Aula 4

0.1 Funções Kernel

Funções Kernel consiste do produto interno entre transformações de atributos. A ideia é que, mesmo que as transformações possam conter vetores de alta dimensões, o produto interno entre eles é relativamente barato de ser calculado.

Vamos tomar por exemplo a transformação ϕ sendo aplicada em dois vetores, x e x':

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

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

O produto interno entre ambas essas transformações é:

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

Ou seja, apesar da transformação aumentar a dimensão de uma forma considerável, a forma de calcular o produto interno é razoavelmente simples.

Definimos a função kernel como o produto interno de vetores de atributos:

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

1 Algumas funções Kernel

É possível escolher vários tipos de funções kernel distintas para poder encontrar o melhor hiperplano que separe os dados em questão, e pode ser difícil saber precisamente quais são adequadas para o problema em questão. Porém, há certos tipos de problema em que algumas funções podem ser mais promissoras que as outras, e ter essa noção é muito proveitoso para o uso prático das SVM's. Dito isso, vamos apresentar os principais kernels (e suas guidelines gerais):

1.1 Linear

Formulação:

$$K(x_1, x_2) = x_1 \cdot x_2 \tag{1}$$

Esse é o kernel mais simples possível: como é bem perceptível, ele não aplica transformação nenhuma aos dados, meramente calculando o produto escalar direto deles (sem qualquer feature transformation). Ele é muito útil para datasets que podem ser linearmente separáveis na dimensão original em que se encontram. Porém, para conjuntos muito complexos de dados, ele é insuficiente.

1.2 Polinomial

Formulação:

$$K(x_1, x_2) = (\gamma(x_1 \cdot x_2) + r)^d = \sum_{k=0}^{d} \binom{d}{k} \gamma^k (x^\top x')^k r^{d-k} \tag{2}$$

Esse kernel busca executar expansões polinomiais no dataset, sendo r uma constante que determina a relação entre termos de alta e baixa ordem, d, o grau da expansão polinomial, e γ, um parâmetro que determina o quanto alteramos o feature space antes de "fazer"a feature expansion (aspas presentes devido ao fato de não computarmos de fato essa expansão, apenas o produto escalar dos vetores expandidos).

Por exemplo: se temos pontos 1D, e desejamos expandi-los para 2D e calcularmos o produto-escalar, usando o kernel polinomial, podemos fazer a seguinte manipulação (para r = 1/2, γ = 1, d = 2):

$$K(x_1, x_2) = (x_1 \cdot x_2 + \frac{1}{2})^2 = (x_1 \cdot x_2)^2 + (x_1 \cdot x_2) + \frac{1}{4} \tag{3}$$

O que equivale a:

$$\Phi(x_1) \cdot \Phi(x_2) = [x_1^2, x_1, 1/2] \cdot [x_2^2, x_2, 1/2] \tag{4}$$

Mostrando como esse kernel relaciona-se à familiar expansão polinomial. Ele é aplicável a dados não-lineares nas dimensões originais, podendo capturar essas relações, mas é propenso a overfitting em expansões altas e computacionalmente custoso.

1.3 Radial Basis Function (RBF)

Formulação:

$$K(x_1, x_2) = e^{-\gamma \|x_1 - x_2\|^2} \tag{5}$$

Matematicamente, esse kernel é fantástico. Caso essa função seja expandida com polinômios de Taylor, obtém-se:

$$K(x_1, x_2) = \sum_{n=0}^{\infty} \frac{(-\gamma)^n}{n!} \|x_1 - x_2'\|^{2n} \tag{6}$$

Equivalente a:

$$K(\mathbf{x}_1, \mathbf{x}_2) = 1 - \gamma \|\mathbf{x}_1 - \mathbf{x}_2\|^2 + \frac{\gamma^2}{2!} \|\mathbf{x}_1 - \mathbf{x}_2\|^4 - \frac{\gamma^3}{3!} \|\mathbf{x}_1 - \mathbf{x}_2\|^6 + \dots \tag{7}$$

