**Página inicial do Tópico**

Os tópicos devem ser distribuídos por páginas, de acordo com a matriz do curso: mínimo de 2 e máximo de 5 páginas de HTML. Deve ter uma página inicial de cada curso, como será visto no final de tudo

**Início da pág. 2**

**Título do tópico:** O processo de classificação de dados

**Título da página:** Classificadores não descansam

**Subtítulo da página:** Uso cotidiano e uso institucional

**Introdução do tema do tópico (obrigatório)**

Contextualização do tema, apresentação dos cenários e problemas onde os conceitos são desenvolvidos e aplicados.

Talvez você não saiba, mas existe um classificador que trabalha para você 24 horas por dia, nos 7 dias da semana! Trata-se do filtro de spam de seu e-mail. Todo programa de filtragem de spam é, em essência, um modelo de classificação. Mais especificamente, um modelo que recebe como entrada um e-mail e a partir de seu texto e de outras informações, o classifica como “normal” ou “spam”. A figura a seguir ilustra o processo.

Inserir figura

Figura 1: Classificador para filtro de spam

Interface gráfica do usuário, Diagrama

Descrição gerada automaticamente

Classificação é o processo de ciência de dados que consiste em utilizar um modelo gerado por um programa de computador para **determinar a classe de um objeto** de maneira automática. Este modelo é chamado de **modelo de classificação** ou simplesmente **classificador**.

Todo classificador possui o mesmo princípio de funcionamento do filtro de spam, independente do problema prático que ele esteja resolvendo: ele recebe um objeto como entrada, analisa as características do objeto e decide qual é a sua classe, considerando um conjunto de classes pré-definidas. No caso do filtro de spam, o objeto a ser a ser classificado é uma mensagem eletrônica e o conjunto de classes possíveis possui apenas dois elementos: {“normal”, “spam”}.

Além da filtragem de spam, existem inúmeras outras aplicações práticas para a classificação, tanto no âmbito das pesquisas científicas quanto dentro das empresas. Vejamos quatro exemplos:

Inserir mosaico

Imagem : máquina de cartão de crédito

Texto revelado na imagem 1: As administradoras de cartão de crédito utilizam modelos de classificação para detectar se uma transação financeira é “legal” ou “suspeita”.

Imagem 2: atendimento gerencial no banco

Texto revelado na imagem 2: Os bancos usam classificadores para classificar um cliente como de “alto”, “médio” ou “baixo” risco para um empréstimo bancário.

Imagem 3: imagem sugestiva de extração de petróleo

Texto revelado na imagem 3: As empresas de petróleo utilizam classificadores que analisam imagens de ressonância magnética de rochas e as classificam como de “alta permeabilidade” ou “baixa permeabilidade”. Rochas com alta permeabilidade possuem maior chance de conter petróleo.

Imagem 4: emojis

Texto revelado na imagem 4: Classificadores de sentimento analisam textos que contém opiniões de pessoas sobre filmes, produtos, notícias e classificam a opinião como “positiva” ou “negativa”.

Imagem 5: texto de uma notícia

Texto revelado na imagem 5: Os portais de notícia utilizam sistemas de classificação para atribuir automaticamente a categoria de um artigo (“Esporte”, “Cultura”, “Economia” etc.).

Imagem 6: Remédio

Texto revelado na imagem 6: As empresas farmacêuticas, utilizam sistemas de classificação para tentar prever o conjunto de reações adversas que novas drogas podem causar.

Fim do mosaico

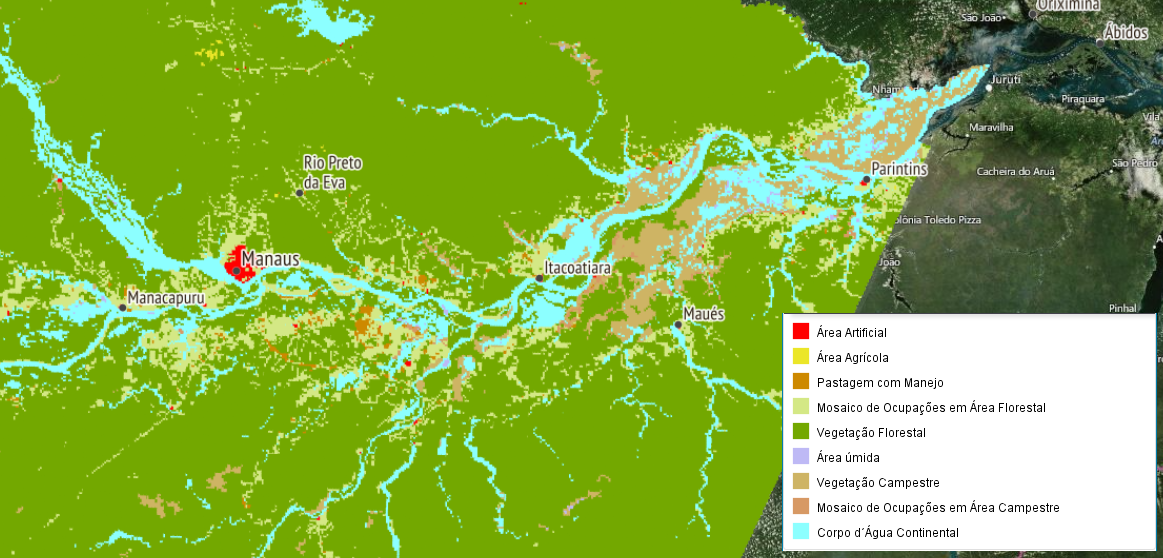
Mas e aqui no IBGE? Existem aplicações práticas para a classificação?

Sim, certamente, e elas também são em grande número. Vejamos três exemplos:

Inserir slider:

Slide 1: Um classificador para identificar fraudes pode ser utilizado em pesquisa domiciliar para detectar se determinado questionário preenchido por um informante é “genuíno” (contém informações verdadeiras) ou “suspeito” (pode conter informações falsas).

Slide 2: Na área de Geociências, um classificador de imagens de satélite pode ser utilizado para mapear a cobertura e o uso da terra, isto é, identificar áreas correspondentes a plantações, rios, áreas urbanas etc.



Slide 3: Um classificador de textos pode ser empregado em pesquisa econômica para determinar automaticamente o código de atividade econômica da empresa em função do texto fornecido por ela para descrever as atividades que realiza. Com isto, torna-se possível separar automaticamente empresas que são da indústria, do comércio, de serviços e da construção civil.

**Fim do slider**

É importante mencionar que o IBGE utiliza algoritmos de classificação há muitos anos, especialmente com o objetivo de apoiar o processo de crítica e imputação de dados de questionários. Porém, a crescente disseminação do conhecimento sobre ciência de dados na instituição e a contínua evolução de nosso parque tecnológico (máquinas com maior espaço de armazenamento e maior capacidade de memória e processamento), permite com que, nos dias atuais, seja possível utilizar modelos de classificação para resolver novos tipos de problemas, como os mencionados acima.

**Inserir atividade**

Pense em um ou mais exemplos de aplicações para a tarefa de classificação no setor em que você trabalha no IBGE.

**Fim da página 2**

**Início da página 3**

**Título do tópico:** O processo de classificação de dados

**Título da página:** Receitas dos classificadores: ingredientes

Pensou em um ou mais exemplos de aplicações para a tarefa de classificação no setor em que você trabalha no IBGE?

Inserir imagem de ingredientes numa bancada.

Tal qual em uma receita culinária, em que é importante saber quais são os ingredientes e como se dá o preparo, você vai ver agora o que é necessário para construir um classificador.

Na tarefa de classificação, a partir de uma base de dados contendo objetos pré-classificados [tooltip: objetos pré-classificados: objetos cuja classe é conhecida], o objetivo é construir um modelo capaz de classificar automaticamente novos objetos [tooltip: novos objetos: objetos cuja classe é desconhecida] em função de suas características. Desta forma, para que seja possível construir um classificador, “apenas” dois ingredientes são necessários: uma boa base de dados de treinamento e um bom algoritmo de classificação.

Uma boa **base de dados de treinamento** (ou base de dados rotulada) deve conter objetos pré-classificados, de acordo com um número finito de classes pré-definidas.

Um bom **algoritmo de classificação** fica encarregado de construir o modelo de classificação a partir da base de dados de treinamento. Os seguintes algoritmos serão tratados em nosso curso: *k*-Nearest Neighbours (*k*-NN), naïve Bayes,Random Forest, Support Vector Machine (SVM) e Rede Neural Multi-layer Perceptron (MLP).

Imagine que você precisa montar uma base de dados de treinamento com o objetivo de possibilitar a criação de um classificador de sentimentos capaz de determinar se é positivo ou negativo cada comentário feito pelos informantes que preencheram questionário de uma pesquisa do IBGE através da internet.

Inserir quadro

Exemplo de base de dados de treinamento para criação de classificador

|  |  |
| --- | --- |
| **X** | **Y** |
| “Fácil e prático, parabéns IBGE!! Achei ótimo mesmo, parabéns!!”, | POSITIVO |
| “Não tive nenhuma dificuldade para realizar o preenchimento. Concluí o processo em 5 minutos” | POSITIVO |
| “Achei o questionário grande demais. Ele travou no meio do processo :(” | NEGATIVO |
| “Gostaria de parabenizar o IBGE pela novidade, é mais prático preencher a pesquisa pela Internet” | POSITIVO |
| “Não gostei” | NEGATIVO |
| “Tudo funcionou perfeitamente” | POSITIVO |
| “Não consegui entender o sistema” | NEGATIVO |

Um classificador deste tipo poderia ser utilizado para monitorar, em tempo real, a opinião das pessoas sobre qualquer pesquisa do IBGE que estivesse na fase de coleta. A base de dados possui 7 observações, sendo 4 delas da classe POSITIVO e as 3 restantes da classe NEGATIVO.

Recorde do Módulo I que a classificação é um problema ao qual se aplica aprendizado supervisionado. Nesse tipo de problema de ciência de dados, a base de dados de treinamento sempre possuirá duas categorias de atributos, X e Y, na qual X é um conjunto de atributos preditivos e Y é um atributo especial, também chamado atributo classe.

Os atributos X, ou atributos preditivos, são aqueles que descrevem as características (*features*) dos objetos, podendo ser tanto numéricos como categóricos. No exemplo acima, a parte X de cada observação da base de treinamento corresponde a uma frase, mais precisamente a um comentário realizado por um informante da pesquisa. É importante apenas relembrar que toda base de treinamento para classificação de textos precisa ser transformada em uma base de dados estruturada na etapa de pré-processamento (detalhes no Módulo II).

Os atributos Y, ou atributo especial ou atributo classe, são aqueles que são alvo da classificação. O atributo classe é sempre do tipo categórico, e essa é a característica-chave que diferencia a classificação da regressão (como introduzido no Módulo I).

No exemplo apresentado, o conjunto pré-definido de classes é {“POSITIVO”, “NEGATIVO”}. As classes também são chamadas de **rótulos de classe** ou simplesmente rótulos. É importante observar que, nos problemas de classificação, em geral, as classes são disjuntas, ou seja, um comentário pode ser “POSITIVO” ou “NEGATIVO”, mas nunca será as duas coisas ao mesmo tempo, da mesma forma que um e-mail é “normal” ou “spam”, nunca os dois.

A criação de um classificador é realizada através do processamento da base de treinamento por um algoritmo de classificação, que se encarregará de aprender a mapear as características dos objetos (X) em rótulos de classes (Y). Uma vez que tenha sido criado, testado e aprovado no teste, um modelo de classificação poderá ser colocado em produção, podendo assim ser utilizado para classificar novos objetos, que são os objetos de classe desconhecida. Aprenderemos como testar e avaliar a qualidade de um classificador na unidade final deste módulo.

No exemplo do classificador de sentimentos, os objetos podem ser classificados em apenas duas classes: “POSITIVO” e “NEGATIVO”. Por isso, o problema é considerado um problema de classificação binário. Problemas que envolvem mais de duas classes são chamados de problemas de classificação multiclasse. Um exemplo seria o problema da classificação de imagens de satélite, onde o objeto de interesse, que é uma imagem, pode ser classificado em ao menos três classes: “plantação”, “rio” ou “área urbana”.

Vamos ver se você está acompanhando tudo até aqui?

Inserir imagem sugestiva de uma base de dados da PME, seguida do texto a seguir.

Considere uma base de dados obtida a partir da extração de um subconjunto da Pesquisa Mensal de Empregos (PME), que armazena o id, escolaridade, idade, sexo, raça e condição de ocupação de pessoas residentes em uma determinada capital brasileira. Considere ainda que a condição de ocupação pode ser “PO” (pessoa ocupada) ou “PD” (pessoa desocupada). Suponha que desejamos utilizar a base de dados para construir um classificador capaz de prever a condição de ocupação de uma pessoa a partir de suas características. Neste caso, quais atributos da base de dados formariam o conjunto X e qual seria o atributo Y? Reflita alguns minutos e depois confira sua resposta com a que se apresenta.

Janela modal.

Veja a resposta.

Os atributos do conjunto X são “escolaridade”, “idade”, “sexo” e “raça”.

O atributo Y é “condição de ocupação”

O atributo “id” não seria usado no modelo, pois ele não descreve qualquer propriedade de uma pessoa, servindo apenas para identificar uma observação da base de dados.

Fim da página 3

**Início da página 4**

**Título do tópico:** O processo de classificação de dados

**Título da página:** Receitas dos classificadores: modo de fazer

**Subtítulo da página:**

Assim como para preparar uma receita culinária é importante saber como se dá o preparo, para analisar dados, é importante saber como se constroem os classificadores.

Inserir imagem de alguém preparando uma receita

A construção de classificadores precisos e eficientes é considerada um dos grandes desafios na área de Ciência de Dados. Por este motivo, foram desenvolvidos diversos algoritmos para a execução desta tarefa, cada qual baseado em um princípio matemático diferente – conforme veremos a partir da próxima unidade. Seria algo como o preparo de um prato que pode ser feito de várias maneiras diferentes.

