Análise de Sentimentos Aplicada à Política

Lucas Romão Silva

Prof Dr. Roberto Hirata Jr.

22 de setembro de 2017

Sumário

Capítulo 1

The First Chapter

Capítulo 2

Materiais e Métodos

2.1 Considerações

Ao longo deste capítulo, se usará n para se referir à quantidade de elementos fornecidas ao nosso modelo, cada entrada é i é um vetor $x_i \in \mathbb{R}^m$. A entrada será referida como X a efeitos de conta e assim cada entrada será dada como X_i nesse caso. Para cada i associaremos duas variáveis t_i e y_i que se referem ao valor esperado e ao valor obtido através do treinamento, respectivamente.

A notação $\mathbb{1}_{i==j}$ é uma função indicadora que vale 1 se i é igual a j e 0 caso contrário.

2.2 Contextualização

Os problemas tratados por *Machine Learning* classificam-se de forma geral em três tipos:

- Aprendizado supervisionado: nesse caso tem-se os elementos de entrada e para cada um desses elementos, tem-se associado um rótulo t_i. Nesse caso o modelo deve ser treinado com base nos elementos dados para que se possa prever o rótulo de uma nova entrada;

- Aprendizado não-supervisionado: nesse caso tem-se apenas os elementos de entrada. O objetivo deste tipo de problema é tentar modelar uma distribuição ou estrutura comum entre os dados para que se possa entendê-los melhor;

- Aprendizado semi-supervisionado: nesse último caso alguns elementos possuem um rótulo associado. Problemas desse tipo aplicam técnicas gtanto de aprendizado supervisionado como de não-supervisionado.

Neste trabalho será tratado um problema de aprendizado supervisionado que é o da classificação.

Na classificação temos k classes e cada elemento i da entrada é associado a uma classe $t_i = \{1..k\}$. O objetivo do problema da classificação é dado entrada $X = (x_1, x_2, \ldots, x_n)$ e $t = (t_1, \ldots, t_n)$ treinar um modelo capaz de prever classes para um x qualquer.

Há diversos algoritmos na literatura que se propõem a resolver o problema da classificação. Bishop (2006)[1] enuncia diversos dos algoritmos comumente utilizados para a classificação, cada algoritmo possui seus prós e contras e utiliza diferentes abordagens.

Para este trabalho escolheu-se implementar os algoritmos *Logistic Regression* e *Support Vector Machines*, que será chamado simplesmente de SVM por facilidade.

Tanto para o *Logistic Regression* quanto SVM será explicado a princípio o problema será inicialmente abordado a partir da classificação binária e, a partir dela, será descrito como estender para o problema com mais de duas classes, que será é o caso deste trabalho.

2.3 Logistic Regression

O nosso modelo será construído de forma probabilística, isto é, a partir de um discriminante linear $w^T x + w_0$ atribuiremos uma probabilidade de um elemento x pertencer à classe C^1 e, consequentemente a probabilidade de pertencer à classe C^2 é dada por $1 - P(C^1|x)$. O termo w_0 é chamado de viés, e para efeito das contas que serão feitas consideraremos vetores w' e x' da forma $x' = (x, 1)$, $w' = (w, w_0)$ note entretanto que os chamaremos daqui pra frente simplesmente de w e x.

No caso da classificação binária, usaremos que $t_n \in \{0, 1\}$ onde $t_n = 1$ se o elemento pertence à classe C^1 e $t_n = 0$ se pertence à classe C^2.

A classificação de um elemento será a classe a qual ele tem maior probabilidade de pertencer.

Para utilizarmos nosso discriminante para atribuir as probabilidades, utiliza-se a função sigmóide definida por:

$$\sigma(a) = \frac{1}{1 + exp(-a)} \tag{2.1}$$

A função sigmóide é usada para obter a probilidade por ter a propriedade de mapear todo o conjunto dos números reais dentro do intervalo $[0, 1]$.

