

Aprendizado supervisionado

Instituto Federal Sul-rio-grandense - IFSul Campus Pelotas

Engenharia Elétrica

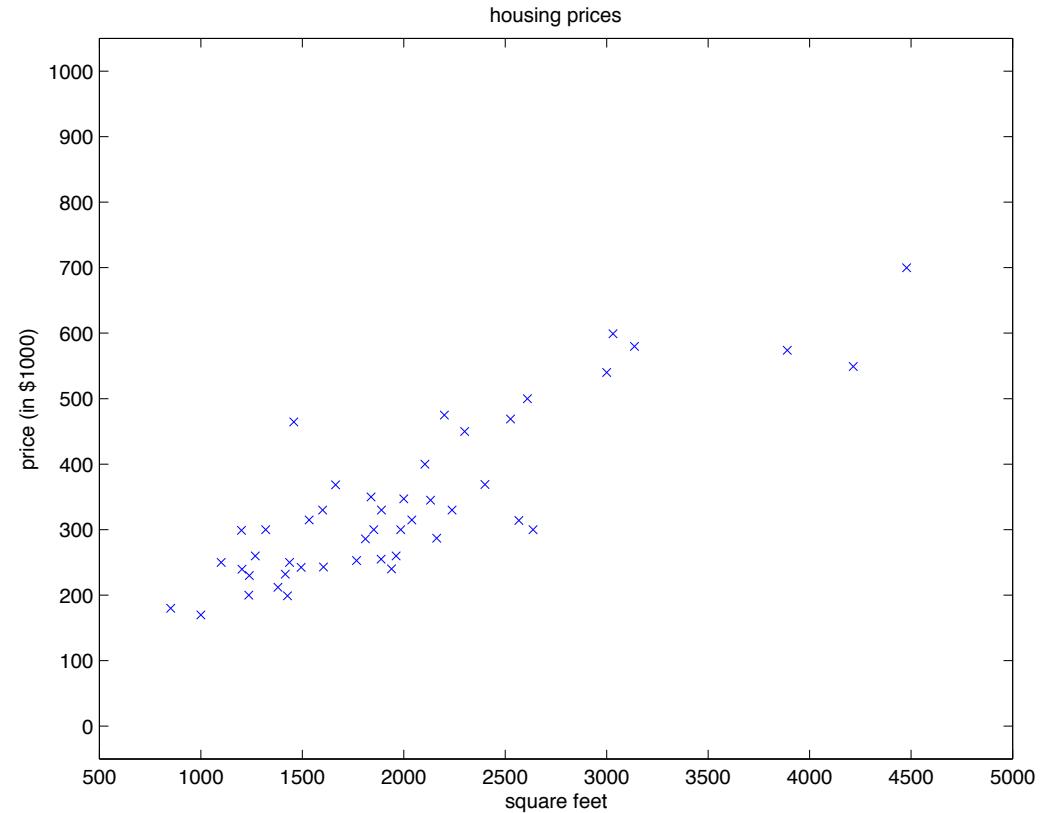
Prof. Lucian Soares Schiavon

Aprendizagem de Máquina

Introdução

- Suponhamos que temos os dados das áreas e preços de 47 casas em uma cidade:

Living area (feet ²)	Price (1000\$)
2104	400
1600	330
2400	369
1416	232
3000	540
:	:

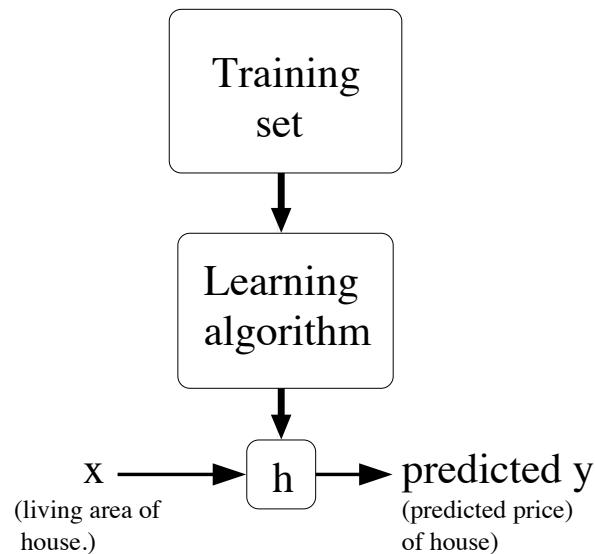


Introdução

- A partir desses dados, como podemos aprender a prever os preços de outras casas dessa mesma cidade, em função do tamanho de suas áreas?
- Utilizaremos a notação $x^{(i)}$ como as variáveis de entrada (as áreas, nesse exemplo), as quais são chamadas de *features*, e $y^{(i)}$ como as de saída, as quais estamos tentando prever (o preço, nesse caso) e podem ser chamadas de *target*.
- O par $(x^{(i)}, y^{(i)})$ é chamado de exemplo de treinamento (*training example*) e os dados que são usados para aprendizagem, uma lista de m exemplos de treinamento $[(x^{(i)}, y^{(i)}); i = 1, \dots, m]$ é chamado de dados de treinamento (*training set*).

Introdução

- Em outras palavras, nosso objetivo é - dado os dados de treinamento - aprender uma função h para que $h(x)$ seja um bom preditor para os valores correspondentes de y .



- Quando a variável que estamos tentando prever é contínua, nós chamamos o problema de aprendizagem como um problema de regressão. Quando y pode assumir somente um pequeno número de valores discretos, nós chamamos de um problema de classificação.

Régressão linear

Regressão linear

- Vamos considerar o *dataset* utilizado no problema das casas mais ricas, considerando o número de quartos em cada casa:

Living area (feet ²)	#bedrooms	Price (1000\$s)
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540
:	:	:

- Agora, os x 's são vetores de duas dimensões. Por exemplo, $x_1^{(i)}$ é a área das casas nos dados de treinamento e $x_2^{(i)}$ é o número de quartos.
- Dependendo do problema, podemos adicionar outras características de entrada. Por exemplo, nesse caso, podíamos adicionar o número de banheiros, se possui lareira etc.

Regressão linear

- Para realizar o aprendizado supervisionado, temos que decidir como vamos representar funções h em um computador.
- Inicialmente, vamos aproximar y como uma função linear de x :

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

- Os θ_i 's são os parâmetros (também chamados de pesos). Para simplificar a notação, vamos convencionar que $x_0 = 1$ e então

$$h(x) = \sum_{i=0}^n \theta_i x_i = \theta^T x$$

- Na parte do lado direito da equação acima, θ e x são ambos vetores, e n é o número de variáveis de entrada (sem contar x_0).

Regressão linear

- Agora, considerando os dados de treinamento, como aprendemos os parâmetros θ ?
- Um método razoável seria fazer os valores de $h(x)$ perto de y , pelo menos para os dados de treinamento que temos.
- Para isso, definimos uma função que mede, para cada valor de θ 's, o quanto perto $h(x^{(i)})$'s são dos $y^{(i)}$'s correspondentes. Portanto, definimos como a função de custo:

$$J(\theta) = \frac{1}{2} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2$$

Algoritmo LMS

Algoritmo LMS

- Queremos escolher θ que minimize $J(\theta)$. Para isso, vamos utilizar um algoritmo que comece com um “chute” inicial para θ , e então repetidamente modifique θ para fazer $J(\theta)$ menor, até convergir para o valor de θ que minimize $J(\theta)$.
- Especificamente, vamos considerar o método do gradiente (*gradient descent*):

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

- Consideramos α como a taxa de aprendizagem (*learning rate*). Esse algoritmo repetidamente dá um passo em direção da diminuição mais íngreme de J .

Algoritmo LMS

- Para implementar esse algoritmo, vamos resolver a derivada na parte direita da equação. Primeiramente, vamos considerar somente um exemplo de treinamento (x, y) , ignorando a soma presente na definição de J .

$$\begin{aligned}\frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2} (h_{\theta}(x) - y)^2 \\ &= 2 \cdot \frac{1}{2} (h_{\theta}(x) - y) \cdot \frac{\partial}{\partial \theta_j} (h_{\theta}(x) - y) \\ &= (h_{\theta}(x) - y) \cdot \frac{\partial}{\partial \theta_j} \left(\sum_{i=0}^n \theta_i x_i - y \right) \\ &= (h_{\theta}(x) - y) x_j\end{aligned}$$

Algoritmo LMS

- Para somente um exemplo de treinamento, temos a seguinte regra de atualização:

$$\theta_j := \theta_j + \alpha (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)}$$

- Essa regra é conhecida como regra de atualização LMS (*Least Mean Squares*) e também conhecida como regra de aprendizagem *Widrow-Hoff*.
- Podemos notar que a atualização dos pesos é proporcional ao termo de erro $(y^{(i)} - h_\theta(x^{(i)}))$.
- Portanto, se encontramos um exemplo de treinamento no qual a nossa predição é praticamente igual o valor real y , então temos pouca necessidade na alteração dos parâmetros.

Algoritmo LMS

- Encontramos a regra LMS somente para um exemplo de treinamento. Basicamente, temos duas maneiras de modificar o método considerando dados de treinamento com mais de um exemplo.
- O primeiro é o algoritmo abaixo:

Repeat until convergence {

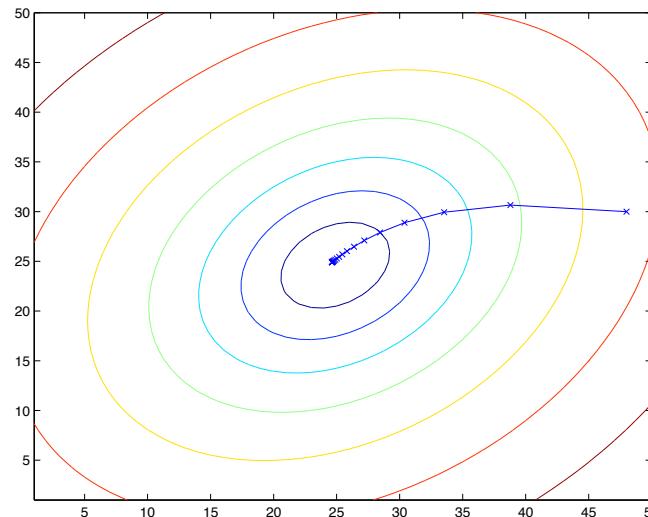
$$\theta_j := \theta_j + \alpha \sum_{i=1}^m (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)} \quad (\text{for every } j).$$

}

- Esse método aborda todos os dados de treinamento em cada passo de atualização dos parâmetros e é chamado de *batch gradient descent* (batch = lote).

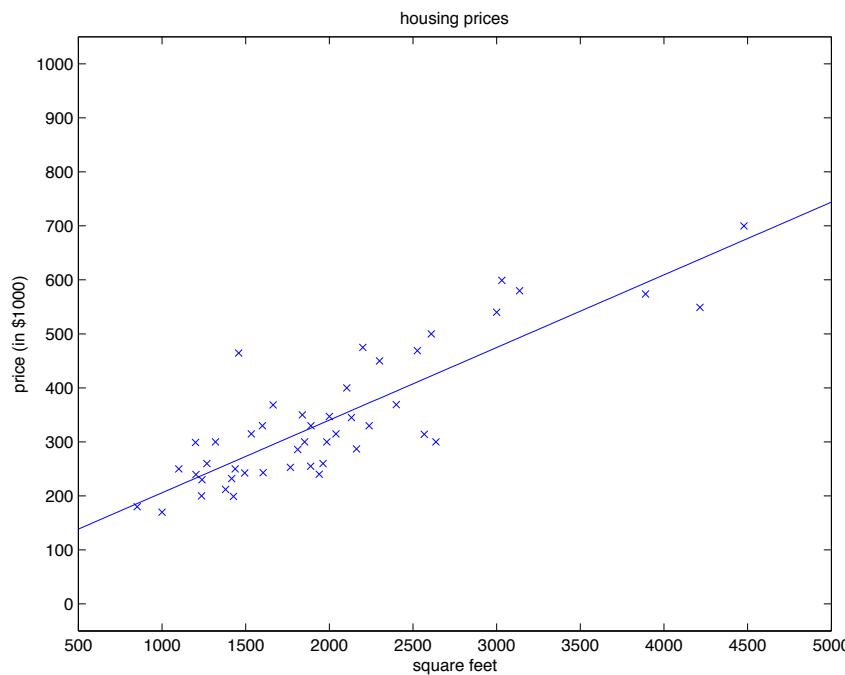
Algoritmo LMS

- Apesar do método do gradiente ser suscetível à mínimos locais em geral, esse método mostrado acima possui apenas um mínimo global e nenhum outro mínimo local. Ou seja, se considerarmos que o valor de learning rate α não é muito grande, o gradiente sempre converge.
- As elipses abaixo mostram os contornos de uma função quadrática. Os x's mostrados na figura marcam os valores sucessivos de θ que o método do gradiente assumiu durante o algoritmo.



Algoritmo LMS

- Utilizando o algoritmo no problema dos preços das casas, obtemos $\theta_0 = 71,27$ e $\theta_1 = 0,1345$.
- Abaixo temos o gráfico de $h_\theta(x)$ em função de x (área), em conjunto com os dados de treinamento.



Algoritmo LMS

- Temos uma alternativa em relação ao algoritmo batch mostrado anteriormente:

```
Loop {  
    for i=1 to m, {  
         $\theta_j := \theta_j + \alpha (y^{(i)} - h_{\theta}(x^{(i)})) x_j^{(i)}$       (for every j).  
    }  
}
```

- Nesse algoritmo, passamos repetidamente pelo conjunto de treinamento e, a cada vez que encontramos um par de dados de treinamento, atualizamos os parâmetros de acordo com o gradiente do erro em relação somente àquele exemplo de treinamento.
- Esse algoritmo é chamado de *stochastic gradient descent* (ou *incremental gradient descent*).

Algoritmo LMS

- Enquanto o algoritmo *batch gradient descent* tem que passar por todos os dados de treinamento antes de atualizar uma única vez os parâmetros θ - o que é uma operação custosa se m for grande - o algoritmo *stochastic gradient descent* começa a ter progresso imediatamente no primeiro exemplo de treinamento e continua a cada dado de treinamento que passa pelo algoritmo.
- Portanto, geralmente o *stochastic gradient descent* alcança o valor de θ perto do mínimo muito antes do *batch*.
- No entanto, ele pode nunca convergir para o mínimo de $J(\theta)$ e os parâmetros continuarão oscilando em torno desse valor. Mas, na prática, esses valores serão boas aproximações do mínimo verdadeiro.
- Por essas razões, se tiver um número muito grande de dados de treinamento, o algoritmo *stochastic gradient descent* é preferido.

Equações normais

Equações normais

- Os algoritmos vistos anteriormente são uma maneira de minimizar J .
- Vamos discutir uma segunda opção, realizando a minimização explicitamente e sem precisar de um algoritmo iterativo.
- Nesse método, iremos minimizar J calculando suas derivadas em relação aos θ_j 's e igualando-as à zero.