Apesar de adotarem princípios diferentes, todos os algoritmos realizam a classificação de dados em um processo composto por duas etapas: **aprendizado (ou treinamento)** e **classificação**.

Na primeira etapa, conhecida como “aprendizado”,o classificador é, de fato, construído.

Na segunda etapa, conhecida como “classificação”,o modelo é utilizado para prever o rótulo de classe de um novo objeto.

Estas etapas são exemplificadas na figura Etapa de Aprendizado e na figura Etapa de classificação, que ilustram, respectivamente, a construção e a utilização de um modelo de classificação a partir de uma base de dados de uma pesquisa sobre mobilidade urbana. Mais especificamente, o modelo classifica uma pessoa como usuária de transporte público ou não (atributo Y) com base em sua renda e idade (atributos X).

Observe como a figura a seguir ilustra o modelo de classificação criado a partir da base de dados de treinamento. Após uma análise mais detalhada, continue a leitura.

Inserir figura

**Figura 2: Etapa de aprendizado**

Diagrama

Descrição gerada automaticamente

Perceba que, na etapa de aprendizado, a base de dados de treinamento (lado esquerdo da figura) deve ser processada pelo algoritmo de classificação para que o classificador seja construído (lado direito da figura). Nesse exemplo, utilizou-se o algoritmo de árvore de decisão, que é apenas um dentre os vários algoritmos de classificação existentes, conforme veremos em breve. A principal característica do algoritmo de árvore de decisão é que ele gera um modelo de classificação gráfico e autoexplicativo. Veja que, de acordo com o modelo gerado, uma pessoa de renda alta é classificada com o rótulo “Não” (não usa transporte público) e as pessoas de renda baixa são classificadas como “Sim” (usuárias de transporte público). Por sua vez, a classificação das pessoas de renda média depende também do atributo idade (os mais velhos são classificados como “Não” e os mais novos como “Sim”).

A figura a seguir ilustra como o modelo de classificação criado na etapa de aprendizado é utilizado para classificar novos objetos. Após uma análise mais detalhada, continue a leitura.

Inserir figura

**Figura 3: Etapa de classificação**

Diagrama

Descrição gerada automaticamente

Perceba agora que, na etapa de classificação, nosso objetivo é classificar novos objetos. No exemplo, o classificador construído na primeira etapa é empregado para obter o rótulo de classe de uma nova pessoa (novo objeto) que tem 28 anos e possui renda média. No caso da árvore de decisão, o processo é bastante intuitivo, consistindo apenas em aplicar os dados dessa nova pessoa sobre o modelo de classificação. Tais dados levarão até uma folha da árvore e é esta folha que definirá a classe do novo objeto. Para o exemplo apresentado, veja que o modelo previu que o novo objeto é da classe “Sim”.

Realce: Uma outra forma de descrever a classificação com uma visão matemática é considerar que se trata, no processo de aprendizado, de uma função alvo *y = f(x)* capaz de mapear um conjunto de valores de atributos *x* para um dos valores pré-determinados de rótulos de classe *yi* *Y*.

Fim da página 4

**Início da pág. 5**

**Título do tópico:** Métodos de classificação

**Título da página:** Algoritmos de classificação

**Subtítulo da página:**

Neste curso, seis diferentes algoritmos de classificação serão apresentados e explicados. A seguir, apresentamos um breve resumo sobre cada um deles.

Inserir lista de definição

* K-*Nearest Neighbours* (*k*-NN)

Este algoritmo executa a tarefa de classificação de um novo objeto *t* em dois passos: primeiro, o algoritmo encontra os *k* objetos da base de treinamento que sejam mais similares a *t*. Tipicamente, a similaridade é computada em termos de alguma medida de distância, como a distância euclidiana, por exemplo. Em seguida, *t* é classificado com a classe mais comum entre os *k* objetos.

* Naïve Bayes

Classificador probabilístico que aplica o Teorema de Bayes e outros conceitos de probabilidade, como independência condicional, para determinar a classe mais provável de um novo objeto.

* Árvore de Decisão

Esta técnica constrói um modelo de classificação na forma de uma árvore, em que cada nó interno representa um teste sobre um atributo (ex.: o texto de um comentário possui a palavra “adorei”?) e cada folha consiste em um rótulo de classe (ex.: comentário POSITIVO ou comentário NEGATIVO). Um novo objeto é classificado através da aplicação de sucessivos testes que encontrarão um caminho na árvore, desde o nó raiz até um nó folha.

* Random Forest

Constrói um grupo (*ensemble*) de *k* árvores de decisão a partir da execução de *k* iterações sobre a base de dados de treinamento. Cada árvore de decisão do grupo é gerada com o uso de subconjuntos independentes e aleatórios de atributos e instâncias. Para prever a classe de um novo objeto, cada classificador individual vota e a classe mais popular é devolvida ao usuário.

* Support Vector Machine (SVM)

Técnica baseada em princípios da geometria analítica e otimização numérica em que os dados de treino são separados em duas classes através da busca por uma fronteira de decisão que maximiza a distância entre os objetos de treino destas duas classes. O classificador SVM possui o formato de uma equação matemática. Uma vez determinada, esta equação poderá ser aplicada para classificar novos objetos.

* Rede Neural Multi-Layer Perceptron (MLP)

Uma rede neural MLP é uma estrutura formada por um conjunto de nós (denominados neurônios artificiais) e arestas estruturados em três camadas: de entrada, oculta (*hidden*) e de saída. Cada aresta possui um peso associado. Um algoritmo de redes neurais emprega um processo de treinamento que visa determinar o melhor conjunto de pesos para a rede, o conjunto capaz de minimizar os erros de classificação. Assim como ocorre com o SVM, o classificador gerado por um algoritmo de rede neural também possui o formato de uma equação matemática.

As próximas páginas explicarão cada um desses algoritmos.

Fim da página 5

**Início da pág. 6**

**Título do tópico:** Métodos de classificação

**Título da página:** Algoritmos de classificação: k- NN

**Subtítulo da página:**

O *k*-NN (*k-nearest-neighbors* – ***k* vizinhos mais próximos**) é provavelmente o mais simples dos algoritmos de classificação. Seu princípio de funcionamento é baseado no aprendizado por analogia: ele determina a classe de um novo objeto levando em conta apenas um pequeno número de objetos da base de treinamento, avaliados como os mais **similares** ao novo objeto. Esses são considerados os seus vizinhos mais próximos, daí o nome do algoritmo.

A seguir, as etapas de treinamento e classificação do *k*-NN serão explicadas. Em seguida, discutiremos algumas características chave do algoritmo, assim como seus pontos fortes e fracos. Utilizaremos a mesma abordagem para os demais métodos de classificação cobertos ao longo desta unidade.

**Título de nível 2:** ETAPA DE TREINAMENTO

O *k*-NN é considerado um algoritmo “preguiçoso” (*lazy learner*), pois ele não cria um modelo na etapa de treinamento. Ao invés disso, a única coisa que o *k*-NN faz nesta etapa é importar a base de dados de treinamento para a memória. É isso mesmo, você não entendeu errado: o *k*-NN praticamente não trabalha na etapa de treinamento, a única coisa que ele faz é a importação do conjunto de dados de treinamento.

Mas, na verdade, o *k*-NN não é tão preguiçoso assim. O que de fato ocorre é que ele prefere se esforçar apenas na etapa de classificação. Neste momento, quando um novo objeto é apresentado, o ­*k­*-NN o classificará com base na sua similaridade em relação aos objetos de treinamento. Ele precisará trabalhar (e muito!) para executar essa ação, como veremos a seguir.

**Título de nível 2**: ETAPA DE CLASSIFICAÇÃO

Uma vez que os objetos de treinamento tenham sido importados para a memória, o *k*-NN poderá realizar a classificação de um novo objeto – que denotaremos por *novoObj* – empregando dois passos:

- Passo 1: o algoritmo irá procurar os *k* objetos que sejam **mais similares** a *novoObj* na base de dados de treinamento. Como veremos em breve, a similaridade é normalmente definida em termos de uma medida de distância, como a distância euclidiana.

- Passo 2: a classe mais frequente entre os *k* objetos encontrados será atribuída para *novoObj*. O valor de *k* é um parâmetro de entrada, definido pelo usuário do algoritmo.

A seguir um exemplo que mostra detalhadamente o funcionamento do *k*-NN. Suponha que desejamos implementar um classificador capaz de determinar se uma pessoa ganha mais de 5 salários mínimos em função de seus anos de estudo e total de horas trabalhadas por semana. Para isso, considere que temos a nossa disposição a base de dados de treinamento da Tabela 1, contendo dados extraídos de um censo hipotético. Veja que ela possui 6 observações (objetos) e 3 atributos: anos de estudo, quantidade de horas trabalhadas por semana e classe de renda. Os dois primeiros atributos formam o conjunto de atributos preditivos (X) enquanto o último corresponde ao atributo classe (Y), possuindo dois valores, “≤ 5 S.M.” (renda mensal igual ou inferior a 5 salários mínimos) e “> 5 S.M.” (renda mensal superior a 5 salários mínimos).

Inserir quadro

Tabela 1: Base de treinamento para a classificação da classe de renda

|  |  |  |
| --- | --- | --- |
| **anos de estudo** | **horas trabalhadas** | **classe de renda** |
| 12 | 50 | > 5 S.M. |
| 7 | 45 | ≤ 5 S.M. |
| 16 | 40 | > 5 S.M. |
| 12 | 30 | ≤ 5 S.M. |
| 13 | 45 | > 5 S.M. |
| 11 | 32 | ≤ 5 S.M. |

Na etapa de treinamento, o *k*-NN irá simplesmente importar a base de dados para a memória. Nenhum modelo será criado, pois o *k*-NN é preguiçoso na hora de treinar.

Na etapa de classificação, suponha que desejamos classificar a renda de uma nova pessoa com 10 anos de estudo e que trabalha 44 horas por semana, que denotaremos por *novoObj* = (10, 44). Considere ainda que o parâmetro *k* foi estabelecido com o valor 3. Dessa forma, o algoritmo *k*-NN irá computar a similaridade entre *novoObj* e todos os objetos contidos no conjunto de treinamento. Após o cálculo, serão selecionados os *3* objetos considerados mais similares de *novoObj*. A classe que mais se repete entre estes *k* vizinhos mais próximos será a classe do novo objeto.

Ok, mas como isso é feito na prática? Como podemos medir a similaridade entre objetos? A abordagem mais comum é considerar que cada objeto é um ponto em um espaço *n*-dimensional (onde *n* corresponde ao número de atributos preditivos do base de dados de treinamento) e então utilizar uma medida capaz de computar a distância entre cada par de objetos. Se a distância entre dois objetos é pequena, então se considera que eles são similares. Caso contrário, são considerados dissimilares.

Um exemplo de medida deste tipo é a distância euclidiana. Dados dois objetos X1 = (x11, x12, ..., x1n) e X2 = (x21, x22, ..., x2n), ambos descritos por *n* atributos, a distância euclidiana entre eles, denotada por *DE(X1, X2)*, pode ser calculada por:

Quase sempre, nós precisamos normalizar os valores de cada atributo antes de utilizar a distância euclidiana. Isso ajuda a impedir com que atributos com faixas de valores mais largas (ex.: idade) dominem atributos com faixas menores (ex.: anos de estudo). Por exemplo, podemos normalizar um valor *v* de um dado atributo *A* para a faixa entre [0, 1], utilizando a fórmula abaixo. Nesta fórmula, considere que *vnorm* é o valor normalizado e *minA* e *maxA*, representam, respectivamente, o menor e o maior valor de *A* na base de dados (para saber mais sobre normalização e outras técnicas de pré-processamento, consulte o Módulo 2 de nosso curso).

Vamos agora mostrar os cálculos de forma prática. Para obter a classe de *novoObj* = (10, 44), é preciso primeiro normalizar tanto os dados da base de treino como do *novoObj*. Vamos exemplificar usando os dados do *novoObj*, mas o processo seria análogo para todos os demais objetos da base de treinamento.

Observando nossos dados, temos que para o atributo anos de estudo, o menor valor registrado é 7 e o maior 16; para o atributo horas trabalhadas, temos 30 como menor valor e 50 como o maior (levando em conta tanto a base de treinamento como os dados do novo objeto a ser classificado). Sendo assim, os valores normalizados dos atributos do *novoObj* podem ser calculados da seguinte forma:

* Valor normalizado de “anos de estudo” para *novoObj* = (10 – 7) / (16 – 7) = 0.33
* Valor normalizado de “horas trabalhadas” para *novoObj* = (44 – 30) / (50 – 30) = 0.70

Então, agora temos *novoObj* = (0.33, 0.70). Já a nossa base de dados de treinamento normalizada teria o seguinte formato:

Inserir quadro.

Tabela 2: Base para classificação da classe de renda, com os atributos anos de estudo e horas trabalhadas normalizados para a faixa [0, 1].

|  |  |  |
| --- | --- | --- |
| **anos de estudo** | **horas trabalhadas** | **classe de renda** |
| 0.56 | 1.00 | > 5 S.M. |
| 0.00 | 0.75 | ≤ 5 S.M. |
| 1.00 | 0.50 | > 5 S.M. |
| 0.56 | 0.00 | ≤ 5 S.M. |
| 0.67 | 0.75 | > 5 S.M. |
| 0.44 | 0.10 | ≤ 5 S.M. |

Agora podemos executar o passo 1 da classificação, que consiste em encontrar os *k = 3* objetos da base de treinamento mais similares a *novoObj*. Para isso, precisamos calcular a distância euclidiana entre elee todos os objetos da base de treinamento. Por exemplo, a distância entre *novoObj* = (0.33, 0.70) e o primeiro objeto da base de treinamento – que denotaremos por *X1* = (0.56, 1.00) – é calculada da seguinte maneira:

De maneira análoga, calcula-se a distância entre *novoObj* e os objetos restantes da base de treinamento, chegando ao resultado obtido na Tabela 3, onde os valores das distâncias euclidianas foram acrescentados na última coluna. Nesta figura, marcamos em vermelho negrito os 3 vizinhos mais próximos de *novoObj* – são os objetos com menor valor calculado para distância euclidiana.

