Universidade do Estado do Rio de Janeiro
Centro de Tecnologia e Ciências
Instituto de Matemática e Estatística

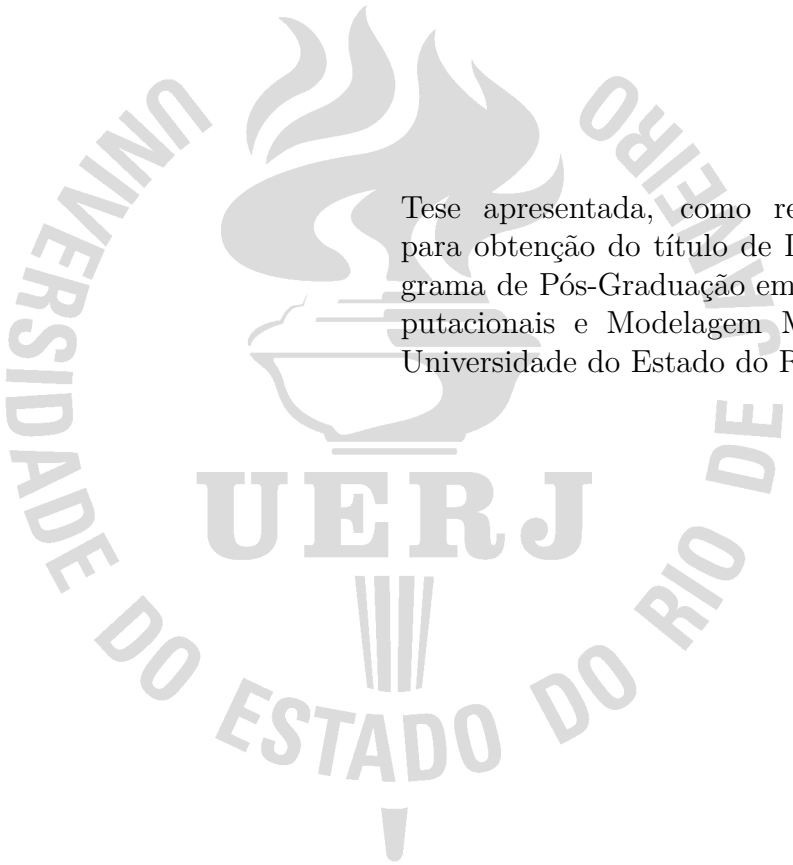
Jorge Luiz de Jesus Goulart

Árvore de Decisão por Agrupamento com DBSCAN Aproximativo

Rio de Janeiro
2024

Jorge Luiz de Jesus Goulart

Árvore de Decisão por Agrupamento com DBSCAN Aproximativo



Tese apresentada, como requisito parcial para obtenção do título de Doutor, ao Programa de Pós-Graduação em Ciências Computacionais e Modelagem Matemática, da Universidade do Estado do Rio de Janeiro.

Orientadores: Prof. Dr. Fabiano de Souza Oliveira
Prof. Dr. Paulo Eustáquio Duarte Pinto

Rio de Janeiro
2024

CATALOGAÇÃO NA FONTE

UERJ/REDE SIRIUS/BIBLIOTECA CTC/A

G694 Goulart, Jorge Luiz de Jesus.
Árvores de decisão por agrupamento com DBSCAN aproximativo/ Jorge Luiz de Jesus Goulart. - 2024.
75 f.: il.

Orientadores: Fabiano de Souza Oliveira, Paulo Eustáquio Duarte Pinto.
Tese (Doutorado em Ciências Computacionais e Modelagem Matemática) - Universidade do Estado do Rio de Janeiro, Instituto de Matemática e Estatística.

1. Aprendizado do computador - Teses. 2. Processo decisório - Teses. 3. Algoritmos - Teses. I. Oliveira, Fabiano de Souza. II. Pinto, Paulo Eustáquio Duarte. III. Universidade do Estado do Rio de Janeiro. Instituto de Matemática e Estatística. IV. Título.

CDU 004

Patricia Bello Meijinhos CRB7/5217 - Bibliotecária responsável pela elaboração da ficha catalográfica

_____ _____

Assinatura Data

Jorge Luiz de Jesus Goulart

Árvore de Decisão por Agrupamento com DBSCAN Aproximativo

Tese apresentada, como requisito parcial para obtenção do título de Doutor, ao Programa de Pós-Graduação em Ciências Computacionais e Modelagem Matemática, da Universidade do Estado do Rio de Janeiro.

Aprovada em 22 de fevereiro de 2024.
Banca Examinadora:

Prof. Dr. Fabiano de Souza Oliveira (Orientador)
Instituto de Matemática e Estatística - UERJ

Prof. Dr. Paulo Eustáquio Duarte Pinto (Orientador)
Instituto de Matemática e Estatística - UERJ

Prof. Dr. Jayme Luiz Szwarcfiter
Instituto de Matemática e Estatística - UERJ

Prof. Dr. Luerbio Faria
Instituto de Matemática e Estatística - UERJ

Prof. Dr. Vinicius Layter Xavier
Instituto de Matemática e Estatística - UERJ

Prof. Dr. Aquiles Braga de Queiroz
Universidade Federal Rural do Rio de Janeiro - UFRRJ

Prof. Dr. Fabio Pereira dos Santos
Universidade Federal do Rio de Janeiro - UFRJ

Rio de Janeiro

2024

AGRADECIMENTOS

É com imensa gratidão que manifesto meus sinceros agradecimentos por esta jornada de aprendizado e descobertas que culmina na conclusão do meu doutorado. Primeiramente, agradeço a Deus por conceder-me força, sabedoria e determinação ao longo deste caminho desafiador. Sua orientação foi fundamental para superar obstáculos e alcançar este marco em minha vida acadêmica.

Aos meus amados pais, Jerônimo e Fátima, e à minha querida esposa, Marina Marques, expresso minha eterna gratidão. O apoio incondicional, amor e compreensão de vocês foram o alicerce que me sustentou nos momentos mais difíceis. Sem o encorajamento e o suporte emocional de vocês, esta conquista não seria possível. Sou profundamente grato por tudo que fizeram por mim.

Aos meus respeitáveis orientadores, Fabiano Oliveira e Paulo Eustáquio, minha admiração e reconhecimento são imensuráveis. Suas orientações precisas, ensinamentos valiosos e incentivo constante foram essenciais para o desenvolvimento do meu trabalho acadêmico. Agradeço pela paciência, dedicação e sabedoria compartilhada ao longo dessa jornada.

À Universidade do Estado do Rio de Janeiro (UERJ), minha gratidão por oferecer um ambiente acadêmico propício ao crescimento intelectual. A UERJ é meu trabalho, meu estudo e minha família. Estou honrado por fazer parte desta instituição referência em ensino e pesquisa.

A todo corpo docente da UERJ, expresso meu profundo agradecimento pela inspiração, conhecimento transmitido e pelo incentivo constante. Cada professor contribuiu significativamente para minha formação, expandindo meus horizontes acadêmicos e intelectuais. Sou grato por cada ensinamento compartilhado e por moldarem meu percurso acadêmico.

À memória do meu amigo, João Felipe, cuja falta é sentida profundamente, expresso um agradecimento carregado de saudade e gratidão. Sua amizade foi um presente valioso em minha vida, iluminando os dias com risos, apoio incondicional e momentos compartilhados. Sua partida deixou um vazio imensurável, porém, seu legado de bondade, lealdade e alegria continuam a ecoar em meu ser. Agradeço por cada conselho, cada riso, cada momento de cumplicidade que tivemos juntos, pois são lembranças que guardo com carinho no meu coração.

Aos meus amados avós, Cacilda, Eduardo e João, cuja presença física não está mais conosco, dedico um agradecimento pleno de amor e admiração. Seus ensinamentos, amor incondicional e valores transmitidos moldaram profundamente minha vida. A sabedoria, paciência e carinho que sempre demonstraram são fontes inesgotáveis de inspiração. Cada lembrança, cada conselho, é um tesouro que guardo com carinho em minhas memórias.

Agradeço por terem sido pilares fundamentais em minha formação, deixando um legado que perdura e orienta meus passos diários. Que suas almas descansem em paz, sabendo que seu impacto em minha vida é eterno e profundamente valorizado.

Por fim, a todos que de alguma maneira contribuíram para esta conquista, meu mais sincero obrigado. Este momento marca não apenas o fim de uma etapa, mas também o início de um novo capítulo. Aprendi, cresci e amadureci ao longo deste percurso e levo comigo lembranças e aprendizados que serão para sempre preciosos.

RESUMO

GOULART, Jorge Luiz de Jesus. *Árvore de Decisão por Agrupamento com DBSCAN Aproximativo*. 2024. 76 f. Tese (Doutorado em Ciências Computacionais e Modelagem Matemática) – Instituto de Matemática e Estatística, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2024.

O aprendizado de máquina tem como objetivo geral criar sistemas que podem aprender com dados, identificar padrões e tomar decisões com o mínimo de intervenção humana. Em geral, métodos de aprendizado de máquina empregam diversos conceitos geométricos, envolvendo pontos no espaço multidimensional. Por outro lado, na literatura de algoritmos para a resolução de conceitos geométricos, há inúmeros algoritmos dessa natureza que envolvem procedimentos randomizados e aproximativos. Tais procedimentos resultam ora em melhoria de complexidade de tempo esperada, ora na simplificação da implementação desses algoritmos. Esta tese apresenta uma variante do modelo de árvore de decisão, na qual novos predicados são considerados para particionar os dados. Ao invés de empregar apenas predicados univariados (associados a uma única característica do dado) como é o caso da árvore de decisão ordinária, a nova variante considera também predicados multi-características. Tais predicados consistem em separar os dados pela pertinência dos mesmos a cada um dos grupos produzidos pelo modelo de agrupamento não-supervisionado DBSCAN. O algoritmo do DBSCAN possui complexidade computacional de tempo elevada para ser diretamente integrado às árvores de decisão. Assim, a proposta consiste no emprego de uma versão aproximativa de tempo linear do algoritmo DBSCAN, para suavizar tal impacto. A tese faz a proposta além de fazer experimentos em diversas bases de referência em aprendizado de máquina.

Palavras-chave: Árvore de Decisão. Algoritmos Randomizados. Aprendizado de Máquina. Algoritmos Aproximativos.

ABSTRACT

GOULART, Jorge Luiz de Jesus. *Cluster Decision Tree with Approximate DBSCAN.* 2024. 76 f. Tese (Doutorado em Ciências Computacionais e Modelagem Matemática) – Instituto de Matemática e Estatística, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2024.

Machine learning aims to create systems that can learn from data, identify patterns, and make decisions with minimal human intervention. Generally, machine learning methods employ various geometric concepts involving points in multidimensional space. On the other hand, in the literature of algorithms for solving geometric concepts, there are numerous algorithms of this nature that involve randomized and approximate procedures. These procedures sometimes result in an improvement in expected time complexity and at other times in simplifying the implementation of these algorithms. This thesis presents a variant of the decision tree model, where new predicates are considered to partition the data. Instead of using only univariate predicates (associated with a single feature of the data), as is the case with the ordinary decision tree, the new variant also considers multi-feature predicates. Such predicates involve separating the data by their relevance to each of the groups produced by the unsupervised clustering model DBSCAN. The DBSCAN algorithm has a high computational time complexity to be directly integrated into decision trees. Therefore, the proposal involves using an approximate version with linear time complexity of the DBSCAN algorithm to mitigate this impact. The thesis not only proposes this but also conducts experiments on various machine learning benchmark datasets.

