Clustering - Tópicos em IA

Arthur Barreto Godoi — arthur.godoi@ufpr.br
Gustavo Gabriel Ripka — gustavo.ripka@ufpr.br
Pedro Henrique Gurski De Oliveira — pedrogurski@ufpr.br
Ulisses Curvello Ferreira — ulissesferreira@ufpr.br

November 19, 2025

Contents

1 Introdução

Para este trabalho pretendemos realizar uma comparação de algoritmos de Clustering, implementando diferentes métodos, sendo eles: K-Means, DBSCAN e clusterização hierárquica. Utilizando três Datasets diferentes e avaliando seus desempenhos de métricas de coesão e separabilidade.

2 *Dataset's utilizados*

Os *dataset's* são dos conjuntos de dados do Sklearn, onde temos conjuntos de dados populares e geradores de dados artificiais.

2.1 make_moons

make_moons gera conjuntos de dados de classificação binária 2D que são desafiadores para certos algoritmos (por exemplo, agrupamento baseado em centróide ou classificação linear), incluindo ruído gaussiano opcional. Produz dois semicírculos intercalados.

Formato básico:
Retorna:

- X: matriz (array) com as coordenadas dos pontos, no formato $(n\_samples, 2)$.
- y: vetor com rótulos binários (0 ou 1), no formato $(n\_samples)$.

Referências e exemplos:

- Documentação oficial: scikit-learn (make_moons)
- Exemplo de geração de dados sintéticos: Kaggle Notebook

2.2 make_blobs

make_blobs gera conjuntos de dados sintéticos para tarefas de clustering (ou classificação), com "blobs" (aglomerados) gaussianos isotrópicos em espaço de características (features). Serve para testar algoritmos que supõem agrupamentos bem definidos (por exemplo, K-Means).

Formato básico:
Retorna:

- X: matriz (array) com as características dos pontos, no formato $(n\_samples, n\_features)$.
- y: vetor com rótulos inteiros indicando o cluster de cada ponto (valores de 0 a $n\_centers - 1$), no formato $(n\_samples)$.

- opcionalmente, centers: matriz contendo as coordenadas dos centros dos clusters, se return_centers=True.

Referências e exemplos:

- Documentação oficial: scikit-learn (make_blobs)
- Exemplo de geração de dados sintéticos: Kaggle Notebook

2.3 make_circles

make_circles gera conjuntos de dados de classificação binária 2D que são úteis para avaliar algoritmos de classificação não linear. O dataset é composto por dois círculos concêntricos (um círculo dentro do outro), podendo incluir ruído gaussiano opcional. Esse formato o torna desafiador para modelos lineares, mas bastante adequado para testes com métodos que capturam fronteiras de decisão não lineares.

Formato básico:
Retorna:

- X: matriz (array) com as coordenadas dos pontos, no formato $(n\_samples, 2)$.
- y: vetor com rótulos binários (0 ou 1), no formato $(n\_samples)$.

Referências e exemplos:

- Documentação oficial: scikit-learn (make_circles)
- Exemplo de geração de dados sintéticos: Kaggle Notebook

3 Análise exploratória de dados (EDA):

Para cada um dos *Dataset's*: analisamos a sua distribuição dos pontos e como se comportavam, seus Boxplots, Histogramas e por apenas finalidade de teste, aplicamos uma regressão logística básica para ver como se desenpenhava.

3.1 Análise do make_moons

Começamos a análise exploratória do dataset *make_moons* olhando para a distribuição dos pontos em um gráfico de dispersão:

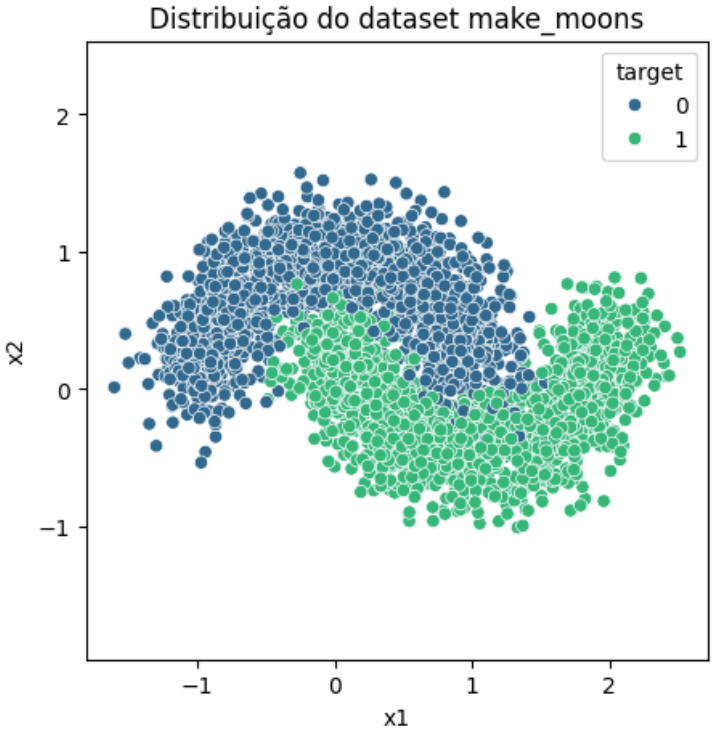


Figure 1: Distribuição do dataset *make_moons*

Nesse gráfico, cada ponto é definido pelas variáveis `x1` e `x2` e recebe uma cor de acordo com sua classe (`target`). O que vemos são dois semicírculos entrelaçados, um padrão típico desse dataset. Esse formato já mostra que não existe uma linha reta capaz de separar as duas classes sem erros. A sobreposição das classes nas regiões de interseção entre os arcos introduz inevitáveis ambiguidades na classificação. Como consequência, modelos lineares, a exemplo da regressão logística, encontram barreiras para representar adequadamente esses padrões. Abordagens não lineares, como SVM com *kernel*, árvores de decisão

ou redes neurais, são mais indicadas nesse tipo de cenário, pois oferecem maior flexibilidade na definição das fronteiras de decisão.

Depois, analisamos os boxplots das variáveis `x1` e `x2` e seus histogramas:

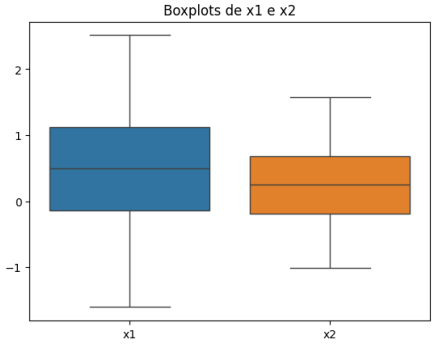


Figure 2: Boxplots das variáveis `x1` e `x2`.

