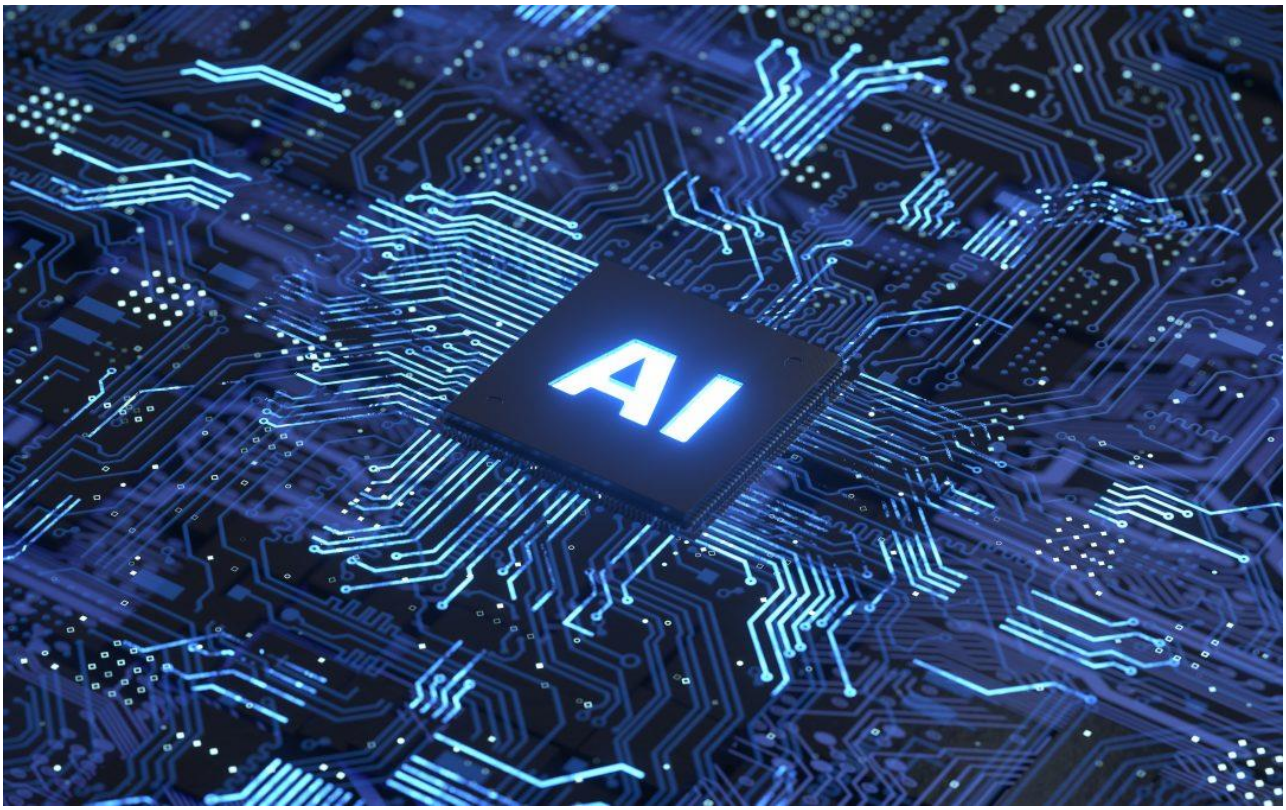
INTRODUCTION TO MACHINE LEARNING – UNSUPERVISED LEARNING

21 DE DEZEMBRO DE 2020



Nome: Miguel Oliveira

Aluno nº: 96200

Email: mdlco@iscte-iul.pt

Docente: Professor Doutor Luís Nunes

Docente: Professor Doutor Sancho Oliveira

Introdução:

Unsupervised Learning (aprendizagem não supervisionada), é um ramo do *machine learning* que procura por padrões num conjunto de dados que não foram previamente identificados e com a mínima interversão humana possível. Também é conhecido como sendo auto-organizada, pois permite a modelagem de densidades de probabilidade sobre entradas.

