Classificadores Lineares

Julio Manuel P Osório

FEA.Dev

2022

O que são classificadores lineares?

Definição

Classificadores lineares são modelos que usam uma combinação linear das features para decidir a qual classe um certo data point pertence.

Vantagens

- Simplicidade no treino
- Geralmente interpretáveis
- Computacionalmente não são muito custosos
- Tendem a não ter problemas de overfitting

Notação

Vetor de features

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_p \end{bmatrix}$$

Vetor de coeficientes (pesos)

$$\mathbf{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_p \end{bmatrix}$$

Determinante de uma matriz

$$|A| = \det(A)$$

Um classificador linear básico

Para começarmos a entender como classificadores lineares funcionam, vamos criar um modelo simples, que serve para classificação binária (1 ou 0).

Defina a função

$$f(\mathbf{x}) = \begin{cases} 1, & \text{se } |\mathbf{w}^\intercal \mathbf{x}| \geq \theta \\ 0, & \text{se } |\mathbf{w}^\intercal \mathbf{x}| < \theta \end{cases}$$

Essa função já é um classificador linear!!

Problemas desse classificador

- Como interpretamos o modelo em termos probabilísticos?
  - Quão mais distante de θ $|\mathbf{w}^\intercal \mathbf{x}|$ estiver, mais provável a observação será de pertencer à classe 1?
  - Como essa probabilidade aumenta? Linearmente, exponencialmente?
- E se houvessem $n > 2$ classes?
  - Fariamos $\theta_1, ..., \theta_{n-1}$ parâmetros?
  - Qual deveria ser a distância entre cada θ_i e θ_{i-1}? (com $n > i > 1$)

Em algum grau, esses problemas são resolvidos pela **regressão logística**

Regressão logística

Vamos começar com a regressão logística para classificação binária. Considere duas classes: 1 e 0 (i.e., y $\in \{0, 1\}$).

Definição

A regressão logística para classificação binária é um modelo que calcula, baseando-se nas features do modelo, a probabilidade de uma observação pertencer a uma classe seguindo a seguinte fórmula:

$$P(y = 1|\mathbf{x}) = \frac{1}{1 + e^{-(w_0 + |\mathbf{w}^\intercal \mathbf{x}|)}} \tag{1}$$

Também, vale

$$P(y = 0|\mathbf{x}) = (1 - P(y = 1|\mathbf{x})) \tag{2}$$

Agora, note que $w_0 + |\mathbf{w}^\intercal \mathbf{x}|$ é o output de uma regressão linear. Daí, a fórmula pode ser deduzida quando aplicamos a função logística $h(z) = \frac{1}{1+\exp(-z)}$ em uma regressão linear em $\mathbf{x}$.

Se manipularmos (1), chegaremos a

$$\ln\left(\frac{P(y=1|\mathbf{x})}{1-P(y=1|\mathbf{x})}\right) = w_0 + |\mathbf{w}^\intercal \mathbf{x}| \tag{3}$$

Por (2), então,

$$\ln\left(\frac{P(y=1|\mathbf{x})}{P(y=0|\mathbf{x})}\right) = w_0 + w_1 x_1 + \cdots + w_p x_p \tag{4}$$

Sendo assim, a regressão logística é um modelo linear para $\ln\left(\frac{P(y=1|\mathbf{x})}{P(y=0)|\mathbf{x})}\right)$, chamado de "log-odds".

Como o modelo escolhe os coeficientes?

Agora que já sabemos como o modelo calcula as probabilidades, precisamos saber como ele faz a escolha dos coeficientes $w_1, \ldots, w_p$

O modelo maximiza uma função de verossimilhança (likelihood) ℓ dada por

$$\ell(w_0, w_1, \ldots, w_p) = \prod_{j=1}^{m} P(y_j = 1|\mathbf{x}_j)^{y_j}(1 - P(y_j = 1|\mathbf{x}_j))^{1-y_j} \quad (5)$$

Considerando um conjunto de treino com m observações, sendo y_j a variável que toma o valor da classe da j-ésima observação. Daí, $y_j \in \{0, 1\},\ \forall j \in \{1, \ldots, m\}$.

