UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE MATEMÁTICA, ESTATÍSTICA E COMPUTAÇÃO CIENTÍFICA
DEPARTAMENTO DE MATEMÁTICA APLICADA

Beatriz Belucci

Introdução à computação científica: aprendizagem de máquina e algoritmos de classificação[1]

Monografia apresentada ao Instituto de Matemática, Estatística e Computação Científica da Universidade Estadual de Campinas como parte dos requisitos para obtenção de créditos na disciplina Projeto Supervisionado, sob a orientação do(a) Prof. Ricardo Biloti.

Campinas
2021

[1]Este trabalho foi financiado pelo SAE.

Conteúdo

1 Introdução

Este projeto foi planejado com o objetivo de realizar o estudo da Computação Científica, através do estudo de modelagens matemáticas e ferramentas teóricas que auxiliassem na resolução de um problema selecionado. Para tal, foi escolhido o tema Inteligência Artificial como área de estudo, focando particularmente nos algoritmos de Aprendizagem de Máquina. Dentre esses algoritmos, os estudados foram os algoritmos de classificação.

Para desenvolver o projeto, foi utilizado como material principal de estudo o livro de Kubat (2017), *An Introduction to Machine Learning*. O material apresenta os fundamentos da aprendizagem de máquina, enfatizando os métodos de classificação, exibindo a teoria relacionada a diferentes métodos classificadores, assim como formas de implementar seus algoritmos e aplicações. Foram estudados o Classificador Bayesiano e o método dos k-Vizinhos Mais Próximos.

Quando necessário, foram realizadas leituras complementares de outras referências, a fim de aprimorar o aprendizado. Para o estudo de probabilidades foi utilizado o texto de Ross (2009). Já para auxiliar na implementação dos algoritmos em Octave, foi utilizado o livro Quarteroni e Saleri (2007); e as vídeo aulas Biloti (2020a) e Biloti (2020b).

2 Fundamentos da aprendizagem de máquina

Nesta seção serão apresentados os conceitos básicos sobre aprendizagem de máquina que serão necessários para a compreensão dos métodos apresentados nas próximas seções.

Nos algoritmos de classificação, queremos que o algoritmo receba um objeto de classe desconhecida e que seja seja capaz de nos retornar a classe a qual esse objeto pertence. Para isso, é necessário, primeiramente, que o algoritmo saiba como realizar uma classificação. Vejamos então, como o algoritmo aprende a classificar.

Para induzir um classificador, é preciso, antes de mais nada, treinar o algoritmo. Isso é feito através do *conjunto de treinamento*. Os elementos do conjunto de treinamento, por estarem já classificados, são explorados na determinação de padrões. Esses padrões são, numa etapa de classificação, empregados na inferência da classe sempre que um novo elemento for analisado. Isso permite que o algoritmo consiga, quando receber um objeto de classe desconhecida, analisar com quais elementos do conjunto de treinamento ele melhor se assemelha e, consequentemente, a qual classe ele tem maiores chances de pertencer.

Para que o algoritmo seja capaz de transformar as informações que recebeu em conhecimento, é necessário que essas informações sejam passadas a ele de maneira adequada. Podemos transmitir essas informações utilizando vetores, os vetores de atributos. No conjunto de treinamento podemos extrair muitas informações a respeito de seus elementos. Entretanto os vetores de atributos armazenam apenas as características que julgamos serem suficientes para a decisão de pertencimento a uma classe ou outra. Essas características podem ser tanto discretas como contínuas.

Uma maneira de testar se o classificador induzido está correto é dividir o conjunto de objetos

conhecidos em dois subconjuntos: conjunto de treinamento e conjunto de teste. Fazendo isso, podemos utilizar o conjunto de treinamento para induzir o classificador e o conjunto de teste para verificar se as classificações estão corretas, umas vez que essas classes eram previamente conhecidas. Vejamos agora métodos que induzem classificadores.

3 Classificador Bayesiano

O classificador Bayesiano é um método que utiliza a teoria probabilística Bayesiana para calcular a probabilidade de um objeto pertencer a uma determinada classe. O algoritmo atribui ao objeto a classe com maior probabilidade.

Para tal implementação, alguns conceitos de probabilidade são necessários. São eles: cálculo de probabilidade utilizando frequência relativa, probabilidade condicional e fórmula de Bayes.