Um dos principais métodos de *unsupervised Learning* é a *cluster analysis* (Análise de clusters) que consiste na categorização por agrupação ou segmentação de dados segundo características partilhadas entre si, sendo capaz de identificar padrões nos dados, pois reage consoante a presença ou abstenção desses mesmos padrões em cada um dos elementos do conjunto de dados. Esta abordagem é útil para se identificar elementos do conjunto de dados que não pertencem ao grupo.

Neste exercício iremos demonstrar como um algoritmo, aplicando conceitos e técnicas de *unsupervised learning* consegue distinguir entre duas distribuições de dados.

Foi elaborado um programa em *python* para auxiliar na conceção e análise deste problema e, assim, tentar dar resposta às questões colocadas.

Todos os testes e resultados obtidos foram feitos num computador HP EliteBook 840 G6 com um processador i5-8365U, 16GB de RAM e um sistema operativo Windows de 64 bits.

De modo a que os resultados obtidos possam ser replicados, recorreu-se a um sistema de "*seeds*" que garantem a replicação dos resultados.

Para se facilmente executarem os resultados de cada exercício, cada um dos exercícios está num ficheiro *.py* respetivo, pois apesar de se ter replicado código comum aos vários exercícios, é assim mais fácil testar e replicar os resultados dos mesmos. Existe a exceção do ficheiro .py que inclui o exercício 2 e 3 juntos, devido à dependência um do outro.

Cada *.py* conta com uma função indicativa com o nome do exercício respetivo e que funciona como "*main*", onde se podem definir alguns modos de execução do programa, por exemplo, analisar os tempos de execução, ou apenas a média de recompensas por estado.

```
if __name__ == '__main__':
    f_name = 'unsupervisedLearningDataSet.txt'
    alfa = 0.9
    runner(one_time=False, ex2_a=False, ex3=False)
```

Já as variáveis do ambiente, como por exemplo, os valores de alfa e discount terão que ser alterados dentro destas funções "*main*".

one_time: *True* → gráficos pontos | *False* → gráficos dos desvios-padrões.

ex2_a: *True* → Algoritmo ex2 a) | *False* → Algoritmo ex2 b)

ex3: *True* → Gráficos do ex3 | *False* → Outros gráficos

Exercício 1

Neste exercício inicial é definido o conjunto de dados onde os algoritmos implementados serão testados afim de demonstrar o uso de técnicas de *unsupervised learning*. Neste caso serão usados pontos correspondentes a coordenadas de uma matriz de duas dimensões usando uma distribuição gaussiana multivariada. Dois conjuntos distintos com centros diferentes são gerados, baralhados aleatoriamente e depois guardados num ficheiro *.txt*

De modo a não ficar preso nesta etapa por muito tempo, foi usado o ficheiro *.txt* fornecido pelo como anexo e previamente preenchido com os dados acima referidos. Os dados contidos no ficheiro *.txt* têm as seguintes características:

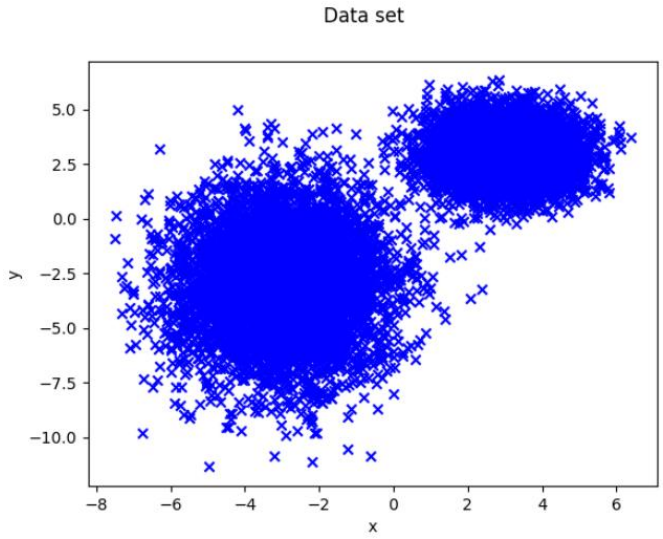
mean1 = [3, 3]

cov2 = [[1, 0], [0, 1]]

mean2 = [-3, -3]

cov2 = [[2, 0], [0, 5]]

