07

Importando, classificando e validando um modelo

Importando, classificando e validando um modelo

Baixe o arquivo e salve como `acesso.csv`. Nesse arquivo estarão todos os dados referente ao acesso de páginas do usuário no nosso site, ou seja, se ele acessou a página home, como funciona e contato, por fim, informamos se ele comprou ou não. Nesse arquivo temos diversos dados do nosso usuário no passado, ou seja, suas características (acessos às páginas) e a marcação (se comprou), no total são 100 linhas.

Agora que temos o arquivo em mãos, precisamos fazer a leitura dele. Para ler esse arquivo, criaremos um novo arquivo python chamado `dados.py` que será o responsável por fazer a leitura desse arquivo e colocar os dados em um array que tenha as informações dos nossos elementos, porém nós precisamos apenas de 1 array ou 2 arrays? Lembra que utilizamos um array para os dados:

```
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]
```

E também um outro array para as marcações:

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

Isso significa que precisaremos de 2 arrays também! Então o nosso arquivo será divido da seguinte forma:

```
acessou_home,acessou_como_funciona, acessou_contato,comprou
1,1,0,0
```

Os valores referentes a `acessou_home, acessou_como_funciona, acessou_contato` (1,1,0) serão o nosso primeiro array que representará os nossos dados. Porém, o valor referente ao `comprou` (0), será o nosso segundo array que representará as nossas marcações.

Observe que os valores dos nossos dados estão em função das marcações, então é comum chamarmos esse tipo de array de X, pois são os dados misteriosos que nós temos, e o que queremos calcular, nesse caso a marcação, chamamos de Y. É comum usar esses nomes para classificar dados, por isso utilizamos o X para indicar os nossos dados e o Y para dados que iremos prever.

Importando os dados

Agora que sabemos a forma que iremos representar os dados do nosso arquivo csv, precisamos importar esse arquivo. Dentro do arquivo `dados.py`, escreveremos o código para importar o csv:

```
import csv
```

Agora precisamos carregar as informações desse csv, então vamos definir a função `carregar_acessos()`:

```
import csv

def carregar_acessos():
```

Mas o que essa função precisa fazer? Ela vai abrir o arquivo csv para leitura e ler todas as linhas. A cada linha que for lida, ela irá associar os valores para o array de dados e os valores para o array de marcações:

```
import csv

def carregar_acessos():

    dados = []
    marcacoes = []
```

Agora que definimos os nossos arrays, precisamos abrir o arquivo utilizando a função `open()` e enviando o nome do arquivo como parâmetro:

```
import csv

def carregar_acessos():

    dados = []
    marcacoes = []

    arquivo = open('acesso.csv', 'rb')
```

Observe que utilizamos também o parâmetro `rb` indicando que queremos ler o arquivo. Agora precisamo ler esse arquivo csv, para isso iremos utilizar a função `reader()` da biblioteca que importamos:

```
import csv

def carregar_acessos():

    dados = []
    marcacoes = []

    arquivo = open('acesso.csv', 'rb')
    csv.reader(arquivo)
```

A função reader devolve um leitor:

```
import csv

def carregar_acessos():
    dados = []
```

```
    marcacoes = []

    arquivo = open('acesso.csv', 'rb')
    leitor = csv.reader(arquivo)
```

Esse leitor irá passar por cada uma das linhas do arquivo csv e adicionar os valores dos dados e a marcação:

```
import csv

def carregar_acessos():

    dados = []
    marcacoes = []

    arquivo = open('acesso.csv', 'rb')
    leitor = csv.reader(arquivo)
    for linha in leitor:
```

Porém, perceba que precisamos pegar os valores das colunas e não uma linha inteira! Então podemos copiar o nome de cada coluna do nosso arquivo e adicionar no `for` :

```
for acessou_home, acessou_como_funciona,
acessou_contato, comprou in leitor:
```