Regressão logística com mais de duas classes: One vs Rest

A regressão logística não serve apenas para classificação binária. Quando tivermos mais do que duas classes, podemos usar duas abordagens: a "One vs Rest" e a "Softmax". Começaremos falando da primeira abordagem.

Definição

Considere que temos $k > 2$ classes no nosso contexto. A abordagem **"One vs Rest"** consiste em treinar $k-1$ classificadores $f_1, \ldots, f_{k-1}$, onde f_1 classifica a classe 1 contra as classes $2, \ldots, k$, f_2 classifica 2 contra $1, 3, \ldots, k$, e assim por diante. Pontos não classificados em $f_1, \ldots, f_{k-1}$ são colocados na classe k.

Potencial problema: e se um datapoint for classificado em mais de uma classe?

Classificação linear com mais de duas classes: Softmax

Ao usarmos esse método para classificação de múltiplas classes, substituímos a função logística que $h(z)$ que comentamos pela **função softmax**.

Definição

Então, considerando que temos k classes ($k > 2$) a probabilidade de uma observação pertencer à classe $j \in \{1, \ldots, k\}$ é dada por

$$P(y = j|\mathbf{x}) = \frac{e^{\mathbf{w}_j^T \mathbf{x}}}{\sum_{l=1}^{k} e^{\mathbf{w}_l^T \mathbf{x}}} \tag{6}$$

Sendo assim, este modelo estima um **vetor de coeficientes** w **para cada classe** j, e estima um **vetor de probabilidades**

$\mathbf{P} = \begin{bmatrix} P(y = 1|\mathbf{x}) \\ \vdots \\ P(y = k|\mathbf{x}) \end{bmatrix}$ usando a equação (6) para cada probabilidade.

Implementação da Regressão Logística em Python: parâmetros

sklearn.linear_model.LogisticRegression

class sklearn.linear_model.**LogisticRegression**(*penalty='l2', *, dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None*) [source]

- "penalty": tipo de penalidade que colocamos nos coeficientes $w_1, \ldots, w_p$. Valores possíveis:
  - "none": nenhuma penalidade é adicionada
  - "l1": penaliza-se a norma ℓ_1 dos coeficientes, i.e., penaliza-se $||w||_1 = \sum_{i=1}^{p} |w_i|$
  - "l2" (default): penaliza-se a norma ℓ_2 dos coeficientes, i.e., penaliza-se $||w||_2 = \sqrt{\sum_{i=1}^{p} x_i^2}$
  - "elasticnet": termos penalizando tanto a norma ℓ_1 quanto a norma ℓ_2 são adicionados
- "C": aqui, $C = \frac{1}{\lambda}$, i.e., C é o inverso da "força" da regularização. Quão maior for este parâmetro, menor será a "força" da regularização.

Implementação da Regressão Logística em Python: parâmetros

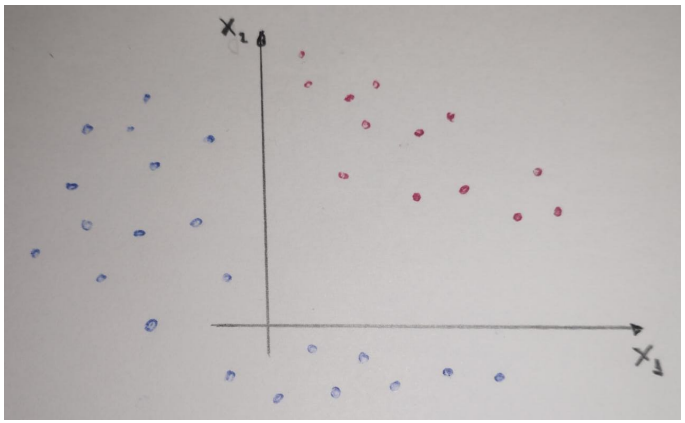
sklearn.linear_model.LogisticRegression

class sklearn.linear_model.**LogisticRegression**(*penalty='l2', *, dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None*) [source]

