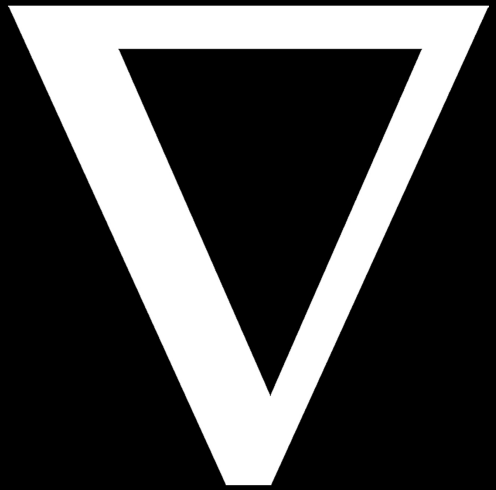


Aula 4

Glauco Fleury Corrêa de Moraes

Presença

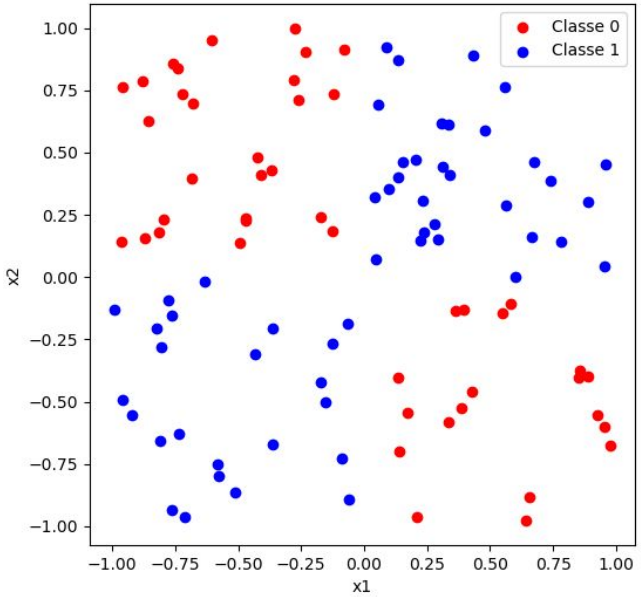
- Linktree: Presente na bio do nosso instagram
- Presença ficará disponível até 1 hora antes da próxima aula
- É necessário 70% de presença para obter o certificado

Presença e Github

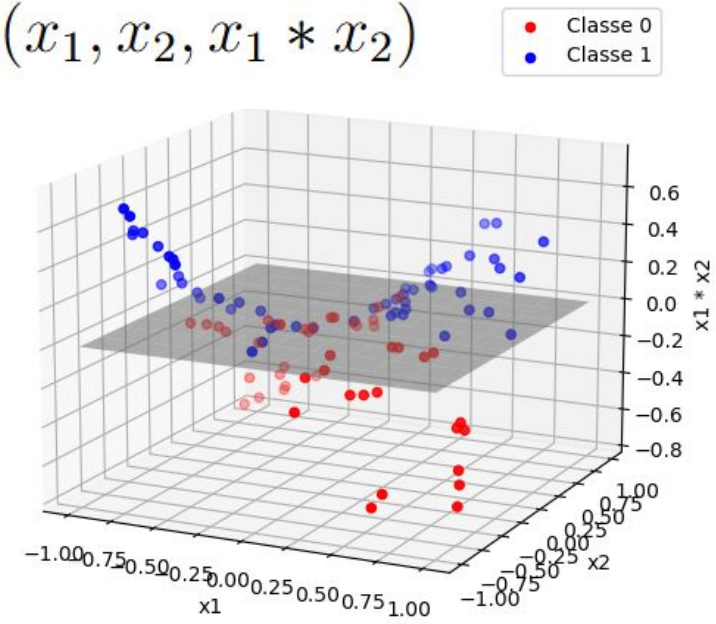
Revisão

Feature Transformation

(x_1, x_2)