Revisitando os mínimos quadrados

- Considerando os dados de treinamento, definimos uma matriz \mathbf{X} como uma matriz $m \times n$ que contem os valores de entrada dos exemplos de treinamento em suas linhas:

$$\mathbf{X} = \begin{bmatrix} \quad (x^{(1)})^T \quad \\ \quad (x^{(2)})^T \quad \\ \vdots \\ \quad (x^{(m)})^T \quad \end{bmatrix}$$

- Também consideramos \vec{y} como um vetor de tamanho m contendo os valores de saída (desejados) dos dados de treinamento:

$$\vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

Revisitando os mínimos quadrados

- Considerando que $h_\theta(x^{(i)}) = (x^{(i)})^T \theta$, podemos verificar que:

$$\begin{aligned} X\theta - \vec{y} &= \begin{bmatrix} (x^{(1)})^T \theta \\ \vdots \\ (x^{(m)})^T \theta \end{bmatrix} - \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix} \\ &= \begin{bmatrix} h_\theta(x^{(1)}) - y^{(1)} \\ \vdots \\ h_\theta(x^{(m)}) - y^{(m)} \end{bmatrix}. \end{aligned}$$

- Então, usando o fato que para um vetor z , temos que $z^T z = \sum_i z_i^2$:

$$\begin{aligned} \frac{1}{2}(X\theta - \vec{y})^T(X\theta - \vec{y}) &= \frac{1}{2} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2 \\ &= J(\theta) \end{aligned}$$

Revisitando os mínimos quadrados

- Considerando os tópicos 4.1 e 4.3 sobre gradiente e funções lineares no material de revisão de álgebra, sabemos que o gradiente de uma função f em respeito à uma matriz \mathbf{A} de tamanho $m \times n$ é:

$$\nabla_{\mathbf{A}} f(\mathbf{A}) \in \mathbb{R}^{m \times n} = \begin{bmatrix} \frac{\partial f(\mathbf{A})}{\partial A_{11}} & \frac{\partial f(\mathbf{A})}{\partial A_{12}} & \dots & \frac{\partial f(\mathbf{A})}{\partial A_{1n}} \\ \frac{\partial f(\mathbf{A})}{\partial A_{21}} & \frac{\partial f(\mathbf{A})}{\partial A_{22}} & \dots & \frac{\partial f(\mathbf{A})}{\partial A_{2n}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f(\mathbf{A})}{\partial A_{m1}} & \frac{\partial f(\mathbf{A})}{\partial A_{m2}} & \dots & \frac{\partial f(\mathbf{A})}{\partial A_{mn}} \end{bmatrix}$$

- Em particular, se \mathbf{A} é somente um vetor x , temos:

$$\nabla_x f(x) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \frac{\partial f(x)}{\partial x_2} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{bmatrix}$$

Revisitando os mínimos quadrados

- Considerando que temos a função $f(x) = x^T Ax$, temos as seguintes propriedades:

$$\nabla_x x^T Ax = 2Ax \text{ (if } A \text{ symmetric)}$$

$$\nabla_x^2 x^T Ax = 2A \text{ (if } A \text{ symmetric)}$$

- Agora, considerando uma matriz **A** com tamanho $m \times n$ e um vetor **b** com tamanho m , podemos encontrar um vetor **x** que aproxime o máximo possível Ax de **b**, medido pelo quadrado da norma euclidiana (3.5):

$$\begin{aligned} \|Ax - b\|_2^2 &= (Ax - b)^T(Ax - b) \\ &= x^T A^T Ax - 2b^T Ax + b^T b \end{aligned}$$

- Calculando o gradiente em relação à **x**, e usando as propriedades anteriores, temos:

$$\begin{aligned} \nabla_x(x^T A^T Ax - 2b^T Ax + b^T b) &= \nabla_x x^T A^T Ax - \nabla_x 2b^T Ax + \nabla_x b^T b \\ &= 2A^T Ax - 2A^T b \end{aligned}$$

Revisitando os mínimos quadrados

- Igualando esta equação à zero e resolvendo para x , temos a equação normal:

$$x = (A^T A)^{-1} A^T b$$

- Relacionando as equações para o caso da função $J(\theta)$, temos a seguinte equação normal:

$$X^T X \theta = X^T \vec{y}$$

- Portanto, o valor de θ que minimize $J(\theta)$ é dada em forma fechada pela equação

$$\theta = (X^T X)^{-1} X^T \vec{y}$$

Interpretação probabilística

- Nessa seção veremos, utilizando conceitos de probabilidade, porque a regressão linear e, mais especificamente, a função de custo dos mínimos quadrados J são uma boa escolha para um problema de regressão.
- Vamos assumir que as variáveis desejadas de saída e as entradas são relacionadas pela seguinte equação:

$$y^{(i)} = \theta^T x^{(i)} + \epsilon^{(i)}$$

- O termo $\epsilon^{(i)}$ é um termo de erro que abrange tanto efeitos não-modelados (por exemplo, características pertinentes à predição do preço das casas que ficaram de fora da regressão) ou ruído aleatório.

Interpretação probabilística

- Vamos assumir que $\epsilon^{(i)}$ possui uma distribuição gaussiana (ou distribuição normal) com média zero e variância σ^2 . Portanto, a densidade de $\epsilon^{(i)}$ é dada por:

$$p(\epsilon^{(i)}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(\epsilon^{(i)})^2}{2\sigma^2}\right)$$

- O que implica em:

$$p(y^{(i)}|x^{(i)}; \theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right)$$

- A notação acima indica que é a distribuição de $y^{(i)}$ dado $x^{(i)}$ e parametrizado por θ .
- Dado \mathbf{X} (matriz com todos os dados de entrada $x^{(i)}$) e θ , qual é a distribuição dos $y^{(i)}$? Essa quantidade é tipicamente vista como uma função de \vec{y} (e talvez \mathbf{X}) para um valor fixo de θ .

Interpretação probabilística

- Quando queremos explicitamente ver isso como função de θ , vamos chamá-lo de função de probabilidade (*likelihood function*):

$$L(\theta) = L(\theta; X, \vec{y}) = p(\vec{y}|X; \theta)$$

- Dado a premissa de independência dos $\epsilon^{(i)}$ - e portanto também dos $y^{(i)}$ dados os $x^{(i)}$ - podemos escrever-la como:

$$\begin{aligned} L(\theta) &= \prod_{i=1}^m p(y^{(i)} | x^{(i)}; \theta) \\ &= \prod_{i=1}^m \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \end{aligned}$$

Interpretação probabilística

- Agora que temos um modelo probabilístico relacionando $y^{(i)}$ e $x^{(i)}$, qual é a melhor maneira de escolhermos os parâmetros θ ?
- O princípio da máxima probabilidade (*maximum likelihood*) diz que devemos escolher θ de modo a termos a maior probabilidade possível. Ou seja, devemos escolher θ de modo a maximizar $L(\theta)$.
- Invés de maximizar $L(\theta)$, também podemos maximizar qualquer função crescente de $L(\theta)$.
- Para tornar o processo mais simples, podemos utilizar o logaritmo da função de probabilidade (*log likelihood*).

Interpretação probabilística

$$\begin{aligned}\ell(\theta) &= \log L(\theta) \\ &= \log \prod_{i=1}^m \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \\ &= \sum_{i=1}^m \log \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \\ &= m \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{\sigma^2} \cdot \frac{1}{2} \sum_{i=1}^m (y^{(i)} - \theta^T x^{(i)})^2\end{aligned}$$

- Portanto, maximizar $\ell(\theta)$ dá a mesma resposta que minimizar a equação abaixo. Podemos reconhecer-la como a equação $J(\theta)$, a nossa função de custo de mínimos quadrados original.

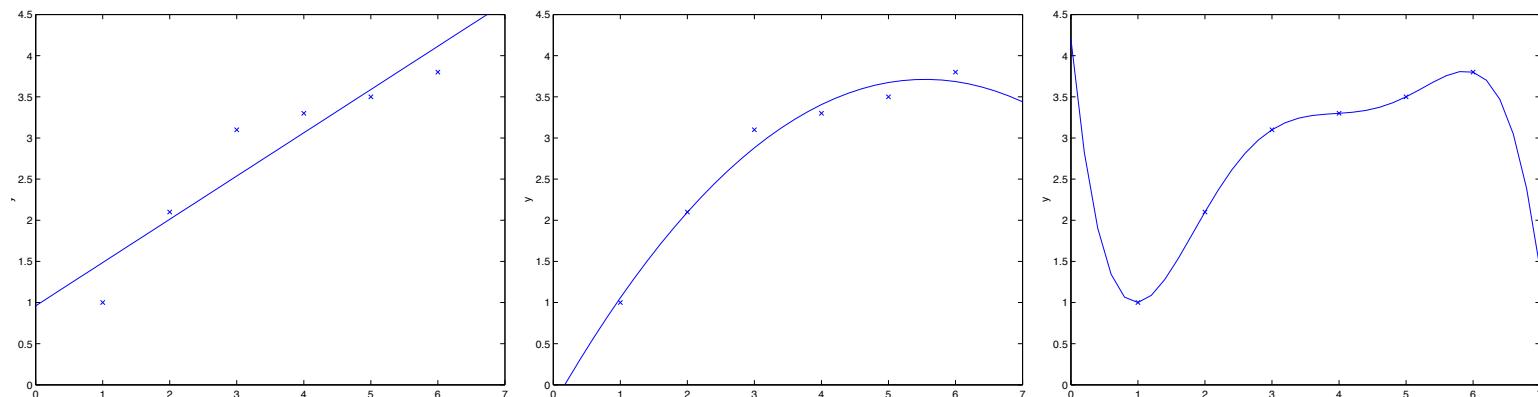
$$\frac{1}{2} \sum_{i=1}^m (y^{(i)} - \theta^T x^{(i)})^2$$

Interpretação probabilística

- Portanto, considerando as premissas probabilísticas feitas em relação aos dados, o método dos mínimos quadrados corresponde à achar a estimativa com máxima probabilidade de θ .
- Podemos notar também que nossa escolha final de θ não depende do valor de σ^2 .

Regressão linear ponderada

- Considere um problema que queremos prever y a partir de x . A figura à esquerda mostra o resultado de uma regressão utilizando $y = \theta_0 + \theta_1x$. Podemos ver que os dados não podem ser perfeitamente representados por uma reta.
- Se adicionarmos o termo x^2 e utilizarmos $y = \theta_0 + \theta_1x + \theta_2x^2$, podemos obter uma melhor representação em relação aos dados (figura central).



Regressão linear ponderada

- Ingenuamente, pode parecer que quanto maior o grau do nosso modelo, melhor. No entanto, isso nem sempre é verdade.
- Por exemplo, a figura mais à direita representa um modelo polinomial de ordem 5. Podemos observar que, apesar da curva passar perfeitamente pelos dados, ele não se comportaria muito bem para prever, por exemplo, preços de casas (y) a partir dos valores das áreas (x).
- Podemos considerar a figura mais à esquerda como um exemplo de *underfitting*, onde claramente os dados mostram uma estrutura não capturada pelo modelo. A figura mais à direita é um exemplo de *overfitting*.

Regressão linear ponderada

- Podemos utilizar o algoritmo de regressão linear ponderada onde, assumindo que têm dados de treinamento suficientes, torna a escolha da ordem do modelo menos crítica.
- No algoritmo original de regressão linear, para realizar uma predição, seguiríamos os seguintes passos:
 1. Fit θ to minimize $\sum_i (y^{(i)} - \theta^T x^{(i)})^2$
 2. Output $\theta^T x$.
- Em contraste, o algoritmo de regressão linear ponderada faz o seguinte:
 1. Fit θ to minimize $\sum_i w^{(i)} (y^{(i)} - \theta^T x^{(i)})^2$
 2. Output $\theta^T x$.

Regressão linear ponderada

- Os valores não-negativos $w^{(i)}$ são chamados de pesos (*weights*). Intuitivamente, se o valor de $w^{(i)}$ for grande para um valor particular de i , então ao escolher θ , tentaremos tornar pequeno o termo $(y^{(i)} - \theta^T x^{(i)})^2$
- Se o valor de $w^{(i)}$ for pequeno, então o fator de erro $(y^{(i)} - \theta^T x^{(i)})^2$ será praticamente ignorado.
- Uma escolha padrão para os pesos é:

$$w^{(i)} = \exp\left(-\frac{(x^{(i)} - x)^2}{2\tau^2}\right)$$

- Note que os valores dos pesos dependem de um ponto particular x no qual queremos avaliar $x^{(i)}$.

Regressão linear ponderada

- Se $|x^{(i)} - x|$ é pequeno, então $w^{(i)}$ tem o valor perto de 1. Se $|x^{(i)} - x|$ é grande, então $w^{(i)}$ é pequeno.
- Portanto, θ é escolhido dando um peso maior aos erros nos exemplos de treinamento perto do ponto particular x .
- O parâmetro τ de largura de banda (bandwidth parameter) controla quão rápido o peso de um exemplo de treinamento diminui baseado na distância de $x^{(i)}$ em relação ao ponto particular x .
- Regressão linear ponderada é o primeiro algoritmo não-paramétrico visto até agora. O algoritmo de regressão linear visto anteriormente é paramétrico porque possui um número finito de parâmetros com valores fixos. Uma vez que achamos os valores de θ e os armazenamos, não precisamos mais dos dados de treinamento para previsões futuras, diferente da regressão linear ponderada.

Classificação e regressão logística

Introdução

- O problema de classificação é basicamente igual ao problema de regressão, exceto que os valores de y que queremos prever são somente um pequeno número de valores discretos.
- Um exemplo seria de classificação binária. Por exemplo, se estamos desenvolvendo um classificador de spam de email, então $x^{(i)}$ pode ser algumas características do email e y pode ser 1 se o email analisado for um spam ou 0 se não for o caso.

Regressão logística

- Podemos abordar o problema de classificação ignorando o fato que y é um valor discreto e usar, por exemplo, o nosso algoritmo de regressão linear para tentar prever y dado x .
- No entanto, há vários exemplos onde esse método funciona muito mal. Intuitivamente, não faz sentido para $h_\theta(x)$ ter valores maiores que 1 ou maiores do que zero se sabemos que $y \in \{0, 1\}$.
- Para corrigir isso, podemos modificar nossa hipótese $h_\theta(x)$:

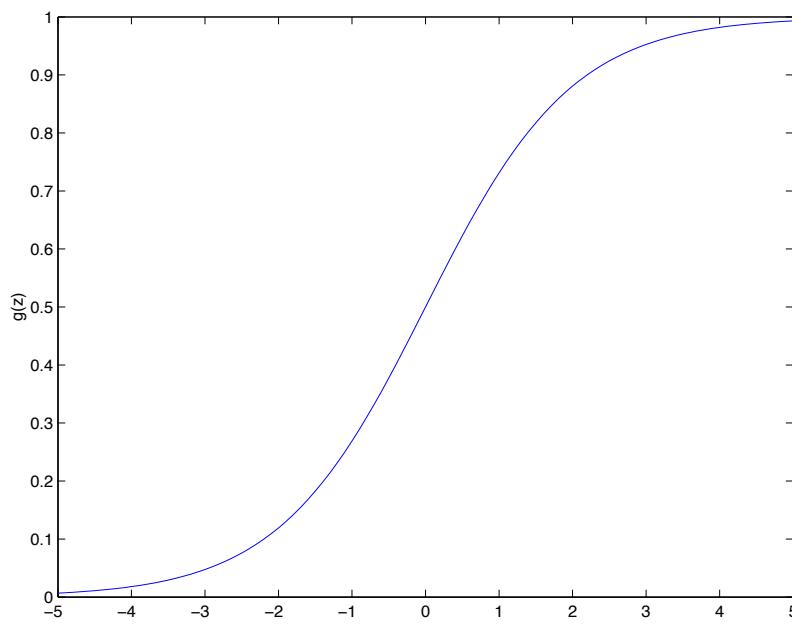
$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

Regressão logística

- Onde

$$g(z) = \frac{1}{1 + e^{-z}}$$

- é chamada de função logística ou função sigmoide.



Regressão logística

- Note que conforme $z \rightarrow \infty$, $g(z)$ tende para 1 e quando $z \rightarrow -\infty$, $g(z)$ tende para 0. Portanto, $g(z)$ está sempre limitado entre 0 e 1.
- Antes de continuarmos, vemos uma importante propriedade da derivada da função sigmoide, que nos será bastante útil.

$$\begin{aligned} g'(z) &= \frac{d}{dz} \frac{1}{1+e^{-z}} \\ &= \frac{1}{(1+e^{-z})^2} (e^{-z}) \\ &= \frac{1}{(1+e^{-z})} \cdot \left(1 - \frac{1}{(1+e^{-z})}\right) \\ &= g(z)(1-g(z)). \end{aligned}$$

Regressão logística

- Então, dado o modelo de regressão logística, como encontramos os valores de θ ?
- Vamos considerar o mesmo caso do modelo de regressão dos mínimos quadrados, onde o encontramos como um estimador da máxima probabilidade (*maximum likelihood*) a partir de algumas premissas.
- Vamos assumir que:

$$\begin{aligned} P(y = 1 \mid x; \theta) &= h_\theta(x) \\ P(y = 0 \mid x; \theta) &= 1 - h_\theta(x) \end{aligned}$$

- Escrevendo de forma mais compacta, temos:

$$p(y \mid x; \theta) = (h_\theta(x))^y (1 - h_\theta(x))^{1-y}$$

Regressão logística

- Considerando que foram gerados m exemplos de treinamento independentes, podemos escrever a probabilidade dos parâmetros como:

$$\begin{aligned} L(\theta) &= p(\vec{y} \mid X; \theta) \\ &= \prod_{i=1}^m p(y^{(i)} \mid x^{(i)}; \theta) \\ &= \prod_{i=1}^m (h_\theta(x^{(i)}))^{y^{(i)}} (1 - h_\theta(x^{(i)}))^{1-y^{(i)}} \end{aligned}$$

- Como anteriormente, é mais fácil maximizar o logaritmo da função acima:

$$\begin{aligned} \ell(\theta) &= \log L(\theta) \\ &= \sum_{i=1}^m y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log(1 - h(x^{(i)})) \end{aligned}$$

Regressão logística

- Como podemos maximizar a probabilidade? Similarmente ao caso de regressão linear, podemos usar o método do gradiente, onde as atualizações são dadas por $\theta := \theta + \alpha \nabla_{\theta} \ell(\theta)$:
- Vamos começar trabalhando somente com um exemplo de treinamento (x, y) :

$$\begin{aligned}\frac{\partial}{\partial \theta_j} \ell(\theta) &= \left(y \frac{1}{g(\theta^T x)} - (1-y) \frac{1}{1-g(\theta^T x)} \right) \frac{\partial}{\partial \theta_j} g(\theta^T x) \\ &= \left(y \frac{1}{g(\theta^T x)} - (1-y) \frac{1}{1-g(\theta^T x)} \right) g(\theta^T x) (1-g(\theta^T x)) \frac{\partial}{\partial \theta_j} \theta^T x \\ &= (y(1-g(\theta^T x)) - (1-y)g(\theta^T x)) x_j \\ &= (y - h_{\theta}(x)) x_j\end{aligned}$$

Regressão logística

- Utilizamos o fato de que $g'(z) = g(z)(1 - g(z))$, gerando o modo incremental do método do gradiente:

$$\theta_j := \theta_j + \alpha (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)}$$

- Podemos ver que o método parece idêntico ao algoritmo LMS; no entanto, não é o mesmo, pois agora $h_\theta(x^{(i)})$ é definido como uma função não linear de $\theta^T x^{(i)}$.