Com *exp* sendo a função exponencial. Aplicando ao nosso modelo obtêm-se a expressão:

$$P(C^1|x) = y(x) = \sigma(w^T x) \tag{2.2}$$

Importante notar que apesar de utilizarmos o vetor x nas equações, é possível aplicarmos uma transformação linear $\phi : \mathcal{R}^m \to \mathcal{R}^d$ à entrada x para obtermos $\phi(x)$. O uso de transformação linear no nosso conjunto de entrada nos permite transformar o domínio para que se obtenha uma separação melhor entre as classes ou até mesmo fazer a redução da dimensão do domínio.

Com essa equação em mãos, nosso objetivo é minimizar o erro na classificação dos dados. Tomamos como erro o negativo do logaritmo da verosimilhança de nossa função que é dada por:

$$E(w) = -\sum_{i=1}^{n} p(t|w) = -\sum_{i=1}^{n} \{t_n ln(y_n) + (1 - t_n) ln(1 - y_n)\} \tag{2.3}$$

A fim de minimizar o erro, utiliza-se métodos de otimização linear (note que por mais que se use uma transformação linear ϕ sobre x nosso problema ainda é linear sobre w).

Dois métodos são comumente usados: método do gradiente e método de Newton-Raphson. Esses métodos são utilizados tanto para o caso da classificação binária quanto o caso da classificação com $k > 2$. A diferença entre um problema e outro será abordada com mais especificidade a seguir.

Uma dúvida natural que surge ao ter que resolver um problema de otimização é o caso de parar o procedimento em um mínimo local ao invés de um mínimo local da função. Entretanto, temos que nossa função $E(w)$ é côncava, isto é, $E(\lambda w + (1 - \lambda)w') = \lambda E(w) + (1 - \lambda)w' \ \forall w, w' \in R^m, \lambda \in [0, 1]$, tal propriedade nos garante que existe um único minizador.

2.3.1 Método do Gradiente

Para este método, minimiza-se a função objetivo, no caso $E(w)$ utilizando apenas o gradiente da função junto de um passo α. Com ambos valores em mãos, o valor w é atualizado usando a equação:

$$w^{(novo)} = w^{(antigo)} + \alpha \nabla E(w) \tag{2.4}$$

Com $\nabla E(w)$ sendo o gradiente do vetor de pesos. O gradiente é calculado usando o fato de que a derivada da função sigmóide com respeito a um vetor a é dada por:

$$\frac{d\sigma}{da} = \sigma(1-\sigma) \tag{2.5}$$

Usando 2.5 tem-se a seguinte equação para o gradiente:

$$\nabla E(w) = X^T(y-t) \tag{2.6}$$

Onde $y_n = P(C^1|x_n) = \sigma(w^T x)$ e t_n tal qual assumido no começo da seção.

O algoritmo de atualização do vetor de pesos descrito a seguir vale tanto para o método do gradiente quanto para o de Newton-Raphson, portanto para o segundo será focado apenas nas diferenças entre os dois.

Algorithm 1 Logistic Regression usando método do gradiente

Input: Matriz $X \in \mathbb{R}^{n \times m}$, vetor de rótulos $t \in \{0,1\}^n$

Output: Vetor de pesos $w \in \mathbb{R}^m$

1: $iteracao \leftarrow 0$
2: $w \leftarrow 0$
3: **while** $|E(w)^{(iteracao)} - E(w)^{(iteracao-1)}| \geq \epsilon$ **and** $iteracao < maxIteracoes$ **do**
4: $\quad y \leftarrow (\sigma(w^T x_1), \sigma(w^T x_2), \ldots, \sigma(w^T x_n))^T$
5: $\quad \nabla E(w) \leftarrow X^T(y-t)$
6: $\quad w \leftarrow w - \alpha \nabla E(w)$
7: $\quad E(w)^{(iteracao)} \leftarrow -\sum_{i=1}^{n}\{t_n ln(y_n) + (1-t_n) ln(1-y_n)\}$
8: $\quad iteracao \leftarrow iteracao + 1$
9: **end while**

Importante notar que em 3 tem-se duas condições de paradas do algoritmo que são o número de iterações e a diferença da diminuição da função objetivo for menor do que um dado ϵ. Tais condições são chamadas de condições de convergência e nos garantem que chegamos a um valor suficientemente próximo do ótimo, uma vez que atingir este valor pode exigir um número muito alto de iterações, o que traz um custo computacional. Na implementação do algoritmo, escolheu-se um valores padrão para ϵ e $maxIteracoes$ como 10^{-4} e 200 respectivamente.

