Obtendo a taxa de acerto

Lembra que calculávamos a nossa taxa de acerto?

# restante do código

resultado = modelo.predict(teste)

diferencas = resultado - marcacoes\_teste

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

total\_de\_acertos = len(acertos)

total\_de\_elementos = len(teste)

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

Perceba que nós temos os dados que representa a variável teste, ou seja, o nosso X, então vamos pedir para o nosso algoritmo representar o resultado prevendo(predict) os nossos dados(X):

from dados import carregar\_acessos

X, Y = carregar\_acessos()

from sklearn.naive\_bayes import MultinomialNB

modelo = MultinomialNB()

modelo.fit(X, Y)

resultado = modelo.predict(X)

Na variável resultado teremos vários 0 e 1 que irá prever se cada um desses usuários vai ou não comprar. Se imprimirmos essa variável:

print(resultado)

E rodarmos o nosso algoritmo:

> python classifica\_acessos.py

> [0 0 0 0 0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 1 1 1 1

0 0 1 0 0 0 1 0 0 1 1 0 0 0 0 1 0 1 0 1 0 1 0 1 0 0 0 1 0 0 1 1 0 0 0 0 0

0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0]

Além dos dados, temos também as marcações(Y), ou seja, por meio dessas marcações que sabemos se os nossos elementos, nesse caso os usuários, que acessaram as páginas do nosso web site, compraram ou não! Se imprirmos também o nosso Y:

print(resultado)

print(Y)

O resultado é:

> python classifica\_acessos.py

> [0 0 0 0 0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 1 1 1 1

0 0 1 0 0 0 1 0 0 1 1 0 0 0 0 1 0 1 0 1 0 1 0 1 0 0 0 1 0 0 1 1 0 0 0 0 0

0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0]

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

Mas e agora? O que faremos com esses 0 e 1? Bom, podemos fazer a mesma comparação de antes, ou seja, se eles forem iguais, significa que acertamos, porém, se eles forem diferentes, significa que erramos. Podemos fazer isso da mesma forma que fizemos anteriormente, ou seja, a conta de diferença! Então o valor da variável diferencas será o nosso resultadomenos o Y:

diferencas = resultado - Y

Agora que calculamos a nossa diferença, precisamos dizer que, para cada valor da diferencas que for igual a 0, iremos contabilizar e atribuir para a variável acertos:

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

O total de acerto era justamente o tamanho da nossa variável acertos, pois ela representa a quantidade de acertos que tivemos:

total\_de\_acertos = len(acertos)

Além do total de acertos, nós também fizemos o total de elementos que é a quantidade de dados que nós temos, ou seja, o tamanho do nosso X:

total\_de\_elementos = len(X)

Por fim, precisamos calcular a nossa taxa de acerto, que é a divisão do total de acertos pela quantidade de elementos multiplicado por 100.0 para apresentar o ponto flutuante, ou seja, os números após a vírgula para uma melhor precisão do percentual de acerto:

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

E agora? Precisamos imprimir a nossa taxa de acerto e quantos elementos foram testados!

print(taxa\_de\_acerto)

print(total\_de\_elementos)

Vejamos como ficou o nosso código:

from dados import carregar\_acessos

X, Y = carregar\_acessos()

from sklearn.naive\_bayes import MultinomialNB

modelo = MultinomialNB()

modelo.fit(X, Y)

resultado = modelo.predict(X)

diferencas = resultado - Y

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)

Vamos testar o nosso código? Vejamos o resultado:

> python classifica\_acessos.py

> 93.9393939394

> 99

Acertou 93,93%? Já podemos ficar felizes e contente e declarar vitória! Porém, lembra que eu havia falado que é muito difícil chegarmos em taxas de acerto absurdamente altas? Como por exemplo, 100%!

Se isso é verdade, então porque o nosso algoritmo está acertando tanto? Se verificarmos quem utilizamos para treinar o nosso algoritmo:

modelo.fit(X, Y)

Foi o nosso X, ou seja, todos os nossos dados. E para testar o nosso algoritmo?

resultado = modelo.predict(X)

Utilizamos o X denovo! Será que estamos testando da maneira correta? Vamos analisar uma situação similar:

Suponhamos que eu dê para você 10 porquinhos e 10 cachorrinhos e lhe digo quais são os porcos e quais são os cachorros.

Você aprendeu quais são os porcos e quais são os cachorros. Em seguida, eu te dou os **mesmos 10 porquinhos** e os **mesmos 10 cachorros** e pergunto para você: "Quais são os porquinhos e quais são os cachorros?"

Quanto você acha que vai acertar? Eu espero que você acerte muito! Pois são os mesmo porquinhos e os mesmo cachorros que eu te ensinei em poucos instantes atrás.

Um outro exemplo similar seria, eu te apresentar a minha mão direita e, logo em seguida, te apresento a minha mão esquerda e então eu mostro a minha mão direita e pergunto: "Que mão é essa?", e depois mostro a minha mão esquerda e pergunto: "Que mão é essa?", com certeza você vai acertar, pois eu acabei de te falar qual é cada uma das mãos que estou te perguntado...

Isso significa que, se você estiver utilizando o mesmo elemento que você treinou o algoritmo para testá-lo, é bem provável que ele vai acertar! E não é pra isso que criamos esse algoritmo!

No mundo real, quando treinamos um algoritmo de classificação nós o treinamos com elementos que conhecemos para que ele tenha um histórico do que já sabemos sobre aquele determinado elemento, porém, quando testamos esse tipo de algoritmo, utilizamos apenas elementos desconhecidos, pois são esses elementos desconhecidos que nós queremos que ele classifique para nós!

Não faz sentido algum nós treinarmos o nosso algoritmo com os 99 registros que conhecemos, ou seja, que já classificamos, e pedirmos para ele testar com os mesmos 99 registros. O nosso teste precisa ser feito com elementos que o nosso algoritmo nunca viu! Mas se nós temos apenas 99 registros anotados, como podemos fazer com que o nosso algoritmo seja treinado e testados por todos esses 99 registros sem que aconteça o mesmo caso problemático que vimos anteriormente?