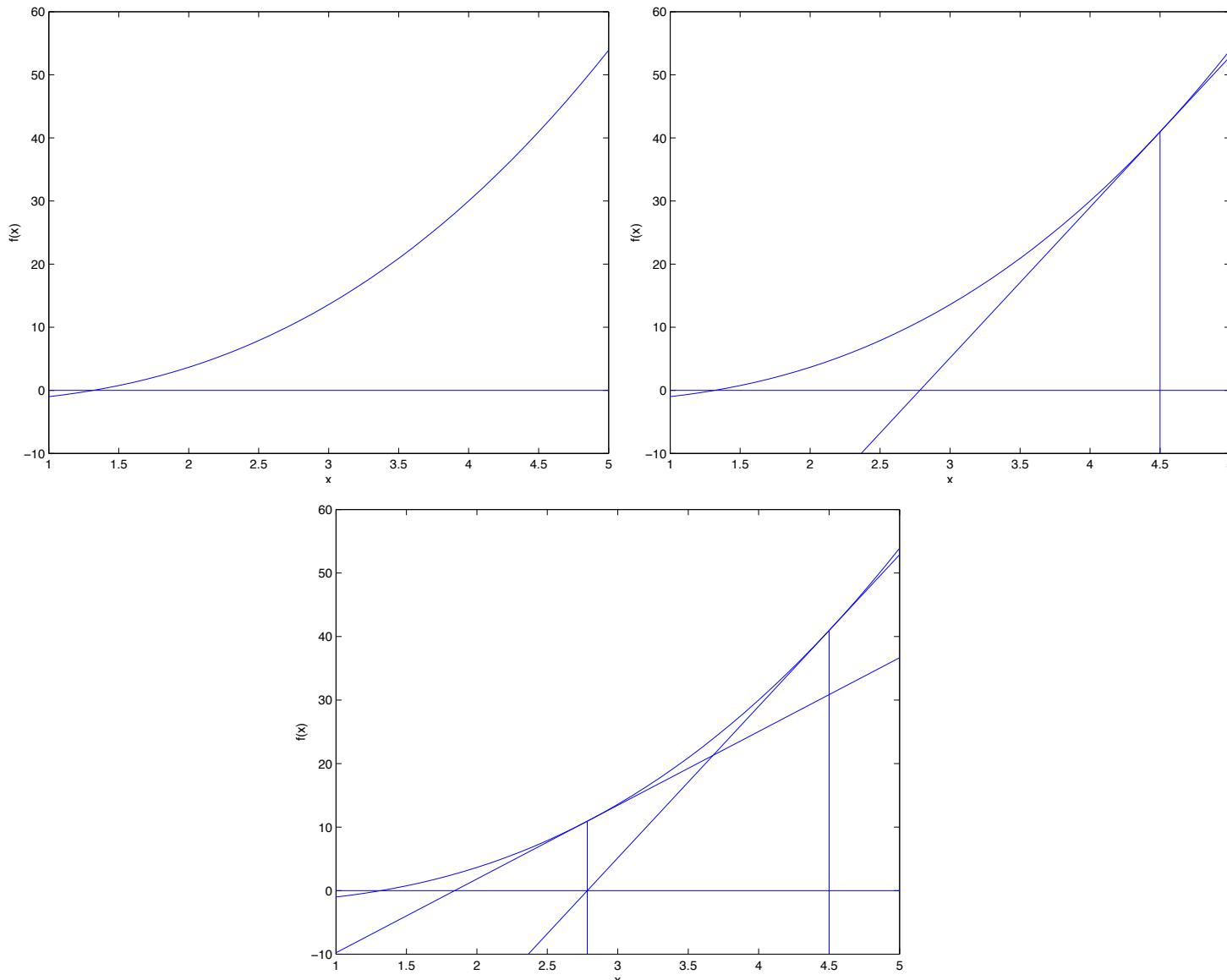
Regressão logística

- Podemos considerar uma outra maneira para minimizar $\ell(\theta)$ utilizando o método de Newton para achar o zero em uma função.
- Especificamente, suponha que temos uma função f e queremos encontrar um valor de θ para que $f(\theta) = 0$, considerando θ um número real.
- O método de Newton utiliza a seguinte atualização:

$$\theta := \theta - \frac{f(\theta)}{f'(\theta)}$$

- Em outras palavras, nós aproximamos a função f através de uma função linear que é tangente à f considerando o valor atual de θ . Resolvendo para que valor a função linear é igual à zero, esse será o novo valor de θ .

Regressão logística



Regressão logística

- O método de Newton fornece uma maneira de encontrar $f(\theta) = 0$. E se quisermos usá-lo para maximizar uma função ℓ ?
- O máximo de ℓ corresponde aos pontos onde sua primeira derivada $\ell'(\theta)$ é zero. Portanto, utilizando $f(\theta) = \ell'(\theta)$, podemos usar o mesmo algoritmo para maximizar ℓ . Abaixo temos a regra de atualização:

$$\theta := \theta - \frac{\ell'(\theta)}{\ell''(\theta)}$$

- Como θ na verdade é um vetor, devemos generalizar o método de Newton de forma multidimensional. Essa generalização é chamada de método de Newton-Raphson e é dado por

$$\theta := \theta - H^{-1} \nabla_{\theta} \ell(\theta)$$

Regressão logística

- O vetor $\nabla_{\theta}\ell(\theta)$ contem as derivadas parciais de $\ell(\theta)$ em relação aos θ_i e \mathbf{H} é uma matriz $n \times n$ chamada de Hessiana e consiste em:

$$H_{ij} = \frac{\partial^2 \ell(\theta)}{\partial \theta_i \partial \theta_j}$$

- O método de Newton tipicamente converge mais rapidamente e com menos iterações do que o método *batch gradient descent* visto anteriormente. No entanto, uma iteração do método de Newton exige bem mais cálculos, pois temos que encontrar a hessiana e calcular a sua inversa.
- Quando o método de Newton é utilizado para maximizar a função logarítmica de probabilidade de regressão logística $\ell(\theta)$ chamamos o método de *Fisher scoring*.

Exercício

- Queremos implementar uma versão ponderada da regressão logística, onde damos pesos à diferentes exemplos de treinamento de acordo com nosso ponto de análise.
- O problema de regressão logística ponderada consiste em maximizar a função abaixo:

$$\ell(\theta) = -\frac{\lambda}{2} \theta^T \theta + \sum_{i=1}^m w^{(i)} \left[y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right]$$

- O primeiro termo do lado direito da igualdade é conhecido como parâmetro de regularização. Para esse problema, utilizaremos um valor fixo de $\lambda = 0,0001$.

Exercício

- Usando essa definição, o gradiente de $\ell(\theta)$ é dado por

$$\nabla_{\theta} \ell(\theta) = X^T z - \lambda \theta$$

- onde z é definido como

$$z_i = w^{(i)}(y^{(i)} - h_{\theta}(x^{(i)}))$$

- e a hessiana é dada por

$$H = X^T D X - \lambda I$$

- onde D é uma matriz diagonal $m \times m$ com:

$$D_{ii} = -w^{(i)} h_{\theta}(x^{(i)}) (1 - h_{\theta}(x^{(i)}))$$

Exercício

- Dado um ponto de análise x , calculamos os pesos como:

$$w^{(i)} = \exp\left(-\frac{\|x - x^{(i)}\|^2}{2\tau^2}\right)$$

1. Utilize o método de Newton-Raphson para otimizar $\ell(\theta)$ para um novo ponto de análise x , e use-o para prever a classe de x . Deverá ser implementada a função no arquivo *lwlrm*, o qual deve, basicamente, calcular os pesos $w^{(i)}$ para cada exemplo de treinamento; maximizar $\ell(\theta)$ usando o método de Newton e gerar a saída $y = 1\{h_\theta(x) > 0,5\}$ como predição.
2. Avalie o sistema utilizando diferentes valores de τ . Particularmente, utilize $\tau = 0.01, 0.05, 0.1, 0.5, 1.0, 5.0$. Como se modifica o limite de classificação ao se modificar esse parâmetro?

Modelos lineares generalizados

Introdução

- Os métodos vistos até agora são casos especiais de uma família mais ampla de modelos, chamados de modelos lineares generalizados.
- Veremos como outros modelos pertencentes à essa família podem ser encontrados e aplicados à outros problemas de classificação e regressão.

Família exponencial

- Vamos começar definindo as distribuições da família exponencial. Podemos dizer que uma classe de distribuições está na família exponencial se puder ser escrita da seguinte forma:

$$p(y; \eta) = b(y) \exp(\eta^T T(y) - a(\eta))$$

- Chamamos η de parâmetro natural (ou canônico) da distribuição. Valores fixos de T , a e b define uma família de distribuições que é parametrizada por η . Conforme variamos η , encontramos diferentes distribuições dentro dessa família.
- Por exemplo, a distribuição de Bernoulli com média ϕ especifica a distribuição sobre $y \in \{0,1\}$, onde $p(y = 1; \phi) = \phi$ e $p(y = 0; \phi) = 1 - \phi$.

Família exponencial

- Conforme variamos ϕ , obtemos distribuições de Bernoulli com diferentes médias. Podemos mostrar que essa classe pertence à família exponencial:

$$\begin{aligned} p(y; \phi) &= \phi^y (1 - \phi)^{1-y} \\ &= \exp(y \log \phi + (1 - y) \log(1 - \phi)) \\ &= \exp\left(\left(\log\left(\frac{\phi}{1 - \phi}\right)\right)y + \log(1 - \phi)\right) \end{aligned}$$

- Portanto, o parâmetro natural é dado por $\eta = \log(\phi/(1 - \phi))$. Além disso, temos:

$$\begin{aligned} T(y) &= y \\ a(\eta) &= -\log(1 - \phi) \\ &= \log(1 + e^\eta) \\ b(y) &= 1 \end{aligned}$$

Família exponencial

- Vamos fazer o mesmo processo em relação à distribuição gaussiana. Lembrando que quando encontramos a regressão linear a partir dela, o valor de σ^2 não tinha efeito sobre a nossa escolha final de θ e $h_\theta(x)$. Portanto, vamos escolher $\sigma^2 = 1$ afim de simplificar os nossos cálculos. Então temos:

$$\begin{aligned} p(y; \mu) &= \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}(y - \mu)^2\right) \\ &= \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}y^2\right) \cdot \exp\left(\mu y - \frac{1}{2}\mu^2\right) \end{aligned}$$

$$\begin{aligned} \eta &= \mu \\ T(y) &= y \\ a(\eta) &= \mu^2/2 \\ &= \eta^2/2 \\ b(y) &= (1/\sqrt{2\pi}) \exp(-y^2/2) \end{aligned}$$

Construindo modelos lineares generalizados

- Vamos considerar um problema de classificação ou regressão onde gostaríamos de prever o valor de uma variável aleatória y em função de x .
- Para calcular um modelo linear generalizado para esse problema, faremos três premissas sobre a distribuição condicional de y dado x e sobre nosso modelo:
 1. Dado x e θ , a distribuição de y segue alguma distribuição da família exponencial com parâmetro η .
 2. Dado x , nosso objetivo é prever o valor esperado de $T(y)$. Na maioria dos nossos exemplos, teremos $T(y) = y$. Isso significa que gostaríamos que nossa predição $h(x)$ satisfaça $h(x) = E[y|x]$. Podemos notar que essa premissa é satisfeita, por exemplo, no caso da regressão logística, onde tínhamos que $h_\theta(x) = p(y = 1|x; \theta) = 0. p(y = 0|x; \theta) + 1.p(y = 1|x; \theta) = E[y|x; \theta]$.
 3. O parâmetro natural η e as entradas x são linearmente relacionadas: $\eta = \theta^T x$.

Construindo modelos lineares generalizados

- Essas três premissas permitirão o cálculo dos modelos lineares generalizados, uma classe de algoritmos de aprendizagem que possui várias propriedades desejáveis, como a facilidade no aprendizado.
- Além disso, os modelos resultantes são geralmente bastante efetivos na modelagem de diferentes tipos de distribuições de y . Por exemplo, regressão logística e os mínimos quadrados ambas podem ser derivadas como modelos lineares generalizados.

Construindo modelos lineares generalizados

- Primeiramente, vamos mostrar que o método dos mínimos quadrados é um caso especial da família de modelos lineares generalizados.
- Considere que uma variável y , a qual queremos prever, é contínua e que modelamos a distribuição condicional de y dado x como uma distribuição gaussiana.
- Como vimos previamente, formulando a distribuição gaussiana como parte da família da distribuição exponencial, tínhamos que $\mu = \eta$. Então, temos:

$$\begin{aligned} h_\theta(x) &= E[y|x; \theta] \\ &= \mu \\ &= \eta \\ &= \theta^T x. \end{aligned}$$

Construindo modelos lineares generalizados

- Podemos fazer esses mesmos passos em relação à regressão logística, onde estamos interessados em uma classificação binária onde $y \in \{0, 1\}$.
- Considerando que y tem um valor binário, seria natural escolhermos a distribuição de Bernoulli para modelar a distribuição condicional de y dado x .
- Como vimos previamente, formulando a distribuição de Bernoulli como parte da família da distribuição exponencial, tínhamos que $\phi = 1/(1 + e^{-\eta})$. Então, seguindo a mesma derivação dos mínimos quadrados, temos:

Construindo modelos lineares generalizados

$$\begin{aligned} h_{\theta}(x) &= E[y|x; \theta] \\ &= \phi \\ &= 1/(1 + e^{-\eta}) \\ &= 1/(1 + e^{-\theta^T x}) \end{aligned}$$

- Podemos observar que $h_{\theta}(x)$ é a função logística. Uma vez que assumimos que y condicionado à x é Bernoulli, isto surge como consequência da definição dos modelos lineares generalizados e das distribuições da família exponencial.

Regressão Softmax

- Vamos observar mais um exemplo de modelo linear generalizado. Vamos considerar um problema de classificação em que a variável y pode ser qualquer valor k , onde $y \in \{1, 2, \dots, k\}$.
- Por exemplo, em vez de classificarmos um email somente entre duas classes - spam ou não spam - queremos agora classificar em três classes, como spam, email pessoal e email de trabalho.
- A variável de resposta continua sendo discreta, mas agora pode ter mais do que dois valores. Portanto, vamos modelá-la como uma distribuição multinomial.