- "fitintercept": (default = True): caso seja igual a "True", vamos colocar um intercepto w_0 nos coeficientes (muitas vezes chamado de viés).
- "solver": algoritmo usado para o problema de otimização.
- "multiclass" (default = "auto"): se escolhermos "ovr", usamos One vs Rest. Se escolhermos "multinomial", usaremos Softmax. Caso usemos "auto", será usado One vs Rest quando os dados forem binários ou quando usarmos solver = "liblinear": do contrário, será usado "multinomial".

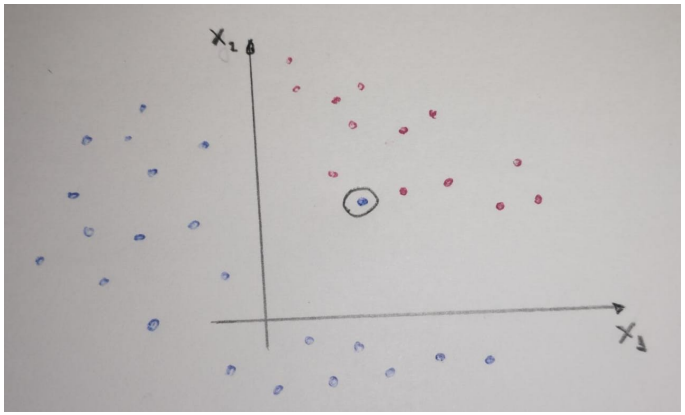
SVM: intuição

Imagine que temos duas classes - 0 e 1 - e duas features x_1 e x_2 (pontos azuis são da classe 0 e pontos vermelhos são da classe 1), representados abaixo:

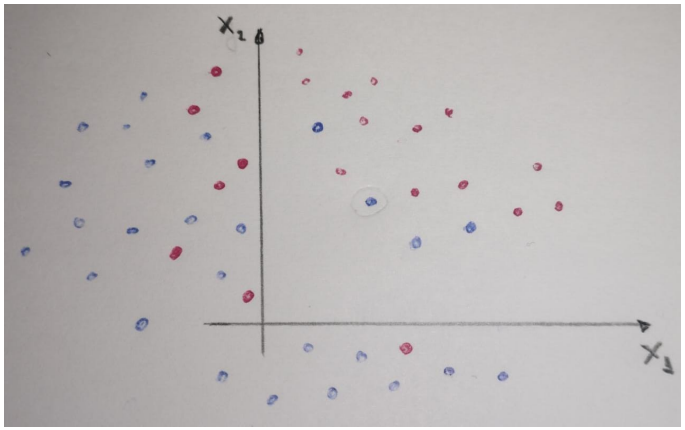


Note que podemos construir um classificador traçando uma reta que separe a classe 0 da classe 1. Pontos abaixo desta reta serão classificados como sendo da classe 0, e pontos acima serão classificados como sendo da classe 1. Mas, e se houvesse um outlier?

SVM: intuição



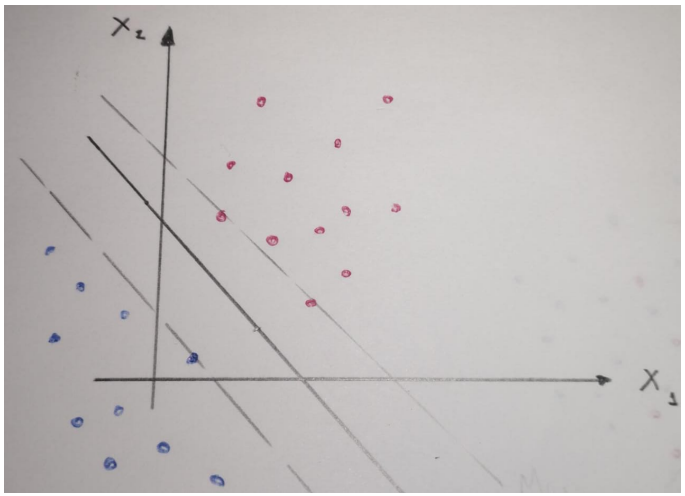
E se as classes estivessem mais misturadas?



