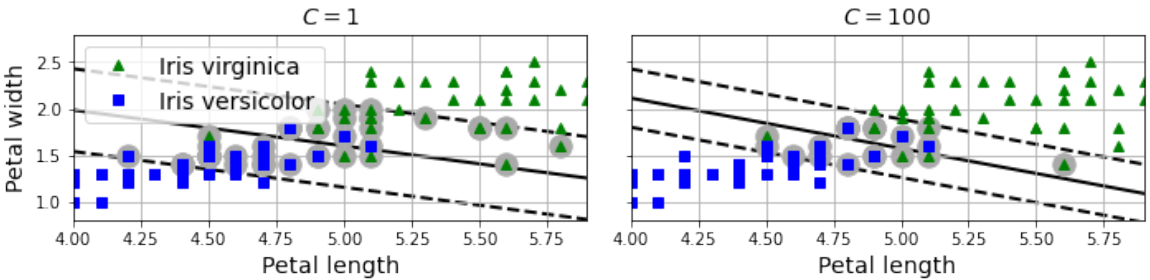


Figure: Margem larga (`C=1`, esquerda) vs. menos violações de margem (`C=100`, direita).

Classificação de Margem Suave (Soft Margin)

Dica

Se o seu modelo SVM está com overfitting, você pode tentar regularizá-lo reduzindo o valor de `C`.

Implementação de SVM Linear com Scikit-Learn

O código a seguir carrega o dataset iris, escala as features e treina um classificador SVM linear para detectar flores *Iris virginica*.

```
from sklearn.datasets import load_iris
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC

iris = load_iris(as_frame=True)
X = iris.data[["petal length (cm)", "petal width (cm)"]].values
y = (iris.target == 2)  # Iris virginica

svm_clf = make_pipeline(StandardScaler(),
                        LinearSVC(C=1, random_state=42))
svm_clf.fit(X, y)
```

O modelo resultante corresponde ao gráfico da esquerda da figura anterior (`C=1`).

Fazendo Previsões com SVM Linear

Como de costume, você pode usar o modelo para fazer previsões:

```
>>> X_new = [[5.5, 1.7], [5.0, 1.5]]
>>> svm_clf.predict(X_new)
array([ True, False])
```

A classe `LinearSVC` não possui um método `predict_proba()`. No entanto, você pode chamar o método `decision_function()`, que retorna uma pontuação para cada instância. Essa pontuação mede a distância sinalizada entre cada instância e o limite de decisão.

```
>>> svm_clf.decision_function(X_new)
array([ 0.66163411, -0.22036063])
```

Sumário

1 Classificação SVM Linear

2 Classificação SVM Não-Linear

Classificação SVM Não-Linear

- Embora SVM lineares sejam eficientes, muitos datasets não são nem de longe linearmente separáveis.
- Uma abordagem para lidar com datasets não lineares é adicionar mais features, como features polinomiais.
- Em alguns casos, isso pode resultar em um dataset (aproximadamente) linearmente separável.

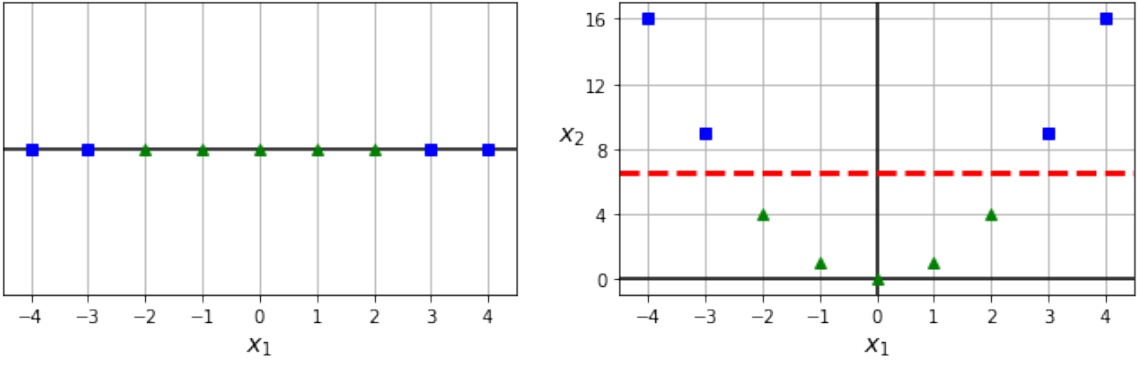


Figure: Adicionando a feature $x_2 = x_1^2$ para tornar um dataset linearmente separável.