Keywords: Decision Tree. Randomized Algorithms. Machine Learning. Approximate Algorithms.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

SUMÁRIO

INTRODUÇÃO

O aprendizado de máquina tem como objetivo criar sistemas que podem aprender funções a partir de dados, pelo reconhecimento de padrões nesses dados. Ou seja, é um método de análise de dados que automatiza a construção de modelos analíticos. Formalmente, dados conjuntos $X = \{X_1, \ldots, X_n\}$ e $Y = \{y_1, \ldots, y_n\}$, com $X_i \in \mathcal{X}$ e $y_i \in \mathcal{Y}$ para todo $1 \leq i \leq n$, seja certa função $f : \mathcal{X} \to \mathcal{Y}$ tal que $f(X_i) = y_i$. Um método de aprendizado de máquina tem por objetivo determinar uma função $g : \mathcal{X} \to \mathcal{Y}$ tal que $g \approx f$. O mapeamento de X_i em y_i por g é desejável, mas não é uma restrição, no sentido que o método pode ter dificuldade em mapear todos os pontos, devido à presença de ruídos nos dados de entrada ou pela natureza complexa da função a ser aprendida. Os conjuntos X e Y são chamados de *exemplos de treinamento*. O elemento y_i é chamado de *classe* de X_i. Quando $\mathcal{Y}$ é um conjunto discreto e finito, o método é chamado de *classificação*. Quando $\mathcal{Y}$ é um conjunto contínuo, o método é chamado de *regressão*. Além disso, se o conjunto Y for omitido da entrada, o método é chamado de *não supervisionado*; caso contrário, de *supervisionado*.

A importância do aprendizado de máquina aumentou expressivamente na última década, uma das causas sendo o volume massivo de dados gerados pela tecnologia contemporânea. Com isso, torna-se imprescindível o contínuo avanço no estudo de métodos que baseiam-se em técnicas específicas de aprendizado de máquina.

Um algoritmo é randomizado (FARIA et al., 2021) quando alguma de suas ações depende do resultado de algum evento probabilístico, por exemplo, quando alguma ação deste algoritmo é determinada pelo resultado da geração de um número aleatório. Os algoritmos randomizados deixaram de ser uma ferramenta na teoria computacional dos números e encadearam, nas últimas décadas, um crescente volume de estudos encontrando aplicações generalizadas em diversas áreas e tipos de algoritmos. Esse crescimento é a razão de duas vantagens que a aplicabilidade da técnica pode trazer: eficência e simplicidade. Algoritmos randomizados tendem a ser mais simples de serem implementados que algoritmos determinísticos para resolução de um mesmo problema, além de sua contribuição na redução da complexidade de tempo médio de execução do algoritmo (FARIA et al., 2021).

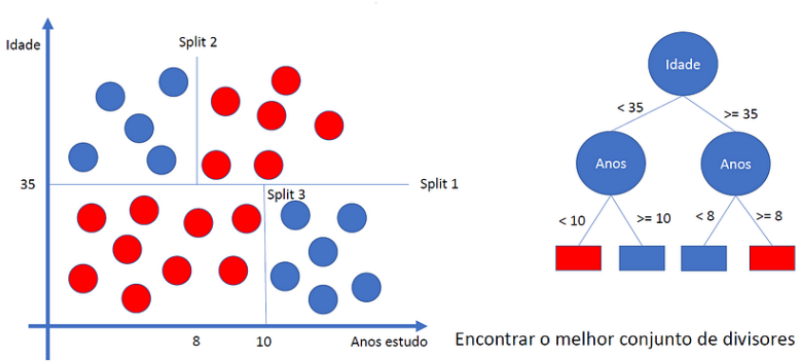
Em contrapartida aos algoritmos determinísticos, que sempre solucionam de maneira exata e em tempo previsível, algoritmos randomizados produzem incertezas durante o processo, sejam elas referentes ao tempo de execução ou à correção dos resultados encontrados. Um algoritmo randomizado pode ser mais rápido e, no entanto, nem sempre retornar a resposta correta. A resposta correta, neste caso, estará associada a uma determinada probabilidade. Alternativamente, o algoritmo pode sempre dar a resposta correta e a incerteza estar associada ao tempo de execução. Estes dois tipos de algoritmos ran-

domizados são respectivamente denominados *Algoritmo de Monte Carlo* e *Algoritmo de Las Vegas*, e serão melhor descritos adiante. O crescente interesse no estudo de algoritmos randomizados, espalhou-se por áreas da computação (FIGUEIREDO et al., 2007) como: computação algébrica, criptografia, geometria computacional, protocolo de redes, computação distribuída, teoria dos grafos, estrutura de dados, entre outros.

Por outro lado, há também a classe dos algoritmos aproximativos que, em linhas gerais, são algoritmos que produzem soluções cuja proximidade da solução ótima está dentro de um limite garantido. Esses algoritmos surgiram em resposta à impossibilidade (na suposição que $P \neq NP$) de resolver em tempo satisfatório inúmeros problemas de otimização NP-difíceis. Sendo assim, passa a ser razoável abrir mão da otimalidade em troca de uma solução computacionalmente viável para o problema em questão com alguma garantia de quão distante a solução obtida está da ótima. Tal compromisso, entre perda de otimalidade e ganho em eficiência, representa o paradigma dos algoritmos aproximativos. Há algumas técnicas que são comumente usadas em algoritmos aproximativos, são elas: Métodos Combinatórios, Programação Linear, Programação Semidefinida, Algoritmos Probabilísticos, Relaxação Lagrangiana, entre outras.

Muitos métodos de aprendizado de máquina possuem forte apelo geométrico em sua solução. As árvores de decisão (que serão formalmente apresentadas na Seção 1.1.1.1), por exemplo, enxergam os dados de entrada como pontos de um espaço multidimensional e tentam dividir o espaço de entrada em regiões menores, com a premissa implícita que pontos dentro de certas regiões definidas geometricamente são classificados da mesma maneira. O método assume que, ao dividir em regiões menores, identificará regiões nas quais todos os pontos (ou uma quantidade significativa deles) estão associados a somente uma classe. A Figura 1 é um exemplo gráfico da divisão do espaço após uma aplicação da árvore de decisão.

Figura 1 - Divisão do espaço em uma árvore de decisão

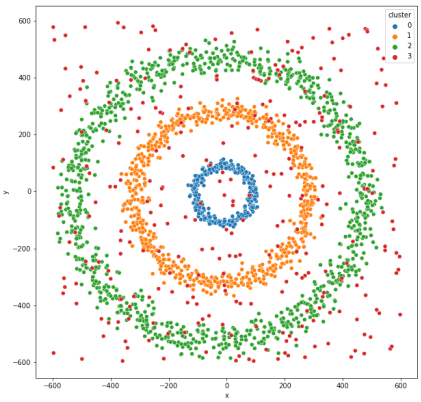


Fonte: https://medium.com/@msremigio/.

No exemplo da Figura 1, as divisões utilizadas pela árvore de decisão foram pararelas aos eixos cartesianos. No entanto, nem sempre essa divisão poderá ser eficaz em

produzir uma divisão concisa. Em comparação, a Figura 2 apresenta padrões de pontos em que o método de tentar definir regiões através de hiperplanos perpendiculares aos eixos não produz uma árvore de decisão pequena.

Figura 2 - Exemplo de agrupamentos em que as metodologias de árvore de decisão univariada e multivariada podem não encontrar uma árvore pequena.



Fonte: https://anderfernandez.com/blog/dbscan-python/

Ao longo do avanço do estudo relacionado às árvores de decisão, novos padrões de divisão das regiões foram propostos, inclusive com a aplicação de métodos não supervisionados de agrupamento. Aplicações com SVM e k-médias já foram propostas e implementados junto às árvores de decisão (KUMAR; GOPAL, 2010; WANG et al., 2020).

Mais especificamente, abordando os métodos não supervisionados, sabe-se que o DBSCAN possui a vantagem de encontrar uma grande variedade de formas de agrupamento, principalmente porque o algoritmo baseia-se no conceito de densidade. O agrupamento k-médias, em comparação, encontra geralmente agrupamentos com formas esféricas. Por outro lado, o DBSCAN possui uma complexidade de tempo alta. É nesta deficiência que foi proposto o DBSCAN aproximativo (GAN; TAO, 2015) e que resolve uma variante relaxada do problema original com complexidade esperada de tempo linear. Sendo assim, este trabalho propõe utilizar tal variante aproximativa do método não supervisionado DBSCAN junto às árvores de decisão, para ser mais uma opção da árvore de decisão no particionamento do espaço em regiões menores. O DBSCAN aproximativo será melhor descrito nos próximos capítulos.

Este trabalho está organizado em quatro capítulos. No Capítulo 1, apresentam-se de forma mais abrangente os conceitos de aprendizado de máquina, geometria computacional e algoritmos randomizados e aproximativos. Também apresentam-se no primeiro capítulo alguns métodos e algoritmos que motivaram esta tese, além de metodologias para quantificar a qualidade de um modelo de classificação encontrado. No Capítulo 2, descrevem-se alguns modelos de árvore de decisão bem como os estudos que serviram de referência e base para este trabalho. No Capítulo 3, apresenta-se o modelo proposto da

árvore de decisão junto à aplicação do DBSCAN aproximativo. No Capítulo 4, descrevem-se as bases utilizadas para teste e apresentam-se os resultados encontrados deste trabalho. Por fim, apresentam-se a conclusão deste trabalho e as propostas para trabalhos futuros.

1 REFERENCIAL TEÓRICO

Neste capítulo, apresentam-se os conceitos de aprendizado de máquina, geometria computacional e algoritmos randomizados.

1.1 Aprendizagem de Máquina

Os *dados* são caracterizados como fatos registrados e a *informação* é o conjunto de padrões ou relações derivadas dos dados. A *Mineração de Dados* é a extração dessas informações ainda subjacentes aos dados e, para tal, necessita-se elaborar algoritmos para encontrar padrões e regularidades investigando automaticamente o conjunto de dados. O aprendizado de máquina, em geral, permite a descrição dos padrões (observados nos dados) que posteriormente será útil para previsão, explicação e compreensão do fenômeno estudado.

Ao reconhecer padrões, o objetivo não é criar um procedimento capaz de identificar o processo em estudo completamente, mas construir uma aproximação boa e útil da realidade. Sendo assim, o aprendizado de máquinas busca detectar certos padrões ou regularidades para entender o processo ou para realizar previsões.

Uma forma específica de descrever uma função a ser aprendida será chamada de *modelo*, que está associada a um algoritmo específico para calibração de parâmetros presentes nesse modelo.

