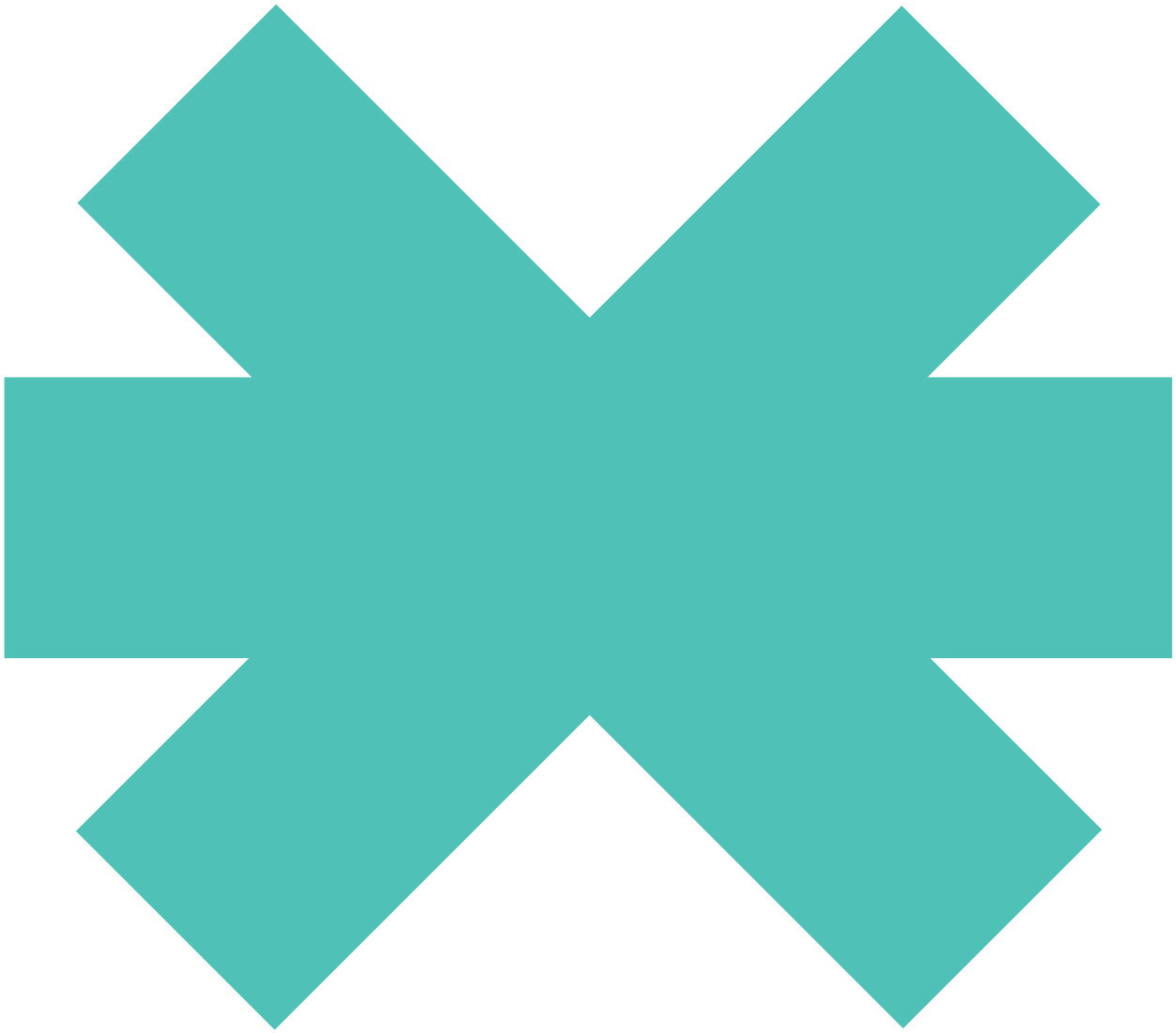
MÓDULO 2
Aprendizagem supervisionada
Especialização em Machine Learning

9

Classificação por SVM

New Technology School

Tokio.



9 Classificação por SVM

Sumário

Os *Support Vector Machines* (SVM) são algoritmos de classificação baseados na regressão linguística, que procuram maximizar a margem de distância entre classes.