Implementando Features Polinomiais com Scikit-Learn

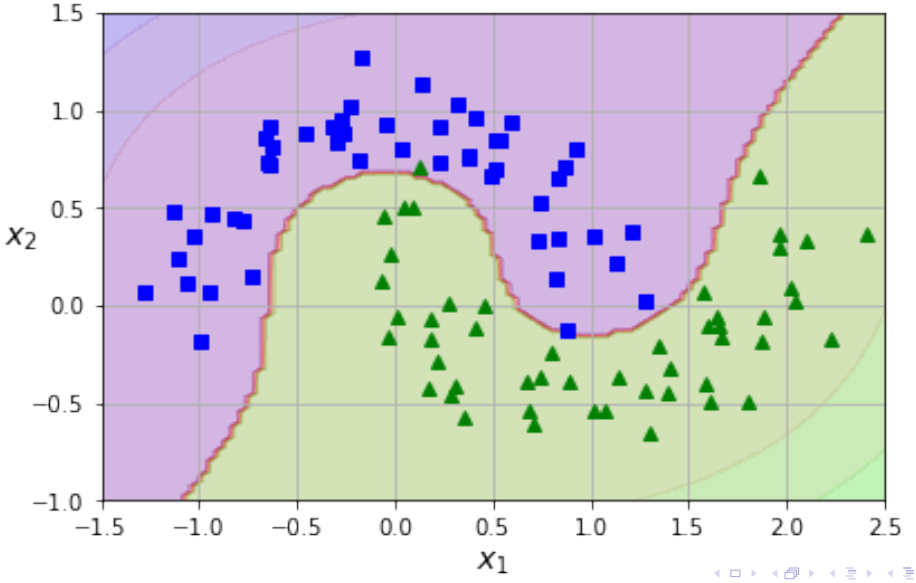
Para implementar esta ideia, podemos criar um pipeline contendo um transformador `PolynomialFeatures`, seguido por um `StandardScaler` e um `LinearSVC`. Vamos testar isso no dataset *moons*.

```
from sklearn.datasets import make_moons
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.svm import LinearSVC

X, y = make_moons(n_samples=100, noise=0.15, random_state=42)

polynomial_svm_clf = make_pipeline(
    PolynomialFeatures(degree=3),
    StandardScaler(),
    LinearSVC(C=10, max_iter=10_000, random_state=42)
)
polynomial_svm_clf.fit(X, y)
```

Resultado no Dataset Moons



Kernel Polinomial

- Adicionar features polinomiais é simples, mas um grau polinomial baixo não consegue lidar com datasets muito complexos, e um grau alto cria um número enorme de features, tornando o modelo lento.
- Felizmente, SVMs permitem usar uma técnica matemática chamada **truque do kernel (kernel trick)**.
- Ela possibilita obter o mesmo resultado como se você tivesse adicionado muitas features polinomiais, mesmo com um grau muito alto, sem realmente ter que adicioná-las.
- Essa técnica é implementada pela classe `SVC`.

```
from sklearn.svm import SVC

poly_kernel_svm_clf = make_pipeline(StandardScaler(),
                                    SVC(kernel="poly", degree=3, C=5))
poly_kernel_svm_clf.fit(X, y)
```