A quantidade de iterações necessárias para a convergência é influenciada fortemente pela escolha de α. Um valor pequeno para α acarretaria em muitas iterações até a convergência ao passo que um valor muito grande pode fazer com que se pare muito longe do valor ótimo.

2.3.2 Método de Newton-Raphson

Vimos em 2.3.1 que o método do gradiente apesar de implementação simples pode levar muito tempo para resolver o problema.

O método de Newton-Raphson acaba convergindo mais rápido do que o método do gradiente, contudo ao custo de uma maior complexidade devido à necessidade de calcular outros elementos.

A atualização agora é feita seguindo a equação

$$w^{(novo)} = w^{(antigo)} - H^{-1}\nabla E(w) \tag{2.7}$$

Onde H é a matriz Hessiano da função erro, que é calculado usando $H = \nabla\nabla E(w) = X^T RX$ onde R é uma matriz diagonal $n \times n$ onde as entradas da diagonal principal valem $R_{kk} = y_k(1 - y_k)$. Substituindo os valores de H e usando 2.6 em 2.7 obtemos

$$\begin{aligned} w^{(novo)} &= w^{(antigo)} - (X^T RX)^{-1}\nabla E(w) \\ &= (X^T RX)^{-1}[(X^T RX)w^{(antigo)} - X^T(y - t)] \end{aligned} \tag{2.8}$$

2.3.3 Extensão para o caso de várias classes

Diversas abordagens podem ser usadas para resolver o problema multiclasse, no caso será usado diversos discriminantes y_k com $k = \{1, \dots K\}$ com K sendo o total de classes. Assim nosso vetor w agora é uma matriz $W \in \mathbb{R}^{m\times k}$.

Quanto à codificação do vetor de rótulos, segue-se a codificação dada em Bishop (2006)[1] de $1-K$, na codificação tem-se que $t_n \in \{0, 1\}^k$ com $t_{nk} = 1$ se o elemento n pertencer à classe k e 0 nas demais entradas.

Quanto a função de probabilidade que desejamos estimar, utiliza-se a função *softmax* que é dada pela equação:

$$P(C^k|x_n) = y_{nk} = \frac{exp(w_k^T x_n)}{\sum_j exp(w_j^T x_n)} \tag{2.9}$$

Que nos dá verossimilhança e consequentemente a seguinte função de erro, tomada usando o negativo do logaritmo da verossimilhança.

$$P(T|w_1,\ldots,w_k) = \prod_{i=1}^{n}\prod_{j=1}^{k} P(C^j|x_i)^{t_{ij}} = \prod_{i=1}^{n}\prod_{j=1}^{k} y_{ij}^{t_{ij}}$$
$$E(W) = -\sum_{i=1}^{n}\sum_{j=1}^{k} t_{ij} ln(y_{ij})$$

Novamente nesse caso pode-se encontrar o valor de W que minimize $E(W)$ usando os dois métodos discutidos em 2.3.1 e 2.3.2, porém agora temos que a derivada com respeito a cada $w_k^T x$ vale:

$$\frac{\partial y_k}{\partial(w_j^T x)} = y_k(\mathbb{1}_{k==j} - y_j) \tag{2.10}$$

Nosso valor de W pode ser interpretado tanto como uma matriz $m \times k$ como um único vetor $1 \times mk$ onde $W = (w_1, w_2, \ldots, w_k)$.

Usando essa representação, podemos calcular o vetor gradiente onde a derivada com respeito a cada w_j é dada pela equação:

$$\nabla_{w_j} E(W) = X^T(Y_j - T_j) \tag{2.11}$$

Com Y_j e T_j correspondendo, respectivamente, às j-ésimas colunas de Y e T.

Com o gradiente em mãos já temos o que é necessário para o método do gradiente e a atualização seria feita da forma $W^{(novo)} = W^{(antigo)} - \alpha\nabla E(W)$.