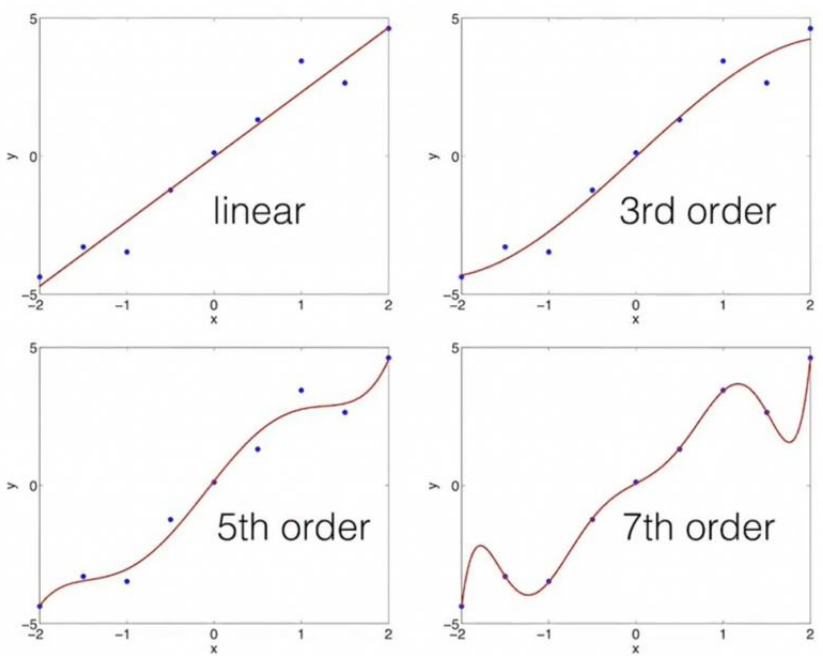
$(x_1, x_2, x_1 * x_2)$



Alta Dimensionalidade

- Alta dimensionalidade pode causar overfitting
- Muito custoso para calcular

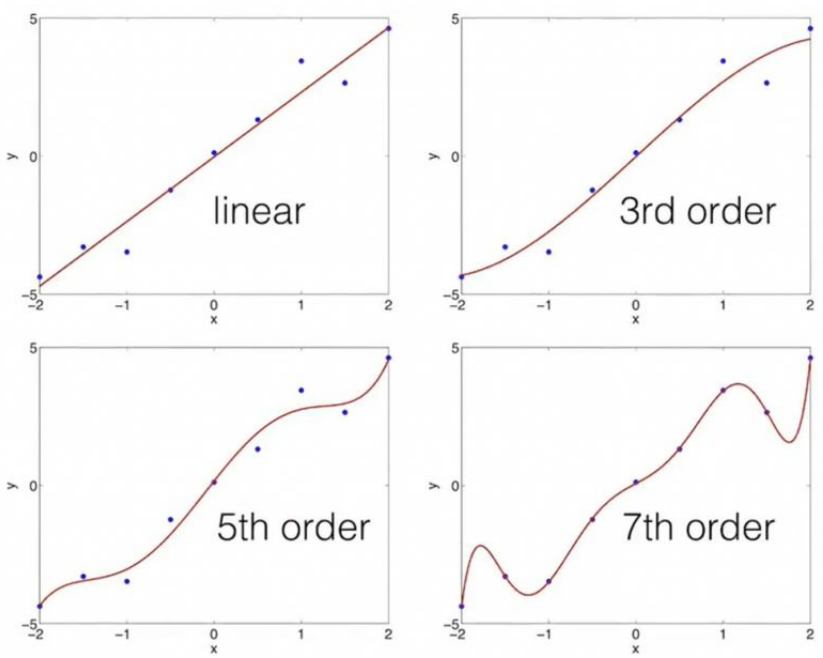
- Funções Kernel (próxima aula)



Alta Dimensionalidade

- Alta dimensionalidade pode causar overfitting
- Muito custoso para calcular

- Funções Kernel (agora!)



