Treinando e testando o algoritmo com metade dos dados

Podemos dividir em duas partes os nossos registros! Por exemplo, podemos deixar os primeiros 50 registros para treinar o nosso algoritmo e os 49 demais para testá-lo, ou então, podemos deixar 80 registros para treiná-lo e os 19 restantes para testá-lo!

Uma das maneiras tradicionais para esses casos é quebrar os nossos dados em 90% e 10%, sendo que, os 90% serão os dados que iremos utilizar para treinar o nosso algoritmo e os 10% serão os dados utilizados para testá-lo! Então vamos alterar o nosso código, observe o X e Y:

X, Y = carregar\_acessos()

Perceba que todos os nossos dados estão concentrados nessas duas variáveis, ou seja, todas as características (X) e todas as marcações (Y) de **treino**! Precisamos separar o X e Y de treino e o de teste. Começaremos pelos nossos dados e marcações de treino:

X, Y = carregar\_acessos()

treino\_dados

treino\_marcacoes

Precisamos adicionar 90% para ambos, então as 90 **primeiras linhas** para cada um deles:

X, Y = carregar\_acessos()

treino\_dados = X[:90]

treino\_marcacoes = Y[:90]

Agora precisamos adicionar os 10% que restaram para as variáveis de teste, porém, dessa vez, precisamos das 9 **últimas linhas**!

X, Y = carregar\_acessos()

treino\_dados = X[:90]

treino\_marcacoes = Y[:90]

teste\_dados = X[-9:]

teste\_marcacoes = Y[-9:]

Vamos testar o nosso código, para isso, abriremos o interpretador do python:

> python

>>>

Agora importaremos o nosso método carregar\_acessos()

>>> from dados import carregar\_acessos

>>>

Então chamamos o método carregar\_acessos() e retornando para o X e Y:

>>> from dados import carregar\_acessos

>>> X,Y = carregar\_acessos()

Vamos criar as nossas variáveis de

treino treino\_dados e treino\_marcacoes:

>>> from dados import carregar\_acessos

>>> X,Y = carregar\_acessos()

>>> treino\_dados = X[:90]

>>> treino\_marcacoes = Y[:90]

Será que funcionou? Vamos imprimir os nossos treino\_dados:

>>> treino\_dados

[[1, 1, 0], [1, 1, 0], [1, 1, 0], [1, 1, 0], [1, 1, 0], [1, 0, 1], [1, 1, 0], [1, 0, 1], [1, 1, 0], [1, 0, 1], [1, 1, 0], [1, 0, 1], [0, 0, 1], [0, 0, 1], [0, 0, 1], [0, 0, 1], [0, 0, 1], [0, 1, 0], [0, 0, 1], [0, 1, 0], [0, 0, 1], [0, 1, 0], [0, 0, 1], [0, 1, 0], [1, 0, 1], [1, 1, 1], [1, 1, 1], [1, 0, 1], [0, 1, 0], [0, 0, 1], [0, 1, 0], [1, 0, 1], [0, 0, 1], [1, 0, 1], [1, 0, 1], [1, 0, 1], [1, 0, 1], [0, 0, 1], [1, 1, 0], [1, 0, 1], [0, 1, 0], [0, 0, 1], [0, 1, 0], [1, 0, 1], [0, 0, 1], [0, 0, 1], [1, 0, 1], [1, 0, 1], [1, 1, 0], [1, 1, 0], [1, 1, 0], [1, 1, 0], [1, 0, 1], [1, 1, 0], [1, 0, 1], [1, 1, 0], [1, 0, 1], [1, 1, 0], [1, 0, 1], [0, 0, 1], [1, 0, 1], [0, 1, 0], [0, 0, 1], [0, 1, 0], [1, 0, 1], [0, 1, 1], [0, 0, 1], [1, 0, 1], [1, 0, 1], [1, 1, 0], [1, 1, 0], [0, 0, 1], [0, 0, 1], [0, 0, 1], [0, 0, 1], [0, 1, 0], [0, 0, 1], [0, 1, 0], [0, 0, 1], [1, 1, 0], [0, 0, 1], [0, 0, 1], [0, 0, 1], [1, 0, 1], [1, 0, 1], [1, 1, 0], [1, 1, 0], [0, 0, 1], [0, 0, 1], [1, 0, 1]]

Aparentemente foram impressos todos os dados, mas e o tamanho desse array? Será que são 90 dados? Vejamos:

>>> len(treino\_dados)

90

Funcionou conforme o esperado! Agora faremos o mesmo para os dados e marcações de teste:

>>> teste\_dados = X[-9:]

>>> teste\_marcacoes = Y[-9:]

>>> len(teste\_dados)

9

>>> len(teste\_marcacoes)

9

>>> teste\_dados

>>> [[1, 0, 1], [1, 1, 0], [1, 1, 0], [0, 0, 1], [0, 0, 1], [1, 1, 1], [0, 0, 1], [0, 1, 0], [0, 0, 0]]