O que implica que estamos fazendo uma feature expansion infinita! Porém, como é a função kernel, não há problema algum, já que só calculamos o produto escalar, sem jamais de fato mappear nossos dados para dimensões infinitas (o que é belíssimo). O kernel RBF é útil para uma variedade enorme de datasets, sendo verdadeiramente o kernel default: tem 1 só parâmetro de ajuste (compare com o polinomial, que tem 3), é bem estudado na literatura, e empiricamente eficaz para a maioria dos problemas que envolvem SVM.

1.4 Sigmoidal

Formulação:

$$K(\mathbf{x}_1, \mathbf{x}_2) = \tanh\left(\alpha\, \mathbf{x}_1^\top \mathbf{x}_2 + r\right) \tag{8}$$

Onde α e r são parâmetros ajustáveis. O kernel sigmoidal é encapsulado pela função tangente hiperbólico, grantindo uma imagem no conjunto $[-1, +1]$. Apesar de não ser especialmente custoso computacionalmente, e capturar relações não lineares entre dados, o kernel sigmoidal não é muito usado na prática, devido à superioridade das outras alternativas existentes.

1.5 Kernel Perceptron

O algoritmo do perceptron levando em conta o ϕ é o seguinte:

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

Se estamos na iteração 4 e sabemos que na iteração 1 e 3 tiveram erros, mas na iteração 2 não. Você concorda que o atual valor de θ será:

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

Ou seja, se estamos na iteração B, mas sabemos os resultados de todas as outras iterações o valor de θ pode ser calculado por:

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

onde α_j é 0 caso o exemplo foi corretamente classificado e 1 caso ele foi classificado de forma errada.

Sendo assim, uma alteração que podemos fazer para o algoritmo é a seguinte:

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. $\alpha_i = 1$

De início isso parece ser uma alteração que aumenta a complexidade do algoritmo, pois agora a cada iteração temos que fazer mais um somatório. Porém, se você prestar bastante atenção este somatório contém o produto interno entre dois ϕ, ou seja:

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

Fazendo com que o algoritmo seja:

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. $\alpha_i = 1$

Ou seja, apesar de termos mais um somatório para resolver agora não precisamos mais trabalhar diretamente com ϕ e sim somente com a função kernel, o que melhora o desempenho do algoritmo.

2 Motivação - SVM

Como visto, funções kernel e feature transformations são técnicas utilizadas para trabalhar com dados não-lineares em dimensões maiores, de modo que transformemos as informações em amostras linearmente separáveis. Com isso, retornamos a um dos problemas de aulas passadas: como encontrar a melhor separação entre as classes?

Mesmo em um espaço onde os dados se tornam separáveis, existem infinitos hiperplanos que poderiam ser usados para dividir as classes. A grande questão é: qual deles oferece a melhor capacidade de generalização para novos dados?

É nesse contexto em que surge o algoritmo SVM e a sua premissa: maximizar a margem, ou seja, escolher o hiperplano que deixa a maior distância possível entre os pontos mais próximos de cada classe. A intuição por trás dessa escolha é que, quanto maior essa margem, menor a chance de o classificador cometer erros em exemplos futuros que estejam próximos da fronteira.

Nas próximas seções, trataremos mais afundo esse algoritmo.

3 Support Vector Machine (SVM) Linear

O *Support Vector Machine* (SVM) é um algoritmo de aprendizado supervisionado utilizado principalmente para tarefas de classificação binária. O objetivo do SVM é encontrar uma **fronteira de decisão ótima** que separe corretamente os dados de diferentes classes com a maior margem possível.