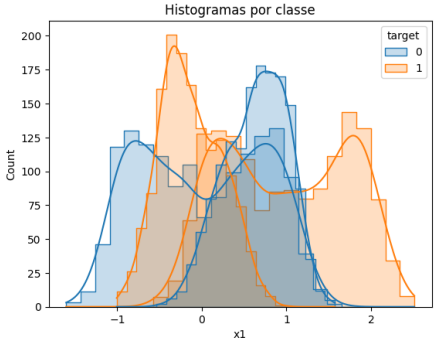


Figure 3: Histogramas das variáveis `x1` e `x2` por classe.

O que eles mostram é que `x1` tem uma variação maior, indo de aproximadamente $-1{,}5$ até 2,5, enquanto `x2` fica mais concentrado, entre -1 e 1,5. Apesar de ambos serem relativamente simétricos, essa diferença de amplitude indica que `x1` poderia ter mais peso no modelo se não houvesse padronização. Isso reforça a importância de aplicar técnicas de pré-processamento, como o `StandardScaler`, que garantem que cada variável contribua de forma equilibrada no processo de aprendizado.

Os histogramas por classe ajudaram a enxergar melhor como cada variável se distribui em relação ao alvo. Apesar de certa separação entre as classes, há muita sobreposição, principalmente em regiões intermediárias. Isso confirma que olhar apenas para `x1` ou `x2` isoladamente não é suficiente para separar bem as classes. Na prática, é a combinação das duas variáveis e o uso de modelos que conseguem explorar relações não lineares que realmente permite identificar melhor os grupos.

Por fim, apenas por finalidade de teste, testamos a regressão logística. Mesmo sendo um modelo linear aplicado a um problema não linear, ela conseguiu um desempenho razoável, com acurácia em torno de 86%. A matriz de confusão mostrou que a maior parte dos pontos foi classificada corretamente: 406 da classe 0 e 372 da classe 1, contra 64 e 58 erros, respectivamente:

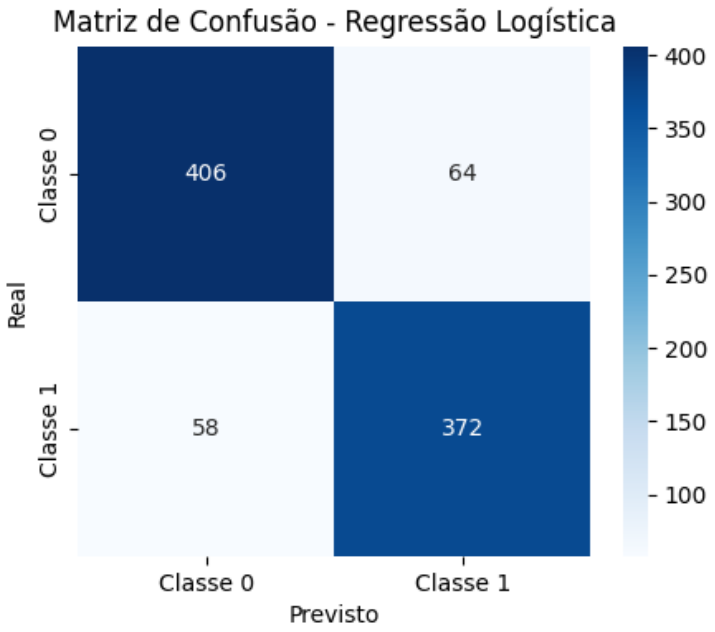


Figure 4: Distribuição do dataset *make_moons*

Isso mostra que a regressão logística é capaz de capturar boa parte do padrão nos dados, mas ainda assim fica limitada pela estrutura do problema.

3.2 Análise do make_blobs

Começamos a análise do dataset *make_blobs* olhando para a distribuição dos pontos em um gráfico de dispersão:

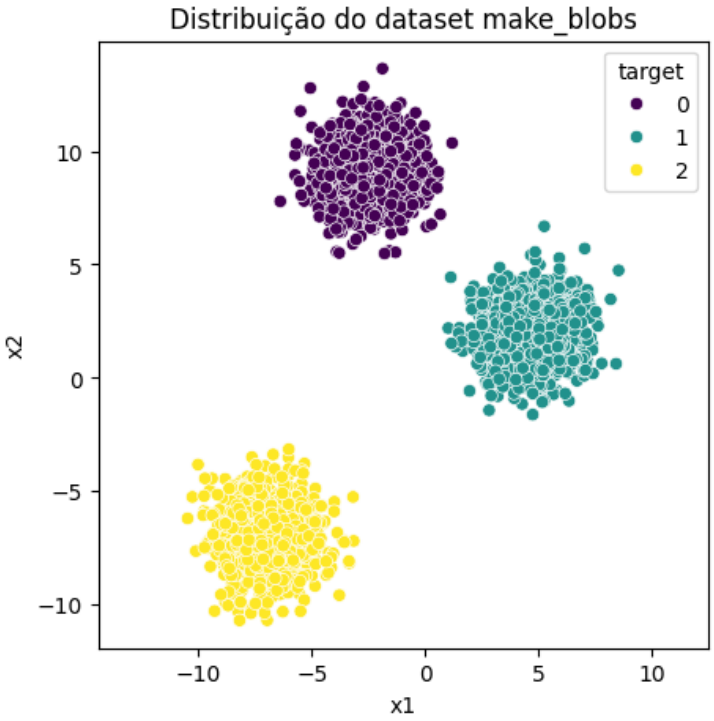


Figure 5: Distribuição do dataset *make_blobs*

O gráfico mostra três grupos bem definidos e separados, cada um representando uma classe distinta. Diferente do *make_moons*, os clusters são esféricos e pouco sobrepostos, o que facilita bastante a tarefa de separação para modelos lineares ou algoritmos de clustering baseados em distância.

Depois, analisamos os boxplots das variáveis `x1` e `x2`, junto com os histogramas por classe:

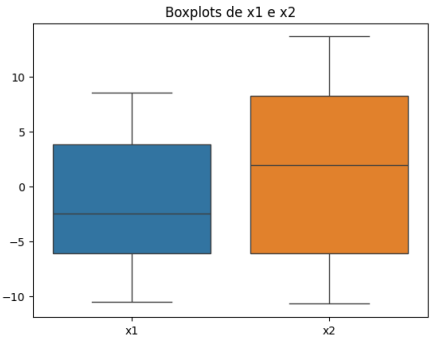


Figure 6: Boxplots das variáveis `x1` e `x2`.

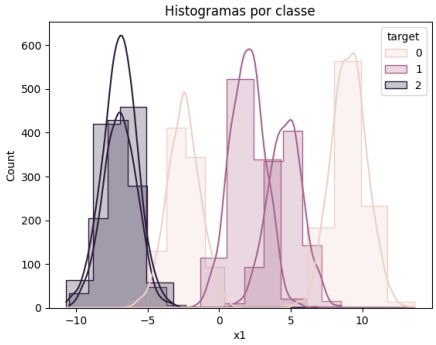


Figure 7: Histogramas das variáveis por classe.