Para aplicarmos o método de Newton-Raphson, seria necessário computarmos o Hessiano que nesse caso seria uma matriz $m * k \times m * k$ com cada bloco j, i contendo uma matriz $m \times m$ calculada pela equação:

$$\nabla_{w_i}\nabla_{w_j} E(W) = -\sum_{k=1}^{n} y_{ki}(\mathbb{1}_{i==j} - y_{kj}) X_k^T X_k \tag{2.12}$$

Onde X_k é a k-ésima linha de X. Com essas equações em mãos nossa atualização de W seria feita usando a fórmula $W^{(novo)} = W^{(antigo)} - H^{-1}\nabla E(W)$.

A classificação de um novo x é feita a partir do cálculo de $P(C^k|x) = y_k(x), \forall k = \{1, \ldots, k\}$.

A classe de x é dada pelo k que tiver a maior probabilidade sobre os demais.

2.4 Support Vector Machine

Assim como fizemos com o logistic regression, começaremos com a definição para o caso binário e depois iremos estender para mais de uma classe. Nesse caso nossas classes serão $t_n \in \{-1, 1\}$ onde $t_n = 1$ se x pertence à classe C^1 e $t_n = -1$ se x pertence à classe C^2.

No algoritmo SVM a classificação é feita a partir de um discriminante linear da forma

$$y(x) = w^T x + b \tag{2.13}$$

Tal y é chamado de hiperplano de decisão e a classificação é baseada no sinal de y. Se $y(x) > 0$, x é atribuído à classe C^1, caso contrário é atribuído à classe C^2.

Porém ao invés de procurarmos um w que separe perfeitamente todas as classes (que não necessariamente existe), nosso objetivo é maximizar a margem do discriminante linear, isto é, a menor distância de um ponto ao hiperplano. A distância de um ponto ao hiperplano é dado pela fórmula

$$\frac{[t_n(w^T x_n + b)]}{||w||} \tag{2.14}$$

Na descrição inicial do problema vamos tratar o caso com o conjunto X linearmente separável, o que indica que é possível obter um hiperplano que separe sem erro todas as classes para, em seguida, tratarmos o caso real que é o do conjunto que não é linearmente separável. O fato de o conjunto ser linearmente separável nos garante que $t_n y(x_n) > 0 \forall n$.

$$\underset{x,b}{\operatorname{argmax}} \left\{ \frac{1}{||w||} min_n [t_n(w^T x_n + b)] \right\} \tag{2.15}$$

Podemos ajustar w e b de forma a termos que $t_n(w^T x_n + b) = 1$ para o ponto mais próximo da margem e $t_n(w^T x_n + b) \geq 1, \forall n$. Com esse reajuste temos que nosso problema de encontrar um vetor de pesos de margem maximizada seria de maximizar $\frac{1}{||w||}$ que é equivalente ao problema de otimização quadrática:

$$\begin{array}{ll} \text{argmin}_{w,b} & ||w||^2 \\ \textbf{sujeito a} & t_k(w^T x_k + b), \quad k = 1, \ldots, n \end{array} \tag{2.16}$$

Agora iremos supor que não necessariamente nossa entrada não é linearmente separável, isto é, não existe um hiperplano que separe perfeitamente as duas classes. Assim iremos permitir que alguns valores estejam classificados incorretamente, para isso será necessário suavizarmos nossa margem penalizando cada uma das entradas incorretamente classificadas. Cortes e Vapnik (1995)[2] descrevem a penalização através da introdução de variváveis de folga $\xi_n \geq 0$ para cada elemento de X. Um elemento corretamente classificado teŕa $\xi_n = 0$, os demais pontos têm $\xi_n = |t_n - y(x_n)|$.

Com isso nossa restrição de $t_n y(x_n) \geq 1$ é modificada e tem-se $t_n y(x_n) \geq 1 - \xi_n \forall n$.

O valor de ξ_n assim nos indica três possíveis casos:

- Se $\xi_n = 0$, x_n está corretamente classificado e se encontra ou na margem ou do lado correto dela.

- Se $0 < \xi_n \leq 1$, x_n está corretamente classificado e se encontra entre a margem e o hiperplano.

- Se $\xi_n > 1$, x_n não está classificado corretamente.