Inserir quadro.

Tabela 3: Base para classificação da classe de renda normalizada e com a informação da distância euclidiana entre cada observação e o *novoObj*.

|  |  |  |  |
| --- | --- | --- | --- |
| **anos de estudo** | **horas trabalhadas** | **classe de renda** | **distância euclidiana para *novoObj*** |
| **0.56** | **1.00** | **> 5 S.M.** | **0.38** |
| **0.00** | **0.75** | **≤ 5 S.M.** | **0.33** |
| 1.00 | 0.50 | > 5 S.M. | 0.70 |
| 0.56 | 0.00 | ≤ 5 S.M. | 0.74 |
| **0.67** | **0.75** | **> 5 S.M.** | **0.34** |
| 0.44 | 0.10 | ≤ 5 S.M. | 0.61 |

Tendo encontrado os vizinhos mais próximos, realizar a classificação torna-se trivial. Basta ver a classe que mais se repete nesses vizinhos. Neste caso, dois dos vizinhos são da classe “> 5 S.M.” e um é da classe “≤ 5 S.M.”. Sendo assim, *novoObj* será classificado como “> 5 S.M.”.

Como último comentário, vale a pena mencionar que a base de dados de nosso exemplo foi inspirada em uma base de dados real que contém dados extraídos do censo dos EUA. Ela pode ser obtida em [https://archive.ics.uci.edu/ml/datasets/census+income]

**Título de nível 2:** DISCUSSÃO

Vamos iniciar comentando algumas características do *k*-NN. Conforme visto, o processo de classificação se baseia em associar o objeto a ser classificado à classe mais comum entre seus *k* vizinhos mais próximos. Note que quando *k* = 1, o novo objeto será associado à classe do objeto de treinamento mais próximo no espaço *n*-dimensional.

Mas como identificar um bom valor para o parâmetro *k*? Em primeiro lugar, é recomendada a utilização de um valor ímpar, para que não ocorra empate no momento de classificar o novo objeto (caso o problema seja de classificação binária). Mas qual valor ímpar? *k* = 1? *k* = 3? *k* = 5? Algum valor maior? Na prática, isso depende do problema e costuma ser determinado em um processo experimental. Neste processo, a base de treinamento é dividida em 2 partições, denominadas partição de treino e partição de validação. Por exemplo, 1/3 das observações podem ser alocadas na partição de validação e 2/3 na de treino. Então, partindo de *k* = 1, utilizamos a partição de validação para estimar a taxa de erro do classificador na partição de treino. O processo é repetido diversas vezes, para diferentes valores de *k*. Será escolhido o valor de *k* que resultar no processo de classificação com menor taxa de erros.

Em nosso exemplo, utilizamos a distância euclidiana para identificar os vizinhos mais próximos. Essa medida pode ser utilizada quando os atributos preditivos são do tipo contínuo. Mas e se nossa base de dados possuir atributos categóricos, como “raça” ou “UF”? A abordagem mais básica consiste em simplesmente comparar o valor do atributo categórico no objeto X1 com o correspondente no objeto X2. Se os valores são idênticos, a diferença entre eles será considerada 0 (ex.: o valor da UF em X1 e X2 é “MG”). Caso contrário, é definida como 1 (ex.: a UF em X1 é “RJ” e em X2 é “MG”). Mas há outras abordagens mais sofisticadas.

Na verdade, o algoritmo *k*-NN é muito simples, porém computar a similaridade entre objetos é algo bem mais difícil. Além da distância euclidiana existem outras fórmulas de distância que podem ser mais ou menos adequadas dependendo do problema e das características dos atributos preditivos da base de treinamento. Detalhes podem ser encontrados em [<https://www.kdnuggets.com/2020/11/most-popular-distance-metrics-knn.html>].

Com relação aos prós e contras, o *k*-NN possui como principal aspecto positivo o fato de ser um algoritmo de fácil implementação e entendimento. Por outro lado, uma importante desvantagem é que o *k*-NN precisa consultar todo o conjunto de treinamento para classificar um novo exemplo. Isso faz com que a memória de armazenamento utilizada seja muito grande. Outra desvantagem relevante é o tempo computacional elevado na etapa de classificação quando se está trabalhando com um espaço de alta dimensão.

Para informações adicionais sobre o *k*-NN, consulte [<https://towardsdatascience.com/knn-in-python-835643e2fb53>].

**Fim da pág. 6**

**Início da pág. 7**

**Título do tópico:** Métodos de classificação

**Título da página:** Algoritmos de classificação: Naïve Bayes

**Subtítulo da página:**

O algoritmo naïve Bayes (NB) é um classificador fundamentado em diversos conceitos da Teoria da Probabilidade, como probabilidade condicional, independência condicional, regra da multiplicação, distribuição conjunta de probabilidades e, especialmente, em um importante teorema chamado Teorema de Bayes. Uma característica atraente deste classificador é a sua capacidade de produzir diretamente estimativas de probabilidade para cada rótulo de classe em vez de simples classificações. Além disso, o classificador se destaca por seu baixo custo computacional na etapa de treinamento, o que significa que ele gera o modelo de classificação em um tempo mais rápido em comparação a outros algoritmos. As subseções a seguir descrevem as etapas de treinamento e classificação do NB.  
  
 **Título de nível 2:** ETAPA DE TREINAMENTO

O princípio básico do NB consiste em treinar um modelo que possibilite a aplicação do **Teorema de Bayes** (ou regra de Bayes) [https://pt.wikipedia.org/wiki/Teorema\_de\_Bayes] para estimar a classe mais provável de um novo objeto. Para tal, na etapa de treinamento o algoritmo computa uma tabela de probabilidades que resume a base de dados de treinamento com informações suficientes para a aplicação deste teorema.

A seguir apresentaremos um exemplo que mostra detalhadamente o processo de treinamento de um classificador NB. Considere a base de dados apresentada na Tabela 4. Suponha que as 15 observações desta base representem dados de uma pesquisa sobre o endividamento de casais jovens. A base registra se o casal possui filhos, a escolaridade do(a) responsável pela família e se a família possui ou não algum tipo de dívida com o cartão de crédito.

Inserir quadro.

Tabela 4: Base com os dados de endividamento de casais jovens.

|  |  |  |
| --- | --- | --- |
| **filhos** | **escolaridade\_resp** | **dívida\_cartão (classe)** |
| Sim | superior | Não |
| Não | médio | Sim |
| Não | médio | Sim |
| Sim | superior | Não |
| Sim | superior | Não |
| Sim | médio | Sim |
| Não | fundamental | Não |
| Não | médio | Sim |
| Sim | médio | Sim |
| Não | médio | Não |
| Sim | médio | Sim |
| Sim | superior | Sim |
| Sim | médio | Sim |
| Não | médio | Não |
| Não | superior | Não |

Suponha que desejamos utilizar o algoritmo NB para treinar um modelo de classificação para prever se uma família possui ou não dívida de cartão de crédito em função de possuir ou não filhos e do nível de escolaridade do(a) responsável pela família. Com o algoritmo NB, a construção do modelo de classificação é muito barata do ponto de vista computacional, consistindo basicamente em montar uma **tabela de probabilidades condicionais** contendo um resumo dos dados contidos na base de dados de treinamento. A tabela referente ao nosso exemplo é mostrada na Tabela 5.

Inserir quadro.

Tabela 5: Modelo de classificação naïve Bayes gerado a partir da análise da base de dados de endividamento de casais jovens.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **dívida\_cartão** | **filhos** | | **escolaridade\_resp** | | |
| Não | Sim | fundamental | médio | superior |
| Não 7/15 | 4/7 | 3/7 | 1/7 | 2/7 | 4/7 |
| Sim 8/15 | 3/8 | 5/8 | 0/8 | 7/8 | 1/8 |

E o que são os valores de cada célula? Aqui vai a explicação. Inicialmente, observe a primeira coluna (“dívida\_cartão”). Ela apresenta as probabilidades “a priori” de uma família estar endividada ou não. Para (dívida\_cartão = “Não”) o valor é 7/15, isto é 7 das 15 famílias entrevistadas não possuem dívida, e para (dívida\_cartão =”Sim”) o valor é 8/15, ou seja, 8 das 15 famílias entrevistadas estão endividadas.

Da segunda célula em diante, as informações apresentadas são mais interessantes. Elas representam uma série de valores de **probabilidades condicionais**. Relembre que nosso objetivo é construir um classificador para prever se uma família está endividada em função de suas características (se possui ou não filhos e a escolaridade do(a) responsável). Por esta razão, as colunas 2 a 5 da tabela apresentam as probabilidades condicionais dos valores dos atributos preditivos “filhos” e “escolaridade\_resp” dados os dois rótulos possíveis da classe “dívida\_cartão” (“Sim” ou “Não”).

Considere, por exemplo, o valor 4/7 localizado na célula referente ao cruzamento da linha (dívida\_cartão = “Não”) com a coluna (possui\_filhos = “Não”). Esse valor indica que, dentre as 7 famílias que não estão endividadas, 4 não possuem filhos. Utilizando a notação de probabilidade condicional, esse valor corresponde à P(possui\_filhos = “Não” | dívida\_cartão = “Não”). Agora veja o valor 3/8 localizado na célula logo abaixo. Ela corresponde ao cruzamento da linha (dívida\_cartão = “Sim”) com a coluna (possui\_filhos = “Não”). Sendo assim, o valor 3/8 indica que, dentre as 8 famílias endividadas, 3 não possuem filhos, representando o valor de P(possui\_filhos = “Não” | dívida\_cartão = “Sim”). E assim ocorre para todos os valores das demais células da tabela: basta você olhar o rótulo de classe referente à linha e o valor de atributo preditivo da coluna para entender o significado do conteúdo armazenado na célula correspondente.

É muito importante que você não esqueça duas coisas. A primeira é que a tabela resumo **é** o modelo de classificação naïve Bayes. A segunda é que é possível gerá-la de forma muito rápida, pois os valores de todas as células da tabela podem ser computados com uma única varredura sobre a base de dados de treinamento (ou seja, percorrendo todos os objetos da base, do primeiro ao último, apenas uma vez).

Ok, nosso modelo NB está criado. Mas como podemos classificar um novo objeto utilizando a tabela resumo? O processo é explicado a seguir.

**Título de nível 2**: ETAPA DE CLASSIFICAÇÃO

No algoritmo NB, a classificação de um novo objeto é realizada com o uso da regra de Bayes. Essa regra possibilita o cálculo da probabilidade de ocorrência de uma hipótese *H*, dada a observação de uma evidência *E*:

Como você já deve estar imaginado, quando a fórmula é usada em problemas de classificação, *E* e *H* possuem o seguinte significado:

* O evento *E* representa o conjunto de valores de atributos preditivos do novo objeto a ser classificado, ou seja, *E =* (*x*1, *x*2, ..., *x*n).
* A hipótese *H* representa um rótulo de classe, ou seja H {*y1*, *y2*, ..., *ym*}.

Observe que a evidência *E*, nosso novo objeto a ser classificado, é um objeto composto por um conjunto de valores de *n* atributos preditivos: *E =* (*x*1, *x*2, ..., *x*n). Devemos avaliar todas as hipóteses possíveis, o que significa determinar a probabilidade de *E* pertencer a cada um dos *m* rótulos de classe existentes {*y1*, *y2*, ..., *ym*}.

O algoritmo NB possui esse nome porque trata todos os atributos como independentes entre si, dado que o valor da classe é conhecido. Embora a suposição seja quase sempre falsa, ela nos permite **dividir a evidência em partes independentes** - exatamente os valores (*x1, x2, ... , xn*), fazendo com que a regra de Bayes possa ser reescrita da maneira a seguir. Veja que ela é utilizada para estimar a probabilidade do novo objeto pertencer a uma classe específica yj:

Essa fórmula reescrita pode então ser utilizada para classificar um novo objeto de uma maneira simples. Mais especificamente, para classificar um novo objeto, o algoritmo NB simplesmente observa as características do novo objeto, recupera os valores correspondentes na tabela de probabilidades e os “pluga” diretamente à fórmula acima. Durante o cálculo, não será preciso utilizar o denominador da fórmula de Bayes, uma vez que ele seria igual tanto para o cálculo da estimativa do rótulo de classe “Não”, como do “Sim”.

Veja agora como isso funciona na prática. Mostraremos, a seguir, como o algoritmo realiza o cálculo da estimativa para os dois rótulos de classe (dívida\_cartão = “Não”) e (dívida\_cartão = “Sim”), considerando o novo objeto *novoObj* = (filhos = “Sim”, escolaridade = “médio”):

Estimativa para endividado = “Não”:

|  |  |
| --- | --- |
| P(dívida\_cartão = “Não”| *novoObj*) = | P(filhos= “Sim”| dívida\_cartão = “Não”) x P(escolaridade = “médio” | dívida\_cartão = “Não”) x P(dívida\_cartão = “Não”) |
| P(dívida\_cartão = “Não”| *novoObj*) = | 3/7 x 2/7 x 7/15 = 0,0571 |

Estimativa para endividado = “Sim”:

|  |  |
| --- | --- |
| P(dívida\_cartão = “Sim”| *novoObj*) = | P(filhos= “Sim”| dívida\_cartão = “Sim”) x P(escolaridade = “médio” | dívida\_cartão = “Sim”) P(dívida\_cartão = “Sim”) |
| P(dívida\_cartão = “Sim”| *novoObj*) = | 5/8 x 7/8 x 8/15 = 0,2916 |

O resultado do exemplo indica que há uma probabilidade estimada bem maior de a família pertencer à classe “Sim” (0,2916) do que à classe “Não” (0,0571). Logo, *novoObj* é classificado com o rótulo “Sim” pelo NB.