Nesse instante, estamos dizendo que, para cada uma dessas colunas (`for acessou_home, acessou_como_funciona, acessou_contato, comprou`) dentro do leitor (`in leitor`). Mas e agora? O que precisamos fazer? Precisamos adicionar ao nosso array `dados` as colunas: `acessou_home, acessou_como_funciona, acessou_contato` utilizando a função `append()` :

```
import csv

def carregar_acessos():
    dados = []
    marcacoes = []

    arquivo = open('acesso.csv', 'rb')
    leitor = csv.reader(arquivo)
    for acessou_home, acessou_como_funciona,
    acessou_contato, comprou in leitor:

        dados.append([acessou_home,
            acessou_como_funciona,
            acessou_contato])
```

Por fim, adicionamos as marcações:

```
import csv

def carregar_acessos():
    dados = []
    marcacoes = []
```

```
    arquivo = open('acesso.csv', 'rb')
    leitor = csv.reader(arquivo)
    for acessou_home, acessou_como_funciona,
    acessou_contato, comprou in leitor:

        dados.append([acessou_home,
            acessou_como_funciona,
            acessou_contato])
        marcacoes.append(comprou)
```

Agora que adicionamos os dados e as marcações podemos retorná-los:

```
import csv

def carregar_acessos():
    dados = []
    marcacoes = []

    arquivo = open('acesso.csv', 'rb')
    leitor = csv.reader(arquivo)
    for acessou_home, acessou_como_funciona,
    acessou_contato, comprou in leitor:

        dados.append([acessou_home,
            acessou_como_funciona,
            acessou_contato])
        marcacoes.append(comprou)

    return dados, marcacoes
```

Vamos testar o nosso código? Vá no terminal e abra o interpretador do python:

```
> python
Python 2.7.10 (default, Oct 14 2015, 16:09:02)
[GCC 5.2.1 20151010] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Primeiro precisamos importa a função `carregar_acessos` do arquivo `dados.py` :

```
>>> from dados import carregar_acessos
>>>
```

Agora podemos utilizar a nossa função! Vamos retorna o valor dessa função:

```
>>> dados, marcacoes = carregar_acessos()
>>>
```

Será que funcionou? Será que não? Vamos dar uma olhada no valor das nossas marcações:

```
>>> marcacoes
```

```
>>> marcacoes
['comprou', '0', '0', '0', '0', '0', '1', '0', '1', '0', '1', '1', '1', '0', '0', '0', '0', '0', '0'
>>>
```

Ele retornou um array com a primeira linha informando ao que se refere, nesse caso, se comprou ou não. Vamos observar as nossas primeiras marcações no arquivo `acesso.csv`:

```
1,1,0,0
1,1,0,0
1,1,0,0
1,1,0,0
1,1,0,0
1,0,1,1
...
```

Ao compararmos os primeiros resultados funcionou conforme o esperado! Agora vamos dar uma olhada nos nossos dados:

```
>>> dados
[['acessou_home', 'acessou_como_funciona', 'acessou_contato'], ['1', '1', '0'], ['1', '1', '0'], ['1
>>>
```

Os dados batem, conforme o esperado! Observe que agora nós separamos os nossos dados das nossas marcações, ou seja, separamos todos os dados que estão em função das marcações! Lembra que para esses casos de classificação é tradicional chamarmos os nossos dados de X, ou seja, os valores que conhecemos do nosso usuário como por exemplo, se ele acessou uma página, se ele tem perninha curta ou qualquer informação que nós já sabemos!

```
def carregar_acessos():
    X = []
    marcacoes = []
    # restante do código
```

Mas e as nossas marcações? Como é mesmo que chamávamos? Lembre-se que as marcações são os valores que não sabemos, ou seja, os valores que queremos calcular, prever! Tradicionalmente, esses valores, essas marcações que queremos prever, chamamos de Y:

```
def carregar_acessos():
    X = []
    Y = []
    # restante do código
```

Vamos substituir todos os `dados` para `X` e `marcacoes` para `Y`:

```
import csv

def carregar_acessos():
    X = []
    Y = []
```

```
    arquivo = open('acesso.csv', 'rb')
    leitor = csv.reader(arquivo)
    for acessou_home,acessou_como_funciona,acessou_contato, comprou in leitor:

        X.append([acessou_home,acessou_como_funciona,acessou_contato])
        Y.append(comprou)

    return X, Y
```

Analisando os valores adicionados

Vamos dar uma olhada novamente no resultado das nossas marcações:

```
>>> marcacoes
['comprou', '0', '0', '0', '0', '0', '1', '0', '1', '0', '1', '1', '1', '0', '0', '0', '0', '0', '0
>>>
```

Repare que veio também o array com a informação do cabeçalho, ou seja, a palavra `'comprou'`, porém nós não queremos isso no nosso array, lembra como era as nossas marcações no primeiro exemplo?

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

Como podemos ver, só tem números! Em nenhum momento especificamos o que significa por meio de uma palavra na primeira posição:

```
marcacoes = ['é_um_porco', 1, 1, 1, -1, -1, -1]
```

Então precisamos descartar esses cabeçalhos simplesmente não lendo a primeira linha! Como podemos fazer isso? Simples, basta apenas, após o momento em que criamos o `leitor` (`leitor = csv.reader(arquivo)`), adicionarmos o comando `next(leitor)` que pulará para a linha a seguir:

```
import csv

def carregar_acessos():
    X = []
    Y = []

    arquivo = open('acesso.csv', 'rb')
    leitor = csv.reader(arquivo)

    next(leitor)

    for acessou_home,acessou_como_funciona,acessou_contato, comprou in leitor:

        X.append([acessou_home,acessou_como_funciona,acessou_contato])
        Y.append(comprou)

    return X, Y
```

Vamos testar o nosso código novamente? Antes de importar a nossa função `carregar_acessos` , feche o interpretador do python e abra novamente para que ele carregue a nova versão da nossa função. Vejamos o resultado:

```
>>> from dados import carregar_acessos
>>>
```

Importou sem nenhum problema! Porém, ao invés de chamar de `dados` e `marcacoes` chamaremos de `X` e `Y` :

```
>>> from dados import carregar_acessos
>>> X, Y = carregar_acessos()
>>>
```

Vamos verificar o valor de `Y` :

```
>>> from dados import carregar_acessos
>>> X, Y = carregar_acessos()
>>> Y
['0', '0', '0', '0', '0', '1', '0', '1', '0', '1', '1', '1', '0', '0', '0', '0', '0', '0', '0', '0',
```

Aparentemente tudo está funcionando como o esperado! Porém, `'0'` ? `'1'` ? Isso é uma *string*! Não queremos *string*s, nós queremos números! Afinal estamos lendo números... E como resolver isso? Podemos converter as *string*s para números, mas que tipo de números? Nesse caso estamos trabalhando com números inteiros, então converteremos para números inteiros! Para converter uma string para inteiro, usaremos a maneira mais tradicional que é enviar, por parâmetro, a coluna desejada na instrução `int()` :

```
import csv

def carregar_acessos():
    X = []
    Y = []

    arquivo = open('acesso.csv', 'rb')
    leitor = csv.reader(arquivo)

    next(leitor)

    for acessou_home,acessou_como_funciona,acessou_contato, comprou in leitor:

        X.append([int(acessou_home),int(acessou_como_funciona)
            ,int(acessou_contato)])
        Y.append(int(comprou))

    return X, Y
```

Testando novamente o nosso código:

```
>>> from dados import carregar_acessos
>>> X, Y = carregar_acessos()
>>>
```

Vejamos agora o valor do `Y`:

```
>>> from dados import carregar_acessos
>>> X, Y = carregar_acessos()
>>> Y
[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,
```

E o `X`?

```
>>> X
[[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],
```

Agora sim nós carregamos os nossos números!

Melhorando a leitura do código

Observe que o nosso código ficou um pouco extenso e repetitivo:

```
import csv

def carregar_acessos():
    X = []
    Y = []

    arquivo = open('acesso.csv', 'rb')
    leitor = csv.reader(arquivo)

    next(leitor)

    for acessou_home,acessou_como_funciona,acessou_contato, comprou in leitor:

        X.append([int(acessou_home),int(acessou_como_funciona)
            ,int(acessou_contato)])
        Y.append(int(comprou))

    return X, Y
```

Veja que utilizamos os nomes `acessou_home`, `acessou_contato`, `acessou_alguma_coisa`! Vamos retirar esse `acessou`, pois já sabemos que nossos dados refere-se a acesso de página do usuário, então não precisamos repetir a mesma palavra em todos os nossos dados! Veja o resultado do nosso código:

```
import csv

def carregar_acessos():
    X = []
    Y = []

    arquivo = open('acesso.csv', 'rb')
```

```
    leitor = csv.reader(arquivo)

    next(leitor)

    for home, como_funciona, contato, comprou in leitor:

        X.append([int(home),int(como_funciona)
            ,int(contato)])
        Y.append(int(comprou))

    return X, Y
```

Também alteraremos no nosso arquivo csv:

```
home, como_funciona, contato, comprou
1,1,0,0
1,1,0,0
1,1,0,0
1,1,0,0
1,1,0,0
1,0,1,1
...
```

Melhoramos uma boa parte do nosso código e deixamos mais limpo, ou seja, com uma leitura mais clara, mas perceba que ainda existe alguns detalhes, vejamos esse trecho de código:

```
X.append([int(home),int(como_funciona)
            ,int(contato)])
```

Observe que enviamos um monte de informação para o `X` que, a primeira vista, não da pra saber a que se refere, nesses casos, podemos extrair todas essas informações para uma variável que deixará mais claro o significado desses valores:

```
dado = [int(home),int(como_funciona)
            ,int(contato)]
```

Observe agora o resultado final do nosso código:

```
import csv

def carregar_acessos():
    X = []
    Y = []

    arquivo = open('acesso.csv', 'rb')
    leitor = csv.reader(arquivo)

    next(leitor)

    for home,como_funciona,contato, comprou in leitor:

        dado = [int(home),int(como_funciona)
```

```
            ,int(contato)]
        X.append(dado)
        Y.append(int(comprou))

    return X, Y
```

Bem mais limpo e de fácil compreensão! Vamos verificar se o nosso código ainda funciona? Porém, dessa vez, ao invés de usar o interpretador python, vamos criar um novo arquivo chamado `classifica_acessos.py` e importar a função `carregar_acessos()` dentro desse arquivo:

```
from dados import carregar_acessos
X, Y = carregar_acessos()
```

Por fim, vamos imprimir o `X` e o `Y` para verificar se está funcionando corretamente:

```
from dados import carregar_acessos
X, Y = carregar_acessos()
print(X)
print(Y)
```

Vamos testar o nosso código:

```
> python classifica_acessos.py
[[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],
[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,
```

Está funcionando perfeitamente! Agora podemos tirar as impressões do `X` e `Y`:

```
from dados import carregar_acessos
X, Y = carregar_acessos()
```

Temos o `X` (dados) e o `Y` (marcações), mas e agora? O que faremos com eles? Lembra que no exemplo anterior, nós tinhamos os nossos `dados` e `marcacoes`:

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

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

O que fazíamos mesmo? Rodávamos o modelo! Então agora vamos importar o algoritmo `MultinomialNB`:

```
from dados import carregar_acessos
X, Y = carregar_acessos()

from sklearn.naive_bayes import MultinomialNB
```

E o que fazíamos com esse algoritmo? Pedíamos para ele criar um modelo e adaptá-lo (`fit`) com os nossos dados(`X`), e marcações(`Y`):

```
from dados import carregar_acessos
X, Y = carregar_acessos()

from sklearn.naive_bayes import MultinomialNB
modelo = MultinomialNB()
modelo.fit(X, Y)
```

Agora precisamos testar o nosso modelo! Vamos verificar esse novo usuário:

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

Observe que ele entrou na home, não entrou na página de como funciona e entrou na página de contato. E agora? Ele comprou ou não? Vamos pedir para que o nosso modelo preveja (`predict`) para nós:

```
from dados import carregar_acessos
X, Y = carregar_acessos()

from sklearn.naive_bayes import MultinomialNB
modelo = MultinomialNB()
modelo.fit(X, Y)

modelo.predict([1,0,1])
```

Porém, lembra que o `predict` espera um array de arrays para não mostrar aquele *warning*? Então vamos adicionar um array:

```
modelo.predict([[1,0,1]])
```

Por fim, vamos imprimir o `predict` para verificar o resultado:

```
from dados import carregar_acessos
X, Y = carregar_acessos()

from sklearn.naive_bayes import MultinomialNB
modelo = MultinomialNB()
modelo.fit(X, Y)
print(modelo.predict([[1,0,1]]))
```

Testando o nosso algoritmo:

```
> python classifica_acessos.py
> [1]
```

Ele imprimiu 1 então isso significa que esse usuário vai comprar! Estamos acreditando que ele vai comprar de acordo com o nosso modelo. Mas e se tivéssemos mais um usuário que caiu direto na página de como funciona e não entrou em contato

([0,1,0])? Ele simplesmente entrou na página de como funciona e viu apenas quais são os nossos planos, nem chegou a ver quais são os nossos produtos ou serviços, e agora? Esse usuário vai comprar ou não vai comprar? Vamos adicioná-lo ao nosso modelo e pedir para ele prever para nós!

```
print(modelo.predict([[1,0,1],[0,1,0]]))
```

Verificando o resultado desse novo usuário:

```
> python classifica_acessos.py
> [1 0]
```

Observe que o nosso modelo está prevendo pra nós que apenas o primeiro usuário vai comprar e o segundo não... Será que apenas esses dois testes já é o suficiente? Com certeza precisamos verificar outros cenários para garantir que está funcionando!

Além de testar apenas 2 casos, podemos testar muito mais, vamos adicionar um terceiro que só acessou apenas a página home([1,0,0]):

```
print(modelo.predict([[1,0,1],[0,1,0],[1,0,0]]))
```

E agora? Será que ele compra? Vamos pedir para que o nosso algoritmo preveja:

```
> python classifica_acessos.py
> [1 0 0]
```

Também não compra... Vamos testar mais um caso em que o usuário entra na página home e na página de como funciona, porém não entra na página de contato([1,1,0]):

```
print(modelo.predict([[1,0,1],[0,1,0],
                      [1,0,0], [1,1,0]]))
```

Será que agora ele vai comprar? Vamos verificiar o resultado:

```
> python classifica_acessos.py
> [1 0 0 0]
```

Nenhum desses usuários comprarão... Provavelmente existe alguma característica em comum entre eles!

Mas e aquele usuário que entra na página home, entra na página de como funciona e também entra na página de contato ([1,1,1]), ele compra ou não?

```
print(modelo.predict([[1,0,1],[0,1,0],
                      [1,0,0], [1,1,0], [1,1,1]]))
```

Vejamos o resultado:

```
> python classifica_acessos.py
> [1 0 0 0 0]
```

Também não compra... Esse usuário acessou todas as páginas, mas, mesmo assim, provavelmente não comprará.

Observe que o modelo está dizendo apenas o que ele acredita! Essa prevenção está boa ou ruim? O algoritmo está chutando bem ou mal? O nível de acerto está alto ou baixo? Como podemos fazer para saber todas essas informações? Podemos testar o nosso modelo da mesma forma que fizemos quando estávamos classificando os porcos e cachorros! 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 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 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]
```