Alguns tipos de aprendizado de máquina são: aprendizado supervisionado, não-supervisionado e semi-supervisionado. A seguir, apresenta-se uma breve introdução de cada um dos tipos de aprendizado.

1.1.1 Aprendizado Supervisionado

O aprendizado supervisionado é aquele em que é fornecido ao algoritmo um conjunto de dados $T = \{(x_i, y_i) \mid 1 \leq i \leq n\}$, chamados de *exemplos de treinamento*, para os quais há um valor y_i associado a cada dado x_i chamado de *classe* ou *rótulo* (MONARD; BARANAUSKAS, 2003). Em geral, cada exemplo de treinamento x_i é representado por um vetor de valores chamados de *características* ou *atributos* e a classe associada. O objetivo do aprendizado de máquina é construir um classificador adequado que possa determinar, com a maior acurácia possível, a classe de novos exemplos que necessitam de classificação. Para classes de domínio discretos, esse problema é conhecido como *classificação*, enquanto que para classes de domínio contínuos como *regressão*.

Alguns métodos de aprendizado supervisionado são: regressão linear, regressão logística, máquinas de vetores de suporte, árvore de decisão, entre outros.

1.1.1.1 Árvore de Decisão

Nesta seção, apresenta-se o modelo de aprendizado supervisionado chamado de Árvore de Decisão. O texto segue a organização e apresentação de FARIA et al. (2021).

Alguns modelos de classificação se limitam ao mapeamento dos dados de treinamento em apenas 2 classes possíveis. No entanto, as árvores de decisão pertencem a um grupo de modelos que não possuem tal restrição.

Em seguida, apresenta-se a construção de uma árvore de decisão para problemas de classificação. Sejam um conjunto de classes C, um conjunto $Y = \{y_1, y_2, ..., y_n\}$ com cada $y_i \in C$, um conjunto de dados

$$X = \{X_1, X_2, ..., X_n\}$$

onde cada dado X_i é representado por um vetor d-dimensional $X_i = (x_{i1}, x_{i2}, x_{id})$ e associado à classe y_i. Os conjuntos X e Y representam os exemplos de treinamento. Cada uma das d posições de cada vetor X_i é chamada de uma *característica* dos dados de X. Os valores correspondentes à j-ésima característica de X consistem de $x_{1j}, x_{2j}, ..., x_{nj}$.

É importante ressaltar que, mesmo que as características de cada vetor X_i possam ter valores numéricos ou categóricos, o modelo trabalhará sem que seja necessário transformar valores categóricos em numéricos. Isto torna-se um fator importante, principalmente em características categóricas em que não seja direta a conversão numérica.

A técnica geral da árvore é criar uma árvore binária enraizada T e empregá-la para classificar novos dados. Para cada nó de T, associam-se informações que dependem da natureza desse nó, conforme os abaixo:

- **Nó Folha**: associa-se a esse nó uma classe específica;
- **Nó Interno**: associa-se a esse nó um predicado (condição verificável) sobre exatamente uma das d características dos dados.

A classificação de certo dado d-dimensional Z segue da seguinte forma. A partir da raiz de T, percorre-se um caminho até uma folha. O caminho exato a ser percorrido é determinado pelos predicados pertencentes aos nós internos de T. Quando atinge-se uma folha, o processo termina, e classifica-se o dado Z conforme a classe encontrada na respectiva folha. O procedimento descrito pode ser implementado por um algoritmo recursivo, como apresentado pelo Algoritmo 1.

Algoritmo 1 - Classificação por árvore de decisão T do dado Z.

```
FUNÇÃO CLASSIFICAR (T, Z)
    se raiz de T é folha então
    |   retornar classe associada ao nó T
    senão
    |   seja π o predicado associado ao nó T
    |   se Z satisfaz π então
    |   |   retornar Classificar(T.Esq, Z)
    |   senão
    |   |   retornar Classificar(T.Dir, Z)
```

Inúmeras árvores de decisão podem ser consideradas, variando a árvore binária quanto às informações associadas aos nós dessa árvore. No entanto, todas elas podem ser produzidas de forma geral pelo processo de construção de árvores de decisão, conforme Algoritmo 2.

Algoritmo 2 - Construção de Árvore de Decisão T associada aos dados X.

```
FUNÇÃO CONSTRUIR (X = {X_1, X_2, ..., X_n})
    seja T um novo nó
    y ← ObterClassificaçãoÚnica(X)
    se y ≠ ∅ então
    |   associar a classe y ao nó T
    senão
    |   melhorPontuação ← −∞
    |   para j ∈ {1, ..., d} fazer
    |   |   se X não possui todos os valores iguais característica j então
    |   |   |   (π, pontuação) ← EscolherPredicado(X, j)
    |   |   |   se pontuação > melhorPontuação então
    |   |   |   |   melhorPontuação ← pontuação
    |   |   |   |   π* ← π
    |   associar o predicado π* ao nó T
    |   X_S ← {X_i ∈ X | X_i satisfaz π*}
    |   X_N ← X \X_S
    |   T.Esq ← Construir(X_S)
    |   T.Dir ← Construir(X_N)
    retornar T
```

O algoritmo recebe como entrada o conjunto $X = \{X_1, X_2, ..., X_n\}$ dos exemplos de treinamento e produz uma árvore de decisão T associada a esses dados. Primeiramente, o algoritmo deve decidir se T será uma árvore que classifica qualquer entrada de uma única maneira e, em caso afirmativo, qual seria essa classe. Essa é a responsabilidade da função ObterClassificaçãoÚnica(X) que, ou retorna a classe a ser usada em toda classificação, ou

retorna $\emptyset$, no caso da necessidade de diferentes classes. Em geral, o segundo caso é mais comum para um conjunto de treinamento. No entanto, para um subconjunto de X, pode ser adequado utilizar o primeiro caso.

No primeiro caso, a decisão é única, ou seja, a classificação se dará pelo único nó raiz e folha de T. No segundo caso, particionam-se os dados com o objetivo de tornar cada subconjunto após a partição o mais homogêneo possível em termos de classificação. Sendo assim, cada predicado abordará uma determinada característica que irá particionar o conjunto de treinamento em dois subconjuntos não vazios. Busca-se um predicado que faça a melhor divisão dos dados naquele nó da árvore. Esse é o objetivo da função EscolherPredicado(X, j). Há algumas métricas que dirão o quão um determinado predicado é adequado para o particionamento do conjunto de treinamento em um determinado nó da árvore.

Após a escolha do melhor predicado, a ideia é produzir duas árvores de decisão recursivamente, uma treinada com os exemplos que satisfazem o predicado, e a outra com os exemplos que não satisfazem. Particiona-se X em X_S (satisfazem o predicado) e X_N (não satisfazem o predicado). Recursivamente, determinam-se as duas árvores de decisão T_S e T_N treinadas com respectivamente os exemplos de treinamento X_S e X_N, definindo T_S como subárvore esquerda de T, e T_N como subárvore direita de T.

Por fim, cabe ressaltar pontos importantes das funções ObterClassificaçãoÚnica e EscolherPredicado. Há casos em que a função ObterClassificaçãoÚnica decidirá retornar uma única classe, mesmo que os dados de X não tenham a mesma classificação. Já a função EscolherPredicado, responsável por escolher o melhor predicado dada uma característica, também é responsável por fornecer uma pontuação acerca da escolha deste predicado. A pontuação indicará o quanto de incerteza é reduzida da previsão com o uso do predicado escolhido. As métricas e metodologias para decidir sobre a classificação única e esta pontuação serão descritas a seguir.

Pureza dos Dados: Uma classificação *pura* é aquela em que todos os dados X_i estão associados a uma única classe $y \in C$. Sendo assim, a função ObterClassificaçãoÚnica deve observar se a classificação de todos os dados de treinamento é pura. No entanto, após alguns particionamentos do conjunto de treinamento, os dados obtidos em um determinado nó podem ser em pouca quantidade, sem uma definição quanto à classificação. Para este problema, pode-se forçar que uma previsão seja determinada pelo algoritmo quando a quantidade de dados se torne abaixo de um certo limite n_{min}.

Outro problema que pode-se encontrar é que o grau de pureza seja alto, mesmo que tenha-se uma grande quantidade de exemplos para o particionamento em um determinado nó. Sendo assim, não se justifica a criação de novos predicados, principalmente pelo aumento do custo de processamento sem nenhum ganho significativo em qualidade de previsão. Para resolver tal problema, quantifica-se a pureza $P(X)$ de um conjunto de dados X, dada por

$$P(X) = \frac{\max_{y \in C}\{|\{X_i \in X \mid y_i = y\}|\}}{|X|}$$

que representa a proporção dos dados que são classificados igualmente como y_{max}, onde y_{max} denota a classificação que ocorre com mais frequência em X. A estratégia acima é apresentada no Algoritmo 3.

Algoritmo 3 - Classificação única dos dados X, quando conveniente.

FUNÇÃO OBTERCLASSIFICAÇÃOÚNICA ($X = \{X_1, X_2, ..., X_n\}$)
    seja $0 \leq P_{max} \leq 1$ um limiar de pureza
    seja n_{min} um limiar de quantidade de dados
    se $n \leq n_{min}$ ou $P(X) \geq P_{max}$ **então**
    | $y_{max} \leftarrow \text{argmax}_{y \in C}\{|\{X_i \in X | y_i = y\}|\}$
    | **retornar** y_{max}
    senão
    | **retornar** $\emptyset$

Escolha de Predicados: Apresenta-se então o Algoritmo 4 para a função EscolherPredicado, com parâmetros X e j. Dados um conjunto X de vetores d-dimensionais e um natural $1 \leq j \leq d$, a função escolherá um predicado associado à característica de índice j. Portanto, tal predicado deve descrever alguma condição que pode ou não ser satisfeita por cada valor $x_{1j}, x_{2j}, ..., x_{nj}$.

No algoritmo, Π_j denota o conjunto de todos os predicados que serão considerados nesse processo de escolha. Sendo assim, o algoritmo irá procurar qual predicado ($\pi \in \Pi_j$) possui melhor pontuação.

A pontuação, do predicado escolhido π, será definida pela função Ganho(X, π).

Algoritmo 4 - Determinando predicado π^* de maior ganho associado aos dados X e característica j.

FUNÇÃO ESCOLHERPREDICADO ($X = \{X_1, X_2, ..., X_n\}, j \in \{1, ..., d\}$)
    Seja Π_j o conjunto de predicados considerados para a característica j.
    MelhorPontuação $\leftarrow -\infty$
    para $\pi \in \Pi_j$ **fazer**
    | pontuação $\leftarrow$ Ganho(X, π)
    | **se** pontuação $>$ melhorPontuação **então**
    | | melhorPontuação $\leftarrow$ pontuação
    | | $\pi^* \leftarrow \pi$
    retornar π^*, melhorPontuação

Apresenta-se a seguir, a definição de Π_j, a estimativa de pontuação dada pela função Ganho(X, π) e os detalhes para a escolha do predicado. Serão consideradas variáveis categóricas e numéricas.