SVM: intuição

Nessa situação, a reta que iremos traçar deve aceitar erros de classificação no conjunto de treino. Caso contrário, esse classificador não seria robusto - levando a problemas de overfitting.

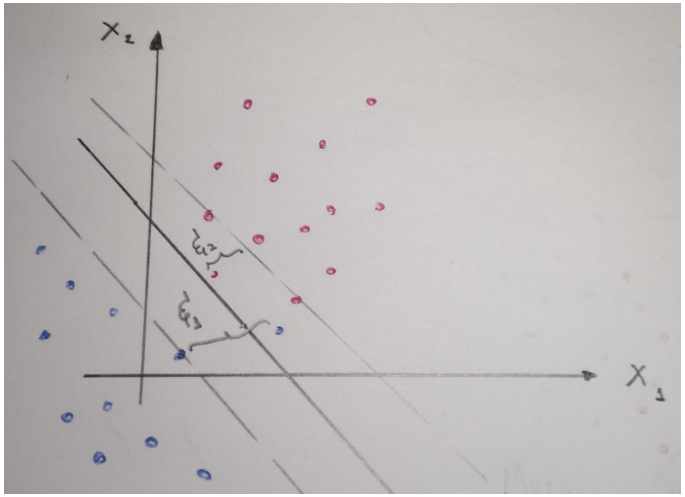
Para fins didáticos, considere outro conjunto de treino, com a mesma ideia do anterior. Já traçamos a reta que separa as duas classes:



A distância entre as duas linhas pontilhadas é aquilo que chamamos de **margem**.

SVM: intuição

Note que a reta que separa as classes é tão apropriada que, mesmo se aparecessem alguns outliers, não haveriam muitos motivos para mudar essa reta:



Aqui, os termos ξ_1 e ξ_2 são as distâncias dos pontos às suas respectivas margens. Note que apesar do ponto vermelho associado a ξ_2 estar corretamente classificado, ainda existe um erro associado a ele. Isto se deve ao fato de que nós não mudamos a margem ao ter um novo ponto vermelho no conjunto de treino.

Support Vector Classifier

Agora que já temos a intuição de como funciona um SVM, vamos formalizar um pouco e entender como o classificador funciona. De início, vamos listar os ingredientes que o nosso classificador precisa:

- Queremos a maior margem possível
- Dada a quantidade e tamanho dos erros que toleramos
- Que classifique corretamente o máximo que puder no conjunto de treino

Estes três ingredientes são suficientes para podermos expressar o problema em termos formais. Agora, considere que estamos em um contexto com p features, e queremos achar o subespaço afim de dimensão p que satisfaça as três condições dadas acima (não se assuste com a terminologia. Em dimensão $p = 2$, esse "subespaço afim" será uma reta. Em dimensão $p = 3$, esse "subespaço afim" será um plano. Em dimensões $p > 3$, esse "subespaço afim" será um **hiperplano**).