Regressão Softmax

- Vamos derivar um modelo linear generalizado para esse tipo de dados, primeiramente expressando a distribuição multinomial como uma distribuição da família exponencial.
- Para parametrizar uma multinomial com k possíveis saídas, precisamos de $k-1$ parâmetros $\phi_1, \dots, \phi_{k-1}$, onde $\phi_i = p(y=i; \phi)$ e $p(y=k; \phi) = 1 - \sum_{i=1}^{k-1} \phi_i$.
- Para expressar a multinomial como uma distribuição da família exponencial, definimos $T(y) \in \mathcal{R}_{k-1}$ como:

$$T(1) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, T(2) = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, T(3) = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \dots, T(k-1) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}, T(k) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Regressão Softmax

- Diferente dos exemplos anteriores, não temos agora $T(y) = y$. $T(y)$ é um vetor dimensional com $k-1$ elementos e não um número real. Representaremos $(T(y))$ como o elemento i do vetor $T(y)$.
- Finalmente representando a distribuição multinomial como membro da família exponencial, temos:

$$\begin{aligned} p(y; \phi) &= \phi_1^{1\{y=1\}} \phi_2^{1\{y=2\}} \cdots \phi_k^{1\{y=k\}} \\ &= \phi_1^{1\{y=1\}} \phi_2^{1\{y=2\}} \cdots \phi_k^{1-\sum_{i=1}^{k-1} 1\{y=i\}} \\ &= \phi_1^{(T(y))_1} \phi_2^{(T(y))_2} \cdots \phi_k^{1-\sum_{i=1}^{k-1} (T(y))_i} \\ &= \exp((T(y))_1 \log(\phi_1) + (T(y))_2 \log(\phi_2) + \\ &\quad \cdots + \left(1 - \sum_{i=1}^{k-1} (T(y))_i\right) \log(\phi_k)) \\ &= \exp((T(y))_1 \log(\phi_1/\phi_k) + (T(y))_2 \log(\phi_2/\phi_k) + \\ &\quad \cdots + (T(y))_{k-1} \log(\phi_{k-1}/\phi_k) + \log(\phi_k)) \\ &= b(y) \exp(\eta^T T(y) - a(\eta)) \end{aligned}$$

Regressão Softmax

- Onde:

$$\begin{aligned}\eta &= \begin{bmatrix} \log(\phi_1/\phi_k) \\ \log(\phi_2/\phi_k) \\ \vdots \\ \log(\phi_{k-1}/\phi_k) \end{bmatrix} \\ a(\eta) &= -\log(\phi_k) \\ b(y) &= 1.\end{aligned}$$

- Podemos então representar η_i como:

$$\eta_i = \log \frac{\phi_i}{\phi_k}.$$

- Por conveniência, também definimos $\eta_k = \log(\phi_i/\phi_k) = 0$. Invertendo a função acima para encontrarmos a função de resposta, temos:

$$\begin{aligned}e^{\eta_i} &= \frac{\phi_i}{\phi_k} \\ \phi_k e^{\eta_i} &= \phi_i \\ \phi_k \sum_{i=1}^k e^{\eta_i} &= \sum_{i=1}^k \phi_i = 1\end{aligned}$$

Regressão Softmax

- Isso implica que $\phi_k = 1 / \sum_{i=1}^k e^{\eta_i}$, a qual podemos substituir na equação anterior para encontrarmos a função de resposta. Essa função mapeando ϕ 's a partir dos η 's é chamada de função **softmax**.

$$\phi_i = \frac{e^{\eta_i}}{\sum_{j=1}^k e^{\eta_j}}$$

- Como discutido anteriormente, sabemos que os η 's são linearmente relacionados aos x 's. Então, $\eta_i = \theta_i^T x$ (para $i = 1, \dots, k-1$), onde $\theta_1, \dots, \theta_{k-1} \in \mathcal{R}^{n+1}$ são os parâmetros do nosso modelo.
- Novamente por conveniência, vamos definir $\theta_k = 0$, e então $\eta_k = \theta_k^T x = 0$

Regressão Softmax

- Portanto, nosso modelo assume que a distribuição condicional de y dado x é dado por:

$$\begin{aligned} p(y = i|x; \theta) &= \phi_i \\ &= \frac{e^{\eta_i}}{\sum_{j=1}^k e^{\eta_j}} \\ &= \frac{e^{\theta_i^T x}}{\sum_{j=1}^k e^{\theta_j^T x}} \end{aligned}$$

- Esse modelo, aplicável à problemas de classificação onde $y \in \{1, \dots, k\}$, é chamado de regressão **softmax**, o qual é uma generalização da regressão logística.
- Então nossa função $h_\theta(x)$ será:

Regressão Softmax

$$\begin{aligned} h_{\theta}(x) &= \text{E}[T(y)|x; \theta] \\ &= \text{E} \left[\begin{array}{c|c} 1\{y = 1\} & \\ 1\{y = 2\} & \\ \vdots & \\ 1\{y = k - 1\} & \end{array} \middle| x; \theta \right] \\ &= \begin{bmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_{k-1} \end{bmatrix} \\ &= \begin{bmatrix} \frac{\exp(\theta_1^T x)}{\sum_{j=1}^k \exp(\theta_j^T x)} \\ \frac{\exp(\theta_2^T x)}{\sum_{j=1}^k \exp(\theta_j^T x)} \\ \vdots \\ \frac{\exp(\theta_{k-1}^T x)}{\sum_{j=1}^k \exp(\theta_j^T x)} \end{bmatrix}. \end{aligned}$$

Regressão Softmax

- Em outras palavras, nossa função nos dará a probabilidade de $p(y = i|x;\theta)$ para qualquer valor de $i = 1, \dots, k$.
- Observamos que $h_{\theta}(x)$ como definida tem dimensão de $k-1$ valores. No entanto, $p(y = k|x;\theta)$ pode ser obtida como:

$$1 - \sum_{i=1}^{k-1} \phi_i$$

- Similarmente aos casos dos mínimos quadrados e de regressão logística vistos anteriormente, se temos um set de treinamento com m exemplos $\{(x^{(i)}, y^{(i)}); i = 1, \dots, m\}$ e gostaríamos de aprender os parâmetros θ_i desse modelo, podemos começar representando a probabilidade logarítmica:

Regressão Softmax

$$\begin{aligned}\ell(\theta) &= \sum_{i=1}^m \log p(y^{(i)} | x^{(i)}; \theta) \\ &= \sum_{i=1}^m \log \prod_{l=1}^k \left(\frac{e^{\theta_l^T x^{(i)}}}{\sum_{j=1}^k e^{\theta_j^T x^{(i)}}} \right)^{1\{y^{(i)}=l\}}\end{aligned}$$

- Podemos agora obter a estimativa da probabilidade máxima dos parâmetros maximizando $\ell(\theta)$ em termos de θ , usando métodos como o do gradiente ou o método de Newton.

Algoritmos de aprendizagem generativos

Introdução

- Considere um problema de classificação no qual queremos aprender a distinguir elefantes ($y = 1$) e cachorros ($y = 0$), baseado em algumas características de um animal.
- Dado um set de treinamento, um algoritmo como regressão logística basicamente tenta encontrar uma linha reta - em outras palavras, um limiar de decisão - que separa elefantes e cachorros.
- Então, para classificar um novo animal como elefante ou cachorro, observamos em qual dos lados do limiar de decisão ele se encontra, e fazemos a predição de acordo.
- Veremos agora uma forma diferente de abordar o problema. Primeiramente, focando em elefantes, podemos desenvolver um modelo da aparência dos elefantes. Então, focando nos cachorros, podemos desenvolver um modelo separado da aparência deles.

Introdução

- Finalmente, para classificar um novo animal, podemos comparar o mesmo com o modelo do elefante e com o modelo do cachorro para conferir com qual deles ele é mais semelhante.
- Algoritmos que tentam aprender diretamente a distribuição condicional de y dado x ($p(y|x)$), como a regressão logística, ou que tentam mapear diretamente do espaço de entradas para $\{0,1\}$ são chamados de algoritmos de aprendizagem discriminativos.
- Agora falaremos sobre algoritmos que, ao invés de modelar $p(y|x)$, iremos modelar $p(x|y)$ e $p(y)$. Esses algoritmos são chamados de algoritmos de aprendizagem generativos.
- Por exemplo, se y indica se um animal é um cachorro (0) ou um elefante (1), então $p(x|y = 0)$ modela a distribuição das características dos cachorros e $p(x|y = 1)$ modela a distribuição das características dos elefantes.

Introdução

- Após modelarmos $p(y)$ - chamado de probabilidade a priori - e $p(x|y)$, nosso algoritmo pode utilizar a regra de Bayes para encontrarmos a probabilidade a posteriori de y dado x :

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)}$$

- O denominador é dado por $p(x) = p(x|y=1)p(y=1) + p(x|y=0)p(y=0)$ e, portanto, pode ser expresso em termos de $p(x|y)$ e $p(y)$, as quais aprendemos.
- Na verdade, se tivermos calculando $p(y|x)$ para fazermos uma previsão, então não precisamos realmente calcular o denominador, pois

$$\begin{aligned} \arg \max_y p(y|x) &= \arg \max_y \frac{p(x|y)p(y)}{p(x)} \\ &= \arg \max_y p(x|y)p(y). \end{aligned}$$

Análise Discriminante Gaussiana

- O primeiro algoritmo de aprendizagem generativo que iremos analisar é a análise discriminante Gaussiana (GDA).
- Assumiremos que $p(x|y)$ é distribuído como uma distribuição normal multivariada, a qual discutiremos antes de vermos o GDA propriamente dito.

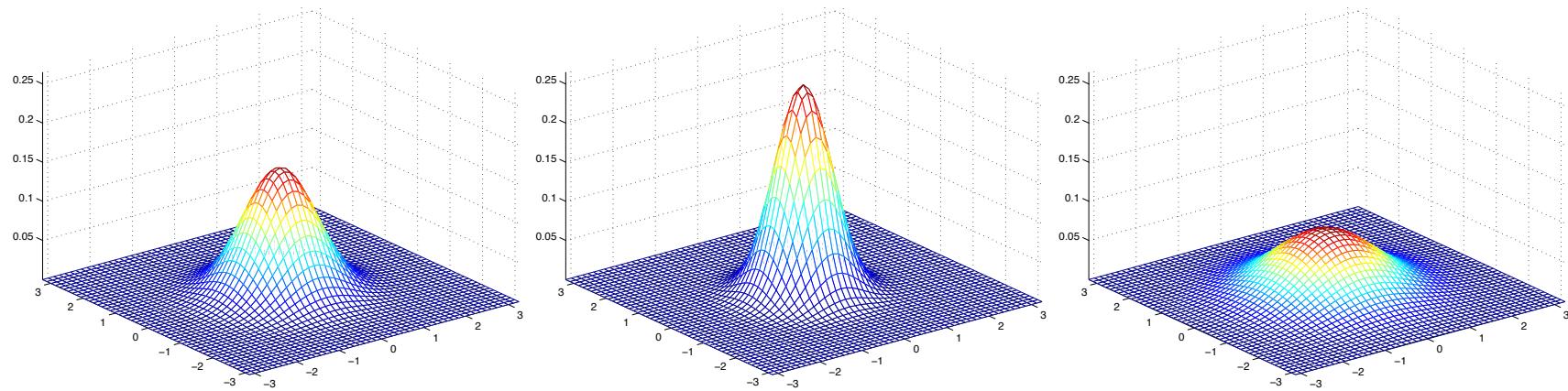
Distribuição normal multivariada

- A distribuição normal multivariada em n dimensões é parametrizada pelo um vetor de médias $\mu \in \mathcal{R}^n$ e uma matriz de covariância $\Sigma \in \mathcal{R}^{n \times n}$, onde $\Sigma \geq 0$ é simétrica e positiva semi-definida.
- Também escrita como $\mathcal{N}(\mu, \Sigma)$, sua densidade é dada abaixo, onde $|\Sigma|$ representa o determinante da matriz Σ .

$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x - \mu)^T \Sigma^{-1} (x - \mu)\right)$$

- A seguir temos alguns exemplos da densidade de uma distribuição Gaussiana:

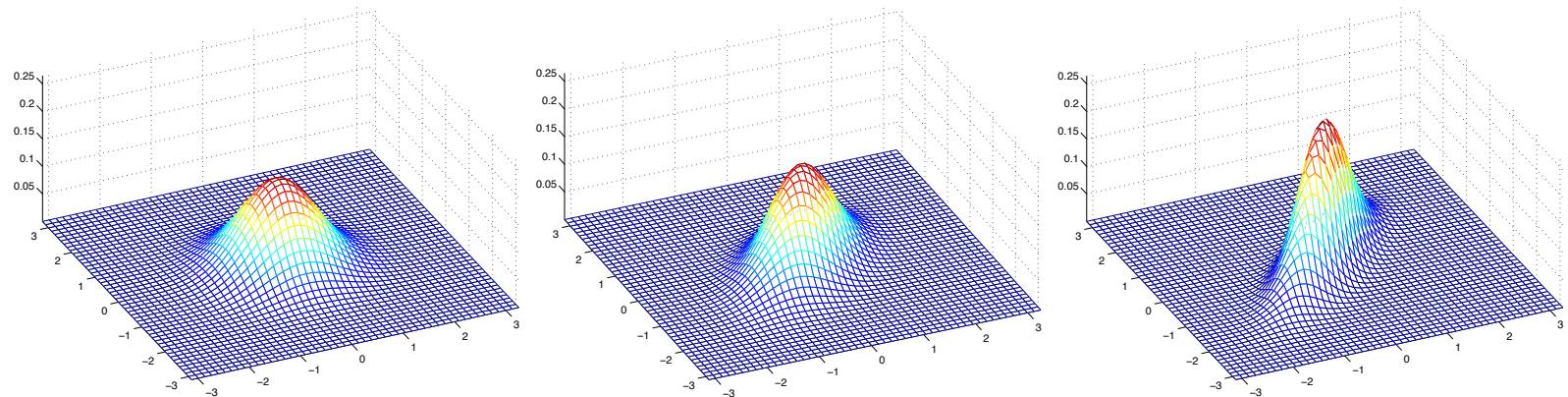
Distribuição normal multivariada



- A figura à esquerda mostra a Gaussiana com média zero (ou seja, um vetor de zeros 2×1) e uma matriz de covariância $\Sigma = I$ (matriz identidade 2×2). Uma Gaussiana com média zero e covariância como uma identidade é chamada de distribuição normal padrão.
- A figura central mostra a densidade de uma Gaussiana com média zero e $\Sigma = 0,6I$ e a figura à direita com $\Sigma = 2I$.

Distribuição normal multivariada

- Podemos observar que quanto maior Σ , a Gaussiana se torna mais “espalhada”, e quanto menor o valor de Σ , a distribuição se torna mais compacta.

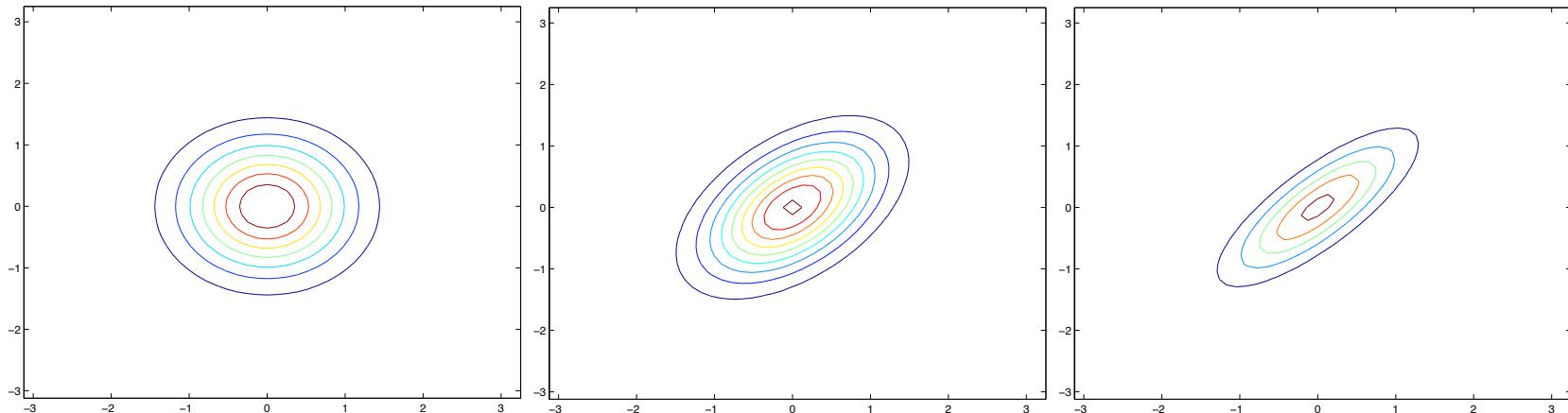


- As figuras acima mostram Gaussianas com média zero e as respectivas matrizes de covariância a seguir:

$$\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}; \quad \Sigma = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}; \quad \Sigma = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix}$$

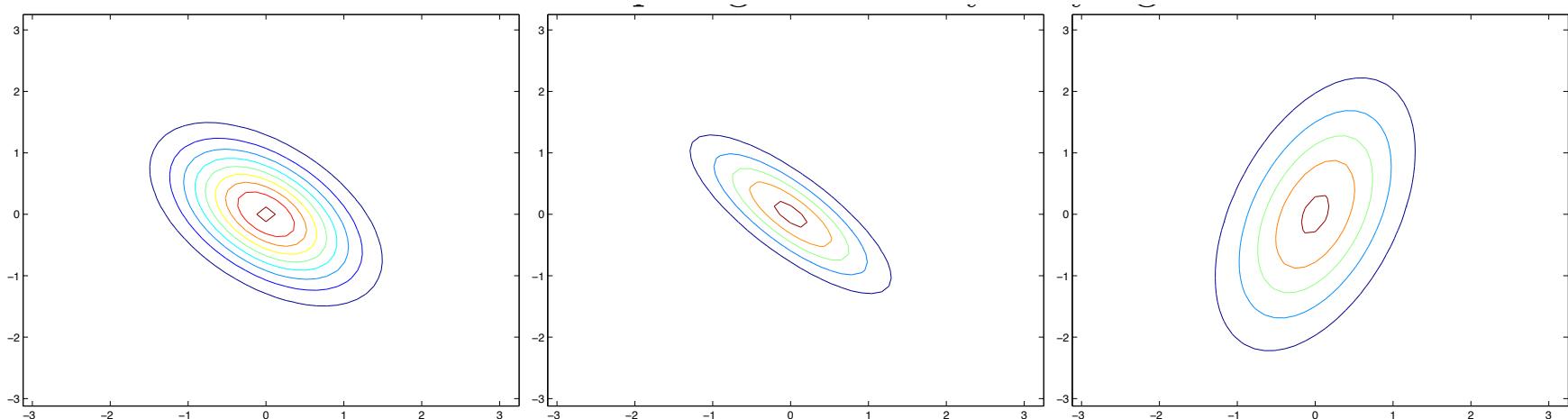
Distribuição normal multivariada

- A figura mais à esquerda mostra a distribuição normal padrão. Podemos observar que ao modificarmos a diagonal secundária, a densidade se torna mais comprimida em relação à linha de 45 graus.
- Podemos observar isso mais claramente olhando os contornos das mesmas três densidades:



Distribuição normal multivariada

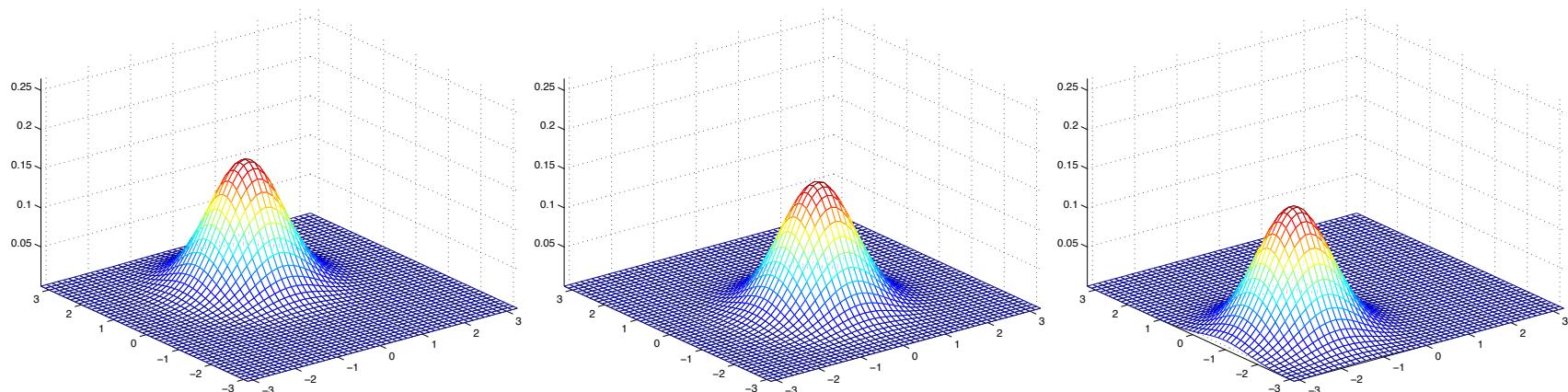
- Abaixo temos mais exemplos do contorno da Gaussiana e suas respectivas matrizes de covariância:



$$\Sigma = \begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix}; \quad \Sigma = \begin{bmatrix} 1 & -0.8 \\ -0.8 & 1 \end{bmatrix}; \quad .\Sigma = \begin{bmatrix} 3 & 0.8 \\ 0.8 & 1 \end{bmatrix}$$

Distribuição normal multivariada

- Como últimos exemplos, fixamos $\Sigma = I$ e variamos a média μ :



$$\mu = \begin{bmatrix} 1 \\ 0 \end{bmatrix}; \quad \mu = \begin{bmatrix} -0.5 \\ 0 \end{bmatrix}; \quad \mu = \begin{bmatrix} -1 \\ -1.5 \end{bmatrix}$$

Modelo de Análise Discriminante Gaussiana

- Quando temos um problema de classificação em que as características de entrada x variáveis aleatórias contínuas, podemos usar o modelo de análise discriminante Gaussiana (GDA), o qual modela $p(x|y)$ usando uma distribuição normal multivariada. O modelo é:

$$\begin{aligned} y &\sim \text{Bernoulli}(\phi) \\ x|y=0 &\sim \mathcal{N}(\mu_0, \Sigma) \\ x|y=1 &\sim \mathcal{N}(\mu_1, \Sigma) \end{aligned}$$

- Escrevendo as distribuições, temos:

$$\begin{aligned} p(y) &= \phi^y(1-\phi)^{1-y} \\ p(x|y=0) &= \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x - \mu_0)^T \Sigma^{-1} (x - \mu_0)\right) \\ p(x|y=1) &= \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x - \mu_1)^T \Sigma^{-1} (x - \mu_1)\right) \end{aligned}$$

Modelo de Análise Discriminante Gaussiana

- Os parâmetros do nosso modelo são ϕ , Σ , μ_0 e μ_1 . Note que temos dois vetores de média diferentes e somente uma matriz de covariância.
- A probabilidade logarítmica dos dados é dada por:

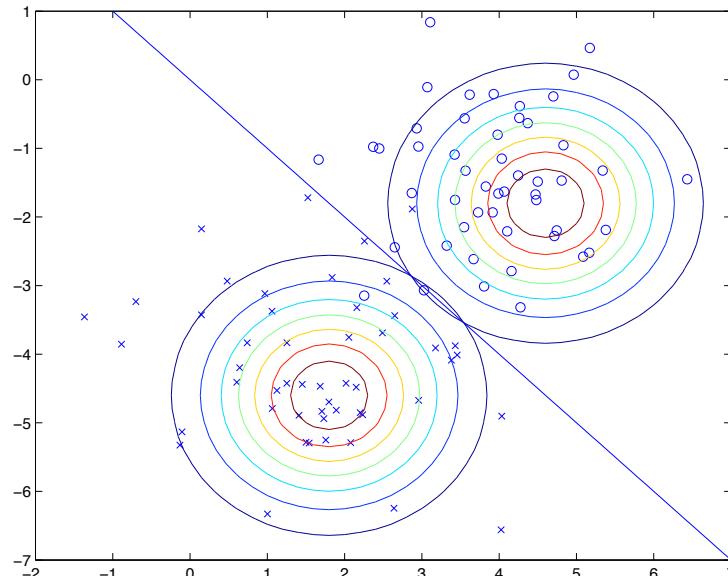
$$\begin{aligned}\ell(\phi, \mu_0, \mu_1, \Sigma) &= \log \prod_{i=1}^m p(x^{(i)}, y^{(i)}; \phi, \mu_0, \mu_1, \Sigma) \\ &= \log \prod_{i=1}^m p(x^{(i)}|y^{(i)}; \mu_0, \mu_1, \Sigma) p(y^{(i)}; \phi)\end{aligned}$$

- Maximizando ℓ em relação aos parâmetros, achamos a estimativa da máxima probabilidade dos parâmetros como:

Modelo de Análise Discriminante Gaussiana

$$\begin{aligned}\phi &= \frac{1}{m} \sum_{i=1}^m 1\{y^{(i)} = 1\} \\ \mu_0 &= \frac{\sum_{i=1}^m 1\{y^{(i)} = 0\}x^{(i)}}{\sum_{i=1}^m 1\{y^{(i)} = 0\}} \\ \mu_1 &= \frac{\sum_{i=1}^m 1\{y^{(i)} = 1\}x^{(i)}}{\sum_{i=1}^m 1\{y^{(i)} = 1\}} \\ \Sigma &= \frac{1}{m} \sum_{i=1}^m (x^{(i)} - \mu_{y^{(i)}})(x^{(i)} - \mu_{y^{(i)}})^T\end{aligned}$$

- Pictoricamente, podemos ver o que o algoritmo está fazendo na figura abaixo:

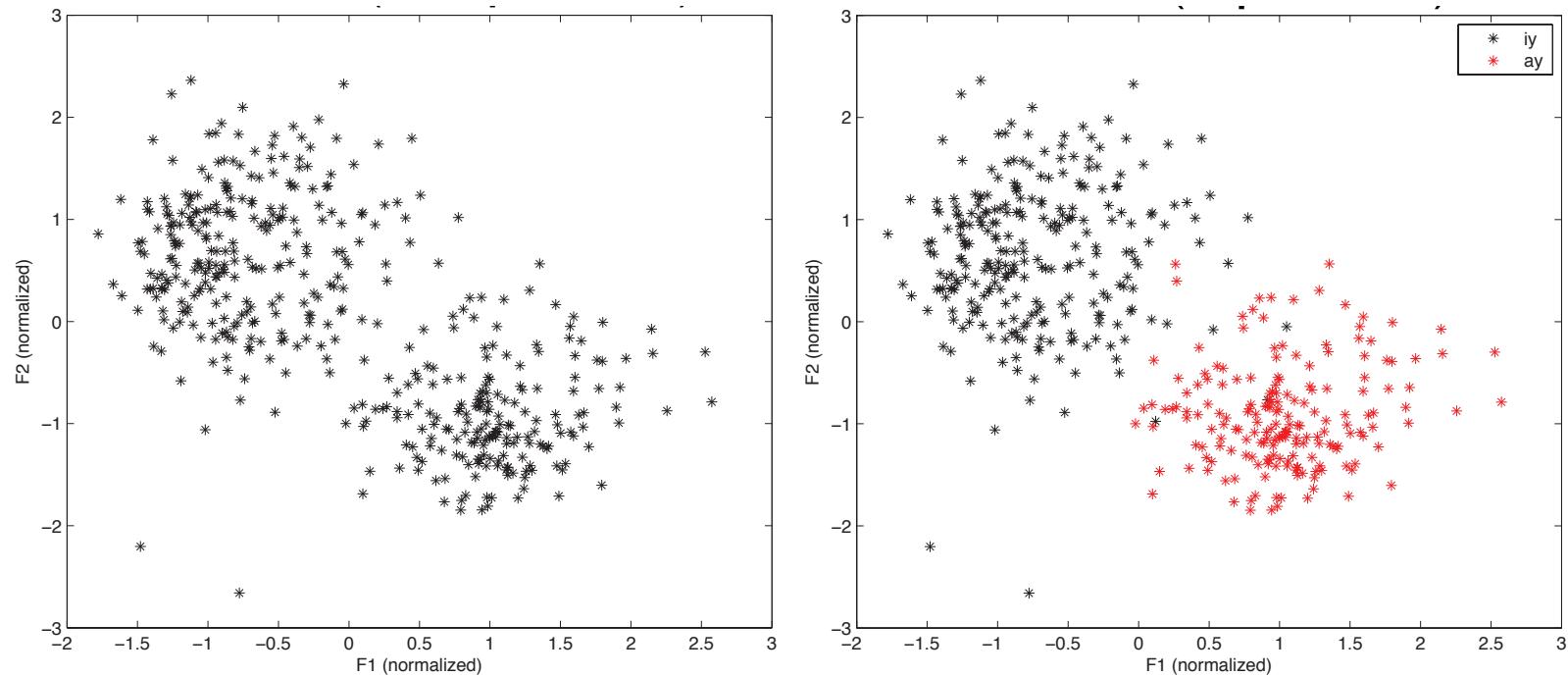


Modelo de Análise Discriminante Gaussiana

- Podemos observar na figura os dados de treinamento, assim como os contornos das duas distribuições Gaussianas para cada uma das duas classes.
- Note que as duas Gaussianas têm contornos com mesmo formato e orientação, já que compartilham a mesma matriz de covariância, mas diferentes vetores de média.
- Também podemos ver na figura a linha reta dando o limite de decisão em que $p(y = 1|x) = 0,5$. Em um lado do limite, prevemos que $y = 1$ é a saída mais provável, e no outro lado, prevemos que $y = 0$.

Exercício

- Os dados disponíveis correspondem aos dois primeiros formantes de duas vogais (/iy/ e /ay/). Os dados estão armazenados na variável D (449 amostras com 2 características cada) e suas respectivas classificações em L (0 para /iy/ e 1 para /ay/).



Exercício

- Represente cada classe como uma distribuição Gaussiana multivariada;
- Encontre a probabilidade de cada ponto sob cada distribuição;
- A partir da probabilidade, escolha a que classe cada um dos pontos pertence, representando-os utilizando diferentes cores no gráfico;
- Desenhe o limite de decisão.

Naive Bayes

- Na análise discriminante gaussiana (GDA), os vetores de entrada x tinham valores contínuos e reais. Vamos falar agora sobre um outro algoritmo de aprendizagem, no qual os x_i 's possuem valores discretos.
- Por exemplo, vamos considerar que queremos construir um filtro de spam para nosso email, classificando-os em emails comerciais (spams) e não-spams.
- Considerando que temos um set de treinamento, vamos começar especificando o vetor x_i , o qual representa as características do email.
- Vamos representar um email por um vetor cujo tamanho é igual ao número de palavras no dicionário. Especificamente, se o email contém a palavra número i do dicionário, então teremos $x_i = 1$; do contrário, temos $x_i = 0$.

Naive Bayes

- Por exemplo, o vetor abaixo é usado para representar um email que contém as palavras “a” e “buy”, mas não as palavras “aardwolf” e “zygmurgy”.

$$x = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \quad \begin{array}{l} \text{a} \\ \text{aardvark} \\ \text{aardwolf} \\ \vdots \\ \text{buy} \\ \vdots \\ \text{zygmurgy} \end{array}$$

- As palavras representadas no vetor x_i são chamadas de vocabulário. Assim, após escolher nosso vetor de características x_i , queremos desenvolver um modelo discriminativo. Portanto, temos que modelar $p(x|y)$.
- No entanto, se temos um vocabulário de 50000 palavras, então $x \in \{0, 1\}^{50000}$ e se modelarmos x como uma distribuição multinomial com 2^{50000} possíveis saídas, teremos um vetor de parâmetros de $(2^{50000} - 1)$ valores.

Naive Bayes

- Para modelarmos $p(x|y)$, faremos uma grande suposição. Assumiremos que x_i 's são condicionalmente independentes dado y . Essa suposição é chamada de Naive Bayes, e o algoritmo resultante é chamado de classificador Naive Bayes.
- Por exemplo, se $y = 1$ significa que o email é um spam, “buy” é a palavra 2087 e “price” é a palavra 39831, estamos assumindo que se $y = 1$, então o conhecimento de x_{2087} não terá efeito algum sobre nossas premissas sobre o valor de x_{39831} .
- Portanto, temos que:

$$\begin{aligned} p(x_1, \dots, x_{50000} | y) &= p(x_1 | y)p(x_2 | y, x_1)p(x_3 | y, x_1, x_2) \cdots p(x_{50000} | y, x_1, \dots, x_{49999}) \\ &= p(x_1 | y)p(x_2 | y)p(x_3 | y) \cdots p(x_{50000} | y) \\ &= \prod_{i=1}^n p(x_i | y) \end{aligned}$$

Naive Bayes

- A segunda igualdade na equação mostra o uso da suposição Naive Bayes. Apesar dela ser uma forte suposição, o algoritmo resultante funciona bem em vários problemas.
- Nosso modelo é parametrizado por $\phi_{i|y=1} = p(x_i = 1|y = 1)$, $\phi_{i|y=0} = p(x_i = 1|y = 0)$ e $\phi_y = p(y = 1)$. Dado um set de treinamento $\{(x^{(i)}, y^{(i)}); i = 1, \dots, m\}$, podemos escrever probabilidade como:

$$\mathcal{L}(\phi_y, \phi_{i|y=0}, \phi_{i|y=1}) = \prod_{i=1}^m p(x^{(i)}, y^{(i)})$$

- Maximizando-o em relação à ϕ_y , $\phi_{i|y=0}$ e $\phi_{i|y=1}$, temos a estimativa da máxima probabilidade:

$$\begin{aligned}\phi_{j|y=1} &= \frac{\sum_{i=1}^m 1\{x_j^{(i)} = 1 \wedge y^{(i)} = 1\}}{\sum_{i=1}^m 1\{y^{(i)} = 1\}} \\ \phi_{j|y=0} &= \frac{\sum_{i=1}^m 1\{x_j^{(i)} = 1 \wedge y^{(i)} = 0\}}{\sum_{i=1}^m 1\{y^{(i)} = 0\}} \\ \phi_y &= \frac{\sum_{i=1}^m 1\{y^{(i)} = 1\}}{m}\end{aligned}$$

Naive Bayes

- Nas equações anteriores, “ \wedge ” significa “and”. Os parâmetros têm uma interpretação natural. Por exemplo, $\phi_{j|y=1}$ é somente a fração dos spams ($y = 1$) nos quais a palavra j aparece.
- Após encontrarmos esses parâmetros, para fazermos uma predição em um novo exemplo x , simplesmente calculamos:

$$\begin{aligned} p(y = 1|x) &= \frac{p(x|y = 1)p(y = 1)}{p(x)} \\ &= \frac{(\prod_{i=1}^n p(x_i|y = 1)) p(y = 1)}{(\prod_{i=1}^n p(x_i|y = 1)) p(y = 1) + (\prod_{i=1}^n p(x_i|y = 0)) p(y = 0)} \end{aligned}$$

- E escolhemos a classe que possui a maior probabilidade a posteriori.
- Note que desenvolvemos o algoritmo Naive Bayes para casos onde x_i assume valores binários.

Naive Bayes

- No entanto, a generalização onde x_i pode assumir valores $\{1, 2, \dots, k_i\}$ é direta. Nesse caso, simplesmente modelaríamos $p(x_i|y)$ como multinomial ao invés de Bernoulli.
- Muitas vezes, mesmo se os atributos de entrada de um problema são contínuos (a área de uma casa, por exemplo), podemos discretizá-los e aplicar Naive Bayes. Por exemplo:

Living area (sq. feet)	< 400	400-800	800-1200	1200-1600	>1600
x_i	1	2	3	4	5

- Portanto, para uma casa com 890 sq. feet de área, usariamos $x_i = 3$. Então, poderíamos aplicar o algoritmo Naive Bayes, e modelar $p(x_i|y)$ com uma distribuição multinomial.
- Quando os atributos originais de valor contínuo não são modelados corretamente por uma distribuição normal multivariada, discretizar esses atributos e usar Naive Bayes ao invés de GDA geralmente nos dará um resultado melhor.