Funções Kernel

Definição

- A função Kernel será definida como o produto interno da feature transformation de vetores de atributos:

$$K(x, x') = \phi(x)\phi(x')$$

.

Definição

- Mesmo que transformações contenham vetores de alta dimensionalidade, o produto interno entre elas é fácil de calcular.

Transformação ϕ:

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

$$\phi(x') = [x'_1, x'_2.x'^2_1, \sqrt{2}x'_1x'_2, x'^2_2]$$

Definição

- O produto interno entre as transformações será

$$\phi(x).\phi(x') = (x.x') + (x.x')^2$$

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

$$\phi(x') = [x'_1, x'_2.x_1'^2, \sqrt{2}x'_1x'_2, x_2'^2]$$

Tipos de função Kernel

É importante saber os diferentes tipos de Kernel, a fim de saber qual tipo é mais útil para o problema que você quer resolver. Os 4 que apresentaremos hoje são:

- Linear
- Polinomial
- Radial Basis Function (RBF)
- Sigmóide

Kernel Linear

- Kernel mais simples possível
- Sem feature transformation (produto escalar dos vetores originais)
- Dados tem que já ser linearmente separáveis nas dimensões originais

$$K(x, x') = x \cdot x'$$

Kernel Polinomial

- aplica a expansão polinomial, já vista antes no curso
- “gamma” e “r” são parâmetros de ajuste dos coeficientes da expansão, enquanto “d” é o grau da expansão polinomial (2°, 3°, 4°, etc)
- computacionalmente custoso para altas dimensões

$$K(x, x') = (\gamma(x \cdot x') + r)^d$$

Kernel RBF (Radial Basis Function)

- feature transformation para um espaço vetorial de dimensão infinita, que considera todas as combinações entre expoentes da dimensão original (somente o kernel é calculável)
- Goal-to-Kernel: altamente versátil, capaz de capturar relações não lineares melhor que o Polinomial, por um custo computacional menor
- somente 1 parâmetro

Feature expansion: infinitas combinações / dimensões

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \qquad \Phi(\mathbf{x}) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1 \\ x_1 x_2 \\ \dots \end{bmatrix}$$

Solução: Kernel RBF

$$\Phi(\mathbf{x_1}) \cdot \Phi(\mathbf{x_2}) = K(x, x') = e^{-\gamma ||x - x'||^2}$$

Kernel Sigmoidal

- utiliza a função tangente hiperbólico (tanh(x)=f(x))
- apresenta 2 parâmetros: “alpha” e “r”
- não é muito utilizado atualmente, por ser em geral pior do que os demais apresentados

$$K(x, x') = tanh(\alpha(x \cdot x') + r)$$

Algoritmo com Kernel

1. $\theta = 0$
2. for i in range (n)
3. if $y^{(i)}(\theta \cdot \phi(x^{(i)})) \leq 0$
4. $\theta = \theta + y^{(i)}\phi(x^{(i)})$

Algoritmo com Kernel

- Se estamos na iteração 4, e sabemos que na iteração 1 e 3 tiveram erros, mas na iteração 2 não:

$$\theta = 1.y^{(1)}\phi(x^{(1)}) + 0.y^{(2)}\phi(x^{(2)}) + 1.y^{(3)}\phi(x^{(3)})$$

1. $\theta = 0$
2. for i in range (n)
3. if $y^{(i)}(\theta \cdot \phi(x^{(i)})) \leq 0$
4. $\theta = \theta + y^{(i)}\phi(x^{(i)})$

Algoritmo com Kernel

- Se estamos na iteração B e sabemos os resultados de todas as outras iterações, o valor de theta pode ser calculado por:

$$\theta = \sum_{j=1}^{B} \alpha_j y^{(j)} \phi(x^{(j)})$$

- α_j é 0, caso, classificado como correto
- α_j é 1, caso, classificado de forma errada