De modo a comprovar que os dados ficaram devidamente importados o seguinte gráfico foi elaborado com os mesmos:



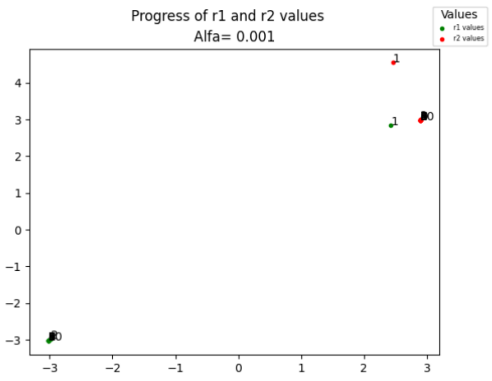
Exercício 2:

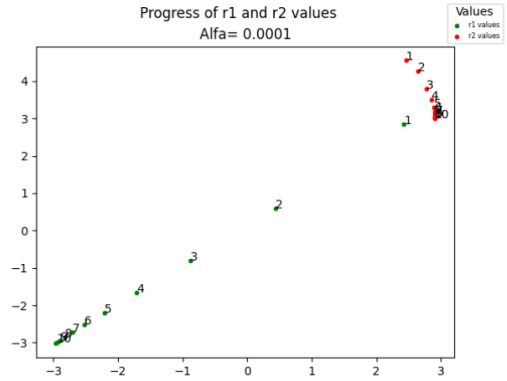
Neste exercício foram escolhidos dois pontos aleatórios do conjunto de dados fornecido, identificados por r1 e r2. Após selecionados os dois pontos, percorreu-se todo o conjunto de dados aplicando a seguinte regra de modo a variar os valores de r1 e r2:

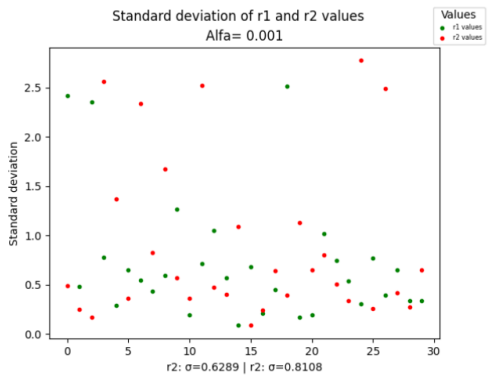
For each point x in the data set:
  If r1 is closest to x than r2
    $r1 = (1 - alfa)\, r1 + alfa * x$
  If r2 is closest to x than r1
    $r2 = (1 - alfa)\, r2 + alfa * x$

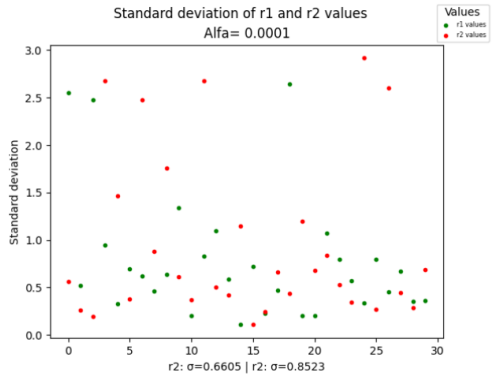
Este processo foi repetido 10 vezes com um alfa = $10e^{-7}$ e guardados os valores consecutivos de r1 e r2. De seguida todo o processo foi repetido 30 vezes para permitir repetição.