Segundo Ross (2009), dado dois eventos A e B, a probabilidade do evento A ocorrer, sabendo que o evento B ocorreu, é chamada de probabilidade condicional e denotada por $P(A|B)$. Podemos estimar tal probabilidade a partir da fórmula de Bayes

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}, \tag{1}$$

onde $P(B)$ representa a probabilidade do evento B ocorrer e $P(A)$ representa a probabilidade do evento A ocorrer.

Por fim, para estimar as probabilidades $P(B)$ e $P(A)$, podemos utilizar a frequência relativa dos eventos A e B no espaço amostral. Ou seja, tome um conjunto com m elementos, nos quais n elementos representam o evento B e p elementos representam o evento A, com $n, p \leq m$. A probabilidade dos eventos A e B ocorrerem é estimada a partir de sua frequência relativa como

$$P(B) = \frac{n}{m}, \tag{2}$$

$$P(A) = \frac{p}{m}. \tag{3}$$

Como estimar $P(B|A)$ será visto em detalhes nas seções seguintes, onde empregaremos a fórmula de Bayes no contexto dos algoritmos classificadores.

Sabendo-se como calcular as probabilidades, podemos utilizá-las no algoritmo de classificação Bayesiano. Para esse algoritmo, vamos dividir sua aplicação em duas partes: quando os elementos dos vetores de atributos são discretos e quando são contínuos.

3.1 Vetores de atributos discretos

Tome um conjunto de treinamento, no qual os objetos são descritos por vetores de atributos discretos $\mathbf{x} = (x_1, \ldots, x_n)$. Cada elemento x_j do vetor $\mathbf{x}$ pertence ao conjunto $R_j = \{1, \ldots, L_j\}$.

Assim, L_j é o número possível de "estados" para o atributo x_j. Supondo que os objetos podem ser de i classes distintas, então a probabilidade de um objeto ser da i-ésima classe dado que ele é descrito pelo vetor **x**, é estima pela fórmula de Bayes

$$P(c_i|\mathbf{x}) = \frac{P(\mathbf{x}|c_i)P(c_i)}{P(\mathbf{x})}, \tag{4}$$

onde, c_i é o conjunto de objetos pertencentes à classe i. Já $P(c_i)$ é a probabilidade de que em uma escolha aleatória dentro do espaço amostral, tenhamos um objeto pertencente à classe i e $P(\mathbf{x})$ é a probabilidade de que, novamente, em uma escolha aleatória dentro no conjunto amostral, tenhamos o vetor **x**.

Usando o conjunto de treinamento, que é um subconjunto do espaço amostral, não podemos calcular tais probabilidades, mas podemos estimá-las a partir das frequências relativas da classe i e do vetor de atributos **x** no conjunto de treinamento. Já para estimarmos $P(\mathbf{x}|c_i)$, não utilizamos a frequência relativa, pois dentro do conjunto de treinamento pode não haver representatividade de um determinado vetor de atributos. Isso faz com que, ao calcularmos a probabilidade condicional de **x** dado c_i, ela seja igual a 0 para toda classe, anulando o numerador da fórmula de Bayes e impossibilitado a escolha da classe mais provável. Entretanto, a falta de representatividade de um atributo individual ocorre com menos frequência, uma vez que há menos valores possíveis de cada atributo quando comparado aos vetores de atributos, posto que, os vetores são formados por combinações de atributos individuais.

Em virtude de tais adversidades, iremos utilizar a *suposição ingênua de Bayes*. Ao utilizar a suposição ingênua de Bayes, estamos assumindo que todos os atributos que descrevem os objetos do conjunto de treinamento são mutualmente independentes, ou seja, os elementos do vetor de atributos não têm relação entre si. A partir disso, temos uma fórmula que combina a probabilidade individual de cada atributo com a probabilidade de um dado vetor

$$P(\mathbf{x}|c_i) = \prod_{j=1}^{n} P(x_j|c_i). \tag{5}$$

onde, com certo abuso de notação, $P(x_j|c_i)$ representa a probabilidade de sortear do conjunto dos objetos da classe i um objeto tal que sua j-ésima coordenada seja x_j.

Logo, unindo (4) e (5), temos:

$$P(c_i|\mathbf{x}) = \prod_{j=1}^{n} P(x_j|c_i) \cdot \frac{P(c_i)}{P(\mathbf{x})}. \tag{6}$$

Note que, para efeito de comparação, o denominador da equação (6) é o mesmo para todas as classes c_i. Com isso, podemos omitir seu cálculo, calculando apenas o numerador. Além disso, $P(x_j|c_i)$ pode ser estimada a partir da frequência relativa do atributo x_j na classe i. Por fim, temos então, que o algoritmo do classificador Bayesiano irá atribuir ao objeto a classe que maximar esse

numerador.

