Análise de Sentimentos Aplicada à Política

Lucas Romão Silva

Prof Dr. Roberto Hirata Jr.

14 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 $x_i \in \mathbb{R}^m$. 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.

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)[**?**] 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 *Supor 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

Para classificar um dado elemento x entre as possíveis classes C_1 e C_2, é utilizado um discriminante linear da forma $y(x) = w^T x + w_0$ sendo w o vetor de pesos associado. A classe atribuída a um vetor x é baseado no sinal de $y(x)$, se $y(x) \geq 0$ ele é designado à classe C^1, caso contrário é designado à classe C^2.

No caso da classificação binária, usamos que $t_n \in \{0, 1\}$

Nesse caso, diz-se que uma superfície de decisão é definida pelo hiperplano $y(x) = 0$ onde sua posição é determinada pelo elemento w_0 que chamaremos de viés. Uma vez que tanto nosso vetor de pesos w quanto nossos vetores x do conjunto de treino possuem m elementos, iremos criar vetores novos $w' = (w_0, w), x' = (1, x)$.

O nosso modelo será construído de forma probabilística, uma vez que o objetivo será obter um vetor w de modo que possamos estimar as probabilidades condicionais $P(C^1|x)$ e consequentemente $P(C^2|x) = 1 - P(C^1|x)$, isto é, a probabilidade de um vetor x pertencer à uma determinada classe.

Para utilizarmos nosso discriminante $y(x)$ para atribuir as probabilidades, utiliza-se a função sigmóide definida por:

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

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 $\phi : \mathcal{R}^m \rightarrow \mathcal{R}^d$ 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 $H = \nabla\nabla E(w) = X^T R X$ 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

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

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)[**?**] 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.

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

$$E(W) = -\sum_{i=1}^{n}\sum_{j=1}^{k} t_{ij} ln(y_{ij})$$