Predicado para Característica Categórica: Uma característica categórica pode ser interpretada como uma escolha dentre um conjunto de valores possíveis. De maneira geral, seja $Dom(j)$ o conjunto de todos os valores existentes da característica j no conjunto de dados, isto é,

$$Dom(j) = \{x_{ij} \mid 1 \leq i \leq n\}$$

O predicado considerado para uma certa característica categórica j será do tipo

$$\text{“}x \in S\text{”}$$

onde $S \subset Dom(j)$ e $S \neq \emptyset$. Em outras palavras, o predicado π associado a uma característica categórica consiste em escolher certo subconjunto próprio não-vazio S de $Dom(j)$ e definir que certo dado $x = (x_1, ..., x_d)$ satisfaz π se $x_j \in S$.

A questão que se levanta imediatamente é quantos subconjuntos S devem ser avaliados. Como o número de subconjuntos de um dado conjunto é exponencial na cardinalidade desse conjunto, deve-se abrir mão de tentar exaustivamente avaliar todos os subconjuntos. Ao invés, avaliam-se todos aqueles que atendam a uma propriedade restritiva. Comumente, emprega-se a restrição de que os subconjuntos S devem ser unitários, decorrendo que há exatamente $|Dom(j)|$ subconjuntos a serem analisados, o que normalmente não é proibitivo (FARIA et al., 2021).

Como conclusão, quando j representa uma característica categórica, tem-se que

$$\Pi_j = \{\text{“}x \in S\text{”} \mid S \subset Dom(j), |S| = 1\}$$

Predicado para Característica Numérica O predicado para características numéricas tem por objetivo particionar o domínio de valores naqueles que estão abaixo e acima e um determinado ponto de corte.

Seja $D = d_1, d_2, ..., d_{|Dom(j)|}$ a sequência ordenada dos elementos de $Dom(j)$ e considere a sequência $M = m_1, m_2, ..., m_{|Dom(j)|-1}$, também ordenada, dos pontos médios dos elementos consecutivos de D, isto é,

$$m_i = \frac{d_i + d_{i+1}}{2} \text{ , para todo } 1 \leq i < |Dom(j)|$$

O predicado considerado para uma certa característica numérica j será do tipo,

$$\text{“}x \leq m_i\text{”}$$

para todo $1 \leq i < |Dom(j)|$. Sendo assim, define-se que um dado $x = (x_1, ..., x_d)$ satisfaz π se $x_j \leq m_i$. Claramente, o número de predicados a serem considerados é exatamente $|Dom(j)| - 1$. Logo, tem-se que

$$\Pi_j = \{\text{“}x \leq m_i\text{”} \mid 1 \leq i < |Dom(j)|\}$$

Entropia dos Dados: Mostra-se agora como calcular a pontuação de um predicado, ou seja, como elaborar a função Ganho(X, π) empregada no Algoritmo 4.

Como dito anteriormente, uma pontuação mais alta de um predicado relaciona-se diretamente com a capacidade deste predicado em dividir o conjunto de dados em subconjuntos mais homogêneos, ou seja, quanto mais uniformes os subconjuntos gerados, maior a pontuação recebida por este predicado. Para tal, recorre-se a um conceito da teoria da informação, normalmente associado à medida de desordem em um sistema. Dado um conjunto de dados $X = \{X_1, ..., X_n\}$, cada X_i associado à classificação $y_i \in C$, define-se a entropia dos dados X ao valor

$$H(X) = -\sum_{y \in C} p_X(y) \log_2 p_X(y)$$

onde $p_X(y)$ corresponde à probabilidade com que um dado X_i escolhido aleatoriamente seja da classe y, isto é,

$$p_X(y) = \frac{|\{X_i \in X \mid y_i = y\}|}{|X|}$$

Note que se X tem pureza absoluta, isto é, todos os dados estão classificados igualmente como y, temos que $p_X(y) = 1$, e portanto $H(X) = 0$, o que corresponde à entropia mínima de qualquer conjunto de dados. Por outro lado, se todas as $|C|$ classes aparecem com a mesma frequência em X, temos $p_X(y) = \frac{1}{|C|}$ para todo $y \in C$ e

$$H(X) = -\sum_{y \in C} p_X(y) \log_2 p_X(y) = -\sum_{y \in C} \frac{1}{|C|} \log_2 \frac{1}{|C|} = \log_2 |C|$$

Empregando o conceito de entropia dos dados, define-se que a entropia de um separador π com respeito a certo conjunto de dados X, denotada por H(X, π), é dada por

$$H(X, \pi) = \frac{|X_S|}{|X|} H(X_S) + \frac{|X_N|}{X} H(X_N)$$

onde X_S e X_N correspondem aos subconjuntos de X que respectivamente satisfazem e não satisfazem o predicado π, isto é,

$$X_S = \{X_i \in X \mid X_i \text{ satisfaz } \pi\}$$

$$X_N = \{X_i \in X \mid X_i \text{ não satisfaz } \pi\}$$

Finalmente, define-se a pontuação de um predicado pela diferença que ele produz na entropia dos dados quando aplicado, da seguinte forma:

$$Ganho(X, \pi) = H(X) - H(X, \pi)$$

Note que, intuitivamente, $Ganho(X, \pi)$ determina o quanto de incerteza quanto à classificação é diminuída se o predicado π for empregado para particionar X.

1.1.2 Aprendizado Não-Supervisionado

O aprendizado não supervisionado é aquele em que é fornecido ao algoritmo um conjunto de exemplos de treinamento para os quais não há classes associadas. Sendo assim, este aprendizado analisa os exemplos fornecidos e tenta determinar se alguns deles podem ser agrupados por alguma métrica de similaridade. Tal processo, após a determinação dos agrupamentos, normalmente, requer uma análise para determinar o que cada agrupamento significa no contexto do problema que está sendo analisado.

Alguns métodos de aprendizado não-supervisionado são: k-médias, DBSCAN, Análise de Componentes Principais (PCA), entre outros.

Nesta seção, o algoritmo DBSCAN será descrito como exemplo de um método de agrupamento. Ele será relevante para a proposta desse trabalho.

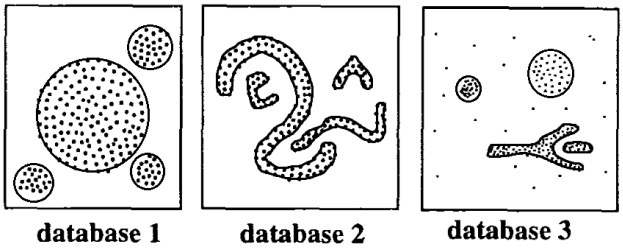
1.1.2.1 DBSCAN exato

O algoritmo DBSCAN (*Density-Based Spatial Clustering of Applications with Noise*) é um método de agrupamento bastante conhecido. Inicialmente foi suposto que o algoritmo DBSCAN proposto possuía complexidade de tempo $O(n \log n)$ (ESTER et al., 1998). No entanto, verificou-se posteriormente que tal algoritmo possuía complexidade $O(n^2)$, e que, no caso particular de duas dimensões, sua complexidade era de $O(n \log n)$.

A ideia original do algoritmo é apresentada: dado um conjunto P de n pontos no espaço d-dimensional $\mathbb{R}^d$, tem-se por objetivo agrupar os pontos de P em subconjuntos de modo que quaisquer dois grupos sejam separados por regiões dispersas. Na Figura 3, há 3 exemplos de dados de entrada. Cada dado é representado por um ponto. Os pontos encontram-se agrupados em linhas envolvendo os pontos que pertencem a cada um dos grupos, que formam um suposto agrupamento desejável de ser encontrado pelo algoritmo. Note que é possível que um determinado ponto não faça parte de nenhum grupo, como é o caso dos pontos fora de qualquer grupo no database 3. Isso visa garantir que o modelo possa reconhecer ruídos presentes nos dados, e não crie mais grupos que o necessário. Em cada método de agrupamento é definida uma métrica de dispersão de interesse.

Cabe ressaltar que, apesar de sua complexidade computacional mais elevada, o

Figura 3 - Exemplificação de agrupamento utilizando o DBSCAN.



Fonte: (GAN; TAO, 2015).

DBSCAN possui uma grande vantagem em relação a outros metodos (como por exemplo, k-médias), pois possui a capacidade de descobrir grupos com formas arbitrárias. Em contraste, k-médias retorna grupos semelhantes a esferas disjuntas. Há inúmeras abordagens para aplicar o agrupamento por densidade, que diferem basicamente na definição de suas regiões dispersas e critérios de como essas regiões devem ser conectadas para formar os agrupamentos.

O algoritmo básico do DBSCAN requer dois parâmetros de entrada:

- ϵ: um valor real positivo.
- $MinPts$: um valor natural.

Define-se $B(p, \epsilon)$ como a bola d-dimensional centrada no ponto p com raio ϵ. A distância utilizada, neste caso, é a Euclidiana. Seja $p_i \in P$. A bola $B(p_i, \epsilon)$ será considerada *densa* se cobrir pelo menos $MinPts$ pontos de P, incluindo p_i.

Forma-se então grupos da seguinte forma. Se $B(p_i, \epsilon)$ é denso, então todos os pontos em B devem ser adicionados ao mesmo grupo. Isso isgnifica que se $B(p_i, \epsilon)$ e $B(p_j, \epsilon)$ são ambos densos e possuem algum ponto em comum, então todos os pontos em $B(p_i, \epsilon)$ e $B(p_j, \epsilon)$ devem pertencer ao mesmo grupo.

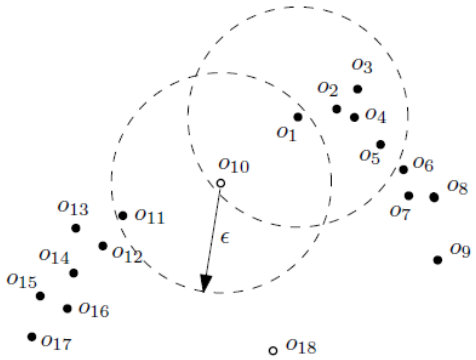
Um ponto $p \in P$ é um ponto *central* se $B(p, \epsilon)$ é denso. A Figura 4 exemplifica a definição de ponto central. Todos os pontos centrais são definidos pelos pontos completamente preenchidos, e os pontos O_{10} e O_{18} são denominados *pontos não centrais*.

Um ponto $q \in P$ é *alcançável em densidade* a partir de $p \in P$ se houver uma sequência de pontos $p_1, p_2, p_3, ..., p_t \in P$ (para algum inteiro $t \geq 2$) tal que:

- $p_1 = p$ e $p_t = q$;
- $p_1, p_2, p_3, ..., p_{t-1}$ são pontos centrais;
- $p_{i+1} \in B(p_i, \epsilon)$ para cada $i \in \{1, \ldots, t-1\}$.

Sendo assim, para exemplificação, na Figura 4, o ponto O_{10} é alcançável por densidade por O_3. Por outro lado, O_{11} não é alcancável por densidade por O_3.

Figura 4 - Exemplificação de ponto central e não central. $MinPts = 4$ e ϵ como na imagem



Fonte: (GAN; TAO, 2015).