3.1 Separação Linear

Dado um conjunto de dados $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, onde $\mathbf{x}_i \in \mathbb{R}^D$ representa o vetor de características e $y_i \in \{-1, +1\}$ o rótulo da classe, o SVM procura um hiperplano da forma:

$$\mathbf{w}^\top \mathbf{x} - b = 0 \tag{9}$$

Este hiperplano deve separar os exemplos positivos ($y_i = +1$) dos negativos ($y_i = -1$), com a maior margem possível entre os dois grupos. Isso não é novidade, já vimos anteriormente essa equação na classificação linear. Vamos entender agora como maximizar a distância da margem, de maneira a diminuir o número de erros em classificações de novos exemplos.

3.2 Margem e Vetores de Suporte

A **margem** é a distância entre o hiperplano de decisão e os pontos mais próximos de cada classe. O SVM busca maximizar essa margem. Os pontos que estão mais próximos do hiperplano são chamados de *vetores de suporte*. É daí que vem o nome Support Vector Machine (Máquina de Vetores de Suporte).

As restrições para separação correta dos dados são:

$$\mathbf{w}^\top \mathbf{x}_i - b \geq +1 \quad \text{se } y_i = +1 \tag{10}$$
$$\mathbf{w}^\top \mathbf{x}_i - b \leq -1 \quad \text{se } y_i = -1 \tag{11}$$

Ou, de forma equivalente:

$$y_i(\mathbf{w}^\top \mathbf{x}_i - b) \geq 1 \quad \forall i \tag{12}$$

3.3 Problema de Otimização

O vetor $\mathbf{w}$ define a direção do hiperplano no espaço das características, e o valor escalar b desloca esse plano em relação à origem. A equação $\mathbf{w}^\top \mathbf{x} - b = 0$ define um conjunto de pontos $\mathbf{x}$ que estão exatamente sobre a superfície de decisão — isto é, o limite entre as duas classes.

A distância de um ponto $\mathbf{x}_i$ ao hiperplano é dada por:

$$\frac{|\mathbf{w}^\top \mathbf{x}_i - b|}{\|\mathbf{w}\|} \tag{13}$$

Portanto, para que os pontos mais próximos (os vetores de suporte) estejam o mais distante possível do hiperplano, queremos **maximizar a margem**, que é exatamente:

$$\text{margem} = \frac{2}{\|\mathbf{w}\|} \tag{14}$$

Esse 2 aparece pois pegamos a distância entre os dois lados do hiperplano. Como é 1 para cada lado, o valor 2 aparece. Maximizar essa margem é o mesmo que **minimizar o valor de** $\|\mathbf{w}\|$. Como $\|\mathbf{w}\|$ envolve uma raiz quadrada, é comum simplificar o problema minimizando $\frac{1}{2}\|\mathbf{w}\|^2$, que tem o mesmo mínimo, mas é mais fácil de resolver computacionalmente.

Assim, o problema de otimização do SVM (também chamado de *SVM de margem rígida*) pode ser formulado como:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{sujeito a } y_i(\mathbf{w}^\top \mathbf{x}_i - b) \geq 1 \quad \forall i \tag{15}$$

O conjunto de restrições garante que todos os exemplos estejam corretamente classificados e fora da margem. A função objetivo representa a busca por um hiperplano com a maior separação entre as classes, ou seja, com a **melhor generalização possível**.

3.4 Classificação

Uma vez treinado o modelo, novos exemplos $\mathbf{x}$ podem ser classificados usando a seguinte função:

$$f(\mathbf{x}) = \text{sign}(\mathbf{w}^\top \mathbf{x} - b) \tag{16}$$

3.5 Margem Suave (Soft Margin)

Em muitos casos práticos, os dados não são perfeitamente separáveis. Para permitir algumas violações das restrições, introduzimos variáveis de folga $\xi_i \geq 0$:

$$y_i(\mathbf{w}^\top \mathbf{x}_i - b) \geq 1 - \xi_i \tag{17}$$

O novo problema de otimização fica:

$$\min_{\mathbf{w},b,\xi} \left[\frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{N} \xi_i \right] \tag{18}$$

O parâmetro $C > 0$ controla o equilíbrio entre maximizar a margem e permitir erros de classificação (quanto maior o C, menos erros são permitidos).