Para finalizar, mostraremos um recurso que costuma ser utilizado por todas as ferramentas de ciência de dados. Para permitir com que os usuários visualizem o resultado de uma classificação feita pelo NB de uma forma mais agradável, essas ferramentas realizam a conversão dos valores calculados para valores percentuais. Isso é feito da seguinte maneira:

* Estimativa para (dívida\_cartão = “Nao”) = 0,0571 ÷ (0,0571 + 0,2916) = 16,38%
* Estimativa para (dívida\_cartão = “Sim”) = 0,2916 ÷ (0,0571 + 0,2916) = 83,62%

**Título de nível 2:** DISCUSSÃO

Vamos iniciar a discussão falando sobre as “suposições ingênuas” usadas pelo NB. O algoritmo NB possui a palavra “naïve” (ingênuo) em seu nome porque se baseia em duas suposições relacionadas aos atributos da base de dados:

1. O efeito do valor de um atributo preditivo sobre um atributo classe é independente do valor dos outros atributos preditivos.
2. Todos os atributos são igualmente importantes.

Ambas as suposições raramente são verdadeiras na prática, porém são elas que permitem que seja possível utilizar a regra de Bayes de uma maneira válida, isto é, respeitando a Teoria da Probabilidade. Você deve ter notado que o exemplo apresentado envolveu apenas atributos preditivos do tipo categórico. Por isso a versão do algoritmo que foi descrita nesta seção é conhecida como **naïve Bayes categórico**. Mas e se a nossa base de dados também possuir atributos preditivos contínuos? Nesse caso, duas soluções são possíveis. A primeira consiste simplesmente em discretizar os atributos contínuos (convertê-los para faixas de valores – veja como fazer no Módulo 2). A outra solução é assumir que todos os atributos contínuos possuem distribuição gaussiana e computar de acordo com essa suposição (nesse caso, temos a versão conhecida como NB gaussiano).

Outro problema em potencial é a possível existência de valores 0 em células da “tabelona de probabilidades”. Eles são gerados quando um valor de atributo preditivo não ocorre com um determinado rótulo de classe na base de dados de treino. No Quadro 5, é isto que a acontece na célula referente a (dívida\_cartão = “Sim”) e (escolaridade = “fundamental”). Mas qual seria o problema do valor 0? O problema é que, no momento de estimarmos a probabilidade de uma classe, ele vai entrar como termo da multiplicação, gerando 0 como resultado da estimativa. Para contornar este problema, as ferramentas de ciência de dados costumam utilizar a técnica conhecida como Correção de Laplace (*laplacian smoothing*), que consiste basicamente em adicionar constantes ao numerador e denominador em cada célula da tabela que representa o modelo de classificação. Um exemplo é apresentado em [https://courses.cs.washington.edu/courses/cse446/20wi/Section7/naive-bayes.pdf].

Por fim, uma breve análise dos pontos fortes e fracos do NB. Com relação aos aspectos positivos, podemos citar principalmente a simplicidade e eficiência (o NB é rápido, de baixo custo computacional). Outra característica positiva é o que o NB produz diretamente estimativas de probabilidade para cada classe. Por fim, como última vantagem, vale mencionar que o NB é uma boa “porta de entrada” para o estudo de Redes Bayesianas e outras técnicas mais sofisticadas que também se baseiam no Teorema de Bayes.

Já o principal ponto fraco do NB é que seu desempenho preditivo costuma ser afetado quando os dados divergem muito das suposições ingênuas.

Realce: **naïve Bayes – versões alternativas**

Além do naïve Bayes categórico, existem versões alternativas do NB. A ferramenta scikit-learn, por exemplo, disponibiliza nada menos do que 5 diferentes versões do algoritmo [link 5 diferentes versões do algoritmo: https://scikit-learn.org/stable/modules/naive\_bayes.html]. Dentre elas, destaca-se a versão conhecida como naïve Bayes multinomial, que é muito utilizada para a construção de classificadores a partir de bases de dados de texto. Detalhes completos sobre NB multinomial podem ser obtidos no livro gratuito de PLN [link livro gratuito de PLN: https://web.stanford.edu/~jurafsky/slp3/]. Outra versão bastante conhecida é o NB gaussiano. Para saber mais, consulte o link [link link: <https://www.inf.ufsc.br/~andre.zibetti/probabilidade/normal.html>]

**Fim da pág. 7**

**Início da pág.8**

**Título do tópico:** Métodos de classificação

**Título da página:** Algoritmos de classificação: Árvores de Decisão

**Subtítulo da página:**

Grande parte dos algoritmos de classificação propostos na literatura baseia-se na construção de **modelos caixa-preta**. Isto é: modelos projetados para maximizar o desempenho preditivo (acurácia) do classificador, mas que não explicam suas previsões de uma maneira que os humanos possam entender.

No entanto, para as instituições governamentais, por questões legais, éticas e de transparência, oferecer ferramentas para que usuários possam interpretar as classificações produzidas por um modelo costuma representar algo tão importante quanto a própria acurácia do modelo. Em outras palavras, grande parte dos problemas no âmbito da administração pública requer a utilização de **modelos de classificação interpretáveis**. Um classificador interpretável possui a habilidade de “explicar” as suas classificações para os usuários através, por exemplo, da apresentação de regras de classificação no formato: SE <condição> ENTÃO <rótulo(s) de classe>.

No campo da ciência de dados, a técnica de árvore de decisão (AD) é a mais conhecida e utilizada para o aprendizado de classificadores interpretáveis. Isto é justificado pelo fato de as ADs possuírem uma estrutura gráfica e intuitiva, que representa um conjunto de regras de classificação e é semelhante a um fluxograma, tornando natural a interpretação do modelo de classificação. A Figura 4 mostra um exemplo baseado em uma AD real, extraída a partir de dados da Pesquisa Mensal de Empregos, que classifica o tipo de ocupação de trabalhadores analfabetos como “trabalhador doméstico” em função de seu sexo e idade. Esta árvore revela que, de acordo com a base de dados da PME, mulheres acima dos 60 anos *têm maior chance* de exercer trabalho doméstico, ao contrário das mulheres abaixo dessa idade e dos homens.

Inserir fluxograma

Figura 4: Árvore de decisão que classifica trabalhadores analfabetos como pessoa que executa serviço doméstico ou não.

Diagrama

Descrição gerada automaticamente

Observe atentamente a figura. Uma AD é uma estrutura que possui as seguintes características:

* É desenhada com a raiz no topo e as folhas na parte inferior. Sendo assim, a raiz dessa árvore é o atributo “Sexo”.
* Cada nó interno (representado como um retângulo na Figura 4) consiste em um teste sobre um atributo preditivo (neste caso, os atributos “Sexo” e “Idade”).
* Uma ramificação, partindo de um nó interno, representa um resultado para o teste (por exemplo, Idade ≤ 60)
* Cada nó folha representa um rótulo de classe (neste exemplo, Trabalhador Doméstico = “Sim” ou Trabalhador Doméstico = “Não”).
* Um novo objeto é classificado seguindo um caminho na árvore, do nó raiz até um nó folha.

Uma AD é formada por um conjunto de regras de classificação, uma vez que existe sempre um único caminho da raiz para cada folha e este caminho representa uma expressão da regra utilizada para classificar um objeto. Folhas diferentes podem produzir a mesma classificação, mas cada uma por uma razão diferente (pois cada caminho representa uma regra diferente). Por exemplo, a AD da Figura 4 é composta por três regras:

* SE (Sexo = ‘M’) ENTÃO (Trabalhador Doméstico = ‘Não’)
* SE (Sexo = ‘F’) e (Idade ≤ 60) ENTÃO (Trabalhador Doméstico = ‘Não’)
* SE (Sexo = ‘F’) e (Idade > 60) ENTÃO (Trabalhador Doméstico = ‘Sim’)

Nas últimas décadas, diversos algoritmos para o aprendizado de ADs foram propostos, sendo os algoritmos CHAID, C4.5 e CART os mais conhecidos. Neste curso, apresentaremos o algoritmo CART, uma vez que ele possui implementações em código aberto nas linguagens R e Python que são muito populares entre os cientistas de dados. Para aprender sobre os algoritmos CHAID e C4.5 você pode consultar os links [https://sefiks.com/2020/03/18/a-step-by-step-chaid-decision-tree-example/] e [https://www.futurelearn.com/info/courses/data-mining-with-weka/0/steps/25391], respectivamente. Na subseção a seguir, vamos mostrar como o CART constrói uma árvore de decisão.

**Título de nível 2:** ETAPA DE TREINAMENTO

Uma AD é composta por uma série de questões. A resposta da primeira questão determina a questão seguinte, e assim sucessivamente até que um nó folha seja alcançado. Caso as questões sejam bem formuladas (na melhor ordem possível), um pequeno número delas poderá ser suficiente para classificar corretamente um objeto.

Portanto, um aspecto fundamental para a construção de uma AD consiste na estratégia para a escolha dos atributos preditivos que estarão mais próximos da raiz da árvore (ou seja, os atributos que são inicialmente avaliados para determinar a classe à qual um objeto pertence). Uma **medida de seleção de atributos** (também chamada de medida de diversidade) é utilizada durante a construção da árvore para produzir um ranking de atributos preditivos. O atributo com melhor escore no ranking é aquele que consegue realizar o melhor trabalho de produzir nós mais puros (nós que separem os objetos da base de treino em grupos onde um único rótulo de classe predomine). Em outras palavras, é o atributo que diminui ao máximo a diversidade dos objetos de treino. Os conceitos de impureza e pureza são exemplificados na Figura 5. Neste exemplo, considere um BD com 18 observações cujo atributo classe possui dois rótulos: “S” ou “N”. No lado esquerdo da figura, temos um exemplo de divisão impura, em que não há predominância de nenhum rótulo classe nas partições produzidas. Já no lado direito, temos um exemplo de divisão que gera partições puras, ou seja, partições em que há predominância de um rótulo de classe.

Inserir figura

Figura 5: Partições impuras *versus* partições puras

Diagrama

Descrição gerada automaticamente com confiança média

O algoritmo CART utiliza o índice de Gini para computar a impureza das partições geradas por um atributo. A fórmula é apresentada a seguir. Considere que *m* é o número de classes e *pi* é a probabilidade de um objeto *t* pertencer à classe *yi* no conjunto de dados *D*.

Para que seja possível entender como o índice de Gini é usado na prática, faremos a demonstração com o uso da base de dados abaixo, baseada no exemplo previamente apresentado sobre as informações de trabalhadores analfabetos na PME.

Inserir quadro.

Tabela 6: Base com os dados de trabalhadores analfabetos.

|  |  |  |  |
| --- | --- | --- | --- |
| **idade** | **UF** | **sexo** | **trabalho\_doméstico (classe)** |
| ≤ 60 | MG | M | Não |
| > 60 | SP | F | Sim |
| > 60 | MG | F | Sim |
| ≤ 60 | SP | M | Não |
| ≤ 60 | SP | F | Não |
| > 60 | MG | F | Sim |
| ≤ 60 | SP | M | Não |
| > 60 | MG | M | Não |
| ≤ 60 | MG | F | Sim |
| > 60 | SP | F | Sim |
| ≤ 60 | MG | F | Não |

A estratégia básica empregada pelo CART é construir a árvore por etapas, ou seja, nó por nó. A cada etapa, todos os atributos preditivos disponíveis deverão ser avaliados, o que significa que o índice de Gini deverá ser calculado para cada um deles. O atributo que resultar no menor valor será escolhido para fazer parte da árvore, pois ele é o que gera as partições que, em média, são as mais puras. Vamos examinar um exemplo passo a passo para entender como é feito.

Na base de dados da Tabela 6, os atributos preditivos são “idade”, “UF” e “sexo” e o atributo classe é “trabalho\_doméstico”. Na primeira rodada do CART, haverá a decisão sobre qual dos três atributos preditivos entrará como nó raiz da árvore. A decisão ocorre em função do resultado dos cálculos a seguir:

PASSO 1: definindo o nó raiz

\* “idade”:

(≤ 60) : (> 60) = 6/11 × [1 – (5/6)2 – (1/6) 2] + 5/11 × [1 – (1/5)2 – (4/5) 2] = 0,30

\* “UF”:

(MG) : (SP) = 6/11 × [1 – (3/6)2 – (3/6) 2] + 5/11 × [1 – (3/5)2 – (2/5) 2] = 0,49

\* “sexo”:

(F) : (M) = 7/11 × [1 – (2/7)2 – (5/7) 2] + 4/11 × [1 – (4/4)2 – (0/4) 2] = 0,26

Como o índice de Gini para o atributo “sexo” é menor, isto significa que ele gera as partições que em média são as mais puras. Portanto, deve ser escolhido para entrar na raiz da árvore.

Ok, mas como os cálculos mostrados foram feitos? Ao definir partições, o CART sempre realiza divisões binárias, ou seja, de cada nó interno sempre serão feitos dois testes que resultam na geração de duas partições. Então, para cada atributo *A* candidato a entrar como nó da AD, o CART realiza a soma ponderada da impureza de cada uma das duas partições resultantes. Isto é feito com o uso da fórmula a seguir, em que *D1* e *D2* são as duas partições geradas pela divisão binária:

Observe, por exemplo, como foi feito o cálculo para o atributo “sexo”. Esse atributo gera a divisão binária (sexo= “F”) e (sexo = “M”). No exemplo apresentado, a primeira parte do cálculo (antes do sinal de adição), refere-se ao cálculo da impureza da partição gerada por (sexo = “F”). O valor 7/11 refere-se ao fato de que das 11 pessoas da base, 7 são do sexo feminino. O valor 2/7 indica que 2 dessas 7 mulheres são da classe “Não”, enquanto 5/7 indica que 5 das 7 mulheres são da classe “Sim”. De maneira análoga, a segunda parte do cálculo refere-se à impureza da partição gerada por (sexo = “M”).