3.2 Vetores de atributos contínuos

Olhando agora para o caso em que os atributos são representados por valores contínuos. Temos então, que o vetores de atributos são da forma $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$. Nessa situação, o calculo de probabilidades fica inviável de ser feito utilizando-se frequências relativas. Isso se dá ao fato de que o número de valores distintos para um determinado atributo pode ser infinito. Logo, a partir da frequência relativa, as probabilidades podem ser infinitesimalmente pequenas. Para solucionar tal dificuldade, iremos introduzir o conceito de *função densidade de probabilidade* (FDP).

Tal função surge do processo de discretização do domínio dos atributos. Podemos discretizá-los subdividindo o conjunto de treinamento em intervalos e separando os vetores dentro desses intervalos. Em uma situação ideal, se temos um conjunto de treinamento infinitamente grande, podemos reduzir o tamanho dos intervalos até que esses sejam infinitesimalmente pequenos. Com isso, construímos uma função contínua $p(x) \in [0, 1]$, a qual assume um alto valor se há muitos objetos descritos pelo atributo x e um baixo valor caso haja poucos objetos descritos pelo atributo x.

Com a FDP podemos utilizar a fórmula de Bayes também em vetores contínuos. Temos então

$$P(c_i|\mathbf{x}) = \frac{p_{ci}(\mathbf{x})P(c_i)}{p(\mathbf{x})}, \tag{7}$$

onde, $p_{ci}(\mathbf{x})$ e $p(\mathbf{x})$ são, respectivamente, a FDP criada a partir dos elementos do conjunto de treinamento que pertencem a classe i e a FDP criada a partir de todos os elementos do conjunto de treinamento. Além disso, $P(c_i)$ é a probabilidade de que em uma escolha aleatória dentro do espaço amostral, tenhamos um objeto pertencente à classe i. Essa probabilidade pode ser estimada a partir da frequência relativa da classe i no conjunto de treinamento. Novamente partindo da suposição ingênua de Bayes, podemos calcular $p_{ci}(\mathbf{x})$ a partir da equação

$$p_{ci}(\mathbf{x}) = \prod_{j=1}^{n} p_{ci}(x_j), \tag{8}$$

onde, com certo abuso de notação, $p_{ci}(x_j)$ é a FDP criada a partir do conjunto c_i dos objetos pertencentes à classe i sendo avaliada no ponto x_j, que é o j-ésimo elemento do vetor de atributos.

Com isso, podemos reescrever (7) como

$$P(c_i|\mathbf{x}) = \prod_{j=1}^{n} p_{ci}(x_j) \cdot \frac{P(c_i)}{p(\mathbf{x})}. \tag{9}$$

Assim como para o caso discreto, não é necessário que seja feito o cálculo de $p(\mathbf{x})$, basta olharmos

para o numerador da equação.

Para aproximar uma FDP podemos utilizar, por exemplo, a *distribuição normal*. Então,

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \tag{10}$$

onde, μ é o ponto onde o máximo da curva está localizado e representa o valor médio dos dados e σ é o parâmetro *variância*, que representa a medida de dispersão dos dados. Esses parâmetros podem ser estimados, para o caso dos vetores de atributos, através das aproximações

$$\mu_j \approx \frac{1}{m} \sum_{j=1}^{m} x_j^d, \tag{11}$$

$$\sigma_j^2 \approx \frac{1}{m-1} \sum_{j=1}^{m} (x_j^d - \mu)^2, \tag{12}$$

onde, m é o número de elementos no conjunto c_i e x_j^d representa o j-ésimo atributo do vetor **x**, em que **x** é um vetor de atributos que descreve um objeto da classe i. Para calcular $p_{ci}(x_j)$ vamos utilizar (10), então

$$p_{ci}(x_j) = k \cdot e^{-\frac{(x_j-\mu)^2}{2\sigma^2}}. \tag{13}$$

Vale notar que, se temos vetores de atributos com n elementos, devemos calcular (13) n vezes, uma para cada atributo.

4 Classificador k-NN

O método dos k-vizinhos mais próximos (k-NN) utiliza as distâncias geométricas entre pontos no espaço para realizar as classificações. O algoritmo atribui ao objeto, a classe com maior representatividade entre k pontos mais próximos.