Laplace Smoothing

- O algoritmo Naive Bayes mostrado funciona bem para uma série de problemas, mas uma pequena modificação faz ele melhorar, especialmente para casos de classificação de textos.
- Considerando ainda o problema de classificação de spam, digamos que surja uma nova palavra nos seus emails, a qual não aparecia no set de treinamento utilizado anteriormente.
- Assumindo que essa é a palavra 35000 no dicionário, seu filtro de spam baseado em Naive Bayes estimou a máxima probabilidade dos parâmetros $\phi_{35000|y}$ como

$$\begin{aligned}\phi_{35000|y=1} &= \frac{\sum_{i=1}^m 1\{x_{35000}^{(i)} = 1 \wedge y^{(i)} = 1\}}{\sum_{i=1}^m 1\{y^{(i)} = 1\}} = 0 \\ \phi_{35000|y=0} &= \frac{\sum_{i=1}^m 1\{x_{35000}^{(i)} = 1 \wedge y^{(i)} = 0\}}{\sum_{i=1}^m 1\{y^{(i)} = 0\}} = 0\end{aligned}$$

Laplace Smoothing

- Ou seja, como essa palavra nunca foi vista anteriormente, o algoritmo considera que a probabilidade de vê-la em outro tipo de email é zero.
- Quando tentamos decidir se uma mensagem contendo essa nova palavra é spam, calculamos a probabilidade a posteriori, e obtemos

$$\begin{aligned} p(y = 1|x) &= \frac{\prod_{i=1}^n p(x_i|y = 1)p(y = 1)}{\prod_{i=1}^n p(x_i|y = 1)p(y = 1) + \prod_{i=1}^n p(x_i|y = 0)p(y = 0)} \\ &= \frac{0}{0}. \end{aligned}$$

- Isso acontece porque cada um dos multiplicadores de $p(x_i|y)$ incluem um termo $p(x_{35000}|y) = 0$ multiplicando. Portanto, nosso resultado é 0/0 e nosso algoritmo não sabe como prever.
- Estatisticamente falando, é uma má idéia estimar a probabilidade de um evento como zero somente porque não foi visto anteriormente em um set de treinamento finito.

Laplace Smoothing

- Considerando o problema de estimar a média de uma variável aleatória multinomial z com valores $\{1, \dots, k\}$, podemos parametrizá-la com $\phi_i = p(z = i)$.
- Dado um grupo de m observações independentes $\{z^{(1)}, \dots, z^{(m)}\}$, a estimativa da máxima probabilidade é

$$\phi_j = \frac{\sum_{i=1}^m 1\{z^{(i)} = j\}}{m}$$

- Como visto anteriormente, alguns casos de ϕ_j podem acabar sendo zero, o que é um problema. Para evitá-lo, utilizamos uma técnica chamada Laplace Smoothing, que substitui nossa estimativa por

$$\phi_j = \frac{\sum_{i=1}^m 1\{z^{(i)} = j\} + 1}{m + k}$$

Laplace Smoothing

- Note que adicionamos 1 ao numerador e k ao denominador. A soma dos valores de ϕ_j continua sendo 1, o que é necessário. No entanto, $\phi_j \neq 0$ para todos os valores de j, resolvendo nosso problema anterior.
- Retornando ao nosso classificador Naive Bayes, agora utilizando Laplace Smoothing, obtemos as seguintes estimativas para os parâmetros:

$$\begin{aligned}\phi_{j|y=1} &= \frac{\sum_{i=1}^m 1\{x_j^{(i)} = 1 \wedge y^{(i)} = 1\} + 1}{\sum_{i=1}^m 1\{y^{(i)} = 1\} + 2} \\ \phi_{j|y=0} &= \frac{\sum_{i=1}^m 1\{x_j^{(i)} = 1 \wedge y^{(i)} = 0\} + 1}{\sum_{i=1}^m 1\{y^{(i)} = 0\} + 2}\end{aligned}$$

Modelo para classificação de texto

- No contexto de classificação de texto, Naive Bayes como foi apresentado usa um modelo chamado Modelo de Bernoulli multivariado.
- Nesse modelo, assumimos que a maneira que o email é gerado é aleatória (de acordo com $p(y)$). Portanto, a pessoa que manda o email utiliza o dicionário, decidindo se inclui cada palavra i no email independentemente e de acordo com as probabilidades $p(x_i=1|y) = \phi_{iy}$. Então, a probabilidade de uma mensagem é dada por $p(y) \prod_{i=1}^n p(x_i|y)$.
- Vamos considerar um novo modelo, chamado modelo multinomial. Vamos considerar algumas notações diferentes, como x_i sendo a identidade da palavra i no email. Então, x_i é agora um número inteiro com valores $\{1, \dots, |V|\}$, onde $|V|$ é o tamanho do vocabulário.
- Um email com n palavras é representado por um vetor (x_1, x_2, \dots, x_n) de tamanho n . Note que cada documento pode ter um valor de n diferente.

Modelo para classificação de texto

- No modelo multinomial, assumimos que o email é gerado de forma aleatória, onde se é spam ou não é determinado por $p(y)$ como anteriormente.
- Então o escritor do email o escreve gerando primeiramente x_1 de alguma distribuição multinomial sobre as palavras ($p(x_1|y)$). Após, a segunda palavra é escolhida independentemente de x_1 , mas tirada da mesma distribuição multinomial. O mesmo acontece com x_3 , x_4 etc.
- Então, a probabilidade de uma mensagem é dada por $p(y) \prod_{i=1}^n p(x_i|y)$. Essa fórmula parece a mesma vista anteriormente, mas significa coisas diferentes. Particularmente, $x_i|y$ é agora uma distribuição multinomial ao invés de Bernoulli.

Modelo para classificação de texto

- Os parâmetros do nosso novo modelo são $\phi_y = p(y)$ como anteriormente, $\phi_{i|y=1} = p(x_j = i|y = 1)$ (para qualquer j) e $\phi_{i|y=0} = p(x_j = i|y = 0)$.
- Se temos um set de treinamento $\{(x^{(i)}, y^{(i)}); i = 1, \dots, m\}$, onde $x^{(i)} = (x_1^{(i)}, x_2^{(i)}, \dots, x_{n_i}^{(i)})$, a probabilidade é:

$$\begin{aligned}\mathcal{L}(\phi, \phi_{i|y=0}, \phi_{i|y=1}) &= \prod_{i=1}^m p(x^{(i)}, y^{(i)}) \\ &= \prod_{i=1}^m \left(\prod_{j=1}^{n_i} p(x_j^{(i)}|y; \phi_{i|y=0}, \phi_{i|y=1}) \right) p(y^{(i)}; \phi_y)\end{aligned}$$

Modelo para classificação de texto

- Maximizando a probabilidade, temos:

$$\begin{aligned}\phi_{k|y=1} &= \frac{\sum_{i=1}^m \sum_{j=1}^{n_i} 1\{x_j^{(i)} = k \wedge y^{(i)} = 1\}}{\sum_{i=1}^m 1\{y^{(i)} = 1\} n_i} \\ \phi_{k|y=0} &= \frac{\sum_{i=1}^m \sum_{j=1}^{n_i} 1\{x_j^{(i)} = k \wedge y^{(i)} = 0\}}{\sum_{i=1}^m 1\{y^{(i)} = 0\} n_i} \\ \phi_y &= \frac{\sum_{i=1}^m 1\{y^{(i)} = 1\}}{m}.\end{aligned}$$

- Finalmente, se utilizarmos Laplace Smoothing, teremos:

$$\begin{aligned}\phi_{k|y=1} &= \frac{\sum_{i=1}^m \sum_{j=1}^{n_i} 1\{x_j^{(i)} = k \wedge y^{(i)} = 1\} + 1}{\sum_{i=1}^m 1\{y^{(i)} = 1\} n_i + |V|} \\ \phi_{k|y=0} &= \frac{\sum_{i=1}^m \sum_{j=1}^{n_i} 1\{x_j^{(i)} = k \wedge y^{(i)} = 0\} + 1}{\sum_{i=1}^m 1\{y^{(i)} = 0\} n_i + |V|}\end{aligned}$$

Exercício

- Queremos classificar um email como spam ou não baseado na reincidência de 5 diferentes palavras no email. Os vetor X representa o número de vezes que cada uma dessas palavras aparece nos 1000 emails presente no set de treinamento. O vetor Y representa, para cada um desses emails, se ele é spam (1) ou não (-1).
- Observe que a soma do número de palavras em cada email, contando as 5 classes, sempre é 20.
- Desenvolva um classificador utilizando Naive Bayes e distribuição multinomial para classificar se um email é spam ou não. Após a criação do modelo, utilize o vetor newX para testar o algoritmo criado, comparando o resultado com o vetor newY e estabelecendo a taxa de acerto.

Support Vector Machine

Introdução

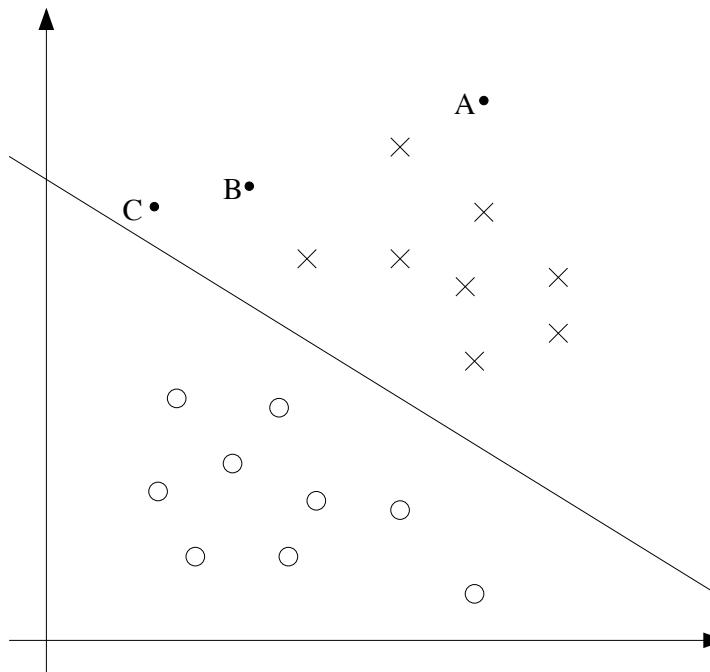
- O algoritmo de aprendizado Support Vector Machine (SVM) é considerado como um dos melhores algoritmos de aprendizado supervisionado.
- Vamos separar SVM em vários tópicos para melhor entendimento. Primeiramente, vamos falar sobre margens e a idéia de separar dados com grande “espaçamento”. Falaremos também sobre um classificador de margens ótimo, kernels - o qual nos permite aplicar SVM eficientemente em um espaço com alta dimensão de características - e o algoritmo SMO (Sequential Minimal Optimization), que nos dá uma implementação eficiente de SVM.

Margens

- Vamos considerar a regressão logística, onde a probabilidade $p(y = 1 | x; \theta)$ é modelado por $h_\theta(x) = g(\theta^T x)$. Iríamos prever “1” se e somente se $h_\theta(x) \geq 0,5$, ou equivalentemente, se e somente se $\theta^T x \geq 0$.
- Considere um exemplo de treinamento onde $y = 1$. Quanto maior o valor de $\theta^T x$, maior também nosso valor $h_\theta(x)$ e, portanto, maior nosso nível de confiança que a saída é realmente 1.
- Informalmente, podemos pensar que a nossa predição é altamente confiável se $\theta^T x \gg 0$ para $y = 1$.
- Da mesma forma, podemos pensar que a nossa predição é altamente confiável se $\theta^T x \ll 0$ para $y = 0$.

Margens

- Portanto, seria um bom objetivo, considerando os dados de treinamento, se achássemos um θ para que $\theta^T x^{(i)} \gg 0$ quando $y^{(i)} = 1$ e $\theta^T x^{(i)} \ll 0$ quando $y^{(i)} = 0$. Formalizaremos essa idéia quando falarmos sobre margens funcionais e geométricas.



Margens

- Antes de continuarmos a discussão sobre margens, vamos introduzir algumas notações sobre classificação.
- Vamos considerar um classificador linear para um problema de classificação binária com saída y e entrada x . A partir de agora, utilizaremos $y \in \{-1, 1\}$ (ao invés de $\{0, 1\}$) para denotar as classes.
- Também, ao invés de parametrizar nosso classificador linear com o vetor θ , usaremos os parâmetros w e b e escreveremos nosso classificador como

$$h_{w,b}(x) = g(w^T x + b)$$

- Então, $g(z) = 1$ se $z \geq 0$ e $g(z) = -1$ caso contrário. Essa notação nos permite tratar o termo b separadamente dos outros parâmetros. Portanto, b realiza o papel que antes era de θ_0 e w assume o papel de $[\theta_1 \dots \theta_n]^T$. Note também que pela nossa definição de g acima, nosso classificador irá prever diretamente 1 ou -1, sem realizar o passo intermediário de estimar a probabilidade de y ser 1 (o que a regressão logística faria).

Margens funcionais e geométricas

- Dado um exemplo de treinamento $(x^{(i)}, y^{(i)})$, definimos margem funcional (functional margin) de (w, b) como

$$\hat{\gamma}^{(i)} = y^{(i)}(w^T x + b)$$

- Se $y^{(i)} = 1$, então para a margem funcional ser grande (isto é, para nossa predição ser confiável e correta) precisamos que $w^T x + b$ seja um número grande e positivo.
- Do mesmo modo, se $y^{(i)} = -1$, então para a margem funcional ser grande, precisamos que $w^T x + b$ seja um número grande e negativo.
- Além disso, se $y^{(i)}(w^T x + b) > 0$, então nossa predição está correta. Portanto, uma grande margem funcional representa um predição correta e confiável.

Margens funcionais e geométricas

- Para um classificador linear com a escolha de g com valores $\{1, -1\}$, há uma propriedade da margem funcional que diminui sua confiabilidade.
- Dada nossa escolha de g , notamos que se substituímos ω por 2ω e b por $2b$, considerando que $g(w^T x + b) = g(2w^T x + 2b)$, isso não mudaria em nada $h_{w,b}(x)$.
- Isto é, g e consequentemente $h_{w,b}(x)$ dependem somente do sinal, mas não da magnitude de $w^T x + b$. No entanto, substituindo ω por 2ω e b por $2b$ também resulta em multiplicar a margem funcional por um fator 2.

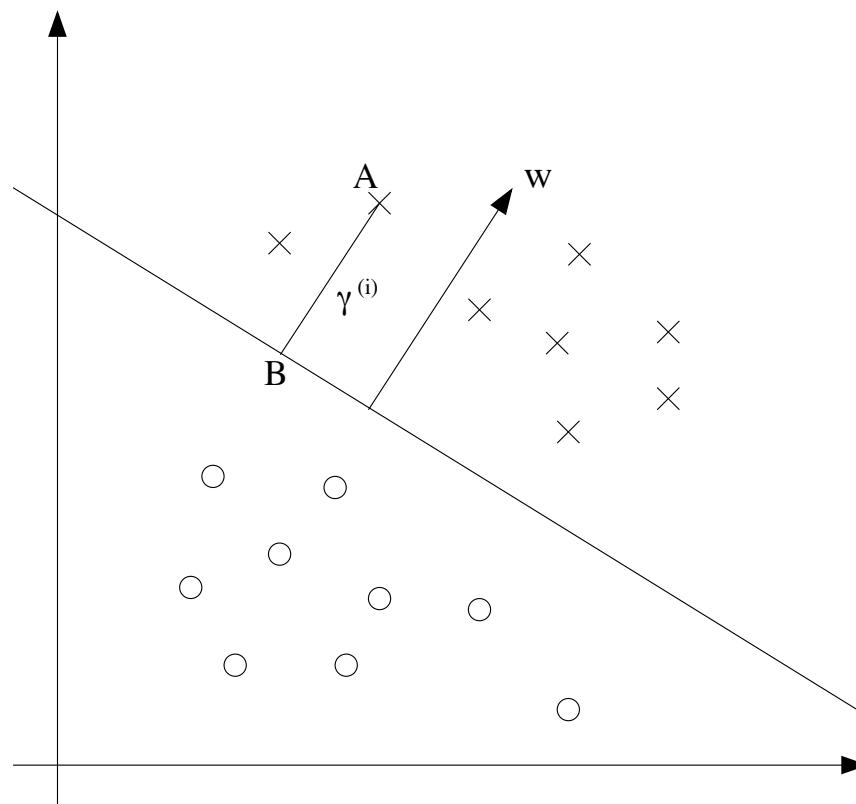
Margens funcionais e geométricas

- Portanto, podemos tornar a margem funcional arbitrariamente grande sem modificar algo significante.
- Intuitivamente, podemos utilizar alguma condição de normalização como $\|w\|_2 = 1$; isto é, substituir (w,b) por $(w/\|w\|_2, b/\|w\|_2)$ e considerarmos a margem funcional de $(w/\|w\|_2, b/\|w\|_2)$.
- Dado um set de treinamento $S = \{(x^{(i)}, y^{(i)}); i = 1, \dots, m\}$, definimos a margem funcional de (w,b) em relação ao menor valor da margem funcional de cada exemplo de treinamento. Portanto:

$$\hat{\gamma} = \min_{i=1, \dots, m} \hat{\gamma}^{(i)}$$

Margens funcionais e geométricas

- Vamos falar agora sobre margens geométricas. Considere a figura abaixo:



Margens funcionais e geométricas

- O limite de decisão em relação à (w, b) é mostrado, assim como o vetor w . Note que w é ortogonal ao plano correspondente ao limite de decisão.
- Considerando o ponto A, o qual representa um exemplo de treinamento $x^{(i)}$ com saída $y^{(i)} = 1$, sua distância do limite de decisão $\gamma^{(i)}$ é dada pelo segmento AB.
- Como podemos encontrar o valor de $\gamma^{(i)}$? Sabemos que $w/\|w\|$ é um vetor unitário com a mesma direção de w . Como A representa $x^{(i)}$, sabemos que o ponto B é dado por $x^{(i)} - \gamma^{(i)} \cdot w/\|w\|$. Como esse ponto se encontra no limiar de decisão, temos que $w^T x + b = 0$. Portanto:

$$w^T \left(x^{(i)} - \gamma^{(i)} \frac{w}{\|w\|} \right) + b = 0$$

Margens funcionais e geométricas

- Isolando $\gamma^{(i)}$, temos:

$$\gamma^{(i)} = \frac{w^T x^{(i)} + b}{\|w\|} = \left(\frac{w}{\|w\|} \right)^T x^{(i)} + \frac{b}{\|w\|}$$

- Esse caso foi trabalhado considerando um exemplo positivo de $y^{(i)}$. Genericamente, definimos margem geométrica de (w, b) em relação a um exemplo de treinamento $(x^{(i)}, y^{(i)})$ como

$$\gamma^{(i)} = y^{(i)} \left(\left(\frac{w}{\|w\|} \right)^T x^{(i)} + \frac{b}{\|w\|} \right)$$

- Note que se $\|w\| = 1$, então a margem funcional e a geométrica se tornam iguais. Além disso, a margem geométrica é invariante à um escalonamento dos parâmetros. Isto é, se substituímos ω por 2ω e b por $2b$, a margem geométrica não é alterada.