Agora que o cálculo foi explicado, veja na Figura 6 como ficam as partições da base de treinamento após o particionamento pelo atributo sexo. É importante notar que a partição gerada por (sexo = “M”) possui maior pureza, uma vez que todos os objetos são da classe “Não”. Já a partição definida por (sexo = “F”) é menos pura, pois temos 2 objetos da classe “Não” e 5 da classe “Sim”.

|  |  |
| --- | --- |
| **Sexo** | |
| F | M |
| >60, SP, F, Sim >60, MG, F, Sim ≤ 60, SP, F, Não >60, MG, F, Sim ≤ 60, MG, F, Sim >60, SP, F, Sim ≤ 60, MG, F, Não | ≤ 60, MG, M, Não ≤ 60, SP, M, Não ≤ 60, SP, M, Não >60, MG, M, Não |

Figura 6. Partições definidas pelo atributo “sexo”

A árvore de decisão parcial, definida em função dessas partições é apresentada na Figura 7:

Inserir figura.

Figura: AD parcial gerada após a primeira rodada do CART

Diagrama

Descrição gerada automaticamente

Figura 7

Não precisamos fazer mais nada com os dados da partição (sexo = “M”) que é perfeitamente pura. Porém, ainda podemos investigar “idade” e “UF” para identificar qual é capaz de gerar uma boa divisão para os dados da partição (sexo = “F”). Basta usar novamente o índice de Gini, porém desta vez considerando apenas as 7 observações referentes à partição (sexo = “F”).

\* “idade”:

(≤ 60) : (> 60) = 3/7 × [1 – (2/3)2 – (1/3) 2] + 4/7 × [1 – (0/4)2 – (4/4) 2] = 0,19

\* “UF”:

(MG) : (SP) = 4/7 × [1 – (1/4)2 – (3/4) 2] + 3/7 × [1 – (1/3)2 – (2/3) 2] = 0,40

Desta forma, o atributo “idade” é o escolhido para particionar o subconjunto de dados definido pela partição (sexo = “F”). A Figura 8 mostra os novos particionamentos por ele gerados.

Figura 8. Partições definidas pelo atributo “idade” na partição (sexo = “F”)

|  |  |  |
| --- | --- | --- |
| **sexo** | | |
| F | | M |
| >60, SP, F, Sim >60, MG, F, Sim ≤ 60, SP, F, Não >60, MG, F, Sim ≤ 60, MG, F, Sim >60, SP, F, Sim ≤ 60, MG, F, Não | | ≤ 60, MG, M, Não ≤ 60, SP, M, Não ≤ 60, SP, M, Não >60, MG, M, Não |
| **Idade** | |  |
| ≤ 60 | >60 |  |
| ≤ 60, SP, F, Não ≤ 60, MG, F, Sim ≤ 60, MG, F, Não | >60, SP, F, Sim >60, MG, F, Sim >60, MG, F, Sim >60, SP, F, Sim |  |

Veja que agora a partição definida por (sexo = “F” e idade > 60) possui todos os objetos da classe “Sim”. Já a partição (sexo = “F” e idade ≤ 60) possui dois objetos da classe “Não” e um da classe “Sim”. No entanto, ela tem um número muito reduzido de objetos e, em seu modo típico de operação, o CART vai considerar que não vale mais a pena tentar dividi-la. Em vez disso, O CART simplesmente irá definir que esta partição gera um nó folha da classe “Não”, pois essa é a classe majoritária da partição. Assim, o algoritmo encerra o seu processamento gerando a AD mostrada na Figura 9, de acordo com a Figura 4 apresentada anteriormente.

Inserir figura.

Figura 9. AD final gerada pelo CART

Diagrama

Descrição gerada automaticamente

**Título de nível 2**: ETAPA DE CLASSIFICAÇÃO

Uma vez que a AD tenha sido gerada, classificar um novo objeto é trivial: basta seguir um caminho da raiz até um nó folha de acordo com as características do objeto. Por exemplo, considere o *novoObj* = (sexo=”F”, idade > 60). De acordo com a AD gerada, o objeto deve ser classificado como “Sim”, independente da variável UF do objeto, pois essa variável não foi considerada pelo modelo criado.

**Título de nível 2:** DISCUSSÃO

Tanto o CART como todos os outros algoritmos para o aprendizado de árvores de decisão usam estratégias parecidas: são recursivos, constroem a árvore utilizando uma abordagem *top-down* e possuem como meta a construção de árvores que possuam o menor tamanho e a maior acurácia possíveis.

O CART consegue gerar apenas árvores binárias, por causa da fórmula que utiliza para seleção de atributos. No exemplo apresentado, isso não foi um problema, pois todos os atributos preditivos possuíam apenas dois valores distintos. Mas e se existir um atributo preditivo com mais de dois valores, como por exemplo “Raça” com os valores {“amarela”, “branca”, “preta”}? Neste caso, O CART avaliará todas as divisões binárias possíveis, que são:

(amarela) : (branca, preta)

(branca) : (amarela, preta)

(preta) : (amarela, branca)

E se houvesse atributos preditivos contínuos? Neste caso, supondo um atributo contínuo *A*, o CART tentará encontrar o melhor ponto de divisão ordenando os valores de *A* de maneira ascendente e depois testando todas as possibilidades considerando os pontos médios entre cada par de valor. Ou seja: se existem *v* valores distintos, então *v-1* possíveis divisões serão consideradas. Por exemplo, suponha um atributo salário que apareça com os quatro seguintes valores distintos na base 1.500, 2.000, 4.000 e 10.000. Neste caso, o CART avaliaria três possíveis divisões:

(≤ 1.750) : (> 1.750) (*1.750 é o ponto médio entre 1.500 e 2.000*)

(≤ 3.000) : (> 3.000) (*3.000 é o ponto médio entre 2.000 e 4.000*)

(≤ 7.000) : (> 7.000) (*7.000 é o ponto médio entre 4.000 e 10.000*)

No exemplo apresentado na Figura 8, o CART encerrou o seu processamento quando ocorreu uma situação em que uma das partições ainda não processada possuía um número pequeno de observações (apenas 3 observações). Na prática, a maioria das ferramentas de ciência de dados permite que o usuário configure este número mínimo de observações que leva ao fim do processamento. Outro critério de parada que pode ser definido para o CART é a profundidade (número de níveis) máxima da árvore.

Quando uma árvore é construída, muitos ramos podem refletir sujeira ou terem sido criados sob influência de *outliers*. O CART pode utilizar uma técnica de poda com o intuito de identificar e remover estes ramos. As árvores podadas tendem a ser menores e menos complexas, portanto, mais fáceis de serem compreendidas pelos usuários de uma ferramenta de ciência de dados. Na maioria das vezes também são mais eficazes para a classificação de novos casos. Uma breve introdução ao tema é apresentada em [https://www.youtube.com/watch?v=u4kbPtiVVB8&t=7s].

Por fim, vamos aos pontos positivos e negativos da técnica de AD. Um primeiro ponto positivo é a eficiência, ou seja, usando o CART ou qualquer outro algoritmo de AD podemos gerar o modelo de classificação em um tempo computacional baixo. No entanto, muito mais importante que isso e o que podemos considerar como principal aspecto positivo é o fato de uma AD ser um modelo interpretável, o que constitui requisito necessário em muitas aplicações relacionadas ao serviço público. Por outro lado, a principal desvantagem é que nem sempre um algoritmo de AD consegue produzir um modelo com desempenho preditivo similar ao de métodos mais sofisticados, como Random Forest e Redes Neurais.

|  |
| --- |
| **EXERCÍCIO:** Dado o BD abaixo, calcule o índice de Gini para os atributos preditivos “Idade” e “Renda” |

**Fim da pág. 8**

**Início da pág. 9**

**Título do tópico:** Métodos de classificação

**Título da página:** Algoritmos de classificação: Random Forest ou Floresta Aleatória

**Subtítulo da página:**

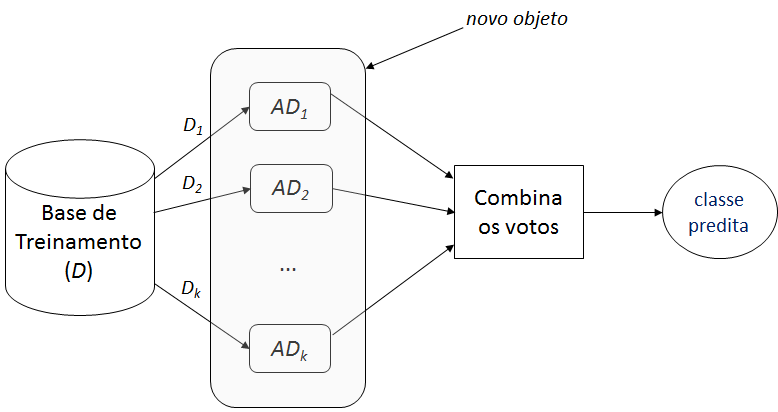
Suponha que seu computador está travando com frequência e você deseja descobrir o motivo. Em vez de chamar apenas um técnico para examiná-lo, você poderia optar por chamar vários. Neste caso, se uma determinada causa para o problema for mais citada do que qualquer outra, você a consideraria como a diagnóstico correto. Ou seja, a causa do problema de travamento seria determinada de acordo com uma espécie de votação majoritária, onde cada técnico daria o seu voto. A sua expectativa é que considerar a opinião mais prevalente de um grupo de técnicos é mais confiável do que levar em conta a opinião de um único profissional.

A Floresta Aleatória (*Random Forest* – RF) é uma técnica de classificação que trabalha com uma ideia similar. Ela combina as decisões de múltiplos modelos para otimizar a eficácia do processo de classificação. Mais especificamente, uma RF é um **modelo de classificação composto**, que é formado por um conjunto (*ensemble*) de classificadores do tipo árvore de decisão. Cada árvore de decisão do grupo é gerada com o uso de subconjuntos independentes e aleatórios de atributos e objetos. Para prever a classe de um novo objeto, cada classificador individual vota e a classe mais popular é devolvida ao usuário.

A Figura 10 resume o funcionamento de um classificador RF. Nas subseções a seguir, utilizaremos essa figura como base para explicar as etapas de treinamento e classificação da RF.

Inserir figura.

Figura 10: Princípio de funcionamento da Random Forest.



**Título de nível 2:** ETAPA DE TREINAMENTO

Um modelo RF combina uma série de *k* modelos de árvore de decisão, *AD1*, *AD2*, ..., *ADk*, com o objetivo de criar um modelo composto otimizado de classificação. Cada árvore de decisão individual é também chamada de classificador base (*base classifier*).

Como ocorre com qualquer técnica de classificação, antes de qualquer coisa precisamos de uma base de dados de treinamento *D*, composta por *d* objetos. O treinamento das *k* árvores de decisão (lado esquerdo da Figura 10) que compõem a RFé feito em *k* iterações. Para cada iteração (*i*=*1*, *2*, ..., *k*), um conjunto de treinamento *Di*, com *d* objetos é gerado a partir de um processo de amostragem com reposição. Sendo assim, cada *Di* tem o mesmo número de objetos do que a base de treinamento original *D*, porém alguns objetos podem aparecer mais de uma vez, enquanto outros podem não ocorrer em *Di*. Esse processo também é conhecido como *bootstrapping*.

A cada iteração *i*, uma árvore de decisão *ADi* é treinada a partir de *Di* com o uso do algoritmo CART (ou outro algoritmo de construção de ADs). As árvores devem ser criadas com tamanho máximo e sem poda. Entretanto, a cada nó de *ADi*, apenas um **subconjunto de atributos candidatos** é considerado para decidir qual será a melhor divisão. Esse subconjunto é **definido de maneira aleatória**.

O modelo final será então composto por diversas árvores, que possuirão estruturas diferentes em termos de quantidade e disposição dos nós (Figura 11), uma vez que cada uma foi treinada a partir de um subconjunto diferente de observações e atributos, sempre determinados de forma independente e aleatória. Daí, o nome Floresta Aleatória.

Inserir figura.

Figura 11. Um classificador Random Forest é composto por um grupo de *k* árvores de decisão com estrutura distinta.

Diagrama, Gráfico de caixa estreita

Descrição gerada automaticamente

**Título de nível 2**: ETAPA DE CLASSIFICAÇÃO

Uma vez que a RF tenha sido gerada, o processo para classificar um novo objeto (lado direito da Figura 10) é muito simples. Durante a classificação, cada árvore vota (ou seja, determina individualmente a classe do objeto) e a classe mais votada é retornada.

**Título de nível 2:** DISCUSSÃO

Um modelo RF tende a ter um desempenho preditivo superior do que cada uma de suas árvores de decisão individualmente. Intuitivamente, podemos ter a percepção de que os classificadores base (árvores) que compõem a RF até poderão cometer erros de classificação, porém a RF propriamente dita só errará a classificação de um novo objeto quando mais da metade deles se equivocarem.

De fato, muitos trabalhos recentes demonstram que, nos mais diversos domínios de aplicação, as RFs possuem desempenho preditivo tão bom quanto o de métodos bem mais complexos e computacionalmente caros, como as redes neurais profundas. Elas oferecem ainda a vantagem de serem eficientes em bases de dados muito volumosas, já que consideram um número reduzido de atributos candidatos para gerar cada nó. Ainda falando sobre eficiência, é importante observar que as etapas de treinamento e classificação da RF são facilmente paralelizáveis, pois cada árvore individual pode ser alocada a uma diferente CPU do computador.

As RFs obtêm melhores resultados quando há diversidade entre os classificadores base que a compõem, ou seja, de maneira ideal, deve existir baixa correlação entre as suas ADs. Diferentes observações empíricas demonstraram que as RFs não são sensíveis ao número de atributos selecionados como candidatos a cada nó. Considerando uma base composta por *d* atributos, tipicamente *log*2(*d + 1*) costumam ser escolhidos.