Se temos vetores de atributos em $\mathbb{R}^n$, podemos representá-lo através de um ponto no espaço n dimensional. A partir disso, podemos, através dos pontos que representam os elementos do conjunto de treinamento, medir as similaridades entre tais ponto e o ponto que representa o objeto ao qual queremos classificar. Uma maneira de calcular essas similaridades é calcular a distância geométrica entre tais pontos. Para isso podemos utilizar a métrica

$$D_M(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n} d(x_i, y_i)}. \tag{14}$$

Para o caso em que os elementos do vetor de atributos são discretos, podemos medir as similaridades entre dois vetores **x** e **y** distintos apenas comparando seus elementos. Logo, para o caso

discreto,

$$\begin{cases} d(x_i, y_i) = 1, \text{ se } x_i \neq y_i, \\ d(x_i, y_i) = 0, \text{ se } x_i = y_i. \end{cases} \tag{15}$$

Para o caso em que os elementos do vetor de atributos são contínuos, temos a métrica Euclidiana usual, então o argumento do somatório de (15) é dado por

$$d(x_i, y_i) = (x_i - y_i)^2. \tag{16}$$

Axiomas de distâncias:

- $D_M(\mathbf{x}, \mathbf{y}) \geq 0 \ \forall \mathbf{x}, \mathbf{y}$
- $\mathbf{x} = \mathbf{y} \Leftrightarrow D_M(\mathbf{x}, \mathbf{y}) = 0$
- $D_M(\mathbf{x}, \mathbf{y}) = D_M(\mathbf{y}, \mathbf{x})$
- $D_M(\mathbf{x}, \mathbf{y}) + D_M(\mathbf{y}, \mathbf{z}) \geq D_M(\mathbf{x}, \mathbf{z})$

Calculadas as distâncias, escolhemos k vetores mais próximos do ponto que representa o objeto que queremos classificar. Feito isso, olhamos para as classes desses k objetos. A classe que possuir maior representação, é a classe atribuída ao objeto. Entretanto, cabe aqui ressaltar que (13) não deve ser aplicada mecanicamente uma vez que as informações que os atributos carregam, devem ser levadas em consideração. Podemos citar dois fatores: a presença de atributos irrelevantes e problemas de escala.

Alguns atributos que descrevem os objetos podem ser irrelevantes. Isso que significa que sua informação não tem significância para a classe a qual o objeto pertence. Contudo, sua presença afeta a distância geométrica entre dois vetores, podendo perturbar a classificação. Já os problemas de escala ocorrem quando o valor de um atributo é muito maior quando comparado aos outros atributos. Isso faz com que sua contribuição seja dominante no cálculo da distância geométrica.

Para mitigar o problema de atributos irrelevantes, uma possibilidade é o pré processamento dos dados. Uma análise dos dados disponíveis por quem está implementando o algoritmo k-NN é fundamental para identificar tais atributos. Para o problema de escala, podemos normalizar os atributos, impondo que todos eles pertençam ao intervalo [0,1].

A normalização é aplicada a cada um dos vetores de atributos **x** do dado de treinamento e, posteriormente, do dado de classificação. Para realizar essa normalização podemos utilizar a equação

$$q_i = \frac{x_i - MIN}{MAX - MIN}, \tag{17}$$

onde x_i é o i-ésimo atributo do vetor de atributos, MIN representa o menor valor do atributo x_i no conjunto de treinamento e MAX, o maior valor. Com isso, construímos o vetor **q**, que representa o vetor de atributos normalizado.

Na atribuição das classes, todos os k-vizinhos mais próximos têm o mesmo peso de votação. Ou seja, estamos levando em consideração apenas a quantidade de vizinhos de uma determinada classe. Em decorrência disso, não consideramos, por exemplo, a seguinte situação: dado duas classes A e B, dentre os k-vizinhos mais próximos, a maioria deles é da classe A, mas os vizinhos da classe B estão geometricamente mais próximos do objeto ao qual queremos classificar, logo, tendem a ser mais semelhantes a ele. Para levar em consideração essa situação, podemos utilizar o método k-NN ponderado.

Nesse método, cada objeto do conjunto de treinamento terá um peso w_i. Esse peso será inversamente proporcional a sua distância ao objeto que estamos classificando, ou seja, quanto maior a distância, menor o peso. Para computar esses pesos, primeiro devemos ordenar as distâncias dos k-vizinhos mais próximos, $d_1, \ldots, d_k$. Sendo d_1 a menor distância e d_k a maior. Temos então,

$$w_i = \begin{cases} \frac{d_k - d_i}{d_k - d_1}, \text{ se } d_k \neq d_1, \\ 1 \quad \text{, se } d_k = d_1. \end{cases} \tag{18}$$

Note que, $w_k = 0$. Com isso, temos que apenas $k-1$ vizinhos são considerados. Logo, o método k-NN só faz sentido quando temos $k > 3$, posto que, se $k \leq 3$, o k-NN ponderado irá degenerar.

