Sabendo a taxa de acerto.

Vimos lá no começo da aula que nós tínhamos 3 porcos e 3 cachorros, cada um com suas variáveis, ou seja, suas características que definiam se eles eram gordinhos (1) ou não (0), se tinham perninhas curtas (1) ou não (0) e, por fim, verificamos se eles faziam *auau* (1) ou não (0) e representamos como nossos elementos:

porco1 = [1, 1, 0]

porco2 = [1, 1, 0]

porco3 = [1, 1, 0]

cachorro4 = [1, 1, 1]

cachorro5 = [0, 1, 1]

cachorro6 = [0, 1, 1]

E nós tínhamos agrupados todos esses elementos em um *array* para representar os nossos dados:

dados = [porco1, porco2, porco3, cachorro4, cachorro5, cachorro6]

E também fizemos as marcações para indicar quais desses elementos eram porcos ou cachorros, marcando com 1 para porco e -1 para cachorro:

marcacoes = [1, 1, 1, -1, -1, -1]

Então treinamos o nosso algoritmo:

modelo = MultinomialNB()

modelo.fit(dados, marcacoes)

E por fim nós testamos 2 elementos misteriosos:

misterioso1 = [1, 1, 1]

misterioso2 = [1, 0, 0]

teste = [misterioso1, misterioso2]

print(modelo.predict(teste))

Mas existe um detalhe no processo que fizemos, pois não adianta nós rodarmos o nosso algoritmo e não fazer ideia de quão bom ele é. Quando treinamos o nosso cérebro, chegamos a ver 300, ou mais, porcos e cachorros na nossa vida, porém nós não temos certeza se estamos bons o suficiente para distinguir entre um cachorro e um porco ou então *spam* e não \*spam ou pessoas que vão me pagar e que não vão me pagar...

E como podemos ter certeza se estamos bons ou não para fazer essas distinções? Precisamos testar! Sim, testar no mundo real, com algum valor que desconhecemos, se iremos funcionar conforme o esperado para verificar o quão bons estamos sendo com esse algoritmo. E como podemos testar? Podemos criar um cenário de teste utilizando os simulando os nossos elementos reais (porcos ou cachorros) por elementos misteriosos:

# [é gordinho?, tem perninha curta?, faz auau?]

misterioso1 = [1, 1, 1]

misterioso2 = [1, 0, 0]

misterioso3 = [0, 0, 1]

teste = [misterioso1, misterioso2, misterioso3]

Agora que temos o nosso cenário de teste, podemos informar os resultados esperados para cada um desses elementos misteriosos, ou seja, as marcações de teste:

marcacoes\_teste

Agora, precisamos classificar cada elemento misterioso e indicar nas nossas marcações de teste. Sabemos que o misterioso1 é um cachorro(-1), o misterioso2 é um porco e o misterioso3 é um cachorro(-1):

marcacoes\_teste = [-1, 1, -1]

Por fim, para ficar mais claro, vamos atribuir o retorno do método predict(), que são os nossos resultados, para uma variável chamada resultado:

resultado = modelo.predict(teste)

print(resultado)

Agora repare o nosso código:

porco1 = [1, 1, 0]

porco2 = [1, 1, 0]

porco3 = [1, 1, 0]

cachorro4 = [1, 1, 1]

cachorro5 = [0, 1, 1]

cachorro6 = [0, 1, 1]

dados = [porco1, porco2, porco3, cachorro4, cachorro5, cachorro6]

marcacoes = [1, 1, 1, -1, -1, -1]

from sklearn.naive\_bayes import MultinomialNB

modelo = MultinomialNB()

modelo.fit(dados, marcacoes)

misterioso1 = [1, 1, 1]

misterioso2 = [1, 0, 0]

misterioso3 = [0, 0, 1]

teste = [misterioso1, misterioso2, misterioso3]

marcacoes\_teste = [-1, 1, -1]

resultado = modelo.predict(teste)

print(resultado)

Será que agora ele vai prever conforme as nossas marcações de teste? Ou seja, falar que os elementos misteriosos são: cachorro, porco e cachorro? Vamos testar:

> python classificacao.py

> [-1 1 -1]

Será que o nosso código se saiu bem? Vamos verificar os valores que estávamos esperando nas nossas marcações de teste:

marcacoes\_teste = [-1, 1, -1]