Em resumo, um modelo RF tende a possuir acurácia muito superior a uma árvore de decisão gerada pelo CART. Esta melhora na acurácia ocorre principalmente porque o modelo composto reduz a variância dos classificadores individuais. Porém, uma importante desvantagem das RFs em relação ao CART – e que é relevante para problemas no âmbito do serviço público – é o fato de que uma **RF não é diretamente interpretável**, ao contrário do que ocorre com uma árvore de decisão.

|  |
| --- |
| Ensembles  A técnica Random Forest é a mais conhecida dentre as técnicas para a construção de modelos compostos, mais conhecidos como *ensembles* (ou *ensemble models*). Porém existem algumas outras técnicas para o aprendizado de *ensembles*, como *Bagging Meta-Estimator,* *AdaBoost* e *Gradient Boosting*, entre outras.Para uma introdução ao tema, consulte o artigo disponibilizado em [https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-for-ensemble-models/] |

Fim da pág. 9

**Início da pág. 10**

**Título do tópico:** Métodos de classificação

**Título da página:** Algoritmos de classificação: *Support Vector Machine* (SVM)

**Subtítulo da página:**

SVM é um algoritmo desenvolvido em grande parte na AT&T Bell, cujo primeiro artigo científico contendo a sua descrição foi publicado no início dos anos 90. Ele é, sem dúvida, o mais complexo dentre os algoritmos de classificação introduzidos neste curso. Porém, a sua intuição não é difícil de ser entendida. Veja a seguir.

Considere a base de treinamento representada graficamente na Figura 12. Nela, temos objetos de duas classes – quadradinhos vermelhos e bolinhas azuis – e uma fronteira de decisão gerada por algum algoritmo de classificação. Esta fronteira de decisão separa os objetos dessas classes. Ela pode também ser chamada de **hiperplano separador**. A figura mostra ainda dois novos objetos *A* e *B*, que deverão ser classificados de acordo com o hiperplano separador.

Inserir figura.

Figura 12. Representação gráfica de uma base de dados de treinamento com objetos de duas classes. Apresenta-se ainda um hiperplano separador dos objetos dessas classes e dois novos objetos *A* e *B*.

Gráfico, Gráfico de dispersão

Descrição gerada automaticamente

Observando a figura, vemos que *A* está muito distante da fronteira gerada pelo algoritmo. Sendo assim, se você “arriscasse” uma predição para *A*, certamente diria que é um objeto da classe quadradinho vermelho com **muita confiança**. Por outro lado, o novo objeto *B* está muito próximo da fronteira, mas ainda do lado dos objetos da classe quadradinho vermelho. É natural pensar que caso essa fronteira se movimentasse um pouco, talvez *B* passasse para o lado das bolinhas azuis. Na prática, a movimentação da fronteira pode ocorrer em diferentes situações. Por exemplo, se obtivermos novos objetos de treinamento e construirmos novamente o modelo com a base de dados modificada. Ou então se usarmos a mesma base de treinamento, porém empregando outro algoritmo para aprender o modelo.

Sendo assim, intuitivamente, temos muito mais confiança na predição de *A* do que de *B*. Em outras palavras, se tivéssemos que escolher um dentre os dois novos objetos para apostar qual deles é da classe quadradinho vermelho, certamente preferiríamos apostar em *A* para não corrermos risco de perder dinheiro! E o SVM trabalha exatamente considerando essa ideia: a partir de uma base de dados de treinamento, é interessante encontrar uma fronteira de decisão que nos permita fazer todas as classificações de forma correta e com a maior confiança possível. Para tal, o SVM busca pela fronteira que tenha a máxima distância para os objetos das duas classes.

Na próxima seção, apresentamos a ideia básica adotada na etapa de treinamento do algoritmo. Como mencionado no início da seção, o algoritmo é complexo e utiliza diversas técnicas matemáticas (desde princípios da geometria analítica até técnicas de otimização). Uma explicação detalhada sobre o SVM necessita de muitas páginas de texto e se encontra fora do escopo de nosso curso. Porém, pode ser encontrada em [https://see.stanford.edu/materials/aimlcs229/cs229-notes3.pdf].

**Título de nível 2:** ETAPA DE TREINAMENTO

O princípio para treinar um classificador SVM será explicado com o uso da base de dados da Tabela 7. Ela contém dois atributos preditivos, X1 e X2, e um atributo classe Y {-1, 1} – o SVM adota essa notação em vez de {0, 1}. Veja que há 4 objetos com o rótulo de classe 1 e que há outros 5 da classe -1.

Inserir figura.

Tabela 7: Base de dados para treinamento de um modelo SVM

|  |  |  |
| --- | --- | --- |
| **X1** | **X2** | **Y** |
| 1 | 5 | 1 |
| 4 | 5 | 1 |
| 2 | 8 | 1 |
| 3 | 6 | 1 |
| 8 | 3 | -1 |
| 7 | 4 | -1 |
| 10 | 2 | -1 |
| 10 | 5 | -1 |
| 8 | 7 | -1 |

Se representarmos essa base de dados graficamente, veremos que os dados são **linearmente separáveis**. Isso significa que uma linha reta (hiperplano) pode ser desenhada de modo que os objetos da classe +1 fiquem de um lado e os da classe -1 do outro (dizemos que a reta gera um modelo com erro de treinamento igual a 0). Mais precisamente, infinitos hiperplanos separadores poderiam ser desenhados como mostra a Figura 13. Todos esses hiperplanos classificam perfeitamente os objetos de treinamento. Porém, não há garantias de que a mesma performance se repetirá para novos objetos. Desta forma, o SVM deverá escolher um deles, baseado em quão bem funcionará para novos objetos.

Inserir figura.

Figura 13. Infinitos hiperplanos separadores com erro de treinamento igual a zero.

Gráfico

Descrição gerada automaticamente

Mas então, qual hiperplano será escolhido? A resposta é: o SVM irá procurar pelo **hiperplano de margem máxima** (*Maximum Margin Hyperplane* – MMH). Basicamente é o hiperplano que consegue classificar perfeitamente os objetos de treino e, ao mesmo tempo, é capaz de produzir a maior distância (margem) entre qualquer objeto da classe -1 e qualquer outro objeto da classe 1. A Figura 14 mostra um exemplo.

Mas por que encontrar o MMH? Da intuição apresentada no início da aula, nós vimos que em um hiperplano onde a margem é pequena, qualquer perturbação na fronteira de decisão pode ter impacto significante na classificação. Em outras palavras, fronteiras com margem pequena são mais suscetíveis ao superajuste – quando o classificador se ajustou em excesso ao conjunto de treinamento. Portanto, quanto maior a margem, maior a confiança da predição e menor a chance de ter havido superajuste.

Inserir figura.

Figura 14: *MMH* é o hiperplano de margem máxima e *d* é a margem máxima. Pontos acima (ou à esquerda) de H1 pertencem à classe +1 e pontos abaixo (ou à direita) de H2 pertencem à classe -1. Os pontos destacados são os mais próximos de MMH para cada classe e são chamados de Support Vectors (SVs).

Uma imagem contendo Gráfico de linhas

Descrição gerada automaticamente

Ok, mas como a busca é realizada? Para começar, a equação de um hiperplano separador pode ser escrita como: *WX* + *b* = 0, onde *W* = {*w1*, *w2*, ..., *wn*} é um vetor de pesos e *b* é um escalar. Qualquer ponto que satisfaça à equação cairá sobre a linha da fronteira. Em nosso exemplo, temos dois atributos preditivos, X1 e X2. Logo a equação do hiperplano separador seria escrita como:

*w1x1* + *w2x2* + *b* = 0

Podemos reescalar os parâmetros w e b de forma que 2 hiperplanos paralelos H1 e H2 sejam expressos na forma abaixo. Assim, qualquer objeto sobre ou acima de H1 pertence à classe +1 e qualquer objeto sobre ou abaixo de H2 pertence à classe -1 (ambos são mostrados na Figura 14).

H1: *w1x1* + *w2x2* + *b* = 1

H2: *w1x1* + *w2x2* + *b* = -1

Observando a Figura 14, podemos notar que temos dois pontos, um quadradinho e uma bolinha, que são os mais próximos de H1 e H2, respectivamente. Eles são chamados de **Support Vectors** (SVs). O objetivo do SVM é exatamente encontrar esses SV’s e, por consequência, encontrar a MMH.

Treinar o SVM, na verdade, envolve estimar os parâmetros *w* e *b* do modelo. Eles devem ser escolhidos de modo que as duas condições sejam satisfeitas:

wxi + b ≥ 1 se yi = 1  
wxi + b ≤ -1 se yi = -1

Essas inequações acima podem ser reescritas como:

yi (wxi + b) ≥ 1

Não seria difícil resolvê-la, mas o SVM impõe um requisito adicional: a margem deve ser máxima! E maximizar a margem equivale a minimizar a seguinte função objetivo:

f(w) = ||w||2 / 2

Então, temos o problema final definido:

Minimizar ||w||2 / 2

s.a. yi (wxi + b) ≥ 1

A função objetivo é quadrática e as restrições lineares em *w* e *b*. Trata-se de um problema de otimização convexo, resolvido pelo método padrão “multiplicador de Lagrange” (consulte [https://see.stanford.edu/materials/aimlcs229/cs229-notes3.pdf] para obter detalhes). Não é algo trivial de ser resolvido, mas felizmente os softwares estatísticos costumam ter pacotes para resolver problemas deste tipo.

**Título de nível 2**: ETAPA DE CLASSIFICAÇÃO

O produto final do treinamento de um classificador SVM é uma equação que define a MMH e é baseada na formulação Lagrangiana:

É a partir dessa fórmula que podemos classificar novos objetos. Basicamente, basta “plugar” as informações de um novo objeto (representado na fórmula por *t*). Se o resultado for positivo, o novo objeto será classificado como +1 e se o resultado for negativo, ele será classificado como -1.

Os componentes da fórmula são os seguintes:

* l: número de SV’s
  + Para dados linearmente separáveis, os SV’s são um subconjunto dos objetos de treino.
  + Veja que no somatório, os SV’s são os únicos objetos da base de treinamento que são levados em consideração.
* xi: é um SV.
* yi: rótulo de classe do SV xi
* t: objeto de teste
* a e bo: parâmetros numéricos determinados pelo algoritmo SVM (ai são multiplicadores Lagrangianos)

**Título de nível 2:** DISCUSSÃO

Para o algoritmo SVM, os SV’s são os objetos de treinamento essenciais. Incrivelmente, se todos os outros objetos forem removidos e o treinamento repetido, o mesmo hiperplano separador será encontrado. Com isso, o SVM consegue definir um modelo de classificação baseado em pouquíssimos elementos da base de dados original.

Nosso exemplo abordou uma situação onde os dados eram linearmente separáveis. Porém, o SVM pode ser estendido para o caso em que os dados possuem fronteira de decisão não linear, isto é, quando não é possível traçar uma reta que separe perfeitamente as classes, como é o caso da base de dados representada na Figura 15.

Inserir figura.

Figura 15: Representação gráfica de uma base de dados de treinamento com objetos de duas classes, onde a fronteira de decisão é não linear.

Uma imagem contendo Gráfico

Descrição gerada automaticamente

Para situações deste tipo, o SVM é estendido com dois passos:

* Passo 1: transformamos os dados de entrada originais para uma dimensão maior utilizando um mapeamento não linear.
* Passo 2: procuramos por um hiperplano separador linear no novo espaço. Mais precisamente, buscamos a MMH no novo espaço.

Mas como escolher o mapeamento? Como assegurar que, no novo espaço, haverá uma separação linear? Felizmente existe uma solução padrão: aplicar uma **função Kernel** aos dados originais. Trata-se de um tipo de função que garante a existência de uma separação linear no novo espaço e que nos possibilita evitar cálculos complexos para realizar a operação de mapeamento. Alguns exemplos de função Kernel são: Kernel polinomial de grau p, Gaussian Radial Basis Function Kernel (RBF) e Sigmoid Kernel. Veja mais em: [https://www.kdnuggets.com/2016/06/select-support-vector-machine-kernels.html].

Conforme visto, o SVM realiza apenas a classificação binária. Porém, é possível estendê-lo para a classificação multiclasse utilizando abordagens simples. Por exemplo, dados *m* rótulos de classes, treinar *m* classificadores binários (um para cada rótulo classe). Neste caso, um objeto de teste será associado ao maior valor positivo entre as previsões de todos os classificadores. Há algumas outras abordagens possíveis e mais informações podem ser obtidas em [https://www.kdnuggets.com/2020/08/one-vs-rest-one-multi-class-classification.html].

O SVM é efetivo em espaços de grande dimensão e, nestes casos, costuma ter performance preditiva superior à de métodos mais simples, como CART, k-NN e naïve Bayes. Ele tem sido utilizado para resolver problemas práticos das mais diversas áreas – o algoritmo é versátil, tendo em vista que diferentes funções Kernel podem ser especificadas como função de mapeamento. Suas principais desvantagens são a complexidade, o fato de ser um classificador caixa-preta e o alto tempo de treinamento.

Fim da pág. 10

**Início da pág. 11**

**Título do tópico:** Métodos de classificação

**Título da página:** Algoritmos de classificação: Multi-Layer Perceptron (MLP)

**Subtítulo da página:**

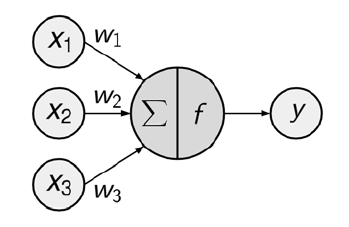
O Multi-Layer Perceptron (MLP) é um subtipo de rede neural, que pode ser considerado como a porta de entrada dos modelos de redes neurais, pois são amplamente utilizados por serem mais simples. Por isso, antes de começarmos a falar sobre o MLP, precisamos fazer algumas definições de redes neurais.