Os boxplots mostram que a variável `x2` tem maior amplitude em comparação a `x1`, reforçando a necessidade de padronização para evitar que diferenças de escala prejudiquem o modelo. Já os histogramas evidenciam que as classes estão bem separadas, com pouca sobreposição, confirmando a facilidade de classificação deste dataset.

Por fim, aplicamos uma regressão logística multinomial para avaliar o desempenho do modelo:

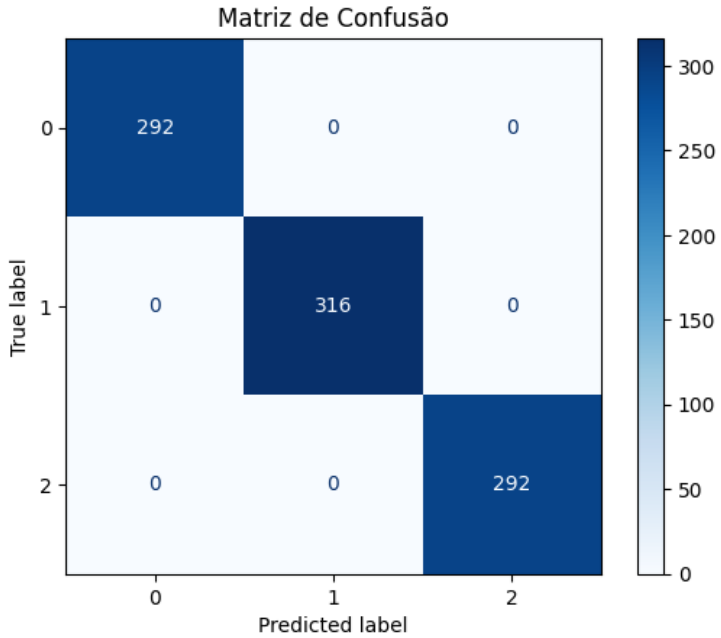


Figure 8: Matriz de confusão – Regressão Logística Multinomial.

O modelo atingiu acurácia de 100%, classificando corretamente todas as instâncias do conjunto de teste. A matriz de confusão ilustra esse resultado, mostrando que não houve erros de classificação. Esse cenário era esperado, já que o *make_blobs* gera dados gaussianos bem definidos e separados, representando um caso "ideal" para algoritmos lineares.

3.3 Análise do make_circles

No caso do *make_circles*, o gráfico de dispersão mostra claramente dois círculos concêntricos que representam as classes 0 e 1. Essa disposição reforça a natureza não linear do problema: não há como traçar uma linha reta que separe bem os grupos. Esse padrão já indica que modelos lineares terão grande dificuldade de capturar a fronteira de decisão correta.

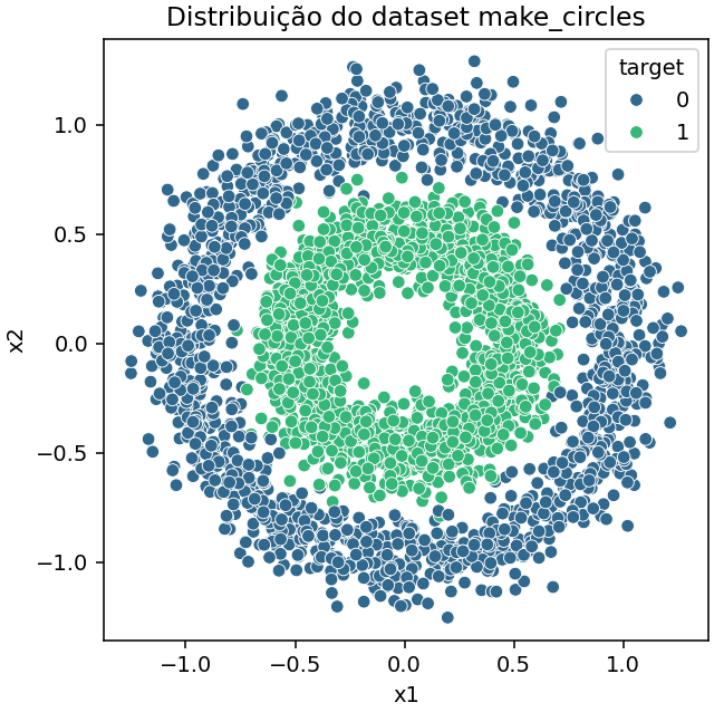


Figure 9: Distribuição do dataset *make_circles*.

A análise dos boxplots evidencia que as variáveis `x1` e `x2` possuem amplitudes semelhantes, o que sugere uma contribuição relativamente equilibrada das duas no modelo. Porém, os histogramas deixam evidente a forte sobreposição entre as distribuições de cada classe. Isso significa que, olhando apenas para `x1` ou `x2` separadamente, não conseguimos distinguir bem os grupos. A separação só faz sentido quando se considera a relação geométrica entre as duas variáveis.

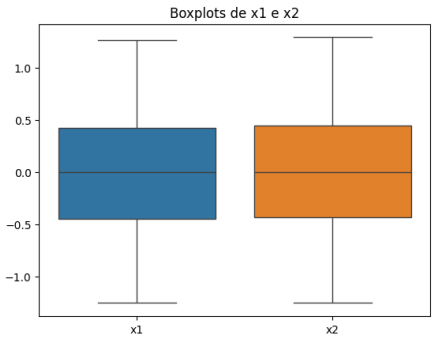


Figure 10: Boxplots das variáveis `x1` e `x2`.

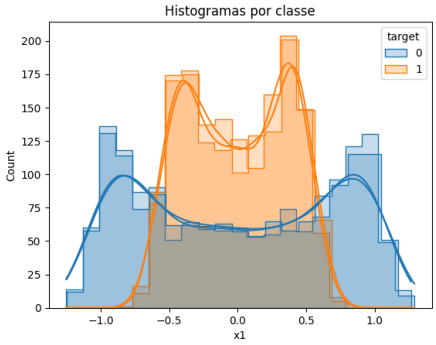


Figure 11: Histogramas das variáveis por classe.

Quando aplicamos a regressão logística, o desempenho foi bastante limitado: a acurácia ficou em torno de 52%. A matriz de confusão mostra muitos erros, especialmente na classe 0, com apenas 166 acertos contra 304 classificações incorretas. Embora a classe 1 tenha tido um resultado um pouco melhor, ainda houve bastante confusão nas previsões. Isso mostra que a regressão logística não é um bom modelo para dados em formato circular, pois não consegue "enxergar" a separação entre os grupos. Para lidar com esse tipo de problema, é preciso recorrer a modelos mais avançados que conseguem aprender fronteiras de decisão mais flexíveis.