3.6 Intuição Geométrica

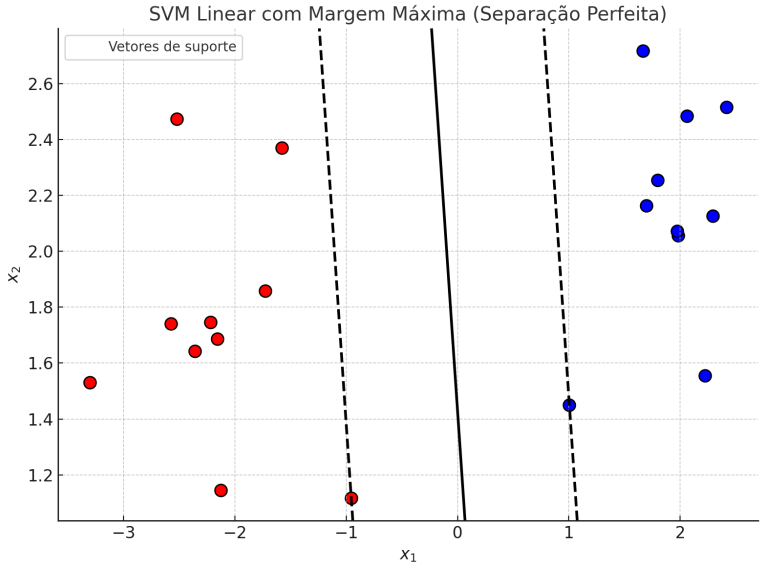

Figura 1: Exemplo de SVM linear com margem máxima.

A figura mostra um exemplo com duas classes separadas por um hiperplano. As linhas tracejadas representam os limites da margem, e os pontos sobre essas linhas são os vetores de suporte.

3.7 Mudando o problema da otimização

Por um motivo que ficará claro ao final dessa explicação, precisamos mudar a nossa função objetivo, transformando-a no que matematicamente chamamos de "versão dual"da otimização original.

A princípio, cabe uma breve explicação sobre um dos métodos mais comuns de otimização funcional: multiplicadores de Lagrange. Em linhas gerais, esse método é útil sempre que desejamos achar MÁX e MÍN de funções multivariáveis ($f(x, y, ..)$) submetidas a restrições ($g(x, y, ..) = k$)

Imaginando exemplos que se enquadram no caso acima, é possível notar que as curvas de nível de f mudam por todo o domínio, e só encontram possíveis máximos ou mínimos quando tangentes à curva restritiva. Matematicamente, como os gradientes sempre são perpendiculares a elas, só haverá esses extremos funcionais nos pontos que satisfazem:

$$\nabla f = \lambda \nabla g; \quad g(x, y, ..) = k \tag{19}$$

Isto é, nos pontos em que a Lagrangiana é 0 e a restrição é satisfeita:

$$\mathbb{L}(\lambda, x, y, ..) = 0 = \lambda \nabla g - \nabla f \quad [\text{restrição: } g(x, y, ..) = k] \tag{20}$$

A equação acima sempre garante um sistema resolvível, com soluções distintas que devem ser analisadas para ver quais as 2 que correspondem de fato aos pontos de extremo. Essa técnica será