SVMs com Kernel Polinomial

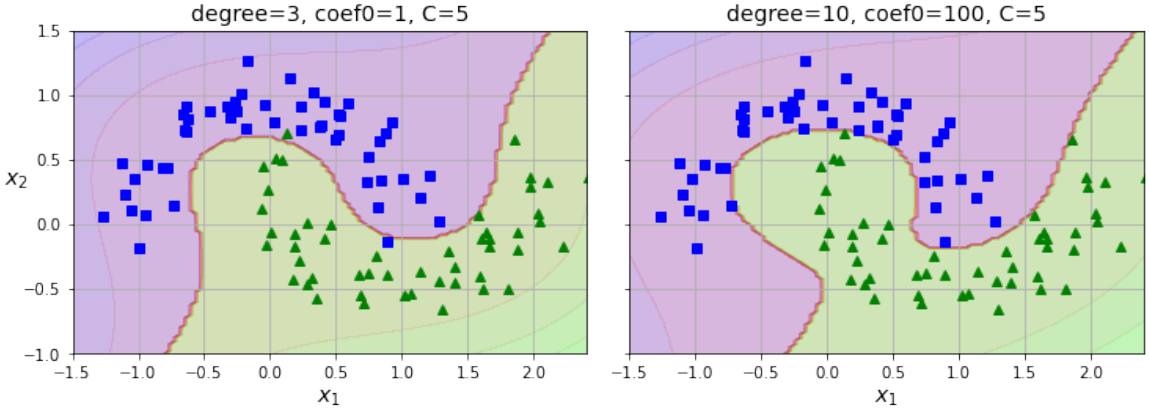


Figure: Classificadores SVM com um kernel polinomial (grau 3 à esquerda, grau 10 à direita).

- `coef0` é um parâmetro da função do kernel polinomial que permite controlar a flexibilidade do modelo através da maior influência dos termos de maior grau (quanto maior o valor, mais flexível).
- Se overfitting: reduzir o grau do polinômio, o C ou o `coef0`. Se underfitting, aumentá-los.

Features de Similaridade

Outra técnica para lidar com problemas não lineares é adicionar features calculadas usando uma **função de similaridade**, que mede o quanto cada instância se assemelha a um **marco (landmark)** particular.

- Ideia: selecionar marcos e criar features que medem a similaridade de cada instância a cada marco.
- Usaremos a função Gaussiana RBF como função de similaridade.

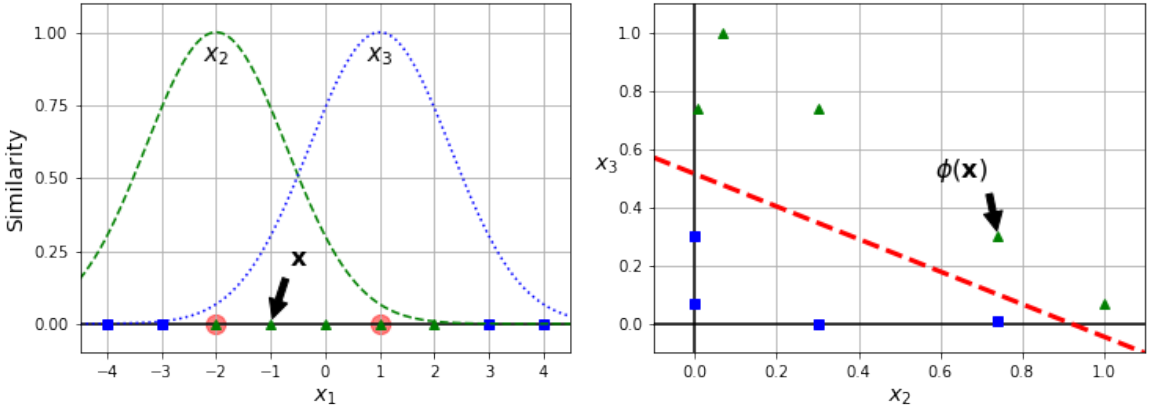


Figure: Features de similaridade usando o RBF Gaussiano. O dataset da direita torna-se linearmente separável.

Kernel Gaussiano RBF

- Assim como o método de features polinomiais, o método de features de similaridade pode ser computacionalmente caro.
- Da mesma forma que o kernel polinomial, o truque do kernel pode ser usado para obter o mesmo resultado como se você tivesse adicionado muitas features de similaridade, sem realmente fazê-lo.

Vamos testar a classe `SVC` com o kernel Gaussiano RBF:

```
rbf_kernel_svm_clf = make_pipeline(StandardScaler(),
                                   SVC(kernel="rbf", gamma=5, C=0.001))
rbf_kernel_svm_clf.fit(X, y)
```

`gamma` é proporcional ao inverso da variância da Gaussiana RBF.
Quanto maior o `gamma`, mais estreita a forma de sino e mais sensível ela se torna a pontos de dados próximos.
Então, reduza o `gamma` para reduzir o overfitting, ou aumente-o para reduzir o underfitting.