No agrupamento produzido pelo DBSCAN, um grupo C é um subconjunto não vazio de P tal que:

- **Maximalidade:** Se $p \in C$ é um ponto central, então todos os pontos alcançáveis por densidade de p também pertencem a C.
- **Conectividade:** Para qualquer ponto $p_1 \in C$, existe um ponto central $p \in C$ tal que p_1 é alcançável por densidade a partir de p.

A definição anterior implica que cada grupo contém pelo menos um ponto central. Outra implicação é que o problema DBSCAN encontrará um único agrupamento de P (ESTER et al., 1998). Além disso, define-se que qualquer ponto não central p em um grupo é denominado *ponto de fronteira*.

Considerando novamente a Figura 4, o agrupamento obtido é o conjunto $\{C_1, C_2\}$ tal que:

$$C_1 = \{O_1, O_2, O_3, ...O_{10}\}$$

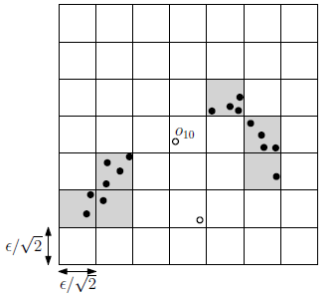
$$C_2 = \{O_{10}, O_{11}, O_{12}, ..., O_{17}\}$$

Observe que:

- Um grupo pode conter pontos centrais e não centrais;
- Todo ponto central pertencerá a um único grupo;
- Um ponto de fronteira pode pertencer a mais de um grupo;
- Pode-se ter pontos que não pertencem a grupo algum; tais pontos serão chamados de *ruídos*.

Em seguida, apresenta-se um algoritmo que encontra o agrupamento do DBSCAN com as características enunciadas, para 2 dimensões.

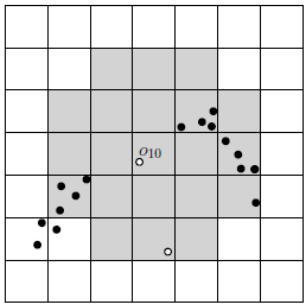
Figura 5 - Grade imposta aos dados da Figura 4



Fonte: (GAN; TAO, 2015).

DBSCAN para 2 dimensões: Considere uma grade arbitrária T no espaço $\mathbb{R}^2$, onde cada célula de T é um quadrado de lado $\frac{\epsilon}{\sqrt{2}}$, exemplificado na Figura 5. Com as especificações descritas, tem-se que quaisquer dois pontos pertencentes a uma mesma célula estão no máximo a uma distância ϵ. Além disso, dada uma célula c, uma célula c' é dita ser ϵ-*vizinha* a c se há pontos do plano $p \in c$ e $p' \in c'$ tais que a distância entre eles seja no máximo ϵ. Veja a Figura 6 para a ilustração do conceito de células vizinhas.

Figura 6 - Células vizinhas à celula σ_{10}



Fonte: (GAN; TAO, 2015).

A grade pode ser construída computacionalmente, levando-se em conta que somente células que contém pontos de P devem ser representadas. Ela é construída conforme algoritmo a seguir:

Algoritmo 5 - Algoritmo para Construção da Grade. Complexidade: $O(n + nCelula)$.

Entrada: Conjunto $P = \{p_1, \ldots, p_n\}$, ϵ e $MinPts$.
FUNÇÃO CONSTRUIRGRADE (P, ϵ, $MinPts$)
    $LarguraCelula \leftarrow \frac{\epsilon}{\sqrt{2}}$
    $minX \leftarrow \min\{p_i.x \mid 1 \leq i \leq n\}$
    $maxX \leftarrow \max\{p_i.x \mid 1 \leq i \leq n\}$
    $minY \leftarrow \min\{p_i.y \mid 1 \leq i \leq n\}$
    $maxY \leftarrow \max\{p_i.y \mid 1 \leq i \leq n\}$
    $nLinhas \leftarrow \lfloor \frac{maxY - minY}{LarguraCelula} + 1 \rfloor$
    $nColunas \leftarrow \lfloor \frac{maxX - minX}{LarguraCelula} + 1 \rfloor$
    $nCelula \leftarrow nLinhas \times nColunas$
    Inicializa uma grade vazia com tamanho $nCelula$
    para cada ponto $p \in P$ **fazer**
    | Adiciona p na célula de índice $(\lfloor \frac{p.x - minX}{LarguraCelula} + 1 \rfloor, \lfloor \frac{p.y - minY}{LarguraCelula} + 1 \rfloor)$
    retornar Grade
Saída: Grade com células de lado $\frac{\epsilon}{\sqrt{2}}$.

O Algoritmo 5 possui complexidade $O(n+nCelula)$ em tempo de pior caso usando-se uma tabela de acesso direto. Se $nCelula$ for muito grande, pode-se implementar em tempo médio $O(n)$ empregando um tabela de dispersão de tamanho n.

Algoritmo 6 - Classificação - Pontos Centrais e Não Centrais: $O(n.MinPts)$.

Entrada: Conjunto P com n pontos, ϵ e $MinPts$.
para cada célula não vazia c da grade **fazer**
| **se** $|c| > MinPts$ **então**
| | Classifique todos os pontos $p \in c$ como centrais
| **senão**
| | **para** cada ponto $p \in c$ **fazer**
| | | $nPontos \leftarrow |c|$
| | | **para** cada célula nc ϵ-vizinha de c **fazer**
| | | | **para** cada ponto $q \in nc$ **fazer**
| | | | | **se** $dist(p, q) \leq \epsilon$ **então**
| | | | | | $nPontos \leftarrow nPontos + 1$
| | | | | | **se** $nPontos \geq MinPts$ **então**
| | | | | | | Classifique p como ponto central
| | | | | | | Encerrar Laço
| | | | **se** $nPontos \geq MinPts$ **então**
| | | | | Encerrar Laço
Saída: Todos os pontos classificados.

Em seguida o algoritmo inicia um processo de rotulagem para decidir para cada ponto $p \in P$ se p é ponto central ou não. Diz-se que uma célula c é *central*, se possuir pelo menos um ponto central. Além disso, como em cada célula a maior distância entre

um ponto e outro é ϵ, se uma célula possuir $MinPts$ ou mais pontos, então todos os pontos desta célula serão pontos centrais. O algoritmo de rotulagem dá-se conforme o Algoritmo 6. Denote por $|c|$ o número de pontos em determinada célula c e por $dist(p, q)$ a distância euclidiana dos pontos p e q.

A complexidade de tempo do Algoritmo 6 é $O(MinPts.n)$ e justifica-se a seguir. Sejam C_1 o conjunto de células c com $|c| > MinPts$, C_2 o conjunto de células com $|c| \leq MinPts$ e $N_\epsilon(c)$ como o conjunto de células ϵ-vizinhas de c. Para cada célula $c \in C_1$ gasta-se $O(|c|)$ tempo. Para cada célula $c \in C_2$ gasta-se $\sum_{c' \in N_\epsilon(c)} O(|c'|)$ tempo para cada ponto em c, então no total temos: $\sum_{c' \in N_\epsilon(c)} O(|c||c'|)$. Sendo assim, o tempo total é dado por

$$\sum_{c \in C_1} O(|c|) + \sum_{c \in C_2} \sum_{c' \in N_\epsilon(c)} O(|c||c'|) \leq O(n) + O(MinPts). \sum_{c \in C_2} \sum_{c' \in N_\epsilon(c)} O(|c'|)$$

Considerando uma célula fixa c', nota-se que c' só será contada para células c que satisfaçam $c' \in N_\epsilon(c)$ que equivale a $c \in N_\epsilon(c')$. Portanto, temos que

$$O(n) + O(MinPts). \sum_{c \in C_2} \sum_{c' \in N_\epsilon(c)} O(|c'|) \leq O(n) + O(MinPts). \sum_{c' \in C_1 \cup C_2} \sum_{c \in N_\epsilon(c')} O(|c'|)$$

$$\leq O(n) + O(MinPts). \sum_{c' \in C_1 \cup C_2} O(|c'|).|N_\epsilon(c)|$$

$$\leq O(n) + O(MinPts).O(n)$$

$$\leq O(MinPts.n)$$

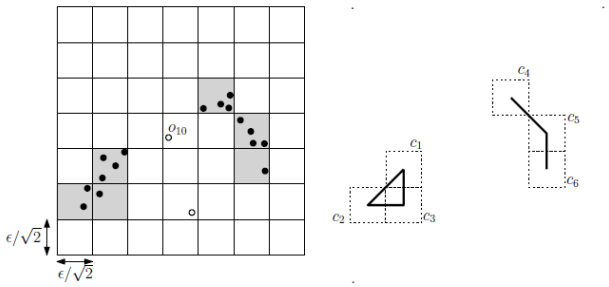
O próximo ponto no método consiste em criar um grafo especial $G = (V, E)$. Define-se esse grafo da seguinte maneira:

- Toda célula central será um vértice em V;
- Dadas duas células centrais c_1 e c_2, $(c_1, c_2) \in E$ se e somente se existir pontos centrais $p_1 \in c_1$ e $p_2 \in c_2$ tais que $dist(p_1, p_2) \leq \epsilon$.

A motivação para a definição desse grafo é que as suas componentes conexas definirão exatamente os grupos encontrados. A Figura 7 mostra exatamente a construção do grafo após a inclusão de cada ponto em sua respectiva célula e o processo de classificação em pontos centrais e não centrais. O processo inicia fixando uma célula central c_1. Seja c_2 uma célula central que é ϵ-vizinha de c_1. Para cada ponto central $p \in c_1$, encontra-se

o ponto central $p' \in c_2$ que é o mais próximo de p. Se $dist(p, p') \leq \epsilon$, uma aresta (c_1, c_2) é adicionada em G. Por outro lado, se todos os pontos em c_1 forem testados e nenhuma aresta foi criada, conclui-se que $(c_1, c_2) \notin E$.

Figura 7 - Grafo gerado pela aplicação do DBSCAN



Fonte: (GAN; TAO, 2015)

Por fim, o algoritmo então busca todas as componentes conexas de G. Cada componente conexa encontrada será um grupo resultante na aplicação do DBSCAN no conjunto P.

Em relação à complexidade, tem-se a seguinte análise. A determinação de G requer $O(n)$ consultas do vizinho mais próximo, cada uma das quais pode ser respondida em tempo $O(\log n)$ após a construção de um diagrama de Voronoi (GAN; TAO, 2015). Portanto, o tempo de execução é limitado por $O(n \log n)$.

Em seguida, após a construção do grafo G, deve-se atribuir os pontos de fronteira aos respectivos grupos. Dados dois pontos p e q, sendo p um ponto central, verifica-se se $dist(p, q) \leq \epsilon$. Em caso positivo, adiciona q ao grupo único de p. Este processo é descrito no Algoritmo 7. No algoritmo, a função $PontoCentralMaisPróximo(p, c)$ retorna o ponto central da célula c mais próximo a p.

Algoritmo 7 - Classificação - Pontos de Fronteira: $O(n.MinPts)$.