>>> teste\_marcacoes

[1, 0, 0, 0, 0, 1, 0, 0, 0]

Conseguimos separar todos os nossos dados para treino e para teste. Agora precisamos alterar o nosso código para utilizar esses novos dados. Começaremos pelo treino:

from sklearn.naive\_bayes import MultinomialNB

modelo = MultinomialNB()

modelo.fit(X, Y)

Ao invés de utilizar os nossos dados reais para treinar (X e Y), utilizaremos os nossos dados de treino, ou seja, treino\_dados e treino\_marcacoes:

from sklearn.naive\_bayes import MultinomialNB

modelo = MultinomialNB()

modelo.fit(treino\_dados, treino\_marcacoes)

Depois de treinar, o que pedíamos para o nosso algoritmo fazer? Prever para nós, quais são os novos dados e retornar o resultado! Vejamos como está atualmente o nosso código:

resultado = modelo.predict(X)

Mas não podemos mais pedir para ele prever com dados que ele já conheça, ou seja, ao invés dos dados reais (X) pediremos para ele classificar os nossos dados de teste (teste\_dados):

resultado = modelo.predict(teste\_dados)

O nosso algoritmo fez o chute para os nossos dados de teste, agora precisamos verificar a diferença entre o resultado e as nossas marcações de teste, vejamos como está no nosso código:

diferencas = resultado - Y

Observe que estamos fazendo a diferença com as marcações reais (Y) que refere-se aos dados reais! Precisamos então, mudar para as nossas marcações que representam os nossos dados de teste (teste\_marcacoes):

diferencas = resultado - teste\_marcacoes

Corrigimos o treino e prevenção do algoritmo, agora só precisamos verificar como está sendo feito o calculo da taxa de acerto:

acertos = [d for d in diferencas if d == 0]

total\_de\_acertos = len(acertos)

total\_de\_elementos = len(X)

taxa\_de\_acerto = 100.0 \* total\_de\_acertos / total\_de\_elementos

print(taxa\_de\_acerto)

print(total\_de\_elementos)

Os acertos ainda serão todos os valores do array diferencas que forem iguais a 0, a taxa\_de\_acertos será também o tamanho do array acertos, porém, o total\_de\_elementos é o tamanho do nosso X? Não! Nós estamos fazendo o teste com os nossos dados de teste, então precisamos pegar o tamanho do nosso teste\_dados para calcular o total de elementos:

acertos = [d for d in diferencas if d == 0]

total\_de\_acertos = len(acertos)

total\_de\_elementos = len(teste\_dados)

taxa\_de\_acerto = 100.0 \* total\_de\_acertos / total\_de\_elementos

print(taxa\_de\_acerto)

print(total\_de\_elementos)

Vamos testar o nosso algoritmo? Vejamos o resultado:

> python classifica\_acessos.py

88.8888888889

9

Veja que o nosso algoritmo treinou com 90 elementos e testou com 9 e o resultado foi de 89%, ou seja, ele acertou 89% das vezes! Agora sim o nosso teste está sendo mais realista.

Por que dessa vez foi diferente da outra que fizemos? Se levarmos em consideração os porcos e cachorros o nosso cenário anterior se resume em:

* Eu te apresento 90 animais entre porcos e cachorros e eu te ensino quais são os porcos e quais são os cachorros.
* Então eu te mostro os mesmos 90 animais entre porcos e cachorros que eu havia te apresentado e peço para que você os classifique mim quais são os porcos e quais são os cachorros.

Com certeza você iria acertar todos ou praticamente quase todos! Mas agora o cenário ficou diferente, estamos agindo da seguinte maneira:

* Eu te apresento 90 animais entre porcos e cachorros e eu te ensino quais são os porcos e quais são os cachorros.
* Então eu te mostro 9 novos animais que também são porcos e cachorros, mas você nunca os viu e então eu peço para que você classifique para mim quais são os porcos e quais são os cachorros

Se você tiver que classificar um novo elemento que você nunca viu, é bem provável que você terá uma taxa de acerto menor! É exatamente por isso que o resultado foi mais baixo do que o anterior, pois agora, o nosso algoritmo está lidando com elementos que ele nunca viu na vida!

Treino

Crie o arquivo classifica\_acesso.py e importe o arquivo dados.py utilizando o comando from dados import carregar\_acessos.

Chame a função carregar\_acessos() e atribua para a variável X e Y.

Agora precisamos dos dados de treino (treino\_dados e treino\_marcacoes) e de teste (teste\_dados e teste\_marcacoes), para os dados de treino, utilizaremos as primeiras 90 linhas de X e Y e para teste as 9 últimas, ou seja, 90% dos nossos elementos para treino e 10% para teste.

Com os dados de treino em mãos, peça para o modelo treinar com os nossos dados de treino utilizando a função fit() passando os arrays treino\_dados e treino\_marcacoes.

Por fim, peça para ele prever (predict) o array teste\_dados e atribuir para a variável resultado e imprima a variável resultado.