Com os pesos calculados, a classificação ocorre da seguinte forma. Primeiro devemos separar os objetos do conjunto de treinamento por classes e, em seguida, somar os pesos de cada classe. Se temos, por exemplo, a classe A e a classe B, onde w_1, w_2 e w_3 são os pesos dos objetos da classe A e w_4, w_5 e w_6 são os pesos correspondentes à classe B. Então, o peso de cada classe é dado por

$$W_A = \sum w_j \delta_A(\mathbf{x}^j) \tag{19}$$

$$W_B = \sum w_j \delta_B(\mathbf{x}^j), \tag{20}$$

onde $\delta_A(x) = 1$ se $x \in A$ e 0 se $x \notin A$, e o da mesma forma para δ_B. O objeto recebe a classe que maximizar essa somatória.

5 Resultados e discussão

Para cada um dos três métodos de classificação citados nas seções anteriores, foram criados algoritmos para implementá-los. Os códigos podem ser vistos na seção Apêndice. Nesta seção, são apresentados os exemplos criados para ilustrar os diferentes classificadores, bem como os resultados das classificações realizadas em cada algoritmo.

Para ilustrar o classificador Bayesiano para atributos discretos, empregamos o algoritmo para classificar animais em duas classes, os "selvagens" e os "domésticos". Para tal, foi criado um conjunto contendo 26 animais de classes conhecidas. Cada animal foi descrito por um vetor com quatro características: pele, tamanho, alimentação e tipo. Foi criado ao todo 13 tipos de atributos. Para o atributo "pele" temos: pelo, escama e pena. Para o atributo "tamanho": pequeno e grande. Para "alimentação": carnívoro, herbívoro, onívoro e insetívoro. E, por fim, para "tipo", temos: ave, mamífero, réptil e peixe. A título de exemplo, temos o animal "Cachorro", que é descrito pelo vetor $\mathbf{x} =$ (pelo, pequeno, carnívoro, mamífero) e pertence à classe "domésticos".

Para calcular as probabilidades necessárias para a implementação do métodos, foram escolhidos aleatoriamente 14 animais. Em seguida, foi utilizada a fórmula de Bayes para classificar os 12 animais restantes. Para que o algoritmo pudesse funcionar com praticidade, foram atribuídos valores numéricos para representar os atributos e as classes. Cada atributo recebeu um número inteiro entre 1 e 13 e as classe receberam 1 ou 2. A classe "domésticos" foi representada pelo número 1 e a classe "selvagens" pelo número 2.

Para esse método, 12 objetos foram classificados. O algoritmo classificou corretamente 7 deles, resultando em aproximadamente 58% de acerto. Apesar de mais da metade dos objetos terem sido corretamente classificados, tal resultado não é o ideal, uma vez que queremos sempre classificar corretamente o maior número possível de objetos.

Tal resultado por ser justificado com a análise de alguns fatores. O primeiro deles é a escolha dos atributos que descrevem os objetos. Os atributos tomados como características suficientes para a decisão de pertencimento a uma classe ou outra, podem não ter sido relevantes o bastante para o conjunto de dados disponível. Além disso, como o conjunto de treinamento foi criado escolhendo-se aleatoriamente os vetores de atributos, pode não ter havido representatividade o suficiente para determinado vetor, ou seja, algum vetor de atributos pode não ter sido representado dentro do conjunto ou não ter sido representado o bastante. Isso causa uma pertubação na classificação, pois afeta diretamente a estimativa das probabilidades.

Uma maneira de aprimorar a solução, seria a subdivisão do conjunto de dados disponível em diversos conjuntos de treinamento distintos. Isso faz com que, a cada divisão, haja uma renovação dos elementos que serão usados para treino, ocasionando diferentes representações dos vetores de atributos. Cada conjunto de treinamento resultará em diferentes probabilidades e, consequentemente, possíveis classificações distintas. Com isso, é possível analisar qual subconjunto resulta na menor taxa de erro e então, utilizá-lo nas classificações.