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 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, 0, 1, 1, 1, 1, 0, 0, 0, 1, 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 `resultado` menos 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 acerta? 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 perdimos 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? 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],
```

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,
>>> 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!

Resumindo

Em um processo de classificação nós temos as características que são todas as informações que utilizamos para poder distinguir um elemento, por exemplo, quais poderiam ser uma as características de um e-mail? Poderia ser o tamanho do e-mail, número de palavras repetidas, se usa letra maiúscula, se o remetente é conhecido, se já foi marcado como *SPAM* entre diversas características! E se forem animais? Outras características. E se forem alunos que irão repetir de ano? Outras características.

Chamamos essas características de X, que são todos os dados que nós temos de entrada, ou seja, todos os dados que nós conhecemos, e esses dados podem ser de e-mails, animais, funcionários, usuários, alunos ou qualquer coisa que queremos classificar, ou seja, todos eles são os nossos elementos.

A partir das características dos nossos elementos, nós treinamos e aprendemos o que é o elemento, de acordo sua característica e marcação, e então tentamos prever um novo elemento, ou seja, se ele se encaixa na classificação 0 ou 1.

Além disso, precisamos sempre lembrar que existe uma taxa de erro, pois nem sempre nós iremos acertar, ou seja, precisamos sempre verificar a forma que o algoritmo foi treinado, analisando a quantidade de dados e características, para determinar um retorno aceitável do nosso objetivo, por exemplo, se o nosso objetivo é conversar com mais pessoas que provavelmente irão pedir para serem demitidas ou com menos pessoas, ou então, se queremos conversar com os alunos que estão com uma chance maior para reprovarem, talvez seja mais interessante conversar com mais alunos do que menos alunos, porém, se fosse entre um cachorro e porco, para qual dos 2 eu gostaria de errar mais? Depende do caso! Isso significa que cada caso, precisaremos verificar o quão interessante é, ou seja, se aceitamos uma taxa de erro maior ou menor.

Temos que nos atentar a todos esses detalhes, pois é dessa forma que o processo de classificação, baseado nas características, funciona.

#

E como transformamos essa teoria em código? Primeiro precisamos representar os nossos elementos, suas características e marcações. Podemos representar todos esses dados por meio de um arquivo csv que é uma abordagem muito comum para tratamento de dados, pois podemos pegar uma planilha eletronica e converter para o formato csv. E vimos que podemos ler um arquivo csv por meio de uma função de leitura de arquivo do próprio python:

```
arquivo = open('acesso.csv', 'rb')
leitor = csv.reader(arquivo)
```

Nesse exemplo nós só trabalhamos com dados inteiros, mas, mais pra frente, veremos como trabalhar com float, string e outras coisas. Além disso, vimos que, de acordo com os dados que lemos:

```
X,Y = carregar_acessos()
```

Aprendemos a importância de treinar um modelo e testá-lo, porém existe um grande desafio nessa abordagem, pois se treinarmos o nosso modelo com diversos dados:

```
modelo.fit(X,Y)
```

Acabamos viciando o nosso modelo com apenas esses dados, ou seja, se testarmos esse modelo perguntando sobre um ou mais elementos que faça parte desses diversos dados:

```
resultado = modelo.predict(X)
```

A chance dele acertar é muito grande! Será que faz sentido esse teste? Aparentemente não... Mas e se o teste for com um elemento que o nosso modelo nunca viu na vida?

```
X,Y = carregar_acessos()

treino_dados = X[:90]
treino_marcacoes = Y[:90]

teste_dados = X[-9:]
teste_marcacoes = Y[-9:]

from sklearn.naive_bayes import MultinomialNB
modelo = MultinomialNB()
modelo.fit(treino_dados, treino_marcacoes)

resultado = modelo.predict(teste_dados)
```

Agora sim estamos verificando se o algoritmo aprendeu de verdade! E é justamente por esse motivo que não utilizamos os mesmos dados, que foram usados para treino, para testar o nosso modelo.

Perceba que utilizamos uma estratégia para separar os dados de treino com os dados de testes que foi:

-90% dos dados serão para treino:

```
treino_dados = X[:90]
treino_marcacoes = Y[:90]
```

-10% restantes dos dados serão para teste:

```
teste_dados = X[-9:]
teste_marcacoes = Y[-9:]
```

O método de treino e teste é tão importante que, na maioria das vezes, formalizamos passo-a-passo o que utilizamos para chegar a um determinado resultado:

```
# minha abordagem inicial foi
# 1. separar 90% para treino e 10% para teste: 88.89%

from dados import carregar_acessos
```

```
X,Y = carregar_acessos()

treino_dados = X[:90]
treino_marcacoes = Y[:90]

teste_dados = X[-9:]
teste_marcacoes = Y[-9:]
```

Podemos ver que nesse teste foram 90% pra teste e 10% pra leitura e o resultado foi 88,89%. Perceba que para cada teste, podemos registrá-lo com essas anotações e verificarmos como é o comportamento do nosso algoritmo para cada cenário.