Algoritmo com Kernel

- Sendo assim, podemos alterar o algoritmo para:

1. $\theta = 0$
2. for i in range (n)
3. if $y^{(i)}(\theta \cdot \phi(x^{(i)})) \leq 0$
4. $\theta = \theta + y^{(i)}\phi(x^{(i)})$

1. $\theta = 0; \alpha_1, \ldots, \alpha_n = 0$
2. for i in range (n)
3. if $y^{(i)}(\sum_{j=1}^{i-1} \alpha_j y^{(j)} \phi(x^{(j)}).\phi(x^{(i)}))$
4. $\alpha_i = 1$

Algoritmo com Kernel

- Aumenta a complexidade? Um for a mais!

1. $\theta = 0$
2. for i in range (n)
3. if $y^{(i)}(\theta \cdot \phi(x^{(i)})) \leq 0$
4. $\theta = \theta + y^{(i)}\phi(x^{(i)})$

1. $\theta = 0; \alpha_1, \ldots, \alpha_n = 0$
2. for i in range (n)
3. if $y^{(i)}(\sum_{j=1}^{i-1} \alpha_j y^{(j)} \phi(x^{(j)}).\phi(x^{(i)}))$
4. $\alpha_i = 1$

Algoritmo com Kernel

- Mesmo tendo um for a mais, este for contêm o produto interno entre dois ϕ, ou seja, eles deixam de utilizar ϕ e passam a utilizar kernel.

1. $\theta = 0$
2. for i in range (n)
3. if $y^{(i)}(\theta \cdot \phi(x^{(i)})) \leq 0$
4. $\theta = \theta + y^{(i)}\phi(x^{(i)})$

1. $\theta = 0; \alpha_1, \ldots, \alpha_n = 0$
2. for i in range (n)
3. if $y^{(i)}(\sum_{j=1}^{i-1} \alpha_j y^{(j)} \phi(x^{(j)}).\phi(x^{(i)}))$
4. $\alpha_i = 1$

Algoritmo com Kernel

- Mesmo tendo um for a mais, este for contêm o produto interno entre dois ϕ, ou seja, eles deixam de utilizar ϕ e passam a utilizar kernel.

$$y^{(i)}(\sum_{j=1}^{i-1} \alpha_j y^{(j)} \phi(x^{(j)}).\phi(x^{(i)})) = y^{(i)}(\sum_{j=1}^{i-1} \alpha_j y^{(j)} K(x^{(i)}, x^{(j)}))$$

Algoritmo com Kernel

- Algoritmo final:

1. $\theta = 0; \alpha_1, \ldots, \alpha_n = 0$
2. for i in range (n)
3. if $y^{(i)}(\sum_{j=1}^{i-1} \alpha_j y^{(j)} K(x^{(i)}, x^{(j)}))$
4. $\alpha_i = 1$

Motivação - SVM

Motivação - SVM

- Utilizamos feature transformations e kernel para transformar dados não-lineares em dados lineares

Motivação - SVM

- Utilizamos feature transformations e kernel para transformar dados não-lineares em dados lineares
- Voltamos a um problema de aulas passadas: como definir a melhor decision boundary (hiperplano)?

Motivação - SVM

- Utilizamos feature transformations e kernel para transformar dados não-lineares em dados lineares
- Voltamos a um problema de aulas passadas: como definir a melhor decision boundary (hiperplano)?
- Solução: **SVM**
  - Melhor Hiperplano -> Maximiza a distância entre os pontos das classes que estão mais próximos do hiperplano

SVM

Como classificar um ponto

- Após o treinamento do modelo, um novo ponto pode ser classificado com:

$$f(\mathbf{x}) = \text{sign}(\mathbf{w}^\top \mathbf{x} - b)$$

Restrições para uma boa margem

Queremos uma margem que:

- Separe corretamente os pontos
- Esteja a uma distância máxima dos pontos extremos de cada classe

Dessa forma, temos uma generalização melhor

Restrições para uma boa margem

Para a correta separação dos dados, temos as seguintes restrições:

$$\mathbf{w}^\top \mathbf{x}_i - b \geq +1 \quad \text{se } y_i = +1$$

$$\mathbf{w}^\top \mathbf{x}_i - b \leq -1 \quad \text{se } y_i = -1$$

Restrições para uma boa margem

De forma equivalente, temos:

$$y_i(\mathbf{w}^\top \mathbf{x}_i - b) \geq 1 \quad \forall i$$

Como calcular a distância entre as margens

Temos que a margem é dada por:

$$\text{margem} = \frac{2}{\|\mathbf{w}\|}$$

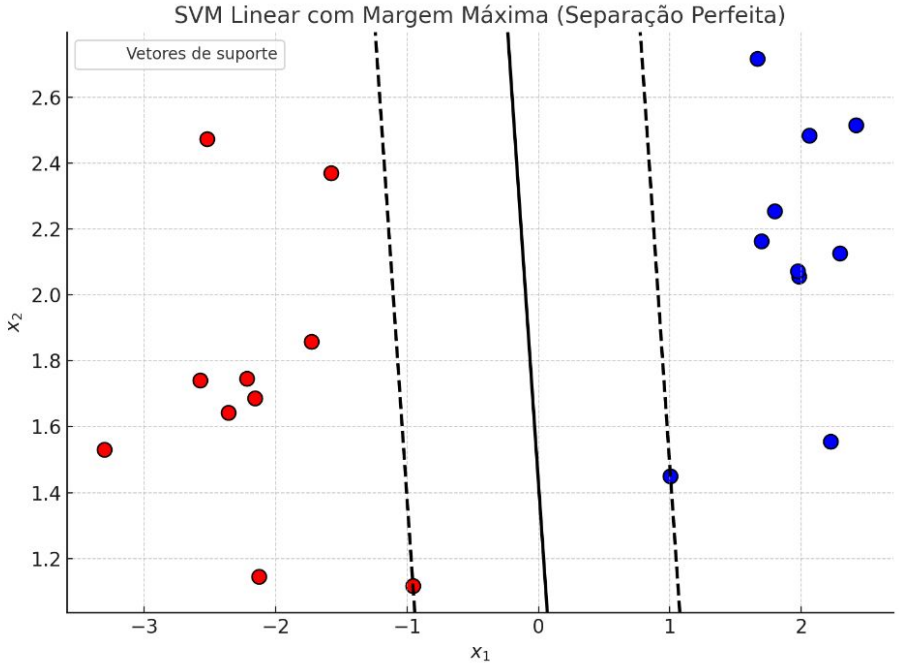
Como maximizar a distância entre as margens

Para maximizar esse valor, isso será a mesma coisa que minimizar o valor de $\|\mathbf{w}\|$

Assim, o problema de otimização do SVM é dado por:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{sujeito a } y_i(\mathbf{w}^\top \mathbf{x}_i - b) \geq 1 \quad \forall i$$

Intuição geométrica



Note que temos a margem cortando dois valores. Esses valores são chamados de vetores de suporte.

Problemas

No mundo real, nem sempre os exemplos conseguem ser separados perfeitamente. Para isso, precisamos modificar a fórmula de minimização anterior.
Vamos incluir uma variável que permite erro, chamada de variável de folga, com $\xi_i \geq 0$

$$y_i(\mathbf{w}^\top \mathbf{x}_i - b) \geq 1 - \xi_i$$

Nova otimização

Assim, a nova otimização segue o seguinte:

$$\min_{\mathbf{w},b,\xi} \left[\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N} \xi_i \right]$$

O parâmetro $C > 0$ controla o equilíbrio entre maximizar a margem e permitir erro. Quanto maior o C, menos erros são permitidos.