Não tem o hiperparâmetro `coef0`.

Efeito dos Hiperparâmetros gamma (γ) e C

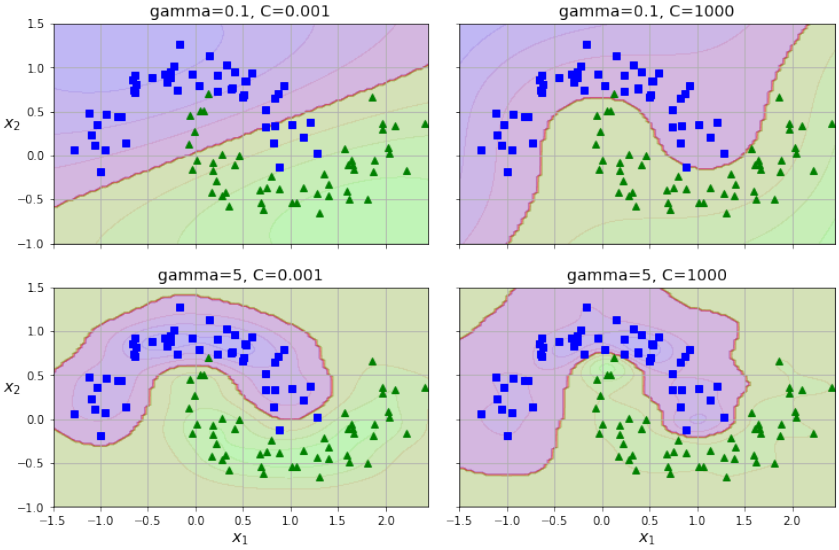


Figure: Classificadores SVM usando um kernel RBF com diferentes valores de gamma e C.

Dicas sobre a Escolha do Kernel

Com tantos kernels para escolher, como decidir qual usar?

Regra geral

1. Sempre tente o **kernel linear** primeiro. Obs.: A classe `LinearSVC` é muito mais rápida que `SVC(kernel="linear")`, especialmente se o conjunto de treinamento for muito grande.
2. Se o conjunto de treinamento não for muito grande, você também deve tentar **kernels SVM**, começando com o **kernel Gaussiano RBF**; ele geralmente funciona muito bem.
3. Se tiver tempo e poder computacional de sobra, você pode experimentar alguns outros kernels, usando busca de hiperparâmetros.

Classes de SVM e Complexidade Computacional

- `LinearSVC`: Baseado na biblioteca *liblinear*, que implementa um algoritmo otimizado para SVMs lineares. Não suporta o truque do kernel, mas escala quase linearmente com o número de instâncias de treinamento e o número de features. Complexidade de tempo $\approx \mathcal{O}(m \times n)$.

- `SVC`: Baseado na biblioteca *libsvm*, que implementa um algoritmo que suporta o truque do kernel. A complexidade do tempo de treinamento geralmente está entre $\mathcal{O}(m^2 \times n)$ e $\mathcal{O}(m^3 \times n)$. Fica terrivelmente lento quando o número de instâncias de treinamento fica grande.

- `SGDClassifier`: É possível treinar um classificador SVM linear usando Gradiente Descendente Estocástico (use o parâmetro `loss="hinge"`). Não suporta kernel trick, mas é eficiente e lida bem com grandes datasets que não cabem na memória. Complexidade $\approx \mathcal{O}(m \times n)$.

Tabela Comparativa das Classes de SVM

Table: Comparação das classes do Scikit-Learn para classificação SVM

Classe	**Complexidade de Tempo**	**Suporte Out-of-core**	**Scaling required**	**Kernel trick**
LinearSVC	$\mathcal{O}(m \times n)$	Não	Sim	Não
SVC	$\mathcal{O}(m^2 \times n)$ a $\mathcal{O}(m^3 \times n)$	Não	Sim	Sim
SGDClassifier	$\mathcal{O}(m \times n)$	Sim	Sim	Não

Fim