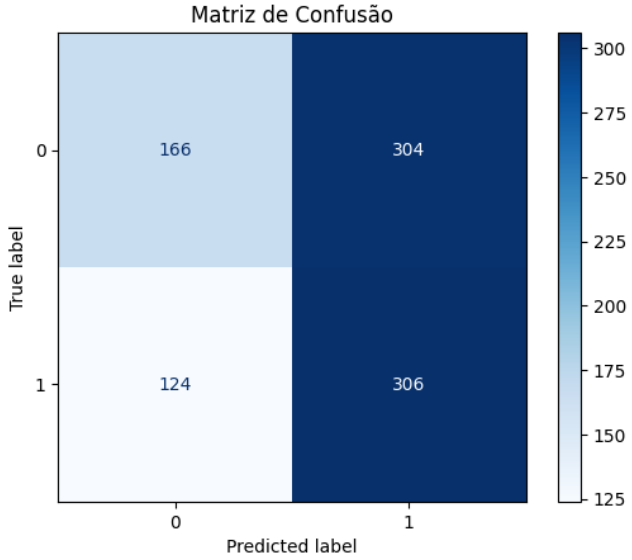


Figure 12: Matriz de confusão Regressão Logística no *make_circles*.

Conclusão Etapa 2

Comparando os três datasets analisados *make_moons*, *make_blobs* e *make_circles* fica claro como a estrutura dos dados influencia a performance de um modelo linear como a regressão logística. Esses experimentos ilustram bem como a escolha do algoritmo precisa estar alinhada ao tipo de padrão presente nos dados: modelos lineares podem ser eficientes em cenários mais simples, mas perdem força em estruturas mais complexas, que exigem abordagens não lineares.

Link do Colab, onde foram realizadas as análises e visualizações dos *Dataset's*: Notebook Colab

4 Resultados

Obtivemos alguns resultados preliminares ao testar os três modelos (K-Means, DBSCAN e clusterização hierárquica) nos nossos datasets escolhidos, com análise inicial dos resultados e possíveis ajustes de abordagem.

As métricas de avaliação de agrupamento foram:
Adjusted Rand Index (ARI)[-1, 1]: 1: agrupamento perfeito; 0: aleatório.
Silhouette Score: usada para avaliar como são bons resultados de clustering no clustering de dados.
Normalized Mutual Information (NMI)[0, 1]: Mede quanta informação é compartilhada entre os rótulos verdadeiros e os clusters gerados.
V-measure [0, 1]: Média harmônica entre homogeneity e completeness.

4.1 K-Means

O algoritmo funciona atribuindo cada ponto de dados ao centroide mais próximo e, em seguida, atualizando iterativamente a posição desses centroides até que a variação interna dos grupos seja minimizada. O objetivo é garantir que os pontos dentro de um mesmo cluster sejam o mais semelhantes possível entre si e o mais diferentes possível dos pontos de outros clusters.

4.1.1 `make_moons`

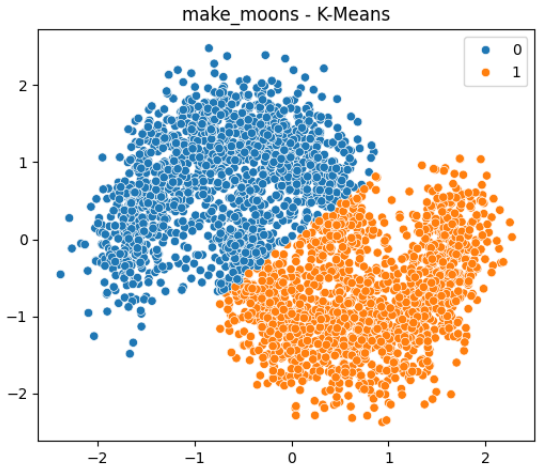


Figure 13: Aplicação do algoritmo K-means sobre o conjunto *make_moons*

- **Adjusted Rand Index (ARI)**: 0.4595
- **Silhouette Score**: 0.4691
- **NMI**: 0.3633

- **V-measure**: 0.3633

O conjunto *make_moons* possui duas regiões semicirculares intercaladas, o que representa uma estrutura **não linearmente separável**. O algoritmo K-Means, por assumir que os clusters possuem formato esférico e fronteiras lineares, apresenta desempenho apenas **moderado**.

O valor intermediário de *ARI* (0.46) e de *Silhouette* (0.47) indica que o modelo conseguiu capturar parcialmente a estrutura dos dados, mas ainda com significativa confusão entre as classes. As métricas *NMI* e *V-measure* confirmam essa limitação.

Conclusão: O K-Means não é adequado para dados com fronteiras não lineares, apresentando segmentação imperfeita neste caso.

4.1.2 make_blobs

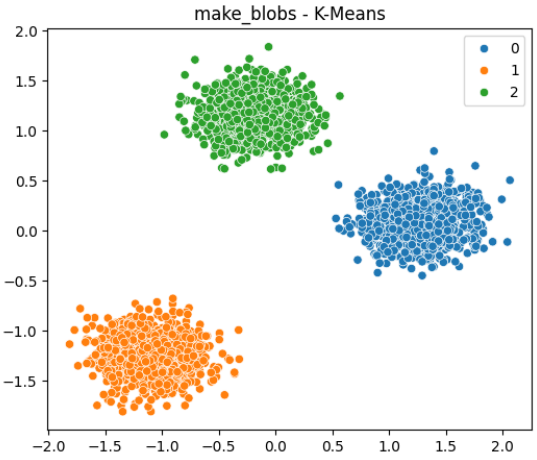


Figure 14: Aplicação do algoritmo K-Means sobre o conjunto `make_blobs`.

- **Adjusted Rand Index (ARI)**: 1.0000
- **Silhouette Score**: 0.8108
- **NMI**: 1.0000
- **V-measure**: 1.0000

O conjunto *make_blobs* gera dados com clusters **esféricos e bem separados**, cenário ideal para o algoritmo K-Means. Os resultados demonstram **desempenho perfeito**, com *ARI*, *NMI* e *V-measure* iguais a 1.0, indicando correspondência total com os rótulos verdadeiros.

O *Silhouette Score* (0.81) reforça a alta coesão interna e a boa separação entre clusters.

Conclusão: O K-Means apresenta desempenho ideal para dados que respeitam suas suposições de clusters convexos e bem definidos.

4.1.3 make_circles

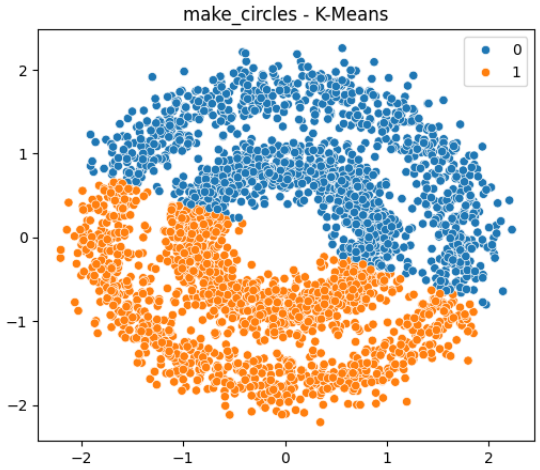


Figure 15: Aplicação do algoritmo K-means sobre o conjunto *make_circles*

- **Adjusted Rand Index (ARI)**: -0.0003
- **Silhouette Score**: 0.3435
- **NMI**: 0.0000
- **V-measure**: 0.0000

O conjunto *make_circles* contém dois círculos concêntricos, o que constitui um problema altamente **não linear**. O K-Means, ao tentar criar fronteiras lineares, não consegue separar os clusters corretamente. As métricas (*ARI*, *NMI*, *V-measure*) próximas de zero indicam desempenho praticamente aleatório. O *Silhouette* também é baixo (0.34), confirmando a má qualidade da separação.