Os SVM substituem a hipótese da regressão logística por outra hipótese usada. Recordemos que a regressão logística apenas pode propor retas de equações lineares para dividir o plano de dados, isto é, não pode criar outro tipo de divisões mais complexas.

Para isso, os SVM transformam o plano de dados antes de encontrar essa solução linear através de outras hipóteses, permitindo-nos ajustar modelos, que uma regressão linear logística não poderia.

Também podem ser usados quando o rácio entre o número de exemplos e características dificulta o treino de um modelo logístico tradicional.

As suas áreas de aplicação são semelhantes à regressão logística, dependendo da distribuição e comportamento dos dados disponíveis.

9.1 Regressão logística vs. SVM

Com a regressão logística, procuramos uma função linear, que delimite as duas regiões atribuídas às classes, mas nem sempre encontramos uma função linear que consiga satisfazer.

A função apenas relaciona linearmente as características proporcionadas.

Assim, por exemplo, se as classes se dividem segundo áreas não divisíveis, como neste caso (uma zona quase circular), uma regressão linear não poderá classificá-las:

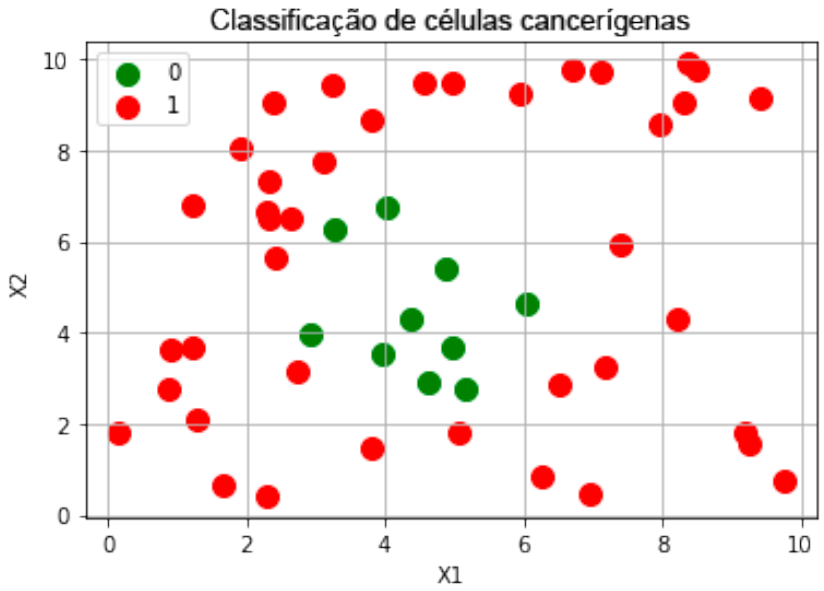


Figura 29

9.2 Hipótese

Transformamos o plano de características dos exemplos num plano diferente através de uma transformação, usando uma função de similaridade entre pontos.

Esse espaço resultante será, então, o valor da "proximidade" ou similaridade de um ponto à sua *landmark* mais próxima, calculada com essa função de similaridade ou *kernel*.

Por outro lado, do valor resultante, teremos uma hipótese diferente: o valor de *Y* dependerá de se a similaridade multiplicada pelo vetor de coeficientes ou "pesos" for maior ou menor de *1* e *-1*, em vez de 0, para maximizar a margem entre classes.

$$h_\theta(f_i) = \theta^T \times f_i$$
$$f_j = \quad similaridade(x, l^j)$$
$$x_i^j \sim l^k \rightarrow f_i^j \approx 1$$
$$x_i^j \neq l^k \rightarrow f_i^j \approx 0$$
$$h_\theta(f_i) \geq 0 \rightarrow y = 1$$
$$h_\theta(f_i) < 0 \rightarrow y = -1$$

Figura 30

Código Latex:

```
h_\theta(f_i) = \theta^T \times f_i \\
f_j = similaridade(x, l^j) \\
x_i^j \sim l^k \rightarrow f_i^j \approx 1 \\
x_i^j \neq l^k \rightarrow f_i^j \approx 0 \\
h_\theta(f_i) \geq 0 \rightarrow y = 1 \\
h_\theta(f_i) < 0 \rightarrow y = -1
```

9.3 *Kernels* e *landmarks*

Uma *landmark* é um ponto ou localização no espaço de características originais. À sua volta, definimos uma região à qual será atribuída uma classe. Logo, a região ou regiões independentes, atribuídas a essa classe serão definidas por uma ou várias *landmarks* e o espaço à sua volta.