Margens funcionais e geométricas

- Por causa dessa invariância ao escalonamento, podemos impor um escalonamento arbitrário em relação à ω sem mudar algo de importante; por exemplo, podemos restringir $\|w\| = 1$, ou $|w_1| = 5$, ou $|w_1+b| + |w_2| = 2$.
- Finalmente, dado um set de treinamento $S = \{(x^{(i)}, y^{(i)}); i = 1, \dots, m\}$, definimos a margem geométrica de (w, b) em relação ao menor valor da margem geométrica de cada exemplo de treinamento. Portanto:

$$\gamma = \min_{i=1, \dots, m} \gamma^{(i)}$$

Classificador de margem máxima

- Dado um set de treinamento, parece uma boa idéia achar um limite de decisão que maximize a margem geométrica, dando confiabilidade ao nosso classificador. Em outras palavras, um classificador que separe os exemplos de treinamento positivos e negativos com um certo “espaço” (margem geométrica).
- Considerando que o set de treinamento é linearmente separável por um hiperplano, como podemos achá-lo de forma a termos a máxima margem geométrica?

$$\begin{aligned} \max_{\gamma, w, b} \quad & \gamma \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq \gamma, \quad i = 1, \dots, m \\ & \|w\| = 1. \end{aligned}$$

- Isto é, queremos maximizar γ , sendo que cada exemplo de treinamento possui uma margem funcional de pelo menos γ . Como $\|w\| = 1$, isso torna a margem funcional igual à geométrica, garantindo que todas as margens geométricas são, no mínimo, γ .

Classificador de margem máxima

- Portanto, precisamos somente resolver esse problema de otimização. No entanto, a restrição $\|w\| = 1$ torna esse problema de difícil solução (não-convexa).
- Vamos então considerar o problema de outra forma:

$$\begin{aligned} \max_{\gamma, w, b} \quad & \frac{\hat{\gamma}}{\|w\|} \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq \hat{\gamma}, \quad i = 1, \dots, m \end{aligned}$$

- Agora temos que maximizar $\hat{\gamma}/\|w\|$, onde as margens funcionais são todas ao menos $\hat{\gamma}$. As margens geométricas e funcionais são relacionadas por $\gamma = \hat{\gamma}/\|w\|$. Além disso, não temos mais a restrição $\|w\| = 1$.

Classificador de margem máxima

- No entanto, continuamos com um problema de otimização de difícil solução (não-convexa).
- Anteriormente, concluímos que se adicionarmos uma restrição escalar arbitrária à w e b , modificamos nada. Baseado nisso, introduziremos a restrição escalar de que a margem funcional de w e b em relação ao set de treinamento deve ser 1:

$$\hat{\gamma} = 1$$

- Considerando que multiplicar w e b por uma constante é a mesma coisa que multiplicar a margem funcional pela mesma constante, essa restrição é realmente escalar, e pode ser satisfeita redimensionando w e b .

Classificador de margem máxima

- Utilizando-o no problema de otimização anterior, notamos que maximizar $\hat{\gamma}/\|w\| = 1/\|w\|$ é o mesmo que minimizar $\|w\|^2$. Portanto, temos agora o seguinte problema:

$$\begin{aligned} \min_{\gamma, w, b} \quad & \frac{1}{2} \|w\|^2 \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1, \quad i = 1, \dots, m \end{aligned}$$

- O problema de otimização obtido é quadrático, cuja solução possui uma ampla e estabelecida teoria matemática. Como a função objetivo sendo minimizada é convexa e os pontos que satisfazem as restrições formam um conjunto convexo, esse problema possui um único mínimo global.
- Problemas desse tipo podem ser solucionados com a introdução de uma função Lagrangiana, que engloba as restrições à função objetivo, associadas a parâmetros denominados multiplicadores de Lagrange α_i :

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^m \alpha_i [y^{(i)}(w^T x^{(i)} + b) - 1]$$

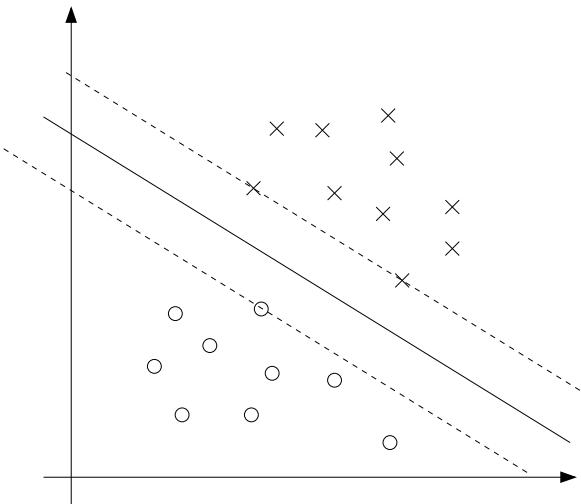
Classificador de margem máxima

- Podemos reescrever a restrição anterior como abaixo, onde temos uma restrição para cada exemplo de treinamento:

$$g_i(w) = -y^{(i)}(w^T x^{(i)} + b) + 1 \leq 0$$

- Pelas condições de Karush-Kuhn-Tucker, teremos que $\alpha_i > 0$ somente para exemplos de treinamento com margem funcional igual à 1 ($g_i(w) = 0$).
- Considere a figura a seguir, onde o hiperplano com margem máxima é mostrado pela linha sólida.

Classificador de margem máxima



- Os pontos com as menores margens são exatamente os pontos mais perto do limite de decisão. Nesse caso, são os três pontos localizados nas linhas tracejadas paralelas ao limite de decisão.
- Portanto, somente três α_i - correspondentes a esses três exemplos de treinamento - serão diferentes de zero na solução ótima para o nosso problema de otimização. Esses três pontos são chamados de vetores de suporte (support vectors). Note que o número de vetores de suporte pode ser bem menor do que o tamanho do set de treinamento.

Classificador de margem máxima

- A função Lagrangiana deve ser minimizada, o que implica em maximizar as variáveis α_i e minimizar w e b . Então temos:

$$\nabla_w \mathcal{L}(w, b, \alpha) = w - \sum_{i=1}^m \alpha_i y^{(i)} x^{(i)} = 0$$

- O qual implica em:

$$w = \sum_{i=1}^m \alpha_i y^{(i)} x^{(i)} \tag{1}$$

- E temos:

$$\frac{\partial}{\partial b} \mathcal{L}(w, b, \alpha) = \sum_{i=1}^m \alpha_i y^{(i)} = 0$$

Classificador de margem máxima

- Substituindo as equações na função Lagrangiana, temos:

$$\mathcal{L}(w, b, \alpha) = \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m y^{(i)} y^{(j)} \alpha_i \alpha_j (x^{(i)})^T x^{(j)} - b \sum_{i=1}^m \alpha_i y^{(i)}$$

- Como o último termo deve ser zero, temos:

$$\mathcal{L}(w, b, \alpha) = \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m y^{(i)} y^{(j)} \alpha_i \alpha_j (x^{(i)})^T x^{(j)}$$

- Como queremos minimizar a função Lagrangiana em relação à w e b , temos o seguinte problema de otimização:

$$\begin{aligned} \max_{\alpha} \quad & W(\alpha) = \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle \\ \text{s.t.} \quad & \alpha_i \geq 0, \quad i = 1, \dots, m \\ & \sum_{i=1}^m \alpha_i y^{(i)} = 0, \end{aligned}$$

Classificador de margem máxima

- Essa formulação é denominada forma dual, enquanto o problema original é referenciado como forma primal. A forma dual possui os atrativos de apresentar restrições mais simples e permitir a representação do problema de otimização em termos de produtos internos entre dados, o que será útil na posterior não-linearização das SVMs.
- É interessante observar também que o problema dual é formulado utilizando apenas os dados de treinamento e os seus rótulos.
- No problema dual, temos um problema de maximização onde os parâmetros são α_i . Falaremos mais tarde sobre o algoritmo específico que usaremos para resolver o problema dual.
- No entanto, se realmente podemos resolvê-lo - isto é, encontrar α que maximize $W(\alpha)$ sujeito às restrições - podemos usar a equação (1) para encontrar os valores ótimos de w em função dos valores de α .

Classificador de margem máxima

- Encontrando w^* , considerando o problema primal e a equação (1), podemos achar o valor ótimo de b como:

$$b^* = -\frac{\max_{i:y^{(i)}=-1} w^{*T} x^{(i)} + \min_{i:y^{(i)}=1} w^{*T} x^{(i)}}{2} \quad (2)$$

- Considere que temos um set de treinamento e queremos fazer uma predição de um novo ponto x baseado nos nossos parâmetros.
- Calcularíamos $w^T x + b$, e preveríamos $y = 1$ se e somente se o valor fosse maior que zero. Utilizando a equação (1), esse valor pode ser escrito como:

$$\begin{aligned} w^T x + b &= \left(\sum_{i=1}^m \alpha_i y^{(i)} x^{(i)} \right)^T x + b \\ &= \sum_{i=1}^m \alpha_i y^{(i)} \langle x^{(i)}, x \rangle + b. \end{aligned} \quad (3)$$

Classificador de margem máxima

- Portanto, se encontrarmos os valores de α_i , para fazermos uma predição devemos calcular uma quantidade que depende somente do produto interno entre x e os pontos do set de treinamento.
- Como vimos anteriormente, os valores de α_i serão todos zero com exceção dos vetores de suporte. Portanto, muitos dos termos da soma anterior serão zero, e precisaremos somente calcular os produtos internos entre x e os vetores de suporte (os quais geralmente são poucos) para realizarmos nossa predição.
- Finalmente, podemos escrever todo nosso algoritmo em termo somente de produtos internos. Usaremos essa propriedade para aplicarmos o conceito de kernel ao nosso problema de classificação. O algoritmo resultante será o SVM (Support Vector Machines).

Kernels

- Voltando a discussão de regressão linear, tínhamos um problema no qual x era a área das casas e consideramos utilizar regressão com, por exemplo, características x , x^2 e x^3 para obter uma função cúbica.
- Para distinguir entre esses dois conjuntos de variáveis, vamos chamar os valores originais de entrada como atributos (attributes) de um problema (nesse caso, a área x).
- Quando mapeamos esses atributos em um novo conjunto que passará pelo algoritmo de aprendizagem, chamaremos esse conjunto de características (features).
- Portanto, chamaremos ϕ como mapa de características, o qual mapeia os atributos em características. Por exemplo, temos

$$\phi(x) = \begin{bmatrix} x \\ x^2 \\ x^3 \end{bmatrix}$$

Kernels

- Ao invés de aplicar SVM usando os atributos originais x , podemos aprender usando as características $\phi(x)$. Ou seja, podemos simplesmente substituir x por $\phi(x)$ no algoritmo anterior.
- Como vimos, o algoritmo que encontramos pode ser inteiramente escrito em termos de produtos internos $\langle x, z \rangle$. Ou seja, iremos substituir todos os produtos internos por $\langle \phi(x), \phi(z) \rangle$.
- Especificamente, dado um mapa de características ϕ , definimos o kernel correspondente como

$$K(x, z) = \phi(x)^T \phi(z)$$

- Portanto, onde tínhamos anteriormente no nosso algoritmo $\langle x, z \rangle$, podemos simplesmente substituir por $K(x, z)$ e nosso algoritmo aprenderá usando as características ϕ .

Kernels

- Agora, dado ϕ , poderíamos calcular $K(x,z)$ encontrando $\phi(x)$ e $\phi(z)$ e fazendo o produto interno. No entanto, em muitas vezes o cálculo de $K(x,z)$ tem um custo bastante baixo, mesmo que o cálculo de $\phi(x)$ tenha um custo bastante alto (talvez porque seja um vetor de dimensão extremamente alta).
- Nesses casos, usando em nosso algoritmo uma maneira eficiente de calcular $K(x,z)$, podemos fazer a SVM aprender em um espaço de características com alta dimensão dado por ϕ , sem em algum momento ter que encontrar ou representar explicitamente os vetores $\phi(x)$.
- Por exemplo, suponhamos que $x,z \in \mathcal{R}^n$ e consideramos

$$K(x,z) = (x^T z)^2$$

Kernels

- Podemos escrevê-lo como:

$$\begin{aligned} K(x, z) &= \left(\sum_{i=1}^n x_i z_i \right) \left(\sum_{j=1}^n x_j z_j \right) \\ &= \sum_{i=1}^n \sum_{j=1}^n x_i x_j z_i z_j \\ &= \sum_{i,j=1}^n (x_i x_j)(z_i z_j) \end{aligned}$$

- Portanto, vemos que $K(x, z) = \phi(x)^T \phi(z)$, onde o mapa de características ϕ é dado (nesse caso para $n = 3$) por

$$\phi(x) = \begin{bmatrix} x_1 x_1 \\ x_1 x_2 \\ x_1 x_3 \\ x_2 x_1 \\ x_2 x_2 \\ x_2 x_3 \\ x_3 x_1 \\ x_3 x_2 \\ x_3 x_3 \end{bmatrix}$$

Kernels

- Note que enquanto para calcular $\phi(x)$ precisamos do tempo $O(n^2)$, para encontrar $K(x, z)$ precisamos apenas de $O(n)$.
- Para um kernel relacionado, consideramos

$$\begin{aligned} K(x, z) &= (x^T z + c)^2 \\ &= \sum_{i,j=1}^n (x_i x_j)(z_i z_j) + \sum_{i=1}^n (\sqrt{2c} x_i)(\sqrt{2c} z_i) + c^2 \end{aligned}$$

- Isso corresponde à um mapa de características (novamente com $n = 3$):

$$\phi(x) = \begin{bmatrix} x_1 x_1 \\ x_1 x_2 \\ x_1 x_3 \\ x_2 x_1 \\ x_2 x_2 \\ x_2 x_3 \\ x_3 x_1 \\ x_3 x_2 \\ x_3 x_3 \\ \sqrt{2c} x_1 \\ \sqrt{2c} x_2 \\ \sqrt{2c} x_3 \\ c \end{bmatrix}$$

Kernels

- O parâmetro c controla o peso relativo entre os termos x_i (primeira ordem) e $x_i x_j$ (segunda ordem).
- Pensando de forma mais abrangente, o kernel $K(x,z) = (x^T z + c)^d$ corresponde ao mapeamento de características em um espaço de características $\binom{n+d}{d}$.
- No entanto, apesar de trabalhar nesse espaço com dimensão $O(n^d)$, calcular $K(x,z)$ necessita somente o tempo $O(n)$, e portanto nunca precisamos representar explicitamente os vetores de característica nesse espaço de características com alta dimensão.