Os seguintes gráficos foram obtidos:









Analisando os dados é possível notar-se o processo acidentado e irregular que o ponto r1 e r2 fazem desde a sua posição inicial até à sua posição final, convergindo sempre para o centro dos dois clusters que o conjunto de dados no seu todo forma.

Contudo também se verifica que para valores muito pequenos do alfa o algoritmo tem mais dificuldade em conseguir atingir o objetivo, mostrando que apenas 10 iterações sobre o conjunto de dados não é suficiente. Já para valores do alfa mais elevados, repara-se que ambos os pontos rapidamente convergem para o objetivo, em apenas uma iteração pelo conjunto de dados.

Assim seria até possível automaticamente atribuir os pontos r1 e r2 aos seus respetivos clusters, medindo qual dos clusters tem maior número de pontos com maior proximidade ao ponto em causa.

Uma vez que fazer o cálculo do desvio-padrão de uma lista de coordenadas não é algo direto, foi aplicada a seguinte fórmula de modo a obter o valor:

$$\sigma_{final} = \sqrt{{\sigma_x}^2 - {\sigma_y}^2}$$

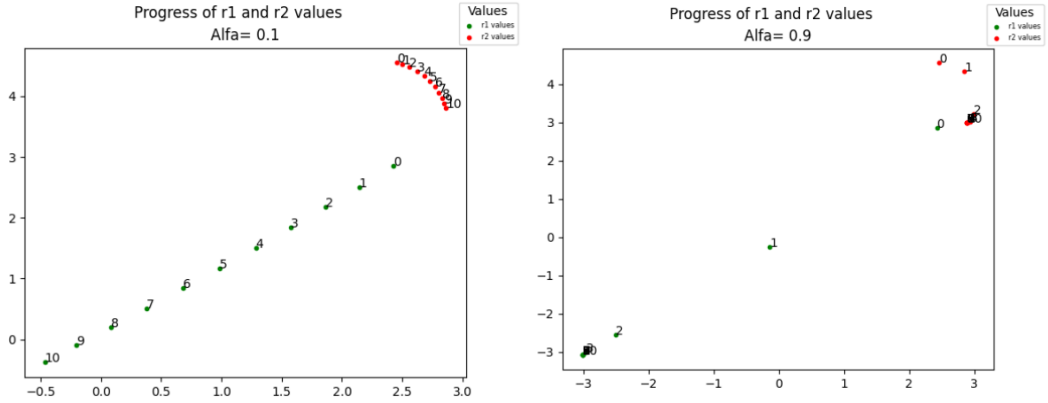
De notar também uma diferença nos resultados quando se analisam os desvios-padrões, sendo que para os valores do ponto r1, durante as sucessivas experiências, tendeu a manter valores não muito dispersos e mais consistentes, já o ponto r2, mostrou ter uma grande variação de desvios-padrões.

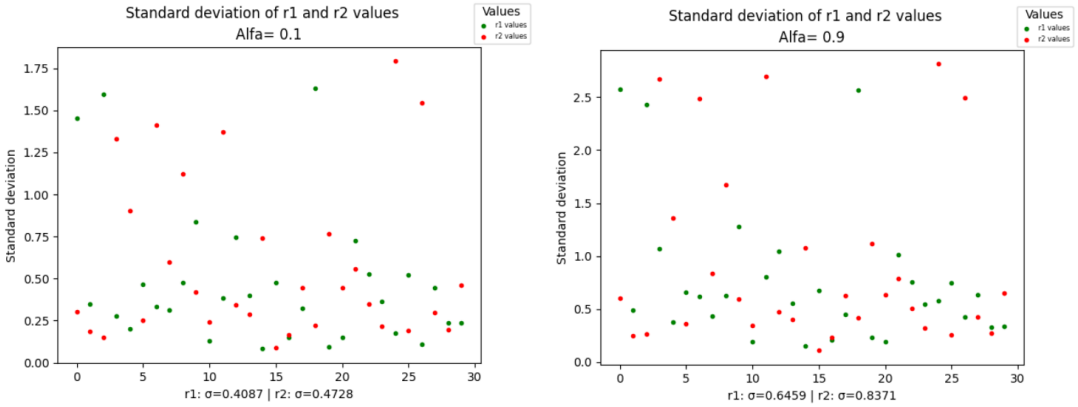
Para a segunda vertente deste exercício foi feita uma alteração na forma de calcular a variância dos pontos r1 e r2, agora em vez de o valor de r1 e r2 ser atualizado a cada exemplo, são acumulados os valores da diferença entre o ponto atual do conjunto de dados e o ponto em causa (r1 ou r2) e só no fim de todos o conjunto de dados ter sido percorrido são os valores de r1 e r2 atualizados.