Um *kernel* é uma função de similaridade ou proximidade de um ponto/exemplo original com cada uma das *landmarks*. Usando a função *kernel* podemos passar de um espaço de exemplos ao novo espaço transformado, que usaremos para treinar o SVM.

Representamos os dados originais, a *landmark* em cor azul e a reta não linear que o SVM ajustará e que dividirá ambas as classes a preto:

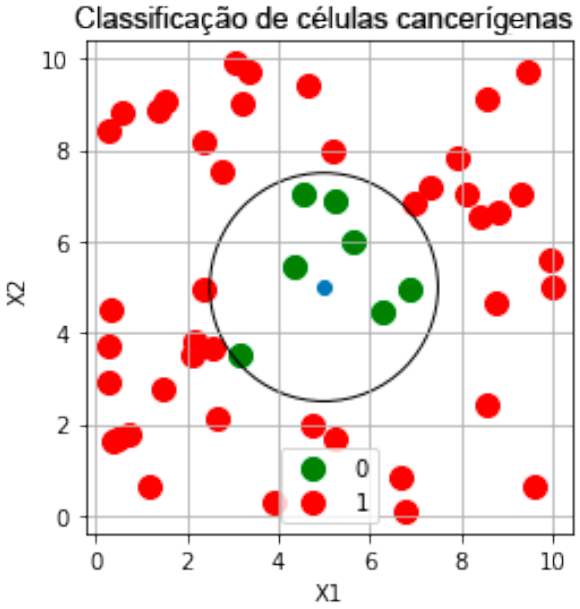


Figura 31

9.4 Transformação da hipótese

Assim, passamos de duas características (X_1 e X_2) a apenas uma, num novo espaço vetorial *F*, que relaciona cada ponto original em *X* com a distância à *landmark* selecionada:

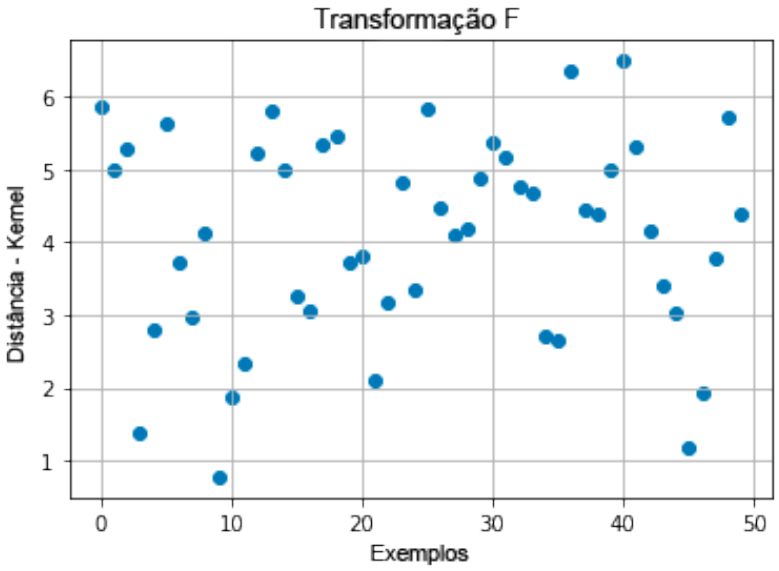


Figura 32

9.5 Tipos de *kernels* disponíveis

Existe um grande número de *kernels* disponíveis. Entre os mais comuns, realçamos os seguintes:

- **Sem *kernel* ou *kernel* "linear":** usamos a hipótese de SVM, mas não transformamos o espaço de características de *X*.
- ***Kernel* gaussiano:** a similaridade é dada pela probabilidade, seguindo uma distribuição normal.
- **Kernel polinomial**: usa polinómios das características originais, permitindo ajustar modelos não lineares; combinando características, também consegue ajustar o modelo sobre a interação entre características.
- ***String kernel*:** permite-nos operar com cadeias de caracteres de diferentes tamanhos, obtendo a semelhança entre duas cadeias diferentes.
- ***Kernel* de chi²:** usado habitualmente em visão computacional ou com variáveis discretas.