Kernels

- Se $\phi(x)$ e $\phi(z)$ estão próximos, podemos esperar que $K(x,z) = \phi(x)^T\phi(z)$ será grande. Se $\phi(x)$ e $\phi(z)$ estão afastados - quase ortogonais entre eles - então $K(x,z) = \phi(x)^T\phi(z)$ será pequeno.
- Portanto, podemos pensar $K(x,z)$ como uma medida do quanto similares são $\phi(x)$ e $\phi(z)$, ou o quanto similares são x e z .
- Dada essa intuição, digamos que estamos trabalhando em um problema onde a similaridade entre x e z parece uma medida razoável. Por exemplo, podemos escolher

$$K(x,z) = \exp\left(-\frac{\|x-z\|^2}{2\sigma^2}\right)$$

- Essa é uma medida razoável da similaridade entre x e z , o qual é perto de 1 quando x e z estão próximos e próximo de 0 quando x e z estão distantes. Esse kernel é chamado de kernel gaussiano e corresponde à um mapeamento de características ϕ com dimensão infinita.

Kernels

- Dada uma função K , como podemos dizer se a mesma é um kernel válido? Isto é, como dizer se existe um mapeamento de características ϕ que então $K(x, z) = \phi(x)^T \phi(z)$ para todo x, z ?
- Suponha que K é um kernel válido correspondente à um mapa de características ϕ . Considere então um set finito de m pontos $\{x^{(1)}, \dots, x^{(m)}\}$ e K como uma matriz quadrada $m \times m$, onde sua entrada (i, j) é dada por $K_{ij} = K(x^{(i)}, x^{(j)})$ (Kernel matrix).
- Se K é um kernel válido, então $K_{ij} = K(x^{(i)}, x^{(j)}) = \phi(x^{(i)})^T \phi(x^{(j)}) = \phi(x^{(j)})^T \phi(x^{(i)}) = K(x^{(j)}, x^{(i)}) = K_{ji}$. Portanto, K é simétrica.
- Além disso, considerando $\phi_k(x)$ ser a coordenada k do vetor $\phi(x)$, então para qualquer vetor z temos:

Kernels

$$\begin{aligned} z^T K z &= \sum_i \sum_j z_i K_{ij} z_j \\ &= \sum_i \sum_j z_i \phi(x^{(i)})^T \phi(x^{(j)}) z_j \\ &= \sum_i \sum_j z_i \sum_k \phi_k(x^{(i)}) \phi_k(x^{(j)}) z_j \\ &= \sum_k \sum_i \sum_j z_i \phi_k(x^{(i)}) \phi_k(x^{(j)}) z_j \\ &= \sum_k \left(\sum_i z_i \phi_k(x^{(i)}) \right)^2 \\ &\geq 0. \end{aligned}$$

- Isso nos mostra que K é positiva semi-definida ($K \geq 0$). Portanto, como dito no Teorema de Mercer, a matriz $K \in \mathcal{R}^{m \times m}$ é um kernel válido se for uma matriz simétrica positiva semi-definida.

Kernels

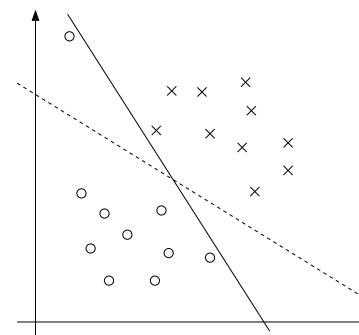
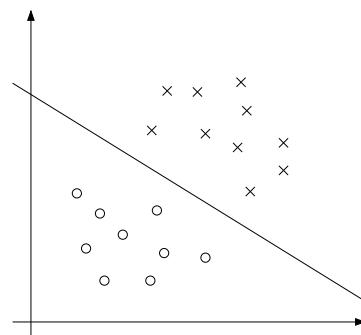
- Vamos considerar o uso de kernels para alguns problemas. Por exemplo, em um problema de reconhecimento de dígito, no qual dada uma imagem (16x16 pixels) de um dígito (0-9) escrito à mão, queremos descobrir qual é o dígito.
- Usando tanto um simples kernel polinomial $K(x,z) = (x^T z)^d$ quanto o kernel gaussiano, SVMs são capazes de obter uma ótima performance nesse problema, o que é de certo modo surpreendente, considerando que os atributos de entrada x é um vetor de tamanho 256 da intensidade dos pixels e que o sistema não tinha conhecimento sobre visão ou sobre quais pixels são adjacentes entre eles.
- Outro exemplo é se x que estamos tentando classificar são strings. Nesse problema, parece difícil construir um set pequeno de características para a maioria dos algoritmos, especialmente se as strings tiverem tamanhos diferentes.

Kernels

- Vamos considerar então $\phi(x)$ um vetor de características o qual conta o número de ocorrências de cada substring de tamanho k em x . Se considerarmos o nosso alfabeto, teremos 26^k strings. Portanto, $\phi(x)$ é um vetor de tamanho 26^k .
- Mesmo para valores moderados de k , esse valor é provavelmente muito grande para trabalharmos eficientemente ($26^4 \approx 460000$). No entanto, é possível calcular eficientemente $K(x,z) = \phi(x)^T \phi(z)$, onde podemos trabalhar implicitamente nesse espaço de características com dimensão 26^k , mas sem explicitamente calcular os vetores de característica nesse espaço.
- O conceito de kernel possui uma aplicabilidade que vai além somente de SVMs. Especificamente, se o algoritmo de aprendizado utilizado pode ser escrito em termos de produtos internos $\langle x, z \rangle$ entre os vetores dos atributos de entrada, então podemos substituí-lo pelo kernel $K(x,z)$, o que permite o algoritmo trabalhar eficientemente em um espaço de características com alta dimensionalidade.

Regularização e o caso não-separável

- O algoritmo SVM apresentado até agora assumiu que os dados são linearmente separáveis. Apesar do mapeamento dos dados em um espaço de características de alta dimensionalidade geralmente aumentar a probabilidade dos dados serem separáveis, não podemos garantir que isso sempre ocorrerá.
- Além disso, para alguns casos encontrar um hiperplano não é exatamente o que queremos, pois podemos estar suscetíveis à um valor atípico (outlier).
- Por exemplo, na figura abaixo é mostrado um classificador ótimo de margem no gráfico à esquerda. Adicionando somente um valor atípico faz com que o limite de decisão modifique drasticamente, resultando em um classificador com uma margem bem menor.



Regularização e o caso não-separável

- Para fazermos o algoritmo funcionar para dados não-linearmente separáveis e se tornar menos sensível à valores atípicos, reformulamos nossa otimização (usando regularização ℓ_1):

$$\begin{aligned} \min_{\gamma, w, b} \quad & \frac{1}{2} \|w\|^2 + C \sum_{i=1}^m \xi_i \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i, \quad i = 1, \dots, m \\ & \xi_i \geq 0, \quad i = 1, \dots, m. \end{aligned}$$

- Portanto, agora é permitido para exemplos terem margem funcional menores do que 1. Se um exemplo tem margem funcional $1 - \xi_i$, há um custo na função objetivo, acrescida por $C\xi_i$.
- O parâmetro C controla o peso relativo entre a minimização dos erros no conjunto de treinamento e assegurar que os exemplos tenham margem funcional de, pelo menos, 1.

Regularização e o caso não-separável

- Como anteriormente, podemos encontrar a lagrangiana como abaixo, onde os α_i 's e os r_i 's são multiplicadores de Lagrange, restritos a serem maiores ou iguais a 0:

$$\mathcal{L}(w, b, \xi, \alpha, r) = \frac{1}{2}w^T w + C \sum_{i=1}^m \xi_i - \sum_{i=1}^m \alpha_i [y^{(i)}(x^T w + b) - 1 + \xi_i] - \sum_{i=1}^m r_i \xi_i$$

- Calculando as derivadas em relação à w e b e igualando-as à 0, substituindo-as elas na equação novamente e simplificando, podemos encontrar a seguinte forma dual do problema:

$$\begin{aligned} \max_{\alpha} \quad & W(\alpha) = \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C, \quad i = 1, \dots, m \\ & \sum_{i=1}^m \alpha_i y^{(i)} = 0, \end{aligned}$$

Regularização e o caso não-separável

- Note que w pode ser expresso em termos de α_i 's como na equação (1). Portanto, após resolver o problema dual, podemos continuar usando a equação (3) para realizar as previsões.
- Surpreendentemente, ao adicionar a regularização ℓ_1 , a única mudança no problema dual é que onde tínhamos uma restrição $0 \leq \alpha_i$ se tornou $0 \leq \alpha_i \leq C$. O cálculo de b^* também foi modificado, sendo a equação utilizada anteriormente não mais válida.
- Além disso, as condições complementares duais KKT são:

$$\begin{aligned}\alpha_i = 0 &\Rightarrow y^{(i)}(w^T x^{(i)} + b) \geq 1 \\ \alpha_i = C &\Rightarrow y^{(i)}(w^T x^{(i)} + b) \leq 1 \\ 0 < \alpha_i < C &\Rightarrow y^{(i)}(w^T x^{(i)} + b) = 1\end{aligned}$$

- Agora, o que nos resta é utilizar um algoritmo para resolvemos o problema dual. O algoritmo escolhido é o SMO (Sequential Minimal Optimization) de John Platt.

Exercício

- Considerando os dados no Q-acadêmico, implemente o algoritmo SMO-SVM para classificá-los em duas classes. Utilize $C = 0,5$ e inicialize α 's e b 's iguais a zero.
- Implemente uma função separada para o kernel, onde o usuário pode escolher entre um kernel gaussiano ou polinomial (primeira, segunda ou terceira ordem).
- Faça dois gráficos em uma figura. No primeiro plote os dados e a classificação originais. No segundo plote a classificação do seu algoritmo, o limiar de decisão e destaque quais pontos são os vetores de suporte.

Regularização e seleção de modelo

- Vamos considerar que queremos selecionar um entre vários modelos para um problema de aprendizagem. Por exemplo, queremos utilizar um modelo de regressão polinomial $h_\theta(x) = g(\theta_0 + \theta_1x + \theta_2x^2 + \dots + \theta_kx^k)$ e precisamos decidir se k deve ser 0, 1, ..., ou 10.
- Como podemos selecionar um modelo que representa um bom trade-off entre, por exemplo, os valores de bias e variância? Ou como selecionamos o valor do parâmetro τ de largura de banda (bandwidth parameter) para uma regressão ponderada? Ou o parâmetro C para um SVM?
- Portanto, queremos selecionar um modelo entre um grupo finito de modelos $\mathcal{M} = \{M_1, \dots, M_d\}$. Por exemplo, no exemplo acima o modelo M_i seria o modelo de regressão polinomial de ordem i . Alternativamente, se estamos tentando selecionar SVM, uma rede neural ou regressão logística, então \mathcal{M} contém esses modelos.

Validação cruzada

- Vamos considerar um set de treinamento S . Uma maneira de escolher um modelo a partir desse set seria através do algoritmo abaixo:
 1. Treine cada modelo M_i usando S , gerando hipóteses h_i ;
 2. Escolha a hipótese com o menor erro de treinamento.
- Esse algoritmo é uma boa alternativa?

Validação cruzada

- A resposta é não. Considere escolher entre a ordem de um polinômio. Quanto mais alta a ordem do polinômio, melhor será o aprendizado considerando o set de treinamento S e, portanto, menor o erro de treinamento.
- Portanto, esse método sempre escolherá um modelo polinomial com ordem alta, o que vimos anteriormente ser uma escolha ruim.
- Abaixo temos um algoritmo que funciona melhor, chamado validação cruzada utilizando o método *hold-out*:
 1. Aleatoriamente separar S em S_{train} (com 70% dos dados) e S_{cv} (30% restante). S_{cv} é chamado de set de validação cruzada.
 2. Treine cada modelo M_i usando somente S_{train} , gerando hipóteses h_i ;
 3. Selecione a hipótese h_i que possui o menor erro $\varepsilon_{\text{Scv}}(h_i)$ utilizando o set de validação cruzada.

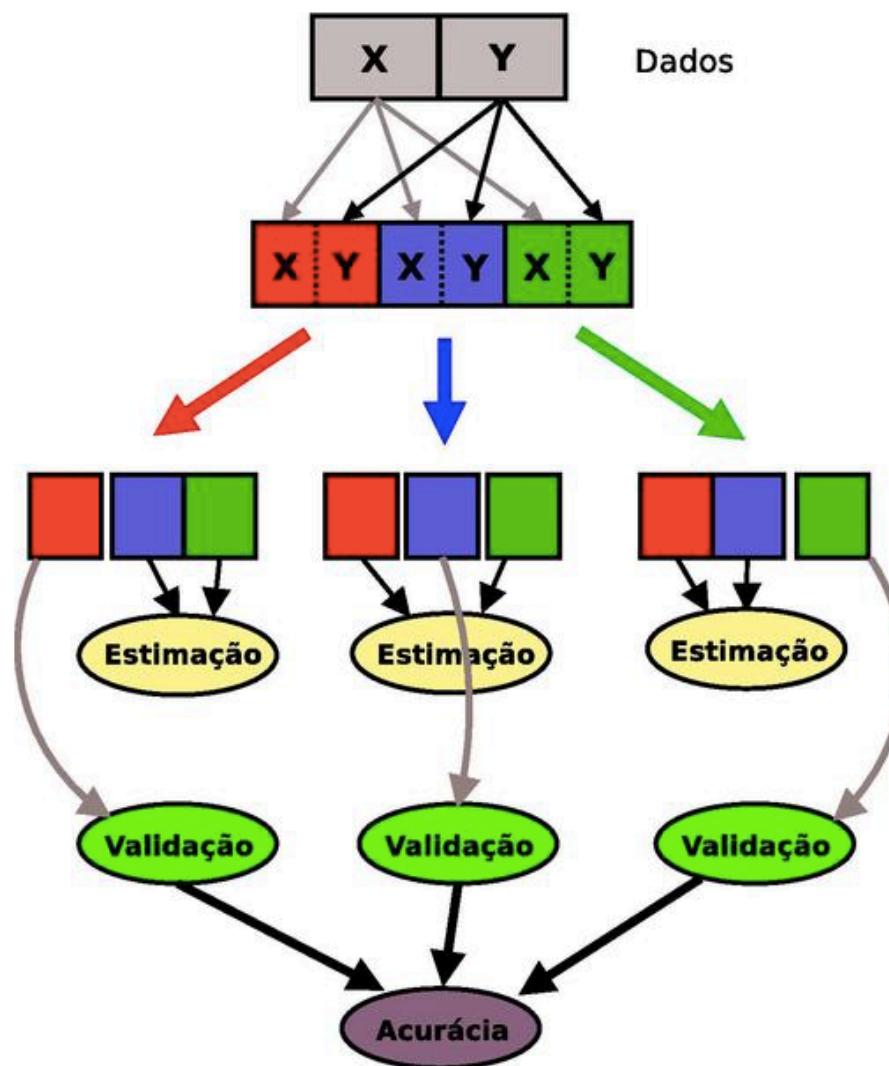
Validação cruzada

- Ao testar no set de exemplos S_{cv} em que os modelos não foram treinados, podemos obter uma estimativa melhor do verdadeiro erro generalizado de cada hipótese h_i , escolhendo a hipótese com menor erro.
- Opcionalmente, após o passo 3 do algoritmo anterior, pode-se treinar novamente o modelo escolhido utilizando agora o set de treinamento inteiro (geralmente essa é um boa alternativa, com exceção de algoritmos de aprendizado muito sensíveis à perturbações).
- A desvantagem do método hold-out é que “desperdiçamos” 30% dos dados. Em casos que temos dados em abundância, não será um grande problema. No entanto, em problemas onde os dados são escassos, precisamos de uma alternativa.

Validação cruzada

- Vamos analisar um outro método de validação cruzada, chamado método *k-fold*:
 1. Separar aleatoriamente S em k subsets com m/k exemplos de treinamento cada. Vamos chamar esses subsets de S_1, \dots, S_k .
 2. Para cada modelo M_i , avaliamos da seguinte maneira:
 - Para $j = 1, \dots, k$
 - Treine o modelo M_i com $S_1 \cup \dots \cup S_{j-1} \cup S_{j+1} \cup \dots \cup S_k$ (isto é, treine com todos os dados exceto S_j) para encontrar hipóteses h_{ij} .
 - Teste a hipótese h_{ij} em S_j para obter $\varepsilon_{Sj}(h_{ij})$.
 - O erro estimado do modelo M_i é então calculado como a média dos $\varepsilon_{Sj}(h_{ij})$'s (média calculada sobre j).
- 3. Escolha o modelo M_i com o menor erro estimado e treine novamente o modelo utilizando todo o set de treinamento S .

Validação cruzada



Validação cruzada

- Uma escolha típica para k seria $k = 10$. Apesar da fração de dados deixada de fora agora ser $1/k$ - bem menor que antes - o procedimento seria mais caro computacionalmente que o método hold-out, considerando que precisamos treinar cada modelo k vezes.
- Enquanto $k = 10$ é comumente usado, em problemas em que os dados são realmente escassos, podemos utilizar a escolha extrema de $k = m$ para deixarmos o menor número possível de dados de lado.
- Nesse caso, treinariíamos repetidamente em todos os exemplos de treinamento de S com exceção de um, e testariíamos nesse exemplo. Então é feita a média dos $m = k$ erros resultantes para obtermos o erro estimado do modelo. Esse método é chamado de método *leave-one-out*.