Multiplicadores de Lagrange

Abordaremos brevemente os multiplicadores de Lagrange para embasar uma reformulação essencial do problema da SVM

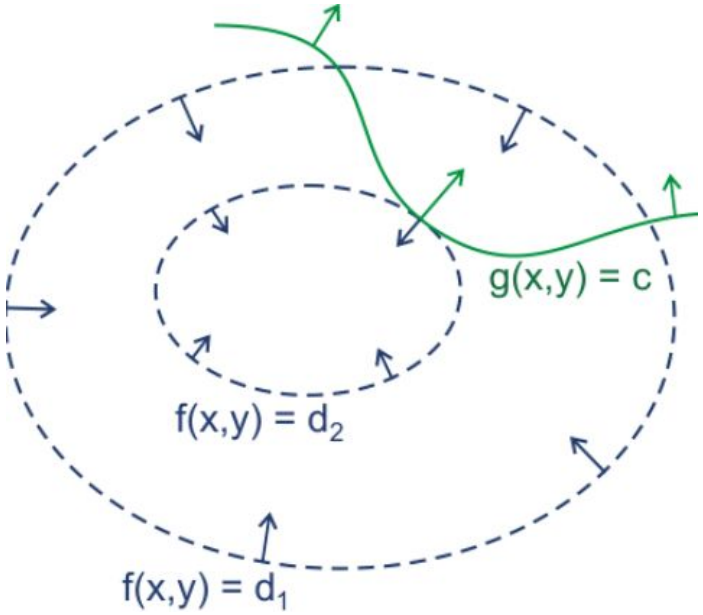
- Situação de uso: otimização de funções multivariáveis $(f(x, y, ..))$ em domínios restritos, do tipo $(g(x, y, ..) = k)$
- Achar os extremos da função f (ponto de Mínima e ponto de Máxima)

- ideia: quando as curvas ou superfícies de nível da restrição se igualam às curvas ou superfícies de nível da função original, teremos candidatos aos extremos
- Como o gradiente é perpendicular à curva para cada função, matematicamente, basta resolver o sistema de equações dado por:

$$\nabla f = \lambda \nabla g; \quad g(x, y, ..) = k$$

Objetivo: achar pontos para a Lagrangiana zerar

$$\mathbb{L}(\lambda, x, y, ..) = 0 = \lambda \nabla g - \nabla f \quad [\text{restrição: } g(x, y, ..) = k]$$



SVM

Nova função objetivo (Lagrangiana)

$$L = \frac{1}{2}\|\mathbf{w}\|^2 - \sum \alpha_i [y_i(w \cdot x_i + b) - 1]$$

Achando os extremos

$$\frac{\delta L}{\delta w} = 0 \qquad \frac{\delta L}{\delta b} = 0$$

Achando os extremos

$$\frac{\delta L}{\delta w} = 0$$

$$w - \sum \alpha_i y_i x_i = 0$$

$$w = \sum \alpha_i y_i x_i$$

Achando os extremos

$$\frac{\delta L}{\delta b} = 0$$

$$-\sum \alpha_i y_i = 0$$

$$\sum \alpha_i y_i = 0$$

Substituindo

$$L = \sum \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)$$

Mudando como classificar um ponto

$$\text{sign}(\mathbf{w}^\top \mathbf{x} - b) = \text{sign}(\sum \alpha_i y_i (x_i \cdot u) \quad + b)$$

Porque fizemos tudo isso?

- Com isso, fica fácil ver porque usamos esse método, pois agora tanto a nossa função objetivo quanto a nossa função de classificação depende apenas do resultado do produto escalar entre vetores de features. Logo, podemos usar as funções kernels para solucionar esse problema.

Prática

data@icmc.usp.br

@data.icmc

/c/DataICMC

/icmc-data

data.icmc.usp.br

obrigado por sua presença!