Conclusão: O K-Means falha completamente nesse tipo de estrutura, pois seus pressupostos geométricos não se aplicam a dados concêntricos.

4.2 DBSCAN

O DBSCAN(Density-Based Spatial Clustering of Applications with Noise) não recebe o número de classes (clusters) como parâmetro, ele descobre automaticamente quantos clusters existem com base na densidade dos dados. Cria clusters baseados em densidade local de pontos, e não na forma geométrica.

Ele usa dois parâmetros principais:
eps: Raio máximo para considerar pontos como "vizinhos".
min_samples: Número mínimo de pontos dentro desse raio para formar um cluster.

Como ele decide os clusters:

1. Escolhe um ponto aleatoriamente;
2. Se esse ponto *tiver* $\geq$ *min_samples* vizinhos em um raio eps, ele é um ponto central, então forma um novo cluster;
3. Os vizinhos também são adicionados e o processo se repete.
4. Pontos que não se encaixam em nenhum cluster (muito isolados) são rotulados como ruído (label = -1).

4.2.1 make_moons

Primeiro Teste
A análise dos resultados obtidos com o **DBSCAN(eps=0.15, min_samples=10)** sobre o dataset **make_moons** pode ser feita observando tanto o gráfico quanto as métricas apresentadas:

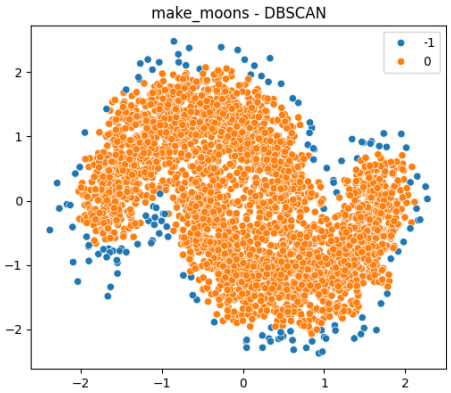


Figure 16: DBSCAN(eps=0.15, min_samples=10) aplicado ao dataset make_moons.

- **Adjusted Rand Index (ARI):** -0.0001 (Valor próximo de 0: não há correspondência com os rótulos verdadeiros)
- **Silhouette Score:** 0.2041 (Baixo: indica clusters mal definidos e pouca separação entre grupos)
- **NMI:** 0.0000 (Nenhuma informação mútua entre clusters e classes reais)
- **V-measure:** 0.0000 (Não houve correspondência com a estrutura real dos dados)

Podemos observar que o DBSCAN identificou apenas um cluster principal e classificou o restante como ruído(-1). Isso indica que os parâmetros usados não foram capazes de separar as duas "luas". A combinação eps=0.15 e min_samples=10 fez com que o DBSCAN fosse muito restritivo, formando apenas um cluster e descartando o resto.

Segundo Teste

A análise dos resultados obtidos com o **DBSCAN(eps=0.1, min_samples=10)** sobre o dataset **make_moons** pode ser feita observando tanto o gráfico quanto as métricas apresentadas:

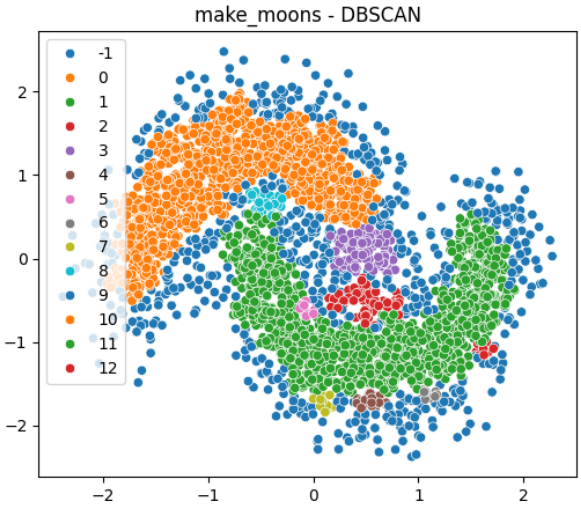


Figure 17: DBSCAN(eps = 0.1, min_samples = 10) aplicado ao dataset make_moons.

- **Adjusted Rand Index (ARI):** 0.4515 (Moderado: há correspondência parcial com os rótulos verdadeiros. Melhor que antes, mas longe do ideal)
- **Silhouette Score:** -0.3886 (Negativo: indica que muitos pontos estão mal alocados, próximos de fronteiras entre clusters)

- **NMI:** 0.4329 (Mostra alguma relação entre clusters e classes reais, mas ainda com muita fragmentação)
- **V-measure:** 0.4329 (Coerente com NMI: reflete a mesma ideia de correspondência parcial)

Diferente do caso anterior (em que havia só um cluster), agora o **DBSCAN** conseguiu identificar múltiplos grupos dentro das duas luas. No entanto, ele fragmentou demais as regiões contínuas, ou seja, as duas luas foram divididas em vários clusters menores (de 0 a 12).
O aumento do número de clusters fez o **ARI** e **NMI** melhorarem, mas o **Silhouette Score** negativo mostra que a separação não é coesa, os clusters estão se sobrepondo ou muito fragmentados.

Conclusão dos Testes
O DBSCAN é um algoritmo poderoso para detectar clusters baseados em densidade, mas ele não se adapta bem ao dataset make_moons.

Se eps é pequeno: o algoritmo fragmenta as luas em vários clusters.
Se eps é grande: ele funde as duas luas em um único cluster.

O DBSCAN até lida bem com formas não lineares, mas só funciona bem quando há clara separação espacial (densidade realmente baixa entre grupos). No make_moons, as luas ficam muito próximas, o espaço de baixa densidade entre elas é insuficiente para que o DBSCAN as separe corretamente.