Kernel gaussiano

É o mais utilizado. Modela os resultados em função da probabilidade e segundo uma distribuição gaussiana ao redor da *landmark*. Assim, a similaridade será maior, quanto mais nos aproximemos da *landmark,* seguindo a distribuição normal, em vez de uma distância linear.

Por sua vez, poderemos representar círculos e ovais, usando os parâmetros da distribuição normal:

- σ: desvio-padrão.
- *l*: média da distribuição ou posição da *landmark*.

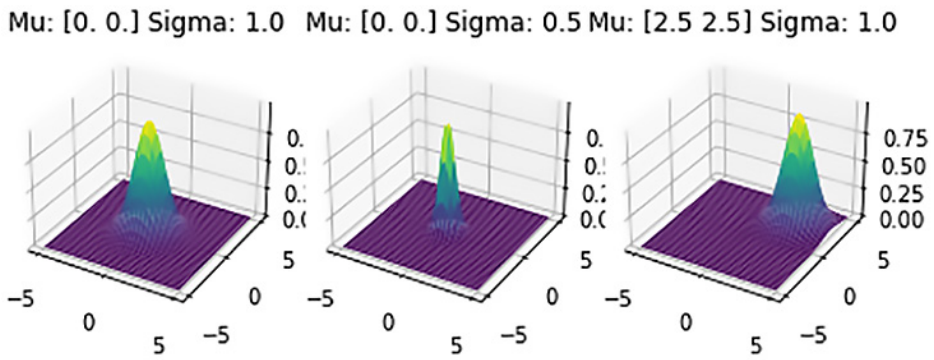


Figura 33

Nota: não normalizar as características antes de aplicar a transformação.

$$f_i = e^{-\frac{||x - l^i||^2}{2\sigma^2}}$$

Figura 34

Código Latex:

```
f_i = e^{- \frac{||x - l^i||^2}{2 \sigma^2}}
```

9.6 Funções de custo

A função de custo substituirá a hipótese usada na regressão logística, com a função de ativação, pela hipótese de SVM e o *kernel* utilizado.

$$J_\Theta = C[\sum_{i=0}^{m} y^i coste_1(\theta^T x^i) + (1 - y^i) coste_0(\theta^T x^i) + \frac{1}{2} \sum_{j=1}^{m} \theta_j^2]$$

Figura 35

Código Latex:

```
J_\Theta = C [\sum\limits_{i=0}^{m} y^i custo_1(\theta^T x^i) + (1 - y^i) custo_0(\theta^T x^i) + \frac{1}{2} \sum\limits_{j=1}^{m} \theta_j^2]
```

9.7 Parâmetro de regularização

Como na regressão linear e logística, podemos regularizar a regressão por SVM; neste caso com o parâmetro *C*: $C = 1/\lambda$.

Assim, C atua inversamente a λ:

- A maior C, maior variância ou sobreajuste e menor desvio ou *bias*.
- A menor C, menor variância ou sobreajuste e maior desvio ou *bias*.

9.8 Algoritmo de treino: classificação multiclasse

1. Compilar os exemplos *X* e os seus resultados previamente conhecidos *Y*.
2. Normalizar os exemplos *X* (exceto para *kernel* gaussiano).
3. Aplicar a transformação do espaço de características *X* no espaço *F*, consoante o *kernel* utilizado.
4. Dividir o conjunto de dados em subconjuntos de treino, validação e teste.

    5.1 Treinar um modelo binário por cada classe:

    5.2 Codificar *Y* para *[0, 1]* segundo a classe.

    5.3 Escolher um rácio de aprendizagem α.

    5.4 Inicializar os pesos Θ, de forma aleatória.

    5.5 Iterativamente, calcular o custo, assim como as suas derivadas/inclinação, e atualizar Θ sobre o subconjunto de treino.

    5.6 Finalizar quando Θ convergir num valor ótimo.

    5.7 Modificar o rácio de aprendizagem α se necessário.

    5.8 Obter a Θ ótima.

    5.9 Otimizar λ iterativamente sobre o subconjunto de validação.

    5.10 Obter a *C* ótima.

    5.11 Avaliar o modelo sobre o subconjunto de teste e obter o custo total final.

    5.12 Obter a Θ e *C* ótimas para o modelo.

Perante novos exemplos, predizemos através da utilização do modelo para cada classe e ficaremos com a única classe predita ou com a que tenha o valor máximo pré-ativado.