De acordo com as nossas marcações, o nosso código funcionou muito bem, ele conseguiu prever! Mas é importante lembrar que no mundo real, ou seja, na prática, esse resultado não costuma ser 100% e o próximo exemplo que veremos logo mais, de compras na web, não iremos conseguir acertar sempre. Porém, observe que estamos testando isso manualmente, ou seja, estamos verificando se o teste passou a olho nu, para o nosso caso atual, não tem problema, mas no mundo real, iremos realizar testes gigantes, como por exemplo, 1000 elementos! E não faz sentido nós ficarmos olhando 1 a 1 para verificar o quanto ele acertou ou o quanto ele errou... O que precisamos saber é: quantos elementos valores da variável resultado(resultados que o algoritmo classificou) são diferentes da variável marcacoes\_teste(resultado que esperamos dos elementos). Mas como podemos fazer para verificar os valores que foram diferentes? Podemos subtrair os 2 arrays:

print(resultado - marcacoes\_teste)

Parece estranho mas essa abordagem funciona da seguinte maneira:

* Resultado e marcação: 1, resultará em: 1 - 1 = 0.
* Resultado: 1 e marcação: -1, resultará em: 1 - (-1) -> 1 + 1 = 2.
* Resultado: -1 e marcação: -1, resultará em: -1 + 1 = 0.
* Resultado: -1 e marcação: 1, resultará em: -1 - 1 = -2

Resumindo todas as possibilidades que fizemos, nos casos em que os valores forem iguais o resultado será 0, ou seja, quando resultar em 0, saberemos que o algoritmo acertou. Vamos testar o nosso código?

> python classificacao.py

> [0 0 0]

O resultado foi tudo 0, então acertou 100% novamente! Mas o ideial seria que o nosso algoritmo mostrasse o percentual ao invés de só a diferença entre o resultado e a marcacoes\_teste. Então vamos fazer com que a diferença dessas duas variáveis sejam retornadas para uma variável chamada diferencas:

diferencas = resultado - marcacoes\_teste

Mas como iremos fazer para verificar os acertos? Sabemos que qualquer valor igual a 0 é um acerto, ou seja, qualquer valor da variável diferencas que seja 0. Isso significa que, para cada diferença dentro da variável diferencas:

for d in diferencas

Se o valor for igual a 0:

for d in diferencas if d == 0

retorne:

acertos = for d in diferencas if d == 0

Porém, nós precisamos passar isso para um array, então fazemos:

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

Se imprimirmos os acertos:

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

print(acertos)

Resultado:

> python classificacao.py

> [0, 0, 0]

Observe que foi impresso foi um array com todos os 0 contidos no array diferencas, ou seja, todos os valores, porém, não queremos os valores, precisamos saber a quantidade de elementos que existem nesse array, e como fazemos para retornar a quantidade de elementos existentes de um array? Utilizamos a função len() que retorna o tamanho do array:

len(acertos)

E esse será o nosso total de acertos, então vamos representar por uma variável chamada total\_de\_acertos:

total\_de\_acertos = len(acertos)

Agora vamos imprimir o total\_de\_acertos:

total\_de\_acertos = len(acertos)

print(total\_de\_acertos)

Se rodarmos novamente:

> python classificacao.py

> [0, 0, 0]

> 3

Porém, para verificarmos a porcentagem, nós precisamos também, do total de elementos que foram testados, podemos extrair esse valor a partir do nosso array teste que contém todos os elementos que foram testados:

teste = [misterioso1, misterioso2, misterioso3]

marcacoes\_teste = [-1, 1, -1]

resultado = modelo.predict(teste)

diferencas = resultado - marcacoes\_teste

print(diferencas)

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

total\_de\_acertos = len(acertos)

print(total\_de\_acertos)

total\_de\_elementos = len(teste)

print(total\_de\_elementos)

Testando o nosso código:

> [0 0 0]

> 3

> 3

Agora que temos tanto o total de acertos quanto o total de elementos, precisamos apenas fazer a divisão de total\_de\_acertos por total\_de\_elementos para extraírmos a nossa taxa de acerto:

total\_de\_acerto / total\_de\_elementos

Então atribuímos para uma variável chamada taxa\_de\_acerto e a imprimimos:

taxa\_de\_acerto = total\_de\_acerto / total\_de\_elementos

print(taxa\_de\_acerto)