Já para o classificador Bayesiano para atributos contínuo, para criar o conjunto de treinamento e

o conjunto de teste, foi utilizado Fisher (1936). Nesse conjunto de dados temos 150 flores *Iris* de 3 espécies distintas: *Iris Setosa*, *Iris Versicolor* e *Iris Virginica*. Cada flor é descrita por um vetor com 4 atributos: comprimento da sépala, largura da sépala, comprimento da pétala e largura da pétala. Dentre esses 150 exemplos, 90 deles foram utilizados para treino e os 60 restantes para teste. Para o uso do algoritmo, cada classe foi representada por um valor numérico. Em ordem de citação, temos as classes 1, 2 e 3.

Para esse caso, o classificador Bayesiano apresentou bons resultados. Dentre 60 classificações, 58 delas foram bem sucedidas, resultando em aproximadamente 96% de acerto. Essa melhoria, quando comparado ao caso de vetores discretos, pode ser resultado do uso de um conjunto de dados melhor, ou seja, um conjunto de dados maior e com atributos mais representativos. Como uma grande quantidade de elementos foi utilizada para treino, as probabilidades puderam ser estimadas com maior acurácia. Todos esses fatores influenciam na qualidade da classificação.

Para o método k-NN foi empregado o mesmo conjunto de dados utilizado no método do classificador Bayesiano para criar o conjunto de treinamento e o conjunto de teste. Esses dados foram preparados para a implementação do algoritmo da mesma forma como feito anteriormente. Foi utilizado $k = 6$. Esse método também classificou corretamente 58 dos 60 objetos, obtendo aproximadamente 96% de acerto. O fato de ter sido utilizado um bom conjunto de dados também pode ser mencionado aqui.

6 Considerações finais

Em conclusão, podemos notar, a partir do resultados obtidos para os diferentes métodos, que o pré processamento dos dados é fundamental para uma boa classificação. Um conjunto de dados com um número significativo de elementos e um conjunto de treino representativo e com atributos relevantes, permite que as estimativas dos dados necessários a cada método sejam feitas de maneira menos perturbada, gerando resultados melhores.

Ademais, podemos notar que tanto o classificador Bayesiano para o caso contínuo, quanto o método k-NN obtiveram os mesmos resultados. Além do já mencionado, tal semelhança pode ter relação com a similaridade de precisão de cada método.

Em relação ao projeto, a pesquisa ainda não está finalizada. O estudo dos métodos de classificação continuará sendo feito e, nos próximos meses, pretende-se estudar o método dos classificadores lineares e polinomiais, bem como redes neurais artificiais.

7 Apêndice

```
function [P_A,P_B,P_a,P_b] = probabilidades_discreto(A,B)

    % function [P_A,P_B,P_a,P_b] = probabilidades_discreto(A,B) calcula as probabilidades
    % necessarias para a aplicacao do metodo do classificador Bayesiano para atributos
    % discretos. Tem como entrada as matriz A e B que armazenam os objetos do conjunto de
    % treinamento de duas classes distintas.
    % Como saida temos:
    % P_a : probabilidade de ser da classe A.
    % P_b : probabilidade de ser da classe B.
    % P_A : frequencia relativa dos atributos no conjunto A.
    % P_B : frequencia relativa dos atributos no conjunto B.

    [m,n] = size(A);
    [j,k] = size(B);

    % m: total de objetos da classe A
    % j: total de objetos da classe B
    % n,k: total de atributos

    % Frequencia relativa de cada classe %

    P_a = m/(m + j); % probabilidade de ser da classe A
    P_b = j/(m + j); % probabilidade de ser da classe B

    % Frequencia relativa de cada atributo em cada um dos conjuntos A e B %

    for i=1:13
        P_A(i,1) = (length(find(A==i)))/m;
        P_B(i,1) = (length(find(B==i)))/j;
    endfor

    % Com a funcao find() encontramos nas matrizes A e B quantos
    % elementos representam o atributo i e retorna um vetor com tais
    % elementos. A funcao length() retorna quantos elementos a funcao
    % find() encontrou. Entao no loop estamos contando quantos atributos
    % de cada tipo existem nas matrizes e dividindo pelo valor de
    % atributos.

endfunction
```