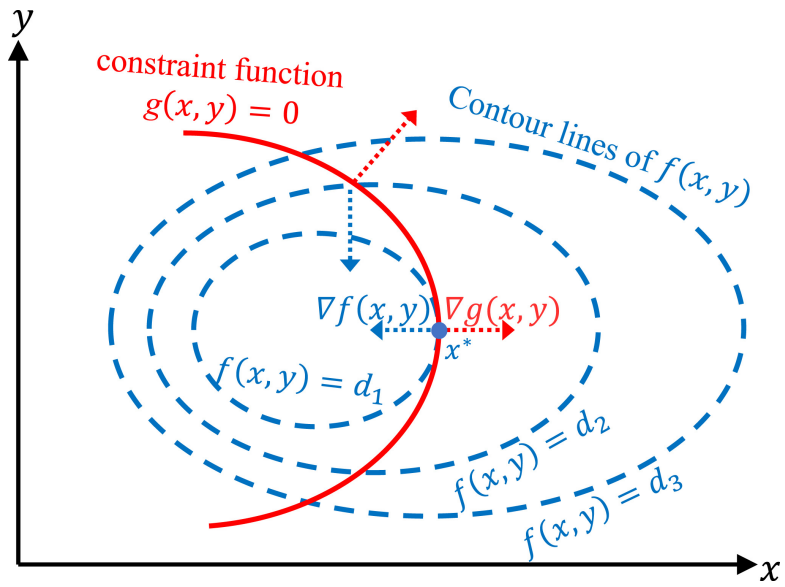

Figura 2: Imagem que captura a essência dos multiplicadores de Lagrange, para uma função com domínio 2D

utilizada agora para buscar esses pontos em nossa função de otimização original, de forma que nossa Lagrangiana será:

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

Onde $[y_i(w \cdot x_i + b) - 1]$ é a restrição que havíamos definido anteriormente.

Agora, fazendo a derivada parcial de L em relação à w e à b e igualando elas à 0, obtemos as seguintes informações para sabermos onde estão os extremos dessa nova função objetivo:

$$\frac{\delta L}{\delta w} = 0 \tag{21}$$

$$w - \sum \alpha_i y_i x_i = 0 \tag{22}$$

$$w = \sum \alpha_i y_i x_i \tag{23}$$

e

$$\frac{\delta L}{\delta b} = 0 \tag{24}$$

$$-\sum \alpha_i y_i = 0 \tag{25}$$

$$\sum \alpha_i y_i = 0 \tag{26}$$

Substituindo isso em L e na nossa regra de classificação, temos:

$$L = \sum \alpha_i \ - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) \tag{27}$$

$$\text{sign}(\mathbf{w}^\top \mathbf{x} - b) = \text{sign}(\sum \alpha_i y_i (x_i \cdot u) \ \ + b) \tag{28}$$

Onde u é o vetor de features que queremos saber a classificação

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.

O que precisamos fazer quando queremos usar os kernels é simplesmente usarmos $K(x_i, x_j)$., ao invés de usar o $(x_i \cdot x_j)$.

Como otimizamos na prática?

Para encontrar os valores ideais dos multiplicadores de Lagrange α_i que maximizam a função objetivo dual, usamos algoritmos de otimização numérica. Um dos métodos mais comuns é o **gradiente ascendente**, já que estamos lidando com um problema de **maximização** (e não de minimização, como ocorre na forma primal).

Nesse método, iterativamente ajustamos os valores de α_i na direção do gradiente da função, até encontrarmos um ponto de máximo (ou um ótimo local no caso de restrições complexas). Em problemas reais, técnicas mais sofisticadas como o método do subgradiente ou algoritmos de otimização quadrática (como o SMO — *Sequential Minimal Optimization*) também são utilizados, porém esse não é o foco agora.

O importante aqui é entender que, ao reformular o problema na forma dual, conseguimos aplicar esse tipo de algoritmo de forma eficiente, aproveitando o fato de que a função depende apenas dos produtos escalares entre vetores de entrada.

O funcionamento do algoritmo do gradiente descendente para minimização de função já foi explicado anteriormente, sendo que a única diferença aqui é que utilizamos os valores que irão maximizar a função, ou seja, estamos pegando o valor do gradiente no sentido do crescimento da função.

$$K(x, x') = e^{-\gamma||x-x'||^2} \tag{29}$$