Support Vector Classifier

Definição

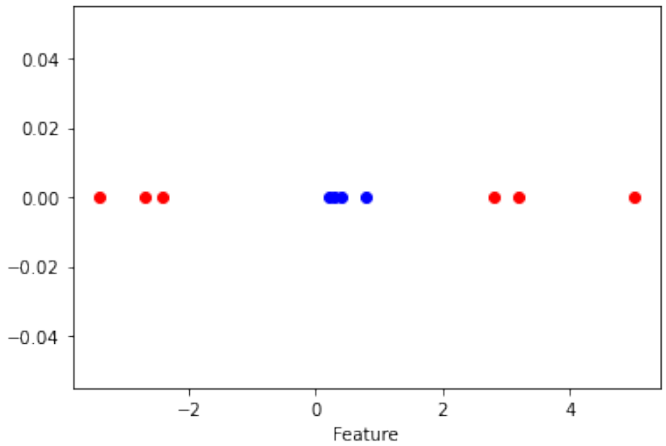
Seja M a margem e um conjunto de treino com m observações, p features e k classes. Então, o **SVC** escolhe um subespaço afim de dimensão p para separar as classes resolvendo o seguinte problema de otimização:

$$\begin{aligned} \max_{\beta_0,\beta_1,\ldots,\beta_p,\xi_1,\ldots,\xi_n} \quad & M \\ \text{s.a.} \quad & \sum_{j=1}^{p} \beta_j^2 = 1 \\ & y_i(\beta_0 + \beta_1 x_{1i} + \cdots + \beta_p x_{ip}) \geq M(1 - \xi_i) \\ & \xi_i \geq 0, \ \sum_{i=1}^{n} \xi_i \leq C \end{aligned} \tag{7}$$

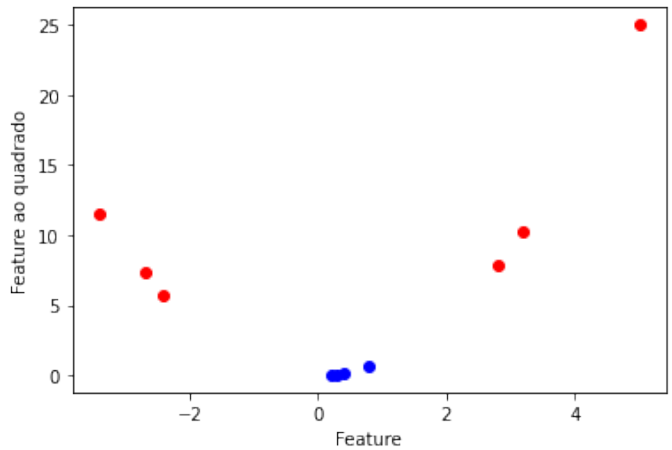
onde C é um hiperparâmetro que controla o tamanho dos erros.

Não linearidade e o Kernel: intuição

E se os dados não forem linearmente separáveis??



Se elevarmos ao quadrado...



Kernel

Quando tivermos dados que não são linearmente separáveis em uma dimensão p qualquer, podemos usar o "truque" do *kernel*, para que possamos fazer isso em uma dimensão maior que p.

O kernel é alguma função matemática K que usamos para transformar os dados de input em outra forma necessária para o processamento.
Algumas funções Kernel bem utilizadas:

- Kernel gaussiano: $K(x, y) = e^{-\frac{||x-y||^2}{2\sigma^2}}$
- Kernel sigmóide: $K(\mathbf{x}, y) = tanh(\gamma \mathbf{x}^\intercal y + r)$
- Kernel polinomial: $K(\mathbf{x}, y) = (\gamma \mathbf{x}^\intercal y + r)^d$

Referências

- Hastie, Trevor, et al. An Introduction to Statistical Learning: With Applications in R. Springer New York, 2013.
- scikit-learn: machine learning in Python — scikit-learn 1.1.2 documentation, https://scikit-learn.org/stable/. Accessed 4 August 2022.
- Yang, Sophia. "Multiclass logistic regression from scratch — by Sophia Yang." Towards Data Science, 18 April 2021, https://towardsdatascience.com/multiclass-logistic-regression-from-scratch-9cc0007da372. Accessed 4 August 2022.
- "Principais funções do kernel na máquina de vetores de suporte (SVM) – Acervo Lima." Acervo Lima, https://acervolima.com/principais-funcoes-do-kernel-na-maquina-de-vetores-de-suporte-svm/. Accessed 5 August 2022.