4.2.2 make_blobs

Primeiro Teste
A análise dos resultados obtidos com o **DBSCAN(eps=0.10, min_samples=10)** sobre o dataset **make_blobs** pode ser feita observando tanto o gráfico quanto as métricas apresentadas:

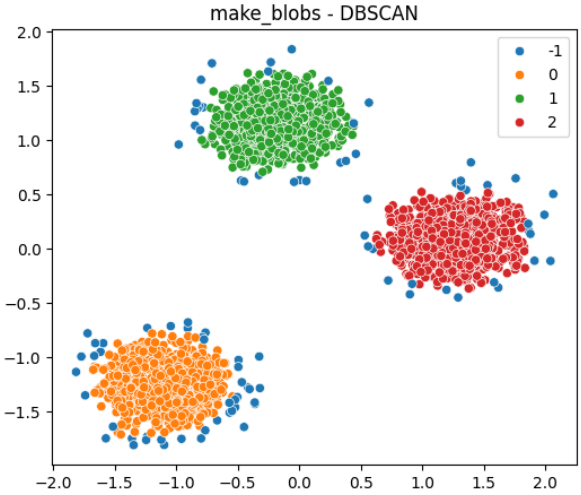
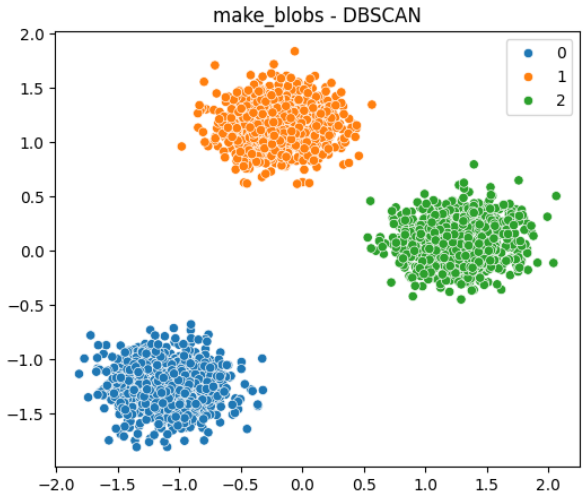


Figure 18: DBSCAN(eps=0.10, min_samples=10) aplicado ao dataset make_blobs.

- **Adjusted Rand Index (ARI):** 0.9529 (Muito alto: quase perfeita correspondência com os rótulos verdadeiros)
- **Silhouette Score:** 0.7524 (Excelente: boa coesão interna e separação entre clusters)
- **NMI:** 0.9261 (Clusters bem alinhados com as classes reais)
- **V-measure:** 0.9261 (Alta homogeneidade e completude)

O DBSCAN detectou ruídos (-1) nas bordas dos grupos. Os três clusters foram corretamente identificados, mas com bordas mais restritas.
O DBSCAN com eps=0.10 e min_samples=10 foi eficaz para o dataset make_blobs, identificando corretamente os três grupos principais. No entanto, o raio de vizinhança pequeno fez com que o modelo considerasse mais pontos como ruído, reduzindo levemente as métricas de desempenho.

Segundo Teste
A análise dos resultados obtidos com o **DBSCAN(eps=0.3, min_samples=10)** sobre o dataset **make_blobs** pode ser feita observando tanto o gráfico quanto as métricas apresentadas:



4

Figure 19: DBSCAN(eps=0.3, min_samples=10) aplicado ao dataset make_blobs.

- **Adjusted Rand Index (ARI):** 1.0000 (Agrupamento idêntico aos rótulos verdadeiros)
- **Silhouette Score:** 0.8108 (Excelente separação entre clusters; valores acima de 0.5 já são muito bons)
- **NMI:** 1.0000 (Correspondência perfeita entre clusters e classes reais)
- **V-measure:** 1.0000 (Medida de homogeneidade e completude também perfeita)

Todos os indicadores apontam para um agrupamento perfeito, o que mostra que o DBSCAN foi extremamente eficaz nesse tipo de dado! Esse resultado ilustra o ponto forte do DBSCAN: detectar clusters densos e bem separados, mesmo sem saber previamente o número de grupos.

Conclusão dos Testes
O DBSCAN é ideal quando há clusters bem definidos, separados e com densidade uniforme, exatamente o caso do make_blobs. Por outro lado, ele falha em datasets com densidades variáveis ou formatos complexos, como o make_moons.

4.2.3 make_circles

Primeiro Teste
A análise dos resultados obtidos com o **DBSCAN(eps=0.2, min_samples=10)** sobre o dataset **make_circles** pode ser feita observando tanto o gráfico quanto as métricas apresentadas:

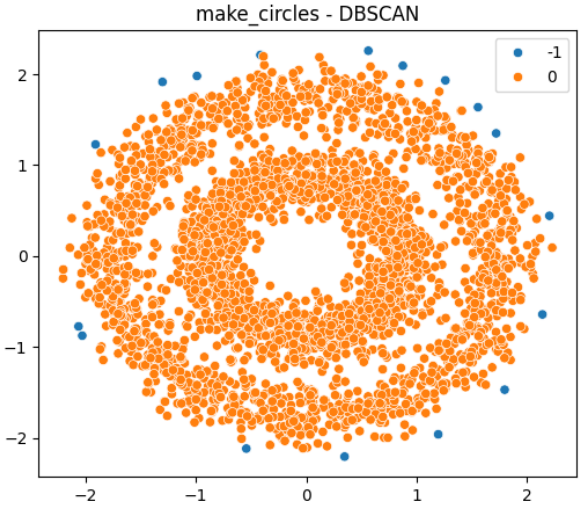


Figure 20: DBSCAN(eps=0.2, min_samples=10) aplicado ao dataset make_circles.

- **Adjusted Rand Index (ARI):** 0.0001 (Nenhuma correspondência com as classes reais)
- **Silhouette Score:** 0.2732 (Baixo: clusters pouco definidos, sobreposição entre regiões)
- **NMI:** 0.0108 (Quase zero: nenhuma relação entre clusters e classes verdadeiras)
- **V-measure:** 0.0108 (Reforça que a estrutura real não foi capturada)

O DBSCAN não conseguiu distinguir os dois círculos. Ele agrupou praticamente todos os pontos como um único cluster grande, ignorando a separação natural entre o círculo interno e o externo.

Segundo Teste

A análise dos resultados obtidos com o **DBSCAN(eps=0.15, min_samples=10)** sobre o dataset **make_circles** pode ser feita observando tanto o gráfico quanto as métricas apresentadas:

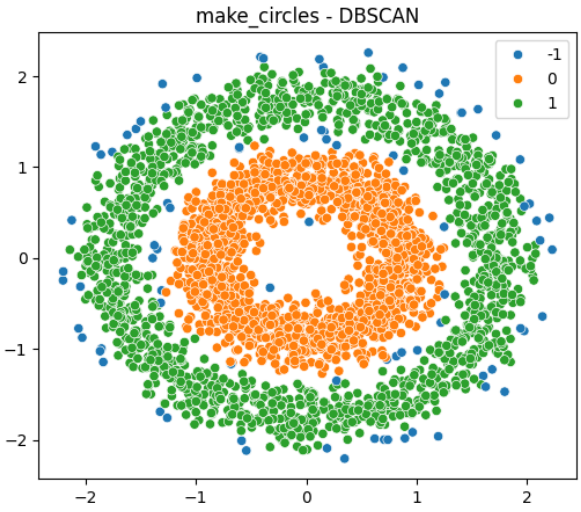


Figure 21: DBSCAN(eps=0.3, min_samples=10) aplicado ao dataset make_circles.