A função objetivo agora precisa conter os valores de ξ e para isso colocamos uma constante $C > 0$ que define a compensação entre a penalização das variáveis de folga e a margem. Com isso temos o seguinte problema de otimização quadrática:

$$\begin{array}{ll} \text{argmin}_{w,b,\xi} & C \sum_{i=1}^{n} \xi_i + \frac{1}{2}||w||^2 \\ \text{sujeito a} & t_k(w^T x_k + b) \geq 1 - \xi_k, \quad k = 1, \ldots, n \end{array} \tag{2.17}$$

Para implementar o SVM basta então resolver o problema de otimização quadrática acima. Isto é feito seguindo os seguintes passos

1. Introduz-se multiplicadores de Lagrange α_n e μ_n e obtem-se o Lagrangiano com as restrições

$$L(w,b,\xi) = \frac{1}{2}||w||^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i\{t_i(w^T x_i + b) - 1 + \xi_i\} - \sum_{i=1}^{n}\mu_i\xi_i \tag{2.18a}$$

$$\alpha_n \geq 0 \tag{2.18b}$$

$$t_n y(x_n) - 1 + \xi_n \geq 0 \tag{2.18c}$$

$$\alpha_n\{t_n(w^T x_n + b) - 1 + \xi_n\} = 0 \tag{2.18d}$$

$$\mu_n \geq 0 \tag{2.18e}$$

$$\mu_n\xi_n = 0 \tag{2.18f}$$

2. Derivamos o lagrangiano com respeito a w, b e ξ e igualamos a 0 para obtermos os valores ótios para essas variáveis e, assim obtemos os seguintes valores:

$$\frac{\partial L}{\partial w} = 0 \Rightarrow w = \sum_{i=1}^{n}\alpha_i t_i x_i \tag{2.19}$$

$$\frac{\partial L}{\partial b} = 0 \Rightarrow \sum_{i=1}^{n}\alpha_i t_i = 0 \tag{2.20}$$

$$\frac{\partial L}{\partial \xi} = 0 \Rightarrow \alpha_n = C - \mu_n \tag{2.21}$$

3. Com isso em mãos, resolve-se não o problema primal e sim o dual (utiliza-se aqui o fato de que se as condições mencionadas no item anterior são satisfeitas, tem-se que vale a dualidade forte e o valor ótimo de ambas as funções coincide). O dual é dado pelo problema

$$\begin{aligned} \min_{\alpha} \quad & \tilde{L}(\alpha) = \sum_{i=1}^{n}\alpha_i - 1/2\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j t_i t_j k(x_i, x_j) \\ \text{sujeito a} \quad & 0 \leq \alpha_i \leq C, i = 1, \ldots, n \\ & \sum_{i=1}^{n}\alpha_i t_i = 0 \end{aligned} \tag{2.22}$$

No problema a cima é importante destacar a expressão $k(x_i, x_j)$, essa expressão é um *Kernel* que é uma função onde $k(x, x') = \phi(x)^T \phi(x')$ com ϕ sendo alguma transformação linear. Tal qual no caso da regressão logística, essas transformações são usadas para mudar o domínio da entrada x.

4. Acha-se o valor de b usando a fórmula:

$$b = \frac{1}{N_{\mathcal{M}}} \sum_{n \in \mathcal{M}} \left(t_n - \sum_{m \in \mathcal{S}} \alpha_m t_m k(x_n, x_m) \right) \tag{2.23}$$

Com $\mathcal{M}$ sendo o conjunto de pontos que satisfazem $0 < \alpha_n < C$ e $\mathcal{S}$ o conjunto de vetores de suporte (pontos que possuem $\alpha_n > 0$ e, consequentemente, contribuem para a classificação do modelo).

Uma vez resolvido o problema dual e encontrado valor de b, podemos classificar um novo x usando o sinal do discriminante $y(x)$ dado por

$$y(x) = \sum_{i=1}^{n} \alpha_i t_i k(x, x_i) + b = \sum_{n \in \mathcal{S}} \alpha_n t_n l(x, x_n) + b \tag{2.24}$$

2.4.1 Extensão ao caso de multiclasses

Diversas abordagens são possíveis para o caso de multiclasses.

Referências Bibliográficas

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning.* Springer, 1 edition, 2006.

[2] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, (20):273–297, 1995.