Uma Rede Neural Artificial (RNA) modela a relação entre um conjunto de entrada e um conjunto de saída. As RNA’s são aprendizes versáteis que podem ser aplicadas a praticamente qualquer tarefa de ciência de dados: classificação (prever o rótulo de classe de cada objeto), regressão (prever uma saída de valor contínuo) e até mesmo reconhecimento de padrões em modelos não supervisionados.

Conforme ilustrado na Figura 16, o diagrama de rede define uma relação entre os dados de entrada (variáveis ​​X) e os dados de saída (variável Y), sendo chamados, respectivamente, de Camada de Entrada e Camada de Saída, e cada elemento das camadas representa um nó (ou neurônio). Portanto, no exemplo da Figura 16 temos três nós na camada de entrada e apenas um nó na camada de saída. Cada ligação dos nós da camada de entrada é ponderado (valores w) de acordo com sua importância, esses pesos são então somados e transmitidos de acordo com uma função de ativação denotada por f.

Inserir figura.

Figura 16: Exemplo de diagrama de Rede Neural com 2 camadas.



Uma rede neural pode ser definida de acordo com as seguintes características:

1. Função de ativação: que transforma a conexão de um neurônio da rede de entrada para ser transmitido adiante.
2. Topologia de rede: que descreve o número de neurônios no modelo, bem como o número de camadas e a maneira como eles estão conectados.
3. O algoritmo de treinamento: que especifica como os pesos das conexões são calculados.
4. **Função de ativação**

É o mecanismo pelo qual o neurônio processa as informações e as repassa por toda a rede. As funções de ativação de RNA podem ser escolhidas com base em sua capacidade de demonstrar características matemáticas desejáveis ​​e relacionamentos de modelo entre os dados.

Talvez a mais utilizada seja a função de ativação sigmóide (ou logística), em que os valores de saída podem ser qualquer valor no intervalo de 0 a 1. Ainda existem várias outras opções de funções de ativação, tais como: linear, tangente hiperbólica, gaussiana, etc. Uma função de ativação linear resulta em uma rede neural muito semelhante a um modelo de regressão linear, enquanto uma função de ativação gaussiana resulta em um modelo chamado de rede Radial Basis Function (RBF).

1. **Topologia de rede**

A topologia determina a complexidade das tarefas que podem ser aprendidas pela rede. Embora existam inúmeras formas de topologia de rede, elas podem ser diferenciadas por três características principais:

1. O número de camadas
2. O número de nós em cada camada
3. A direção em que as informações caminham na rede

**2.a) Número de Camadas**

Nós vimos na Figura 16 uma rede muito simples, com apenas duas camadas. Os sinais resultantes dos nós de entrada são recebidos pelo nó de saída, que usa sua própria função de ativação para gerar uma previsão final. Esse tipo de rede é chamado de rede de camada única (em inglês, single-layer network), pois só tem um conjunto de pesos.

Como você pode esperar, uma maneira de criar redes mais complexas é adicionando camadas intermediárias (ou ocultas). Uma rede multicamada (em inglês, multilayer network) adiciona uma ou mais camadas ocultas (em inglês, hidden layers) que transmitem os sinais dos nós de entrada antes de alcançar o nó de saída. Na Figura 17 temos uma rede com apenas uma camada oculta (com três nós) e dois conjuntos de pesos (um conjunto que liga a camada de entrada com a camada oculta, e outro conjunto que liga a camada oculta com a camada de saída).

Inserir figura.

Figura 17: Exemplo de Rede com 3 camadas: Entrada, Oculta, Saída



Além disso, no exemplo da Figura 17 temos uma rede multicamada totalmente conectada (em inglês, fully connected), o que significa que cada nó de uma camada anterior está conectado a cada nó da próxima camada, e embora ocorra na maioria dos casos, isso não é necessário.

**2.b) O número de nós em cada camada**

O número de nós da camada de entrada é predeterminado pelo número de atributos nos dados de entrada. Em geral, é dado pela quantidade de variáveis X de interesse. Da mesma forma, o número de nós da camada de saída é predeterminado pelo número de resultados a serem modelados ou pelo número de rótulos de classe a serem classificados, de acordo com a variável alvo Y.

No entanto, o número de nós na camada oculta é deixado para o usuário decidir antes de treinar o modelo. Infelizmente, não existe uma regra confiável para determinar esse número, mas deve-se levar em conta que grandes redes neurais geralmente são mais lentas para treinar.

Uma prática recomendada é usar a parcimônia, ou seja, o menor número de nós que apresentem um desempenho satisfatório em um conjunto de dados de validação. Na maioria dos casos, mesmo com apenas um pequeno número de nós ocultos, a rede neural pode oferecer uma enorme capacidade de aprendizado.

**2.c) A direção do caminho da informação**

Você deve ter notado que nos exemplos anteriores as pontas das setas foram usadas para indicar a direção para onde os sinais são enviados. Redes nas quais o sinal de entrada é alimentado continuamente em uma única direção até alcançar a camada de saída são chamadas de redes *feedforward*.

Entretanto, se é permitido que os sinais sejam transmitidos em ambas as direções usando loops, essa rede é chamada de *feedback*. Apesar de seu potencial, as redes feedback ainda são pouco utilizadas na prática.

Por outro lado, as redes feedforward têm sido amplamente aplicadas aos problemas do mundo real. Na verdade, a rede feedforward multicamada é a topologia padrão de uma RNA, e por isso também é chamada de **Multi-layer Perceptron (MLP)**.

1. **Algoritmo de Treinamento com retropropagação**

Treinar uma rede neural ajustando os pesos de conexão é muito intensivo computacionalmente. À medida que a rede neural processa os dados de entrada, as conexões entre os neurônios são fortalecidas ou enfraquecidas, e os pesos de conexão refletem esses padrões observados ao longo do tempo. O algoritmo, que usa uma estratégia de retropropagação de erros, agora é conhecido simplesmente como retropropagação (em inglês, backpropagation).

Em sua forma mais geral, pode ser descrito resumidamente:

1) iniciar os pesos;

2) propagar os dados de entrada (feedforward);

3) retropropagar os erros;

4) Repete 2 e 3 até que o critério de parada seja satisfeito.

Na inicialização, normalmente, os pesos iniciais são definidos aleatoriamente. O algoritmo de retropropagação passa, portanto, por duas fases iterativas: feedfoward dos dados e retropropagação dos erros, em que cada iteração do algoritmo é conhecida como uma época (em inglês, epoch).

Na fase Feedforward os neurônios são ativados em sequência, partindo da camada de entrada, passando pela camada oculta, até alcançar a camada de saída. Ao longo do caminho, são aplicados os pesos e a função de ativação em cada neurônio. Ao atingir a camada final, temos um valor estimado na saída.

Na fase de retropropagação o valor estimado na saída da fase anterior é comparado com o valor real dos dados de treinamento. Essa diferença resulta em um erro que é propagado para trás na rede, para assim modificar os pesos e reduzir os erros das próximas épocas. Aqui, os erros partem da camada de saída, passando pela camada oculta, até alcançar os pesos da camada de entrada. Com o tempo, a rede usa as informações enviadas para trás para atualizar os pesos, e assim reduzir o erro total da rede.

Ao fim do algoritmo, os critérios de parada podem ser: os erros da época anterior estão abaixo do limite especificado; ou a porcentagem de registros classificados incorretamente na época anterior está abaixo do limite especificado; ou quando atinge o número de épocas pré-determinado.

O algoritmo de retropropagação usa a derivada da função de ativação de cada nó para identificar a direção do gradiente de cada um dos pesos. O gradiente sugere o quanto o erro pode ser reduzido caso haja uma mudança no peso. O algoritmo tentará atualizar os pesos que resultam na maior redução no erro total da rede, através de uma taxa de aprendizado (em inglês, learning rate). Quanto maior a taxa de aprendizado, mais rápido o algoritmo tentará otimizar os gradientes, o que pode reduzir o tempo de treinamento.

**Título de nível 2:** DISCUSSÃO

Entre as vantagens da utilização de redes neurais podemos enumerar: adaptação a problemas de classificação ou regressão, pois a saída pode ser discreta ou contínua; abordagens de modelagem mais precisas; necessita de poucas suposições sobre os relacionamentos subjacentes dos dados; geralmente a precisão da previsão é elevada; robusta, pois funciona mesmo quando os dados de treinamento contêm erros.

Por outro lado, as desvantagens seriam: a dificuldade de interpretação de como os resultados foram gerados; o treinamento pode gerar modelos que subestimam ou sobrestimam os dados; dificuldade em obter a topologia mais adequada ao problema; computacionalmente intensivo e tempo longo para treinar, principalmente se a topologia da rede for complexa.

Em relação a esse último ponto, quando há mais de duas camadas ocultas, já pode ser considerado um problema de Deep Learning (que não está no escopo desse curso). Onde temos alguns outros tipos de redes neurais, tais como: Siamese Neural Network, Convolutional Neural Network, Long short term memory (LSTM) network, Deep belief network, etc.

**Exemplo**

Pegar uma base de exemplo e fazer o passo-a-passo via R ou Python.

|  |
| --- |
| Se você desejar aprofundar seus conhecimentos sobre os algoritmos apresentados nessa seção, poderá consultar o livro disponibilizado no site Datamining. [link Datamining: <https://dataminingbook.info/book_html/>. |

**Fim da pág. 11**

**Início da pág. 12**

**Título do tópico: Como avaliar a qualidade de um classificador**

**Título da página:** Estimativa da eficácia de um modelo de classificação

**Subtítulo da página:**

Ao longo desse módulo, aprendemos que, para criar um classificador basta termos em mãos uma base de dados de treinamento e escolhermos um algoritmo de classificação. Entretanto, antes de colocar um classificador em produção é preciso **estimar a sua eficácia** (ou desempenho preditivo), isto é, determinar, com algum grau de certeza, se o classificador terá um bom desempenho para classificar novos objetos. Afinal de contas, um classificador que faz muitas previsões erradas não teria utilidade prática.

Mas como fazer essa estimativa? O princípio fundamental empregado na avaliação de classificadores consiste em **utilizar um conjunto de dados de teste** formado por objetos que **não estiveram envolvidos no processo de treinamento** do classificador. Basicamente, deve-se aferir se o modelo gerado com os dados de treinamento conseguirá acertar uma quantidade significativa de classificações quando aplicado aos objetos de teste. Ou seja, iremos verificar se a **acurácia** (percentual de classificações corretas) do modelo será alta com os objetos de teste – objetos que não fizeram parte do treinamento e que, por isso, são “novos objetos” para o classificador.

As duas próximas subseções introduzem, respectivamente, os métodos para avaliação de classificadores e as medidas que podem ser empregadas para mensurar diferentes aspectos relacionados ao desempenho preditivo de um classificador.

**1. Métodos para a Avaliação de Modelos de Classificação**

O método conhecido como *holdout* é o mais simples dentre os utilizados para a avaliação de modelos de classificação. Nesta abordagem, a base de dados rotulada é dividida de forma aleatória em dois conjuntos (ou partições) independentes: conjunto de treinamento e conjunto de teste (Figura 18). Tipicamente, dois terços dos dados são alocados para treino e o terço restante é alocado para teste. A divisão deve ser feita de maneira aleatória, mas idealmente o processo deve garantir que cada classe seja adequadamente representada tanto no conjunto de treinamento como no de teste (processo conhecido como estratificação). Uma vez que as duas partições tenham sido definidas, a partição de treino é usada para criar o modelo e a partição de teste para testá-lo – testar significa medir a acurácia (e outros indicadores) do modelo na partição de teste.

Inserir figura.

Figura 18: Partições geradas pelo método *holdout*.

Forma, Retângulo

Descrição gerada automaticamente

O método *holdout* é simples, mas tem uma desvantagem importante: para estimar se um classificador possui boa qualidade ou não, ele aposta todas as suas fichas em uma **única avaliação** envolvendo um modelo construído com uma partição de treino e testado com uma partição de teste, ambas definidas de forma aleatória. Fazer um único teste usando apenas essas duas partições parece um pouco arriscado, não?

Felizmente, existem técnicas capazes de obter estimativas mais confiáveis para o desempenho preditivo de um classificador. Uma delas é a validação cruzada (*cross-validation*). Nesta técnica, a base rotulada é dividida em *k* partições (*folds*) *D1*, *D2*, ..., *Dk*, onde todas deverão ter o tamanho igual ou aproximadamente igual. Na Figura 19, apresenta-se um exemplo onde *k*=10, um dos valores mais comumente utilizados. Após a divisão, são realizadas *k* rodadas de treino e teste. A cada iteração *i*, a partição *Di* é reservada para teste e as *k-1* partições restantes são utilizadas para treinar o modelo. Por exemplo, para *k*=1, reserva-se as observações pertencentes a *D1* para teste, e utiliza-se as observações pertencentes a *D2* até as pertencentes a *D10* para treinamento. Quando *k*=2, reserva-se as observações de *D2* para teste, e utiliza-se as observações de *D1* e *D3* até *D10* para treinamento. E assim, sucessivamente. A acurácia final estimada para o modelo será igual à acurácia média das *k* rodadas. Sendo assim, ao contrário do que ocorre com o método *holdout*, na validação cruzada a estimativa da acurácia não é feita considerando o resultado obtido sobre uma única partição de treino e uma única teste, mas é na verdade uma média do desempenho preditivo de *k* modelos diferentes, testados sobre *k* bases de testes diferente. Isso constitui uma importante vantagem.

Inserir figura.

Figura 19: Partições geradas pelo método de validação cruzada, quando *k*=10.

