

sumarizar_por_classe

Código geral

```
def sumarizar_por_classe(self):  
  
    separado = self.separar_por_classe()  
  
    sumarios = {}  
  
    for valorClasse, instancias in separado.items():  
  
        atributos = zip(*instancias)  
  
        medias = [self.calcularMedia(atributo) for atributo in atributos]  
  
        atributos = zip(*instancias)  
  
        desvios = [self.calcularDesvio(atributo) for atributo in atributos]  
  
        mediasDesvios = zip(medias, desvios)  
  
        mediasDesvios = list(mediasDesvios )  
  
        sumarios [valorClasse] = mediasDesvios  
  
    self.sumarios = sumarios
```

Códigos específicos explicados

separado = self.separar_por_classe()

separar_por_classe retorna um dicionário com três chaves.

Associado a cada chave, uma lista.

Cada lista contém 50 instâncias da Íris.

sumarios = {}

sumarios recebe um dicionário vazio. Sem chaves e sem valores associados às chaves.

atributos = zip(*instancias)

instancias tem cinquenta instâncias da Íris contendo cada uma quatro atributos.

De certa forma, são cinquenta listas de quatro valores cada.

zip(*instancias) retorna quatro listas de cinquenta valores cada.

medias = [self.calcularMedia(atributo) for atributo in atributos]

Para cada lista dentre as quatro listas contidas em atributos, self.calcularMedia calcula sua média.

medias recebe quatro médias. A média de cada um dos quatro atributos da Íris.

desvios = [self.calcularDesvio(atributo) for atributo in atributos]

Para cada lista dentre as quatro listas contidas em atributos, self.calcularDesvio calcula seu desvio.

desvios recebe quatro desvios. O desvio de cada um dos quatro atributos da Íris.

mediasDesvios = zip(medias, desvios)

medias e desvios são duas listas de quatro valores cada.

zip(medias, desvios) retorna quatro listas de dois valores cada.

Cada lista guarda a média e o desvio de um atributo da Íris.

sumarios[valorClasse] = mediasDesvios

sumarios recebe a corrente/atual chave.

Associado à ela, as quatro listas contendo, cada uma, a média e o desvio de um atributo.

for valorClasse, instancias in separado.items():

sumarios: Conteúdo das chaves

Ao fim das três iterações do for, sumarios tem três chaves.

Associado a uma chave, uma lista de quatro sumários.

Um sumário é uma lista que contém a média e o desvio de um atributo.

sumarios: Conteúdo específico

sumarios, independente das chaves, tem doze listas que dizem respeito à média e desvio.

Cada lista contém uma média e um desvio.

Três dessas listas dizem respeito à sepal length, três à petal length, três à sepal width e três à petal width.

Das que dizem respeito à sepal length, uma diz respeito à setosa, outra à virgínica e outra à versicolor.

O mesmo acontece com as que dizem respeito aos outros três atributos.

calcularProbabilidadeValorAtributo

Código geral

```
def calcularProbabilidadeValorAtributo(self, x, media, desvio):  
  
    exponent = math.exp(-(x - media) ** 2 / (2 * desvio ** 2))  
  
    return (1 / (math.sqrt(2 * math.pi) * desvio)) * exponent
```

Códigos específicos explicados

Desnecessário. Código simples.

O que o método faz

Ele recebe um valor de um atributo de uma instância da Íris e a distribuição normal (média e desvio padrão) do mesmo atributo da mesma classe da qual a instância cujo valor pertence faz parte.

Então ele calcula a probabilidade de, por assim dizer, localizar aquele valor naquela distribuição normal.

calcularProbabilidadeClasse

Código geral

```
def calcularProbabilidadeClasse(self, vetorInstancia):  
  
    probabilidades = {}  
  
    for valorClasse, sumariosClasse in self.sumarios.items():  
  
        probabilidades [valorClasse] = 1  
  
        for i in range(len(sumariosClasse)):  
  
            media, desvio = sumariosClasse[i]  
  
            x = vetorInstancia[i]  
  
            probabilidades [valorClasse] *=  
            self.calcularProbabilidadeValorAtributo(x, media,  
            desvio)  
  
    return probabilidades
```

Códigos específicos explicados

for valorClasse, sumariosClasse in self.sumarios.items():

Dentre as três chaves de sumarios, os comandos dentro desse for serão executados sobre a sobre a Setosa.

for i in range(len(sumariosClasse)):

Dentre os quatro sumários contidos na Setosa, os comandos dentro desse for serão executados sobre o primeiro.

media, desvio = sumariosClasse[i]

media e desvio recebem respectivamente os elementos 1 e 2 do primeiro sumário.

x = vetorInstancia[i]

x recebe o valor do primeiro atributo da instância da Íris cujo cálculo da probabilidade da classe está sendo calculado.

probabilidades[valorClasse] *= self.calcularProbabilidadeValorAtributo(x, media, desvio)

O valor 1 de probabilidades[valorClasse] é multiplicado com o resultado do cálculo da probabilidade de encontrar o valor do atributo 1 da instância da Íris dentro da distribuição normal do mesmo atributo da mesma classe.