Listing 1: Cálculos das probabilidades para o caso discreto do classificador Bayesiano.

```
function [Classificacao] = bayesiano_discreto(A,B,T)

    % function [Classificacao] = bayesiano_discreto(A,B,T) e um algoritmo para o classificador
    % Bayesiano.Ela tem como entrada as matrizes A e B, que armazenam os vetores de atributos
    % dos objetos da classe A e B, respectivamente. Recebe tambem, a matriz T
    % que armazena o conjunto de teste.

    [P_A,P_B,P_a,P_b] = probabilidades_discreto(A,B)


    [s,t] = size(T)

    % s: numero de objetos que serao classificados
    % t: numero de atributos


    for i = 1:s
        G(i,1) = prod(P_A(T(i,:)));
        G(i,2) = prod(P_B(T(i,:)));
    endfor

    % O loop calcula o produto das frequencias relativas dos atributos da
    % matriz T. A matriz G armazena na primeira coluna esse produto referente a
    % classe A e na segunda coluna referente a classe B


    %  Calculando o numerador da formula de Bayes %

    G(:,1) = G(:,1)*P_a;
    G(:,2) = G(:,2)*P_b;

    % Calculado a classe e a porcentagem de confianca da classificacao %

    r = G(:,1) + G(:,2);

    % r: soma das probabilidade ser da classe A e da classe B

    Porcentagem = (G./r)*100;

    % Porcentagem: porcentagem de confianca da classificacao

    [P,C] = max(Porcentagem(k,:);

    % P: maior porcentagem entre as duas classes
    % C: classes associadas a P

    Classificacao = [C P];

    % Classificacao: saida da funcao. Retorna C e P

endfunction
```

Listing 2: Classificador Bayesiano para atributos discretos.

```
function [n,P,M,V] = probabilidades_continuo(T)

    % function [n,P,M,V] = probabilidades_continuo(T) calcula as probabilidades necessarias
    % para a aplicacao do metodo do classificador Bayesiano para atributos
    % continuos. Como entrada recebe a matriz T que armazena o conjunto de treinamento.
    % Como saida tem:
    % n: numero total de classes.
    % P: vetor onde cada elemento e frequencia relativa de uma classes
    % M: matriz onde o elemento (i,j) e a media do atributo j na classe i
    % V: matriz onde o elemento (i,j) e a variancia ao quadrado do
    % atributo j na classe i

    n = max(T(:,end)); % n:  numero de classes

    % Separando os exemplos de treinamento por classes %

    [d,e] = size(T(:,1:end-1));

    % Separando os objetos de treino por classes %

    for ii=1:n
        I=find(T(:,end)== ii);
        C{ii}=T(I,1:end-1);
        a(1,ii) = rows(C{ii});
    endfor

    % Em cada iteracao, o loop encontra na matriz T qual vetor p de
    % indice u pertence a classe q e armazena esse vetor em uma lista
    % C{q} Ao final temos 3 listas C{q} uma para cada classe
    % a : armazena, por coluna, o numero de elementos em cada classe

    % Calculando a frequencia relativa de cada classe %

    P = a./d;

    % Calculando a media e a variancia de cada atributo por classe %

    for t = 1:n
        M(t,:)= mean(C{t});
        V(t,:)= var(C{t}).^2;
    endfor
    % M: matriz onde o elemento (i,j) e a media do atributo j na classe i
    % V: matriz onde o elemento (i,j) e a variancia ao quadrado do
    % atributo j na classe i
endfunction
```

Listing 3: Cálculos das probabilidades para o caso contínuo do classificador Bayesiano.

```
function [classe] = bayesiano_continuo(T,x)

    % function [classe] = bayesiano_continuo(T,x) e um algoritmo para o metodo do
    % classificador Bayesiano para atributos continuos. Como entrada recebe a matriz T que
    % armazena o conjunto de treinamento e o vetor x , que
    % representa o vetor de atributos do objeto que queremos classificar.
    % Como saida temos a classe do objeto.

    [n,P,M,V] = probabilidades_continuo(T)

    % Calculando a Funcao Gaussiana %
    for t = 1: n
      k = 1./sqrt(2*pi*V(t,:));
      aux = ((x-M(t,:)).^2)./(2.*V(t,:));
      g = e.^(-aux);
      p_x(t,:) = k.*g;
    endfor

    % k: coeficiente de normalizacao
    % p_x: Gaussiana. Matriz onde o elemento (i,j) e a FDP do atributo i
    % na classe j

    z = prod(p_x');
    B = z.*P;
    [ii,jj] = max(B);
    classe = jj;
    % z: p_ci(x)
    % B: numerador da formula de Bayes
    % ii: o valor maximo do numerador da formula de Bayes
    % jj: saida da funcao. Classe associada a ii

endfunction
```