- **Adjusted Rand Index (ARI):** 0.9328 (Excelente: o agrupamento está quase idêntico às classes reais)
- **Silhouette Score:** 0.1031 (Baixo, mas esperado: os clusters são próximos e circulares, o que reduz a separação média)
- **NMI:** 0.8859 (Alta correlação entre clusters e classes verdadeiras)
- **V-measure:** 0.8859 (Alta homogeneidade e completude, ou seja, boa correspondência geral)

As duas cores principais (laranja e verde) representam os dois círculos, o interno e o externo. Os pontos azuis (-1) são considerados ruído (outliers), localizados nas bordas mais dispersas.
Diferente da execução anterior (onde o DBSCAN agrupava tudo em um único cluster), agora ele capturou corretamente a estrutura concêntrica dos dados, dois anéis bem definidos. Isso mostra que o ajuste do parâmetro eps para 0.15 fez o modelo encontrar a densidade correta para diferenciar os círculos.

Conclusão dos Testes
O DBSCAN se mostrou muito eficiente para o dataset make_circles, conseguindo classificar seus círculos interno e externo. Porém, por os dados serem muito próximos, acabaram gerando ruído ou classificando apenas uma classe predominante ao alterar os parâmetros eps.

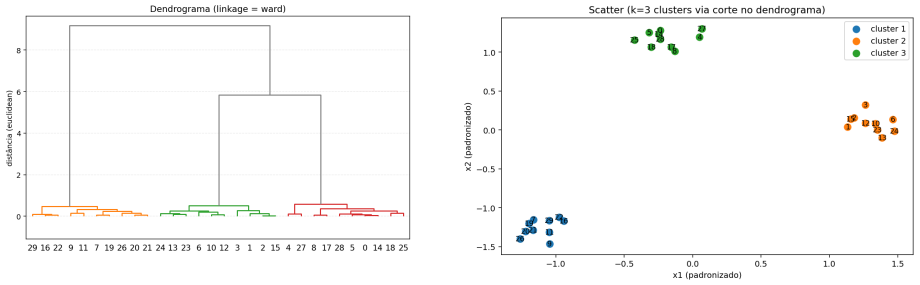
4.3 Clusterização Hierárquica

A **clusterização hierárquica aglomerativa** é uma técnica de agrupamento que parte do princípio "do menor para o maior": cada ponto começa como um cluster próprio e, a cada passo, dois clusters "mais próximos" são fundidos. Esse processo *bottom-up* continua até restar o número desejado de grupos (k) ou até que a distância mínima entre clusters ultrapasse um limite definido.

O histórico dessas fusões pode ser visualizado como uma **árvore do dendrograma**, que mostra a ordem em que os clusters se juntaram e o "custo", isto é, a distância de cada junção. Isso torna o método especialmente interpretável, pois, além de produzir rótulos finais, ele revela a estrutura hierárquica dos dados.

A clusterização hierárquica aglomerativa começa tratando cada amostra como um cluster e, passo a passo, funde o par de clusters mais próximos segundo uma regra de proximidade (*linkage*). Após cada fusão, recalcular as distâncias e repetir até atingir o número desejado de grupos (k) ou um limiar de distância. Quando há um dendrograma, costuma-se escolher o "corte" onde aparece um salto grande nas distâncias, pois geralmente separa grupos mais naturais.

O *linkage* determina o comportamento do método: **ward** minimiza o aumento de variância (excelente para *blobs* esféricos, requer distância euclidiana); **complete** usa a maior distância entre pontos (clusters compactos, pode fragmentar grupos grandes); **average** usa a média das distâncias (meio-termo estável); **single** usa a menor distância (capta formas alongadas, mas sofre com *chaining*). A métrica de distância (euclidiana, Manhattan, cosseno etc.) e a padronização das variáveis (ex.: `StandardScaler`) influenciam diretamente as fusões. No fim, você obtém os rótulos de cluster e, se quiser interpretar, o dendrograma que explica como as fusões aconteceram.



(a) Dendrograma — método Ward (b) Grupos formados após corte

Figure 22: Exemplo de árvore do dendrograma representando a agrupação do gráfico lateral.

4.3.1 make_moons

O dataset tem duas meias-luas entrelaçadas. *Linkages* que favorecem compactação tendem a "cortar" as luas; critérios que seguem conectividade podem ligar tudo por cadeias de pontos.

- **ward -** ARI = `0.518` - Sil = `0.455`: privilegia grupos compactos; impõe uma fronteira quase reta que corta as curvas. Resultado razoável, mas erra na zona de contato.
- **average -** ARI = `0.0` - Sil = `NaN`: usa distância média; acaba conectando as luas e, ao forçar $k = 2$, isola um ponto e deixa o resto junto. ARI≈0 e Silhouette indefinido.
- **complete -** ARI = `0.505` - Sil = `0.431`: mais conservador contra "pontes", separa melhor que o *average*, mas ainda com fronteiras retas; desempenho parecido ao *ward*.
- **single -** ARI = `0.0` - Sil = `NaN`: sofre com o *chaining effect*; ao cortar em $k = 2$, geralmente isola um *outlier* e mantém o resto unido.

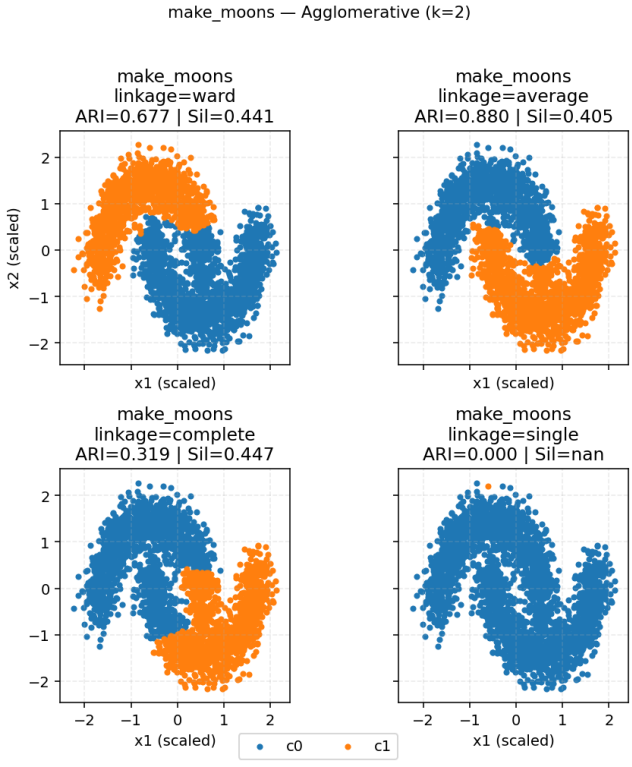


Figure 23: Distribuição dos dados do `make_moons`.