Entrada: Conjunto P com n pontos, ϵ, $MinPts$.
para cada célula não vazia c na grade **fazer**
  para cada ponto $p \in c$ **fazer**
    PontoFronteira ← VERDADEIRO
    se p não for um ponto central **então**
      para cada célula $c' \in N_\epsilon(c)$ **fazer**
        q ←PontoCentralMaisPróximo(p, c')
        se $dist(p, q) \leq \epsilon$ **então**
          Adiciona p ao grupo de q
          PontoFronteira ← FALSO
      se PontoFronteira **então**
        Marque p como ruído
  Saída: Todos os pontos não centrais atribuídos em algum grupo.

O Algoritmo 7 é similar ao Algoritmo 6, com a diferença que apenas verifica pontos em células vizinhas para pontos não centrais em vez de todos os pontos. Portanto, não pode custar mais que o algoritmo para classificação de ponto central. Sendo assim, o Algoritmo 7 é limitado por $O(MinPts.n)$.

Em linhas gerais, além da determinação do grafo G, o restante do algoritmo é calculado em $O(MinPts.n) = O(n)$ tempo esperado ou $O(n \log n)$ tempo de pior caso.

Verifica-se que o algoritmo anterior, para obter a complexidade de $O(n \log n)$, emprega o auxílio de um diagrama de Voronoi, tornando a solução como um todo uma tarefa não trivial. No capítulo seguinte, apresenta-se uma solução aproximativa para o cálculo do DBSCAN. Tal proposta, em linhas gerais, reduz a complexidade do algoritmo em troca da possibilidade de encontrar mais de um resultado possível, ou seja, o processo passaria a não ter mais uma única solução para um mesmo grupo de pontos. No próximo capítulo, então, ilustra-se o algoritmo proposto por GAN; TAO (2015).

1.1.3 Aprendizado Semi-Supervisionado

O aprendizado semi-supervisionado parte do princípio que os dados de treinamento serão parcialmente rotulados, isto é, alguns exemplos de treinamento estarão associados a classes (como no aprendizado supervisionado), enquanto outros não (como no aprendizado não supervisionado). O objetivo principal é fazer uso de todos os dados disponíveis. Para isso, deve-se fazer uso efetivo dos dados não rotulados, exigindo o uso de métodos não supervisionados. Depois que grupos ou padrões são descobertos, prossegue-se para aplicação do método supervisionado.

1.1.4 Métricas Para Avaliação de Um Modelo de Aprendizado Supervisionado

Há inúmeras métricas para avaliação de um modelo de classificação em Aprendizado de Máquinas. Cada métrica é util em diferentes aspectos do desempenho do modelo. Várias métricas podem ser utilizadas em conjunto para a obtenção de uma completa avaliação. Alguns métodos mais comumente usados são: Precisão, Sensibilidade (ou *Recall*), F1-score, Acurácia, Erro de Classificação, Especificidade, Cálculo de Falsos Positivos, Matriz de Confusão, Curva Roc (*Receiver Operating Characteristic Curve*), entre outros.

Nesta tese, utilizaremos para avaliação o erro de classificação e o F1-score.

O *erro de classificação* é a porcentagem de exemplos classificados de maneira incorreta. Já o *F*1*-score* (GERON, 2019) é uma métrica utilizada principalmente quando há possibilidade de desbalanceamento de classes, ou seja, há mais dados de uma classe do que de outra.

Para descrição da precisão e do recall precisamos descrever previamente 4 conceitos: verdadeiro positivo, falso positivo, verdadeiro negativo e falso negativo. Seguem, então, as devidas definições:

- **Verdadeiro Positivo:** Refere-se a um caso em que o modelo classifica uma amostra como pertencente a uma determinada classe e essa amostra realmente pertence a essa classe;
- **Falso Positivo:** Ocorre quando o modelo prevê que uma amostra pertence a uma determinada classe, mas na realidade ela não pertence a essa classe;
- **Verdadeiro Negativo:** Representa a previsão do modelo de que uma amostra não pertence a uma classe específica e essa amostra realmente não pertence àquela classe;
- **Falso Negativo:** Refere-se a uma situação em que o modelo prevê que uma amostra não pertence a uma determinada classe, mas na realidade ela pertence a essa classe.

Sendo assim, a *Precisão* é a proporção de verdadeiros positivos em relação a todos os dados classificados como positivos pelo modelo,

$$\text{Precisão} = \frac{\text{Verdadeiros Positivos}}{\text{Verdadeiros Positivos} + \text{Falsos Positivos}}$$

A *Sensibilidade* é a proporção de exemplos positivos reais corretamente classificados. É calculada da seguinte forma

$$\text{Sensibilidade} = \frac{\text{Verdadeiros Positivos}}{\text{Verdadeiros Positivos} + \text{Falsos Negativos}}$$

O F1-score é calculado com base na precisão e na sensibilidade do modelo, da seguinte forma

$$\text{F1} = 2 \times \frac{\text{Precisão} \times \text{Sensibilidade}}{\text{Precisão} + \text{Sensibilidade}}$$

O cálculo padrão do F1-score é feito para problemas de classificação binária. No entanto há variações do cálculo para problemas multiclasses. Sendo assim, considerando a possibilidade de classes desbalanceadas nos exemplos utilizados nessa tese, utilizaremos essa variação para o cálculo do F1-score, que é calculada da seguinte forma

$$\text{F1}_{ponderado} = \frac{\sum_{i=1}^{|C|} (\text{Quantidade de Exemplos da Classe } i) \times \text{F1}_i}{\text{Tamanho Total da Amostra}}$$

onde $F1_i$ representa a métrica F1-score da classificação binária considerando a Classe i uma classe e todas as demais uma segunda classe.

Além disso, há alguns métodos de aprendizado de máquina projetados para a avaliação da generalização de um modelo de classificação. Alguns exemplos são: Validação

Cruzada, *Holdout* Estratificado, Reamostragem, entre outros. Nesta tese, utilizou-se o método da Validação Cruzada.

A *Validação Cruzada* é uma técnica fundamental no campo de Aprendizado de Máquina, estimando o desempenho de um modelo em dados fora da amostra, empregando o conjunto de dados tanto como treinamento quanto como teste para estimar o erro.

O processo da Validação Cruzada (JAMES et al., 2013) consiste em dividir o conjunto de dados em partes menores. Realizam-se, então, iterações de treino e teste da seguinte maneira:

- **Passo** 1**:** O conjunto é dividido em k partes aproximadamente iguais;
- **Passo** 2**:** O modelo é treinado k vezes, utilizando $k-1$ partes como dados de treinamento e a parte restante como teste;
- **Passo** 3**:** O desempenho do modelo é calculado através de uma métrica de avaliação em cada iteração de treino/teste;
- **Passo** 4**:** Ao final das k iterações, calcula-se a média dos resultados das métricas em cada parte para estimar o desempenho geral do modelo.

A escolha do valor de k depende do tamanho do conjunto de dados (BISHOP, 2006). Os valores comuns para k são 5 e 10.

Algumas vantagens do uso da validação cruzada são: estima a capacidade de generalização do modelo, uso eficiente dos dados disponíveis, redução de viés na avaliação do modelo, avaliação robusta do modelo. A validação cruzada é usualmente empregada para a seleção de hiperparâmetros.

1.2 Geometria Computacional

A *geometria computacional* pode ser descrita como uma área da Ciência da Computação que estuda algoritmos e estruturas de dados para aplicação e resolução de problemas geométricos, importantes em outras áreas da computação que necessitam de uma interpretação geométrica, como por exemplo: Computação Gráfica, Robótica, Sistemas de Informações Geográficas, Visão Computacional, Otimização Combinatória, Processamento de Imagens, entre outras.

A geometria computacional, emergiu do campo de projeto e análise de algoritmos no final da década de 1970 (BERG et al., 1998) e tornou-se assunto frequente em periódicos e conferências com um grande número de pesquisadores ativos na área. No entanto, as primeiras soluções algorítmicas, para muitos problemas geométricos, eram ineficientes na prática e/ou difíceis de implementar. Nas últimas décadas, novas técnicas algorítmicas

seguem sendo desenvolvidas que melhoram e simplificam as abordagens iniciais (BERG et al., 1998). Uma técnica importante, com muito emprego na área de geometrica computacional, é a dos algoritmos randomizados.

Nesta tese, apresentam-se dois problemas de geometria computacional, a título de ilustração: círculo envolvente mínimo e o problema do par de pontos mais próximos.

1.3 **Algoritmos Randomizados**

O objetivo desta seção é apresentar os conceitos básicos para o projeto de algoritmos randomizados. Esta seção é baseada nas contribuições de Juraj Hromkovic (HROMKOVIČ, 2004), Rajeev Motwani e Prabhakar Raghavan (MOTWANI; RAGHAVAN, 2007).

1.3.1 Modelos de Algoritmos Randomizados

Algoritmos randomizados são controlados por um processo aleatório. Isto significa que algumas decisões em trechos do algoritmo resultam de um sorteio aleatório. Um algoritmo randomizado geralmente é analisado pelo seu tempo de execução e pela sua probabilidade de obter o resultado correto em uma entrada escolhida aleatoriamente.

Em seguida, apresentam-se dois tipos básicos de algoritmos randomizados.

1.3.2 Algoritmo de Las Vegas

Formalmente, um algoritmo randomizado A é chamado de algoritmo de Las Vegas computando uma função F se, para qualquer entrada x para A, temos que, $Prob(A(x) = F(x)) = 1$ e $T(x)$ é uma variável aleatória, onde $T(x)$ representa o tempo de execução de A sob a entrada x.

Sendo assim, a incerteza do algoritmo de Las Vegas não estará presente na correção do resultado obtido, mas no tempo de execução do algoritmo. É esperado que as execuções do algoritmo em uma entrada possam também variar em tempo de execução, porque se todas as execuções do algoritmo em dada entrada tivessem aproximadamente o mesmo tempo, então seria possível construir um algoritmo determinístico igualmente eficiente para esta tarefa, que simplesmente simula uma execução fixa do algoritmo randomizado (HROMKOVIČ, 2004). O sucesso de um algoritmo de Las Vegas está associado portanto se a sua complexidade de tempo médio é adequada.

1.3.3 Algoritmo de Monte Carlo

Em contrapartida ao algoritmo de Las Vegas, há o algoritmo de Monte Carlo. Enquanto o primeiro possui incerteza associada à complexidade pela complexidade de tempo de execução, o segundo produz uma resposta em geral de tempo eficiente, mas não garante a sua correção absoluta. Em geral, o algoritmo de Monte Carlo é aplicado a problemas de decisão. Nesse caso, o algoritmo pode garantir a correção na decisão "Sim" (algoritmo de Monte Carlo com garantia no "sim") ou na decisão "Não" (algoritmo de Monte Carlo com garantia no "não"). Há também a possibilidade do algoritmo não garantir a correção em ambos os casos. Como parte do projeto de um algoritmo de Monte Carlo, deve-se mostrar que a probabilidade de que o algoritmo produza a saída correta esteja limitada por alguma função conhecida.

1.3.4 Construção Incremental Aleatória

Em inúmeros problemas em ciência da computação, sabe-se que a aplicação de técnicas de randomização em algoritmos pode torná-los mais eficientes em tempo do que seus equivalentes algoritmos determinísticos (MOTWANI; RAGHAVAN, 2007). No entanto, em geral, a grande vantagem da aplicação de randomização em algoritmos para problemas de geometria computacional é a simplicidade para compreensão do algoritmo e facilidade de implementação.