Uma imagem contendo Diagrama

Descrição gerada automaticamente

|  |
| --- |
| **Para saber mais:**  É importante comentar que além da validação cruzada, existem outras técnicas para estimar a qualidade de classificadores, dentre as quais:   * Validação cruzada repetida [https://machinelearningmastery.com/repeated-k-fold-cross-validation-with-python/] * *leave-one-out* [https://www.statology.org/leave-one-out-cross-validation/] * *bootstrap* [https://sebastianraschka.com/blog/2016/model-evaluation-selection-part2.html] |

**2. Métricas de Desempenho Preditivo**

Nessa seção, apresentaremos métricas capazes de medir diferentes aspectos que definem a efetividade de um classificador. As métricas que apresentaremos podem ser calculadas a partir de uma importante estrutura conhecida como **matriz de confusão** (MC).

A finalidade de uma MC é armazenar todos os diferentes tipos de erros e acertos realizados pelo classificador ocorridos durante o processamento de um conjunto de objetos de teste. Trata-se simplesmente de uma matriz quadrada que indica as classificações corretas e erradas após o classificador ter sido testado. Mais formalmente, uma MC é uma matriz quadrada *m* x *m*, onde *m* representa o número de rótulos de classe envolvidos no problema. Cada célula *cij* denota o número de objetos de teste que o classificador associou à classe *i* e que, de fato, pertencem à classe *j*. Desta forma, as células da diagonal principal sempre irão conter o número de objetos corretamente classificados pelo modelo.

A Tabela 8 apresenta o formato de uma MC hipotética produzida após o processo de teste de um classificador em um problema com *m* = 3, ou seja, um problema de classificação multiclasse envolvendo 3 classes, neste caso {“a”, “b”, “c”}. Nessa matriz, como em qualquer outra, as classes reais estão nas linhas e as preditas nas colunas. Veja que neste exemplo, o classificador acertou a maioria das classificações (a maior parte dos resultados caiu na diagonal principal).

Inserir figura.

Tabela 8: Matriz de confusão hipotética em um problema com 3 classes (*m* = *3*)

|  |  |  |  |
| --- | --- | --- | --- |
|  | **a** | **b** | **c** |
| **a** | 49 | 14 | 7 |
| **b** | 7 | 315 | 0 |
| **c** | 8 | 0 | 77 |

Considere a primeira linha, que trata dos resultados referentes ao rótulo de classe “a”. O valor 49 indica que 49 objetos da base de teste cuja classe real é “a” foram corretamente classificados como classe “a”. Já o valor 14 indica o número de objetos cuja classe real é “a”, mas que foram incorretamente classificados como “b”. Por fim, o valor 7 indica o número de objetos cuja classe real é “a”, mas que foram incorretamente classificados como “c”. De maneira análoga a segunda e terceira linha da tabela apresentam os resultados referentes às classes “b” e “c”, respectivamente.

A partir dessa matriz, torna-se possível calcular diferentes medidas de desempenho. A seguir apresentaremos as 4 mais conhecidas: Acurácia (*acuraccy*), Precisão (*precision*), Revocação (*recall*) e Medida F1 (*f1 measure)*.

**Acurácia (*Ac*)**: esta medida avalia o desempenho médio do classificador, consistindo na proporção de instâncias corretamente classificadas durante o processo de teste. Nesta fórmula, *d* é o total de objetos de teste, *m* é o número de classes e *TPk* corresponde ao número de objetos da classe *k* que foram corretamente classificados como *k* durante o processo de teste do classificador. *TP* é a abreviação de *True Positives* – ou verdadeiro positivos (sempre corresponde a um valor que faz parte da diagonal principal da MC).

No caso da MC da Tabela 8, o total de objetos da base de teste é *d* = 477 (soma dos valores de todas as células da matriz). Sendo assim, a Acurácia do classificador na base de teste é calculada da seguinte forma:

Ac = (49 + 315 + 77) / 477 = 0,9245

Ou seja, o classificador acertou a classe de 92,45% dos objetos de teste.

Entretanto, também é interessante avaliar o desempenho do classificador **por classe**, pois muitas vezes o classificador pode ser efetivo para classificar objetos de uma determinada classe, mas não ser efetivo para outras. Para tal, podemos fazer uso das medidas de Precisão e Revocação.

**Precisão (*Pr*)**: a precisão da classe *ck* mede a quantidade de exemplos classificados como positivos, que de fato são positivos.

Então vamos começar pela classe “a”. Considerando a MC da Tabela 8, quem são os TPs (*True Positives)* dessa classe? São os objetos de teste que são “a” e foram classificados como “a”. Esta informação está representada em uma única célula da matriz, que é a célula [1,1] (49 casos). E quem são os FPs (*False Positives –* ou falsos positivos) da classe “a”? São os objetos que não são “a”, mas que o classificador equivocadamente classificou como “a” durante o processo de teste. Esses objetos podem ser achados na 1a coluna da matriz, nas linhas 2 e 3 (linhas referentes às classes reais “b” e “c”). Ou seja, o denominador é o total da coluna “a”.

Tendo identificado os TPs e FPs, o cálculo de Pr(“a”) é trivial. Basta fazer:

Pr(“a”) = TPa / (TPa + FPa) = 49 / (49 + 7 + 8) = 49 / 64 = 0,7656

Analogamente, para as classes “b” e “c”, temos:

Pr(“b”) = TPb / (TPb + FPc) = 315 / (315 + 14 + 0) = 315 / 329 = 0,9574

Pr(“c”) = TPc / (TPc + FPc) = 77 / (77 + 7 + 0) = 77 / 84 = 0,9167

Veja que a classe “a” tem um valor de precisão bem menor que o das outras duas classes. Observando a matriz, podemos notar que realmente esta é a classe com a maior proporção de falsos positivos.

Agora que temos a Precisão de cada classe, precisamos arrumar um jeito de computar um valor que represente a Precisão do classificador. Para tal, basta calcular a média dos 3 valores. Esse cálculo pode ser feito usando a abordagem macro ou a abordagem micro.

Prmacro = somar os 3 valores e dividir por 3 = (0,7656 + 0,9574 + 0,9167) / 3 = 0,8799

Prmicro = soma ponderada dos 3 valores, dividido pelo total de registros da base de dados de teste. Ou seja, para cada classe *k* eu pego a *Prk* e multiplico pelo número de registros da classe *k* na base de dados de teste (total das linhas da Tabela 8).

Prmicro = (0,7656 \* 70 + 0,9574 \* 322 + 0,9167 \* 85) / 477 = 0,9220

**Revocação (*Re*)**: a revocação ou *recall* da classe *ck* mede a porcentagem de exemplos positivos reais que foram classificados como positivos.

Mais uma vez, vamos começar pela classe “a”. Considerando a MC da Tabela 8, os FNs (*False Negatives* ou falsos negativos*)* são os objetos cuja classe real é “a”, mas que o classificador equivocadamente classificou como “b” ou “c” durante o processo de teste. Esses objetos podem ser achados na 1a linha da matriz, nas colunas 2 e 3 (linhas referentes às classes preditas como “b” e “c”). Ou seja, o denominador é o total da linha “a”.

Tendo identificado os TPs e FNs, o cálculo de Re(“a”) é trivial. Basta fazer:

Re(“a”) = TPa / (TPa + FNa) = 49 / (49 + 14 + 7) = 49 / 70 = 0,7000

Analogamente, para as classes “b” e “c”, temos:

Re(“b”) = TPb / (TPb + FNc) = 315 / (315 + 7+ 0) = 315 / 322 = 0,9783

Re(“c”) = TPc / (TPc + FNc) = 77 / (77 + 8 + 0) = 77 / 85 = 0,9059

Podemos então usar a abordagem macro ou micro para obter o Recall do classificador:

Remacro = (0,7000 + 0,9783 + 0,9059) / 3 = 0,8614

Remicro = (0,7000 \* 70 + 0,9783 \* 322 + 0,9059 \* 85) / 477 = 0,9246

**Medida F1 (*F1*)**: trata-se da média harmônica entre a Precisão e a Revocação. É útil, pois na maior parte das situações práticas um classificador costuma ter um bom desempenho em uma das medidas, mas ruim na outra (não foi o caso de nosso exemplo, onde a MC ficou muito boa e, por isso, os valores de todas as medidas ficaram altos).

Também calculamos a F1 por classe e depois tiramos a média macro ou micro. Sendo assim, abaixo estamos apresentando apenas o exemplo do cálculo de F1 para a classe “a”:

F1(“a”) = 2 x Pr(“a”) x Re(“a”) / (Pr(“a”) + Re(“a”)) = 2 x 0,7656 x 0,7000 / (0,7656 + 0,7000) = 0,7313

**Caso Especial – Classificação Binária**

Provavelmente, o maior número de problemas práticos de classificação é binário. Por exemplo, considerando o IBGE, um problema real deste tipo seria a identificação de fraudes em questionário (classes {“Sim”, “Não”}, onde “Sim” significa questionário fraudulento e “Não” questionário legítimo).

Como o problema binário é muito comum, o jeito mais usado pelas ferramentas de ciência de dados para calcular as medidas neste tipo de problema é considerar (arbitrariamente) uma das classes como “positiva” e outra como “negativa”. No exemplo da Tabela 9 (hipotético), temos duas classes “C1” e “C2”, onde “C1” foi considerada a classe “positiva” e “C2” a “negativa” pela ferramenta de ciência de dados.

|  |  |  |
| --- | --- | --- |
|  | **C1** | **C2** |
| **C1** | 318 (TP) | 6 (FN) |
| **C2** | 6 (FP) | 170 (TN) |

Tabela 9. Matriz de confusão hipotética em um problema binário (*m* = *2*)

A partir dessa matriz, se o usuário desejar, é possível calcular a Pr, Re e F1 do classificador diretamente, usando a chamada “abordagem binária”. Veja o exemplo a seguir, onde realiza-se o cálculo da precisão:

Pr = TP / (TP + FP) = 318 / (318 + 6) = 0,9815

**Problemas com classes desbalanceadas**

Os exemplos das Tabelas 8 e 9 apresentaram matrizes de confusão onde o desempenho preditivo para todos os rótulos de classe foi muito bom. Porém, não é isso que ocorre sempre na prática, especialmente em problemas onde as classes são **desbalanceadas**, isto é, quando o classificador foi treinado com uma base em que havia predominância de objetos de uma determinada classe. Nesta situação, o classificador tende a apresentar maior eficácia para os objetos da classe majoritária e, muitas vezes, apresenta um desempenho ruim para os objetos da(s) classe(s) minoritária(s).

Por exemplo, considere o problema de identificar fraudes em questionários. Neste caso, é comum que a base rotulada possua muito mais objetos da classe “Não” (questionário legítimo) do que da classe “Sim” (questionário fraudulento). Então, quando ela for dividida nas partições de treino e teste, a partição de treino ficará naturalmente com mais objetos da classe “Não” do que da classe “Sim”. Tipicamente, o classificador aprenderá classificar melhor os objetos “Não” do que os objetos “Sim”, já que ele foi apresentado a mais exemplos de objetos da classe “Não”.

Por exemplo, a MC da Tabela 10, poderia ser uma MC gerada após o teste do classificador de fraudes em questionários. Veja que para a classe majoritária (“Não”), o classificador acertou 35033 classificações e errou 2122, obtendo acurácia de 0,94 (ou 94%). Porém, na classe minoritária (“Sim”), o classificador teve um desempenho bem pior, acertando 7009 classificações e errando 4678, ou seja, a acurácia para essa classe específica é 0,6 (ou 60%). Isso pode representar um problema em um bom número de aplicações, onde deseja-se um desempenho parecido para a classificação de qualquer rótulo. Para conhecer algumas técnicas básicas para o tratamento de bases de dados com classes desbalanceadas, consulte [https://tatianaesc.medium.com/trabalhando-com-classes-desbalanceadas-em-problemas-machine-learning-29ee8db4a049]

Inserir figura.

Tabela 10: Matriz de confusão de um problema binário com classes desbalanceadas

|  |  |  |
| --- | --- | --- |
|  | **Sim** | **Não** |
| **Sim** | 7009 | 4678 |
| **Não** | 2122 | 35033 |

|  |
| --- |
| Há diversas outras medidas de desempenho preditivo para classificadores. Para uma referência completa não apenas sobre essas medidas, mas sobre outros assuntos relacionados à avaliação e classificadores, consulte a apresentação disponibilizada em: <https://www.site.uottawa.ca/~nat/Talks/CanadianAI-Tutorial.pptx> |

Fim da pág. 12.

**ÚLTIMA PÁGINA**

**Curiosidade (opcional)**

Uma situação sobre o tema que seja interessante. Fatos, números e informações históricas, por exemplo

**Ponto de Reflexão (obrigatório)**

Este pode ser um destaque para um desafio ou dilemas éticos do tema. A ética e a qualidade dos produtos e serviços devem ser temas transversais nos cursos do IBGE. Pode também dar ênfase a uma aplicação social do conceito ou tema, estimulando o pensamento crítico e formulação de hipóteses.

**Fechamento [Texto+hiperlinks+mídias] (obrigatório)**

O fechamento da página de introdução é muito importante para manter o interesse e o entendimento das ideias propostas. Busque antecipar os assuntos que virão a seguir ao longo do fechamento de forma crítica e instigante, criando curiosidade. Pode usar perguntas sobre os conceitos e processos que serão desenvolvidos nas próximas páginas do tópico.

**Atividade (opcional)**

Para elaborar as atividades, parta do objetivo específico de aprendizagem do conteúdo que está descrito na Matriz do Curso. As atividades devem fornecer uma gradação de desafios que auxiliem o participante na retomada dos temas e na produção de insights, ou seja, as questões podem fornecer novas formas de conectar a informação lida com novas possibilidades de aplicação do conhecimento, não bastando avaliar a memorização. Para saber mais sobre produção de questões acesse: [Oficina de elaboração de questões.](https://www.academia.edu/37675823/OFICINA_DE_ELABORA%C3%87%C3%83O_DE_QUEST%C3%95ES)

**Para Saber Mais (obrigatório)**

Links para sites, vídeos, artigos, livros, podcasts, etc que se relacionem e complementem de forma interessante a ideia apresentada na página.