Assim usou-se a seguinte nova regra:

$$\begin{aligned} &for\ all\ x: \\ &\quad d \mathrel{+}= (x - r) \\ &r \mathrel{+}= \left(\frac{alfa}{n_{examples}}\right) * d \end{aligned}$$

E assim obtiveram-se os seguintes gráficos:





Analisando os dados obtidos, verifica-se que desta vez, o novo algoritmo não consegue ter tanta precisão para valores de alfa mais pequenos, contudo para maiores valores, demonstra rapidamente conseguir levar o ponto r1 e r2 ao objetivo.

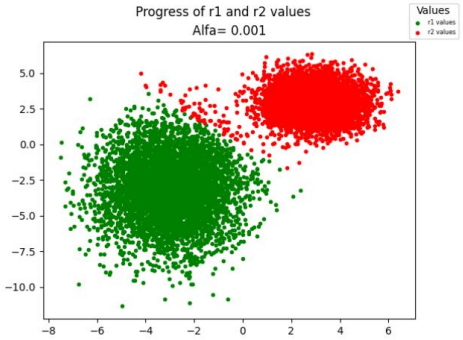
Quanto aos registos dos desvios-padrões não demostraram sofrer grandes alterações em relação à iteração anterior, o desvio padrão do ponto 1 continua a apresentar valores mais consistentes enquanto que o do ponto r2, continua com valores mais dispersos.

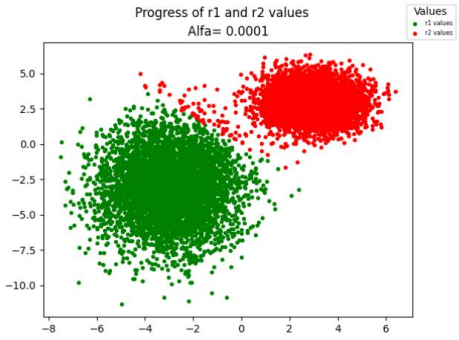
Exercício 3:

Para este exercício mantiveram-se os algoritmos usados no exerício anterior, contudo a analise aos resultados mudou. Desta vez tendo os valores finais do r1 e do r2, foi feita uma verificação de todos os elementos do conjunto de dados e verificando qual dos dois pontos, r1 ou r2 estava mais próximo, sendo que o mais próximo seria agrupado consoante.

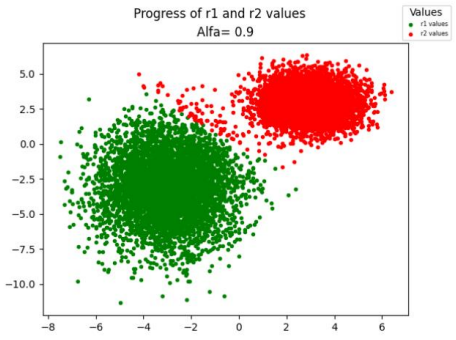
Este processo foi feito para valores diferentes de alfa bem como usando cada um dos dois algoritmos do exercício 2.