Uma abordagem importante para descrição e entendimento de algoritmos em geometria computacional é a Construção Incremental Aleatória, responsável pela facilidade e simplicidade no compreensão do algoritmo. O conceito da técnica resume-se em ordenar aleatoriamente os n elementos que compõe a entrada do problema e, em seguida, considerar um elemento de cada vez e calcular o efeito da inclusão desse elemento na solução. Para melhor entendimento da técnica, apresenta-se a seguir um algoritmo incremental aleatório para ordenação de um vetor, conforme descrito por Motwani et al. (MOTWANI; RAGHAVAN, 2007).

Seja um vetor P com n números a serem ordenados. A ideia geral do algoritmo será como se segue. Iterativamente, espera-se que após a i-ésima iteração ($1 \leq i \leq n$) o vetor possua os i primeiros elementos ordenados. A iteração i consiste em escolher uniformemente ao acaso (aleatoriamente) um dos $n - i + 1$ números restantes e inserí-lo, na sua posição correta, na lista dos $i - 1$ primeiros elementos já ordenados. Procedendo assim, após a n-ésima iteração, o vetor encontra-se ordenado. Note que essa estratégia é basicamente aquela do bem conhecido algoritmo de ordenação *InsertionSort*, com a diferença que o próximo elemento a ser inserido entre os já ordenados é escolhido aleatoriamente.

O algoritmo concreto consiste em manter, após i inserções de chaves, uma lista dos

$i+1$ intervalos da reta real que representam a partição da reta pelas i chaves inseridas. Cada intervalo mantém uma lista de ponteiros para todas as chaves ainda não inseridas que estão contidas naquele intervalo, e cada chave não inserida mantém um ponteiro para o seu intervalo associado.

No começo da i-ésima iteração do algoritmo, na qual $i-1$ chaves já foram inseridas, há portanto i intervalos definidos pelas $i-1$ chaves já ordenadas. Ao inserir uma nova chave x, o intervalo I associado a x deve ser quebrado em dois, e todas as chaves, dentre as chaves ainda não inseridas que apontam para I, devem ter seus ponteiros atualizados. O número esperado de tais chaves é $\Theta((n-i)/i)) = \Theta(n/i)$. Portanto, o número total esperado de operações é de $\Theta(\sum_{i=1}^{n} n/i) = \Theta(n \log n)$. Note que, para conseguir essa complexidade de pior caso, diversos outros algoritmos empregam técnicas bem mais elaboradas.

Em seguida, apresentam-se alguns algoritmos para problemas de geometria computacional com suas respectivas soluções randomizadas (utilizando o paradigma da construção incremental aleatória). O texto segue a organização e apresentação de Motwani et al. (MOTWANI; RAGHAVAN, 2007).

1.3.4.1 Círculo Envolvente Mínimo

Dado um conjunto P de n pontos no plano, $P = \{p_1, p_2, p_3, ..., p_n\}$, o problema do círculo envolvente mínimo consiste em encontrar o círculo de menor raio que contém todos os pontos de P.

Pode-se mostrar facilmente que, em qualquer solução, ou o círculo envolvente mínimo:

- passa por exatamente dois pontos do conjunto e, neste caso, o segmento de reta formado por esses dois pontos é o diâmetro do círculo; ou
- passa por três pontos do conjunto e, neste caso, o triângulo formado por esses três pontos não é obtuso.

O algoritmo mais ingênuo para resolver o problema é o de força bruta que, basicamente, faz uma enumeração de todos os pares e trios de pontos de P e determina o círculo que passa por cada uma dessas duplas ou trios de pontos. Para cada círculo, verifica-se em seguida se ele envolve todos os demais pontos. Dentre todos que responderem afirmativamente a essa verificação, aquele de menor diâmetro é a resposta do problema. A complexidade de tempo desse algoritmo é $O(n^4)$.

Há outros algoritmos determinísticos que resolvem o problema do círculo envolvente mínimo com melhor complexidade. Dentre eles: Algoritmo de Chrystal com complexidade

$O(hn)$ (CHRYSTAL, 1885), Algoritmo de Elzinga e Hearn com complexidade $O(h^3n)$ (ELZINGA; HEARN, 1972), Algoritmo de Shamos e Hoey com complexidade $O(n \log n)$ (SHAMOS; HOEY, 1975), entre outros. Considere h como o número de pontos que estão no limite do círculo.

Algoritmo Incremental Aleatório: Apresenta-se um algoritmo incremental aleatório (BERG et al., 1998) para a resolução do círculo envolvente mínimo. Inicialmente geramos uma permutação aleatória dos n pontos de P. Seja p_i o i-ésimo ponto da permutação gerada. Incluem-se sucessivamente os pontos dessa ordenação à solução. Seja D_i o círculo envolvente mínimo com respeito a $P_i = \{p_1, \ldots, p_i\}$

- Se $p_i \in D_{i-1}$ então $D_i = D_{i-1}$
- Se $p_i \notin D_{i-1}$ então p_i está com certeza na fronteira de D_i.

Tem-se então, o algoritmo incremental aleatório para resolução do problema do círculo envolvente mínimo, dado pelo Algoritmo 8.

Algoritmo 8 - Círculo Envolvente Mínimo - Randomizado

Entrada: Conjunto P com n pontos
Saída: Centro e Raio do Círculo Envolvente Mínimo
FUNÇÃO $CirculoMinimo\ (P)$
    Computar uma permutação uniformemente randomizada dos pontos de P.
    Tome D_2 como o menor círculo envolvente criado por p_1 e p_2.
    para $i \leftarrow 3$ até n **fazer**
        se $p_i \in D_{i-1}$ **então**
            $D_i \leftarrow D_{i-1}$
        senão
            $D_i \leftarrow CirculoMinimoComPonto(\{p_1, p_2, p_3, ..., p_{i-1}\}, p_i)$
    retornar D_n

A etapa crítica para o algoritmo ocorre quando $p_i \notin D_{i-1}$. Sendo assim, deve-se criar uma sub-rotina ao algoritmo que faça com que encontre o menor círculo envolvendo P_i, dado que p_i deve estar em sua fronteira. Toma-se $q = p_i$. Adicionam-se os pontos de P_{i-1} em ordem aleatória e mantém-se o menor círculo envolvente de $P_{i-1} \cup q$ sob a restrição de que q está em sua fronteira. Tem-se então o Algoritmo 9.

Algoritmo 9 - Círculo Mínimo Com Ponto

Entrada: Conjunto P com n pontos e um ponto q tal que existe um círculo envolvente com q em sua fronteira.
Saída: Centro e Raio do Círculo Envolvente Mínimo para P com q em sua fronteira.
FUNÇÃO $CirculoMinimoComPonto\ (P, q)$
| Computar uma permutação uniformemente randomizada dos pontos de P.
| Tome D_1 como o menor círculo envolvente criado por p_1 e q.
| **para** $j \leftarrow 2$ até n **fazer**
| | **se** $p_j \in D_{j-1}$ **então**
| | | $D_j \leftarrow D_{j-1}$
| | **senão**
| | | $D_j \leftarrow CirculoMinimoComDoisPontos(\{p_1, p_2, p_3, ..., p_{j-1}\}, p_j, q)$
| **retornar** D_n

Novamente, outra restrição é apresentada para resolução do problema. Desta vez, precisa-se encontrar o menor círculo envolvente para um conjunto sob a restrição de que dois pontos dados, q_1 e q_2, estejam em sua fronteira. O Algoritmo 10 descreve esta rotina.

Algoritmo 10 - Círculo Mínimo com 2 pontos.

Entrada: Conjunto P com n pontos com dois pontos, q_1 e q_2, tal que existe um círculo envolvente para P com q_1 e q_2 em sua fronteira.
Saída: Centro e Raio do Círculo Envolvente Mínimo para P com q_1 e q_2 em sua fronteira.
FUNÇÃO $CirculoMinimoComDoisPontos\ (P,\ q_1,\ q_2)$
| Computar uma permutação uniformemente randomizada dos pontos de P.
| Tome D_0 como o menor círculo envolvente de P criado com q_1 e q_2 em sua fronteira.
| **para** $k \leftarrow 1$ até n **fazer**
| | **se** $p_k \in D_{k-1}$ **então**
| | | $D_k \leftarrow D_{k-1}$
| | **senão**
| | | $D_k \leftarrow$ Círculo com q_1, q_2 e p_k em sua fronteira
| **retornar** D_n

Essa última rotina, encerra a apresentação completa do algoritmo randomizado para o problema do Círculo Envolvente Mínimo. Em seguida, apresenta-se o cálculo da complexidade do algoritmo.

Teorema 1. *O menor círculo envolvente para um conjunto de n pontos no plano pode ser determinado em tempo esperado $O(n)$ usando espaço auxiliar linear de pior caso (BERG et al., 1998).*

Demonstração. O Algoritmo 10 é executado em tempo $O(n)$ porque cada iteração do comando de repetição leva um tempo constante e usa armazenamento linear. O Algoritmo 9

e o Algoritmo 8 também precisam de espaço linear, então o que resta é analisar o tempo de execução esperado dos Algoritmos 8 e 9.

O tempo de execução do Algoritmo 9 é $O(n)$, desde que não conte o tempo gasto em chamadas para o Algoritmo 10. Para analisar seu tempo esperado, deve-se observar a probabilidade de ter que fazer tais chamadas. Limita-se essa probabilidade da seguinte forma: Fixa-se um subconjunto $\{p_1, p_2, ..., p_i\}$ e consideremos D_i o menor círculo envolvendo $\{p_1, p_2, ..., p_i\}$ com q em sua fronteira. Considere que tenhamos a remoção de um dos pontos $\{p_1, p_2, ..., p_i\}$. O círculo envolvente do conjunto remanescente só pode mudar quando remove-se um dos três pontos em sua fronteira. No entanto, um dos pontos no limite é q, então há no máximo dois pontos que fazem com que o menor círculo envolvente diminua. A probabilidade do ponto removido ser um desses pontos é $\frac{2}{i}$. (Quando há mais de três pontos no limite, a probabilidade de que o menor círculo envolvente mude só pode diminuir.) Portanto, podemos limitar o tempo total de execução esperado do Algoritmo 9 por

$$O(n) + \sum_{i=2}^{n} O(i) \cdot \frac{2}{i} = O(n)$$

Aplicando argumento análogo ao Algoritmo 8, verifica-se que o tempo de execução esperado do algoritmo geral para o problema do Círculo Envolvente Mínimo é $O(n)$. □

1.3.4.2 Problema do Par de Pontos Mais Próximo

O problema do par de pontos mais próximo em geometria computacional é definido da seguinte forma: dado um conjunto $P \subset \mathbb{R}^d$ de n pontos em d dimensões, encontre dois pontos distintos p, $q \in P$ que minimizam a distância euclidiana $dist(p, q)$, entre todos os pares de pontos em P.