Resumo

Recebe a nova instância da Íris.

Pega a classe Setosa.

probabilidades [Setosa] = 1.

Pega o sumário do atributo 1.

probabilidade1 = probabilidade do atributo 1 da nova instância da Íris.

probabilidades [Setosa] *= probabilidade1.

Pega o sumário do atributo 2.

probabilidade2 = probabilidade do atributo 2 da nova instância da Íris.

probabilidades [Setosa] *= probabilidade2.

Pega o sumário do atributo 3.

probabilidade3 = probabilidade do atributo 3 da nova instância da Íris.

probabilidades [Setosa] *= probabilidade3.

Pega o sumário do atributo 4.

probabilidade4 = probabilidade do atributo 4 da nova instância da Íris.

probabilidades [Setosa] *= probabilidade4.

Pega a classe Versicolor.

probabilidades [Versicolor] = 1.

Pega o sumário do atributo 1.

probabilidade1 = probabilidade do atributo 1 da nova
instância da Íris.

probabilidades [Versicolor] *= probabilidade1.

Pega o sumário do atributo 2.

probabilidade2 = probabilidade do atributo 2 da nova
instância da Íris.

probabilidades [Versicolor] *= probabilidade2.

Pega o sumário do atributo 3.

probabilidade3 = probabilidade do atributo 3 da nova
instância da Íris.

probabilidades [Versicolor] *= probabilidade3.

Pega o sumário do atributo 4.

probabilidade4 = probabilidade do atributo 4 da nova
instância da Íris.

probabilidades [Versicolor] *= probabilidade4.

Pega a classe Virgínica.

probabilidades [Virgínica] = 1.

Pega o sumário do atributo 1.

probabilidade1 = probabilidade do atributo 1 da nova
instância da Íris.

probabilidades [Virgínica] *= probabilidade1.

Pega o sumário do atributo 2.

probabilidade2 = probabilidade do atributo 2 da nova instância da Íris.

probabilidades [Virgínica] *= probabilidade2.

Pega o sumário do atributo 3.

probabilidade3 = probabilidade do atributo 3 da nova instância da Íris.

probabilidades [Virgínica] *= probabilidade3.

Pega o sumário do atributo 4.

probabilidade4 = probabilidade do atributo 4 da nova instância da Íris.

probabilidades [Virgínica] *= probabilidade4.

O que o método faz

Utiliza calcularProbabilidadeValorAtributo para a sepal length de uma instância da Íris, passando a distribuição normal desse atributo na classe Setosa.

Depois faz o mesmo passando a distribuição da Virgínica.

Depois da Versicolor.

E esse processo de três partes é repetido com os outros três atributos da instância.

predizer

Código geral

```
def predizer(self, vetorInstancia):
```

```
    probabilidades = self.calculaProbabilidadeClasse(vetorInstancia)
```

```
    melhorRotulo, melhorProbabilidade = None, -1
```

```
for valorClasse, probabilidade in probabilidades.items():  
  
    if melhorRotulo is None or probabilidade > melhorProbabilidade:  
  
        melhorProbabilidade = probabilidade  
  
        melhorRotulo = valorClasse  
  
return melhorRotulo
```

Códigos específicos explicados

Desnecessário. Código simples.

main

Códigos específicos explicados

```
instancias = list(zip(iris.sepal_length, iris.sepal_width, iris.petal_length,  
iris.petal_width))
```

instancias recebe 150 listas com 4 valores cada.

```
instanciasTreino, instanciasTeste, classesTreino, classesTeste =  
train_test_split(instancias, classes, test_size=0.3, random_state=42,  
stratify=classes)
```

O argumento `test_size = 0.3` define que as variáveis `instanciasTeste` e `classesTeste` receberão 30 %.

`random_state` é a semente que está sendo utilizada para gerar o valor aleatório que será utilizado para dividir os dados. De modo que, sempre que necessário realizar a divisão dos dados, a mesma semente pode ser utilizada para que o gerador gere a mesma divisão.

`stratify = classes` define que, assim como há 33 % da Setosa, 33 % da Versicolor e 33 % da Virgínica nos dados originais, os dados de treino e de teste também tenham cada um a mesma divisão em três partes iguais.

`instanciasTreino` terá 105 listas de 4 valores cada.

`instanciasTeste` terá 45 listas de 4 valores cada.

classesTreino terá 105 valores.

classesTeste terá 45 valores.