Usando o algoritmo do exercício 2 a)





Usando o algoritmo do exercício 2 b)



Observando os resultados conseguidos, consegue-se ter uma boa ideia da existência dos dois clusters presentes no conjunto de dados, tanto usando o algoritmo do exercício 2 a) como o do b), contudo de notar que o algoritmo do 2 a) parece demonstrar melhores resultados e mais precisão na diferenciação dos dois clusters.

Exercício 4:

Para este exercício pretende-se implementar uma simples versão de um cluster hierárquico aglomerativo. Um Cluster hierárquico aglomerativo é uma técnica de clusters, sendo um cluster nada mais é que um agrupamento de dados sob uma mesma característica que neste caso se irá basear-se no tamanho e distância dos dados de um conjunto.

Para isso usou-se a seguinte lógica:

While there are more than two points
Find the closest two points
Replace both points by their average

Uma vez que o conjunto de dados inicial tem um tamanho considerável, 10.000 pontos, e tendo em o nível computacional da implementação acima referida, optou-se antes da aplicação do algoritmo, selecionar aleatoriamente apenas amostras de 100 e 500 elementos do conjunto de dados, como sendo uma amostra representativa do mesmo, pois para uma amostra de 1000 elementos o nível computacional já demonstrava ser muito exigente.

Posto isto, foi testado o seu desempenho com dois valores de alfa diferentes possibilitando a existência de um meio de comparação.

Pontos / *Amostras*	*R1*	*R2*
100	(3.145, 2.516)	(-2.87, -4.368)
500	(4.636, 3.495)	(-2.234, -2.442)

Esta abordagem apesar de obter valores não muito longe do centro dos respetivos clusters existentes, apresenta resultados menos precisos do que nas etapas anteriores. Contudo é de notar que a falta de precisão nos resultados, pode se dever às pequenas amostras usadas, visto que usar todo o conjunto de dados, tornava-se demasiado exigente computacionalmente, ou seja, talvez usando todos o conjunto de dados na sua integridade pudesse aumentar a precisão.

Outro fator que também prejudica os resultados obtidos, é a simplificação do algoritmo que de um cluster hierárquico aglomerativo, pois ao simplificar desta maneira leva a que todos os pontos estejam a ser classificados da mesma maneira, ou seja, falta a definição dos clusters existentes no conjunto de dados, pois sem essa definição não é possível definir quais pontos são o que.

Exercício 5:

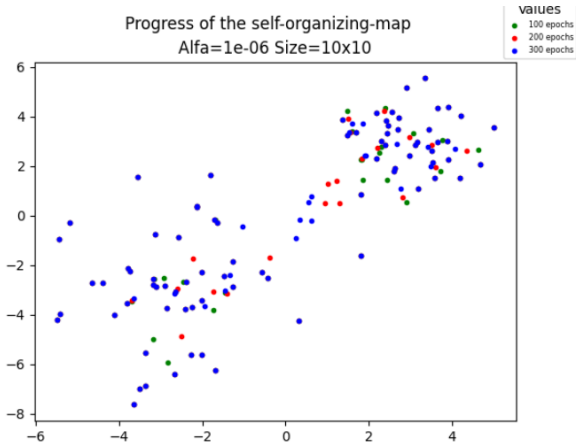
Esta questão pretende simular uma versão simples de um "*self-organizing map*". Um "*self-organizing map*" é um tipo de rede neuronal artificial que é treinada recorrendo ao método de aprendizagem não supervisionada, que aprende a produzir uma representação discreta, de baixas dimensões (duas dimensões) de acordo com o espaço da amostra de dados de treino que são fornecidos. A isto chama-se de "mapa" e é assim um método de reduzir dimensões.

Neste caso em concreto a versão simplificado deste processo será feita adaptando o processo descrito no exercício 2. Desta vez, ao contrário de selecionar-se aleatoriamente 2 pontos do conjunto de dados, é criada uma matriz de "representativos" do conjunto de dados, esta matriz pode variar entre 4x4 e 10x10, sendo que cada um dos seus elementos, é um ponto selecionado aleatoriamente do conjunto de dados, ou seja, para o caso de se trabalhar com uma matriz de tamanho 4x4, 16 pontos aleatórios seriam escolhidos do conjunto de dados.