4.3.2 make_blobs

Três "bolhas" esféricas e bem separadas cenário ideal para o hierárquico. A noção de distância é consistente e não há cadeias conectando clusters, então todos os *linkages* convergem para a partição verdadeira.

- **ward -** ARI = `1.000` - Sil = `0.844`: separa os três centros com fronteiras limpas.
- **average -** ARI = `1.000` - Sil = `0.844`: a média entre pontos preserva a mesma hierarquia de fusões.
- **complete -** ARI = `1.000` - Sil = `0.844`: evita "pontes" e mantém os grupos intactos.
- **single -** ARI = `1.000` - Sil = `0.844`: mesmo propenso a *chaining*, acerta porque as bolhas estão bem separadas.

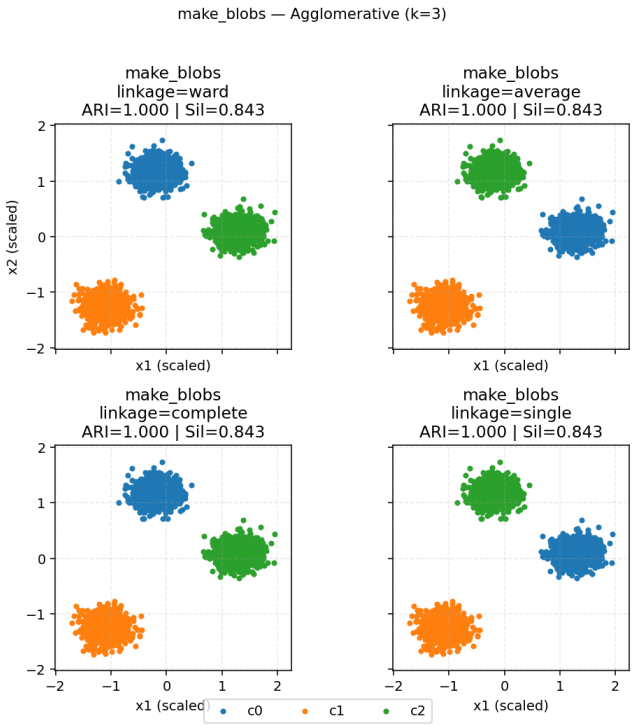


Figure 24: Distribuição dos dados do `make_blobs`.

4.3.3 make_circles

Dois anéis concêntricos (não convexos). *Linkages* que assumem convexidade tendem a misturar os anéis; já o critério que segue a menor distância consegue

percorrer o círculo, mas é sensível a ruído e tem Silhouette baixo.

- **ward -** ARI = `0.028` - Sil = `0.324`: busca formas esféricas; cria cortes quase retos e mistura partes dos anéis.
- **average -** ARI = `0.0` - Sil = `0.319`: a distância média não evita conexões indesejadas; mistura os círculos.
- **complete -** ARI=`0.003` - Sil = `0.328`: mais rígido contra "pontes", mas ainda quebra a geometria circular.
- **single -** ARI = `1.000` - Sil = `0.111`: segue a conectividade e recupera os dois anéis; Silhouette baixo por clusters longos e pouca separação radial (além de sensível a ruído).

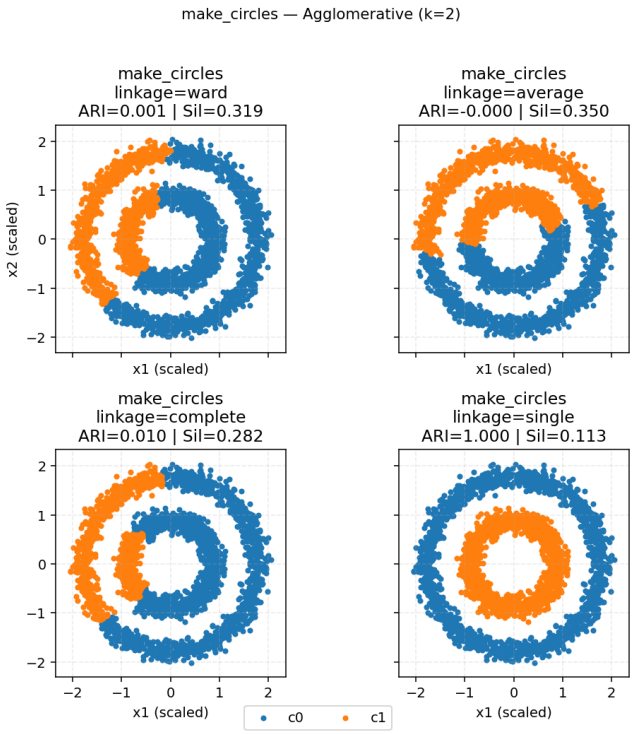


Figure 25: Distribuição dos dados do `make_circles`.

5 Conclusão Final

O **K-Means** apresentou desempenho excelente apenas em cenários compatíveis com suas hipóteses geométricas, como no *make blobs*, onde os clusters são esféricos e bem separados. Em contraste, falhou completamente em estruturas não lineares, como as presentes nos datasets *make moons* e *make circles*. Isso reforça sua limitação a fronteiras lineares e clusters convexos.

O **DBSCAN** mostrou-se mais flexível, sendo capaz de capturar estruturas complexas quando seus parâmetros são bem ajustados. No entanto, sua sensibilidade às escolhas de *eps* e *min_samples* se destacou: pequenas variações podem resultar em clusters excessivamente fragmentados ou na fusão de grupos distintos. Apesar disso, o algoritmo obteve resultados quase perfeitos no *make blobs* e, com o ajuste adequado, também conseguiu separar corretamente os círculos concêntricos do *make circles*.

Já a **Clusterização Hierárquica** apresentou comportamento altamente dependente do método de ligação (*linkage*) utilizado. Enquanto métodos como *ward*, *average* e *complete* funcionaram muito bem em dados esféricos, somente o *single linkage* foi capaz de recuperar a estrutura do *make circles*, mesmo que com baixo *silhouette* devido à própria natureza do formato dos clusters. Esse comportamento evidencia tanto a interpretabilidade quanto a sensibilidade dessa técnica.

De modo geral, os resultados mostram que **não existe um algoritmo universalmente superior**: cada método possui vantagens e limitações que o tornam mais ou menos adequado dependendo da geometria, da distribuição e da densidade dos dados. Assim, a escolha do algoritmo deve sempre considerar a estrutura esperada do problema, acompanhada de uma análise exploratória cuidadosa e da calibragem de hiperparâmetros.

References

[1] A. A. Ribeiro e E. W. Karas, Otimização contínua: aspectos teóricos e computacionais, Cengage Learning, 2013.

[2] https://medium.com/@will.lucena/agrupamento-hierrquico-329e30a9f32d

[3] https://www.datacamp.com/pt/tutorial/hierarchical-clustering

[4] https://www.inf.ufpr.br/diego/IA06.pdf