Verificando o resultado:

> python classificacao.py

> [0 0 0]

> 3

> 3

> 1

Vamos melhorar a nossa impressão, vamos retirar todas as impressões. Então imprimimos primeiro o nosso resultado: print(resultado), depois a diferença entre o resultado e as marcações de teste: print(diferenca), por fim, faremos a impressão da taxa de acerto: print(taxa\_de\_acerto), porém vamos multiplicar por 100.0: taxa\_de\_acerto = 100.0 \* total\_de\_acertos / total\_de\_elementos, para que seja apresentado como percentual, vejamos o resultado do nosso código final:

porco1 = [1, 1, 0]

porco2 = [1, 1, 0]

porco3 = [1, 1, 0]

cachorro4 = [1, 1, 1]

cachorro5 = [0, 1, 1]

cachorro6 = [0, 1, 1]

dados = [porco1, porco2, porco3, cachorro4, cachorro5, cachorro6]

marcacoes = [1, 1, 1, -1, -1, -1]

from sklearn.naive\_bayes import MultinomialNB

modelo = MultinomialNB()

modelo.fit(dados, marcacoes)

misterioso1 = [1, 1, 1]

misterioso2 = [1, 0, 0]

misterioso3 = [0, 0, 1]

teste = [misterioso1, misterioso2, misterioso3]

marcacoes\_teste = [-1, 1, -1]

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

print(resultado)

print(diferencas)

print(taxa\_de\_acerto)

Rodando o nosso algoritmo, obtemos o seguinte resultado:

> python classificacao.py

> [-1 1 -1]

> [0 0 0]

> 100.0

100.0! Porém, é sempre válido lembrar que 100.0 é um número muito difícil de acontecer no mundo real. Um exemplo bem simples que demonstra o nosso algoritmo errando seria modificar a nossa marcacoes\_teste informando que o último elemento misterioso era um porco que fazia *auau*, ou seja, um porquinho que faz *auau*:

misterioso1 = [1, 1, 1]

misterioso2 = [1, 0, 0]

misterioso3 = [0, 0, 1]

teste = [misterioso1, misterioso2, misterioso3]

marcacoes\_teste = [-1, 1, 1]

Vamos verificar se o nosso algoritmo vai adivinhar que esse porquinho que faz *auau* é um porquinho? Vejamos o resultado:

> python classificacao.py

> [-1 1 -1]

> [ 0 0 -2]

> 66.6666666667

Como vimos, ele errou! perceba que o úlitmo valor da diferenca foi diferente de 0, ou seja, um erro! Por fim, a nossa taxa de acerto foi de 66%.

Repara que a taxa de acerto é fundamental para que o nosso algoritmo, no momento em que recebe novos dados, nesse caso, animais novos com características novas, seja possível verificar o quão bom ele foi em uma situação do mundo real, como por exemplo, nesse último teste, ele foi capaz de acertar 66%.

Daqui a pouco a nossa situação será diferente, iremos utilizar um exemplo real em uma aplicação web onde iremos pegar muito mais valores para realizarmos o nosso teste com diversas características e um melhor cálculo para a taxa de acerto. E nesse mesmo exemplo, veremos como a nossa taxa de acerto dificilmente acertará 100%.

Quando estamos classificando, é bem comum mostrarmos um percentual que indica a taxa de acerto que o nosso algoritmo obteve para uma melhor analise.

Crie uma marcação de teste para os elementos misteriosos utilizando o array marcacoes\_teste indicando o que é cada elemento misterioso.

Calcule a diferença das variáveis resultado e as marcacoes\_teste e atribua a uma variável chamada diferencas.

Percorra o array diferencas e devolva todos os seus elementos que forem iguais a 0 e atribua a uma variável chamada acertos.

Calcule o total de acertos pegando o tamanho do array acertos com o método len() e atribua à variável total\_de\_acertos.

Calcule o total de elementos pegando o tamanho do array teste com o método len() e atribua à variável total\_de\_elementos.

Calcule a taxa de acerto multiplicando 100.0 pela divisão das variáveis total\_de\_acertos e total\_de\_elementos e atribua à variável taxa\_de\_acerto.

Por fim, imprima as variáveis resultado, diferencas e taxa\_de\_acerto.