Após ser feita a escolha dos representativos continuar-se-á a atualizar o valor do representativo com a fórmula do exercício 2, contudo desta vez todos os pontos vizinhos do representativo serão igualmente atualizados usando a seguinte regra:

$$r = (1 - alfa/2)\, r + (alfa/2) * x$$

Para avaliação da evolução do "self-organizing-map" foram feitos "*snapshots*" à medida que o mapa executava 301 iterações, sendo que esses "*snapshots*" foram feitos nas iterações 100, 200 e 300. Para uma questão de comparação, escolheu-se representar estes três "*snapshots*" num só gráfico, sendo que cada "*snapshot*" tem a sua cor correspondente.



Olhando para os resultados conseguidos, é possível ver que os representantes apesar de aleatoriamente escolhidos acabam por convergir num resultado que tenta emular ao máximo aquele que é o aspeto do conjunto de dados inicial, sendo possível ver o progresso, desde os 100

epochs em que os dados ainda não davam uma boa perspetiva do conjunto de dados, até ao resultado final.

Exercício 6:

Para esta secção foi implementado um algoritmo de *DBScan*. O principal conceito deste algoritmo é localizar regiões de alta densidade que se encontram separadas umas das outras por regiões de baixa densidade. Para se medir a densidade de uma região recorre-se a dois passos:

Densidade num dado ponto P: Número de pontos dentro de um circulo com um raio *Eps* (ϵ) do ponto P.

Densidade de uma região: Para cada ponto num cluster, o círculo com raio ϵ contem pelo menos um número mínimo de pontos (*MinPts*).

Seguindo a definição de uma região densa, um ponto é considerado como ponto nuclear quando $|N(p)| \geq MinPts$, tal como o nome o sugere, estes pontos encontram-se normalmente dentro de um *cluster*. Um ponto "fronteira" tem menos de *MinPts* dentro da sua vizinhança com raio ϵ, contudo encontra-se na vizinhança de outro ponto nuclear. Já os pontos classificados como "barulho" é qualquer ponto que não seja, nem fronteira nem nuclear.

Assim este algoritmo consiste nos seguintes passos:

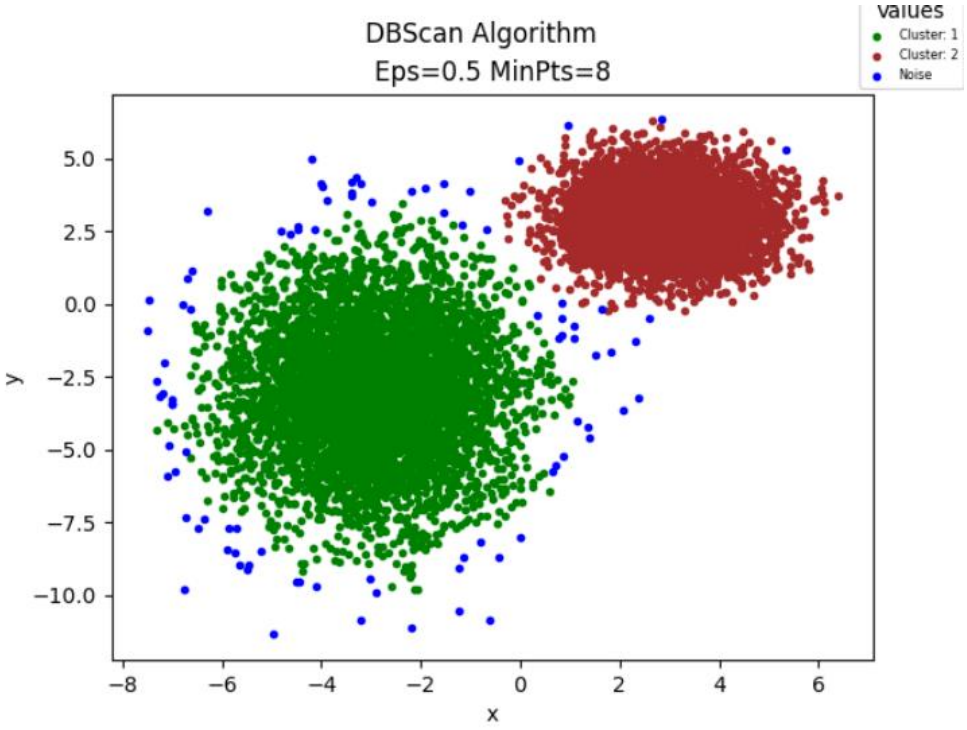
1. O algoritmo começa com um ponto arbitrário que não tenha ainda sido visitado e cuja sua vizinhança é obtida segundo o parâmetro ϵ.
2. Se este ponto contem *MinPts* dentro da sua vizinhança de raio ϵ, começa a formação de um *cluster*, caso contrário o ponto é classificado como "barulho". Este ponto pode ser mais tarde encontrado na vizinhança de raio ϵ de um outro ponto diferente, e assim, pertencer a um *cluster*.
3. Se este ponto é considerado um ponto nuclear, então os seus pontos dentro da sua vizinhança de raio ϵ pertencem também aquele cluster. Assim todos os pontos encontrados na vizinhança de raio ϵ são adicionados ao cluster, bem como todos os pontos pertencentes à vizinhança de cada um.
4. O processo acima mencionado continua até que todos os pontos do cluster sejam encontrados.
5. O processo reinicia com um novo ponto que pode ou não ser um ponto nuclear ou considerado "barulho".

Assim sendo foi implementado o algoritmo em cima descrito, que neste caso baseia-se no pseudo-código que pode ser encontrado na *wikipédia* (https://en.wikipedia.org/wiki/DBSCAN) referente ao mesmo:

```
DBSCAN(DB, distFunc, eps, minPts) {
    C := 0                                                  /* Cluster counter */
    for each point P in database DB {
        if label(P) ≠ undefined then continue               /* Previously processed in inner loop */
        Neighbors N := RangeQuery(DB, distFunc, P, eps)     /* Find neighbors */
        if |N| < minPts then {                              /* Density check */
            label(P) := Noise                               /* Label as Noise */
            continue
        }
        C := C + 1                                          /* next cluster label */
        label(P) := C                                       /* Label initial point */
        SeedSet S := N \ {P}                                /* Neighbors to expand */
        for each point Q in S {                             /* Process every seed point Q */
            if label(Q) = Noise then label(Q) := C          /* Change Noise to border point */
            if label(Q) ≠ undefined then continue           /* Previously processed (e.g., border point) */
            label(Q) := C                                   /* Label neighbor */
            Neighbors N := RangeQuery(DB, distFunc, Q, eps) /* Find neighbors */
            if |N| ≥ minPts then {                          /* Density check (if Q is a core point) */
                S := S ∪ N                                  /* Add new neighbors to seed set */
            }
        }
    }
}
```

```
RangeQuery(DB, distFunc, Q, eps) {
    Neighbors N := empty list
    for each point P in database DB {                   /* Scan all points in the database */
        if distFunc(Q, P) ≤ eps then {                  /* Compute distance and check epsilon */
            N := N ∪ {P}                                /* Add to result */
        }
    }
    return N
}
```

Posto isto, os seguintes resultados foram obtidos:



Analisando o resultado final, é possível perceber que o algoritmo implementado conseguiu com sucesso classificar o conjunto de dados que lhe foi fornecido, tendo conseguido fazer a identificação de cada um dos dois clusters existentes, bem como de todos os pontos que não se enquadravam corretamente em nenhum dos clusters, e ficaram classificados como “barulho”.