Listing 4: Classificador Bayesiano para atributos contínuos.

```
function [classe] = kNN_otimizado(T, x, k)
    % function [classe] = kNN_otimizado(T, x, k) implementa o metodo k-NN ponderado.
    % Tem como entrada uma matriz T que armazena o conjunto de treinamento, um vetor x de
    % atributos que descreve o objeto que queremos classificar e um escalar k  que representa
    % o numero de vizinhos que iremos analisar. Como saida temos a classe do objeto.

    ii = max(T(:,end));

    % ii armazena o numero de classes existentes no conjunto de treinamento

    [m,n] = size(T(:,1:end-1));

    % m armazena quantos objetos existem no conjunto de treinamento;
    % n armazena o numero de elementos do vetor de atributos.

    % Normalizacao do conjunto de treinamento e do objeto %

    maximo = max(T); % vetor com o maior valor de cada atributo
    minimo = min(T); % vetor com o menor valor de cada atributo

    for j=1: n
        T(:,j) =  (T(:,j) - minimo(j))/(maximo(j)- minimo(j));
        x(:,j) = (x(:,j) - minimo(j))/(maximo(j)-minimo(j));
    endfor

    % Cada coluna da matriz T e do vetor x foram normalizadas de acordo
    % com a equacao (16).

    % Calculando as metricas %

    for j = 1: m
        d(j,1) = norm(x-T(j,1:n));
    endfor

    % a j-esima linha do vetor d armazena a metrica entre o
    % vetor x e o j-esimo vetor da matriz T

    [D,I] = sort(d);

    % D armazena os elementos de d por ordem crescente e I armazena a
    % posicao que os elementos de D estavam em d.

    D = D(1:k);       % escolhe as k menores distancias
    u = I(1:k);       % seleciona as k primeiras posicoes

    % Calculando os pesos de cada atributo %

    if D(1) == D(end)
        W = ones(m,1);
    else
        W = (D(end)- D)/(D(end)- D(1));
    endif

    % w armazenas os pesos dos atributos.
    % Se a menor distancia e a maior sao iguais, todos os pontos estao a
    % uma mesma distancia, logo o peso nao e necessario e o vetor w passa
    % a ser um vetor de entradas iguais a 1.

```

```
59      C = T(u,end); % armazena as classes dos u-esimos elementos de T
60
61      % Separando os objetos de treino por classes %
62
63      for q=1:ii
64        p = find(T(u,end)== q);
65        C{q}=T(p,1:end-1);
66      endfor
67
68      % Em cada iteracao, o loop encontra na matriz T qual vetor p de
69      % indice u pertence a classe q e armazena esse vetor em uma lista
70      % C{q}. Ao final temos 3 listas C{q} uma para cada classe
71
72
73
74
75      classe1 = 0;  classe2 = 0; classe3 = 0; % iniciando variaveis
76
77      for h = 1:k
78          if C(h)==1
79            classe1 = classe1 + W(h);
80          endif
81          if C(h)==2
82            classe2 = classe2 + W(h);
83          endif
84          if C(h)==3
85            classe3 = classe3 + W(h);
86          endif
87      endfor
88
89      % Esse loop calcula a soma dos pesos de cada classe. Para cada valor
90      % de k procura nas listas as posicoes dos vetores, pois, por
91      % contrucao, a posicao do vetor na lista e a sua classe. Separa os
92      % pesos para efetuar a soma, de acordo com as classes.
93
94      % Classificando %
95
96      Y = [classe1,classe2,classe3];
97      [peso, classe] = max(Y);
98
99      % Y: vetor que armazena os pesos correspondentes a cada classe
100     % peso: valor maximo dos pesos
101     % classe: saida da funcao. Classe associada a variavel peso
102
103 endfunction
```

Listing 5: k-NN ponderado.

Referências

Biloti, R. (2020a). Introdução ao Octave – Parte I. https://www.youtube.com/watch?v=ryn5y53L3kc (acessado em 29/11/2020).

Biloti, R. (2020b). Introdução ao Octave – Parte II. https://www.youtube.com/watch?v=xwb0tHSs0Hc (acessado em 29/11/2020).

Fisher, R. (1936). Iris data set. https://archive.ics.uci.edu/ml/datasets/Iris (acessado em 02/03/2021).

Kubat, M. (2017). *An Introduction To Machine Learning*. Springer.

Quarteroni, A. M. e Saleri, F. E. (2007). *Cálculo Científico com MATLAB e Octave*. Springer.

Ross, S. (2009). *Probabilidade: um curso moderno com aplicações*. Bookman Editora.