Novamente, a solução do problema por um algoritmo de força bruta é facilmente apresentada. Bastaria verificar a distância entre todos os pares de pontos e retornar o par com menor distância encontrada. Tal algoritmo é executado em tempo $\Theta(n^2)$.

No entanto, alguns algoritmos determinísticos foram sendo implementados para redução da complexidade do processo. Na década de 1970, o problema foi resolvido no plano em tempo $O(n \log n)$ aplicando em seu processo a Triangulação de Delaunay (SHAMOS; HOEY, 1975; SHAMOS; PREPARATA, 1985). Além disso, para qualquer $d \geq 2$, o algoritmo clássico de divisão e conquista de Bentley e Shamos também atinge tempo $O(n \log n)$ (BENTLEY; SHAMOS, 1976). Posteriormente, outros algoritmos foram implementados, dentre eles o proposto por Buchin et al (BUCHIN; MULZER, 2011), em tempo esperado $O(n \log \log n)$.

Em seguida, apresenta-se o algoritmo randomizado de Rabin, que resolve o pro-

blema no plano em tempo esperado $\Theta(n)$.

Algoritmo de Rabin: Apresenta-se nesta seção o algoritmo incremental aleatório de Rabin (RABIN, 1976), como abordado em (BANYASSADY; MULZER, 2007) para o caso de 2 dimensões, embora a discussão possa ser generalizada para $d \geq 3$ dimensões.

Seja $P \subset \mathbb{R}^2$ um conjunto de n pontos no plano tal que todas distâncias entre pontos em P são distintas. Além disso, assume-se, por conveniência, que P está completamente no quadrante superior direito, ou seja, que todos os pontos em P têm coordenadas positivas.

Para $i \in \mathbb{Z}$, define-se a grade G_i como uma subdivisão do plano em células quadradas de diagonal 2^i. A grade G_i é alinhada de modo que a origem dos eixos cartesianos apareça como um canto de quatro células de grade. Duas células serão consideradas vizinhas se compartilham ao menos um vértice de grade. Define-se o identificador de uma célula $\sigma \in G_i$ como um par de $\mathbb{Z} \times \mathbb{Z}$, indicando a coluna e linha do canto inferior esquerdo de σ. Como exemplo, a célula cujo canto inferior esquerdo é a origem, tem identificação $(0, 0)$. A célula imediatamente à direita tem identificação $(1, 0)$, e aquela imediamente acima tem identificação (0,1), e assim por diante. Supõe-se também, que há uma função chamada $EncontrarCelula(i, p)$ que retorna o identificador da célula $\sigma \in G_i$, que contém o ponto p. Tal função pode ser implementada em tempo constante.

A ideia do algoritmo é construir uma tabela de dispersão (estrutura de dados também conhecida como dicionário) para armazenar células que estão em uma grade, cada célula armazenando uma lista de seus pontos.

Seja $P \subset \mathbb{R}^2$ o conjunto de n pontos no plano e $k = \lfloor \log_2 n \rfloor - 1$. Calcula-se então uma gradação aleatória de subconjuntos de $P = P_0 \supset P_1 \supset P_2 \supset ... \supset P_i \supset ... \supset P_k$, onde o conjunto P_i é um subconjunto aleatório de P_{i-1} com exatamente $|P_i| = \lfloor \frac{n}{2^i} \rfloor$ elementos.

O algoritmo procederá etapas numeradas descendentemente de $k + 1$ até 1. O objetivo da etapa i é determinar um dicionário de células D_{i-1} que armazene todos os pontos de P_{i-1} tal que:

- Cada célula de D_{i-1} contém no máximo 1 ponto de P_{i-1};
- Seja p, $q \in P_{i-1}$ pontos que constituem o par mais próximo em P_{i-1}. Então, as células que contém p e q devem ser vizinhas.

Como $|P_k| = O(1)$, pode-se obter isto facilmente na etapa $k + 1$, verificando todos os pares em P_k e calculando a distância do par mais próximo δ_k. Então, define-se $j = \lceil \log_2 \delta_k \rceil - 1$, cria-se um dicionário de célula D_k para a grade G_j e insere-se todos os pontos de P_k em D_k (cada ponto $p \in P_k$ é inserido na lista associada a célula $EncontrarCelula(j, p)$ de D_k). Como o diâmetro das células de G_j é $2^j \in [\frac{\delta_k}{2}, \delta_k)$, cada célula em D_k tem no máximo um ponto de P_k e os pontos mais próximos estão em células vizinhas.

Nas próximas etapas, o algoritmo sempre terá o dicionário de células D_i da etapa anterior, e construirá o dicionário de célula D_{i-1} da seguinte forma:

1. Insira todos os pontos de P_{i-1} em D_i;
2. Para cada célula σ não vazia em D_i:
   (a) encontre o conjunto Q_σ dos pontos que estão na célula σ;
   (b) Por força bruta, encontre a distância δ_σ do par mais próximo em Q_σ
3. Defina $\delta_{i-1} = \min_{\sigma \in D_i}\{\delta_\sigma\}$;
4. Defina $j = \lceil \log \delta_{i-1} \rceil - 1$ e crie o dicionário de células D_{i-1} para G_j;
5. Insira todos os pontos de P_{i-1} em D_{i-1}.

A diagonal das células em D_{i-1} é $2^j \in [\frac{\delta_{i-1}}{2}, \delta_{i-1})$, logo cada célula contém no máximo um ponto de P_{i-1}. Além do mais, as células do par mais próximo devem ser vizinhas.

Pode-se mostrar que o problema do par de pontos mais próximo tem tempo esperado de $O(n)$ (RABIN, 1976).

2 VARIANTES DE ÁRVORE DE DECISÃO E AGRUPAMENTO

Neste capítulo, apresentam-se os principais temas utilizados para a proposta desta tese. Serão descritos em mais detalhes variantes de árvores de decisão e um algoritmo aproximativo para o cálculo do método de agrupamento DBSCAN, chamado DBSCAN ρ-aproximado.

2.1 DBSCAN ρ-aproximado

Nesta seção, apresenta-se o método aproximativo que resolve o problema do DBSCAN (DBSCAN ρ-aproximado).

Sabe-se que a aplicação do algoritmo DBSCAN exato pode ser implementada em tempo $O(n \log n)$ no espaço em 2 dimensões. Para $d \geq 3$ dimensões, o tempo de computação torna-se $O(n^2)$ com um algoritmo direto, mas pode ser implementada em tempo subquadrático de $O(n^{2(1-1/(d+2))} \log^c n)$, para certa constante $c > 0$. Apresentaremos um algoritmo aproximado para a técnica de agrupamento por densidade (GAN; TAO, 2015).

Além do conjunto de pontos P, dos parâmetros ϵ e $MinPts$, tem-se agora de entrada um parâmetro real ρ que controla o grau de aproximação.

As definições básicas do DBSCAN exato se aplicam, algumas com pequenas modificações. As noções de ponto central e não central mantém-se inalteradas. O conceito de acessibilidade de densidade é ajustado como se segue.

Um ponto $q \in P$ é *ρ alcançável por densidade (ρ-alcançável)* de p se existe uma sequência de pontos $p_1, p_2, p_3, ..., p_t \in P$ tal que:

- $p_1 = p$ e $p_t = q$;
- $p_1, p_2, p_3, ..., p_{t-1}$ são pontos centrais;
- $p_{i+1} \in B(p_i, \epsilon(1 + \rho))$ para cada $i \in \{1, \ldots, t-1\}$.

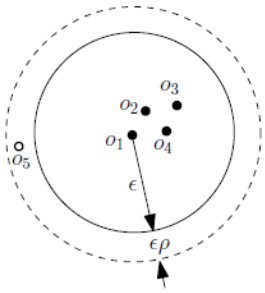
Considere a Figura 8, o círculo interno possui raio ϵ e o externo, $\epsilon(1 + \rho)$. Os pontos centrais e não centrais são representados, respectivamente, em preto e branco. O ponto O_5 é ρ-alcançável de O_3, mas não é alcançado por densidade de O_3.

Um agrupamento C ρ-aproximado é um subconjunto não vazio de P tal que:

- **Maximalidade:** Se $p \in C$ é um ponto central, então todos os pontos densidade alcançável de p também pertencem a C.
- **Conectividade ρ-aproximado:** Para qualquer ponto $p_1 \in C$ existe um ponto $p \in C$ tal que p_1 é ρ-alcançável de p.

Nessa nova formulação, o requisito de conectividade foi relaxado para um raio de $\epsilon(1+\rho)$. Então, o problema DBSCAN ρ-aproximado é encontrar um conjunto C de grupos ρ-aproximado. Além disso, ao contrário do problema original, o problema aproximado pode não ter um resultado único. Na Figura 8, por exemplo, o ponto O_5 poderia ser incluído ou não no grupo $\{O_1, O_2, O_3, O_4\}$

Figura 8 - Exemplo dos raios utilizados no DBSCAN ρ-aproximado



Fonte: (GAN; TAO, 2015)

Ambos os processos, exato e aproximativo, são parametrizados por ϵ e $MinPts$. Pode-se afirmar que os agrupamentos obtidos pelo DBSCAN ρ-aproximado são um superconjunto e um subconjunto respectivamente daqueles obtidos do processo exato pelos parâmetros $(\epsilon, MinPts)$ e $(\epsilon(1+\rho), MinPts)$ (GAN; TAO, 2015).

Resumidamente, dado um ponto $p \in C$, todos os pontos alcançáveis por densidade de p farão parte do grupo C. No entanto, os pontos ρ-alcançáveis por densidade podem fazer parte ou não de C. É nessa relaxação dos pontos ρ-alcançáveis que se permite a possibilidade de obter-se mais de um resultado possível para o mesmo conjunto de pontos e com os mesmos parâmetros aplicados.

Em seguida, apresenta-se a estrutura de dados utilizada no algoritmo e o processo denominado de Contagem de Intervalo Aproximada, útil para a execução do DBSCAN aproximativo.

Contagem de Intervalo Aproximada: Seja P um conjunto de n pontos em $\mathbb{R}^d$ onde d é uma constante. Dado qualquer ponto $q \in \mathbb{R}^d$, raio ϵ, e uma constante ρ, uma *consulta de intervalo aproximada* é definida como uma consulta que retorna um número inteiro garantido estar entre $|B(q, \epsilon) \cap P|$ e $|B(q, \epsilon(1+\rho)) \cap P|$.

Estrutura de dados: A estrutura representa um particionamento de grades hierárquicas. Primeiro cria-se uma grade regular em $\mathbb{R}^d$ onde cada célula será um hiper-cubo d-dimensional de lado $\frac{\epsilon}{\sqrt{d}}$. Para cada célula não vazia da grade (considerando-se a presença dos pontos de P), divide-se agora em 2^d células de mesmo tamanho (isto é, divide-se cada hipercubo em cubos menores, com metade do lado original). Novamente, para cada célula não vazia resultante, repete-se o processo recursivamente até que o comprimento da célula resultante seja inferior a $\frac{\epsilon\rho}{\sqrt{d}}$.

Dada uma consulta de contagem de intervalo aproximada com os parâmetros q, ϵ, ρ, calcula-se sua resposta da seguinte forma. Iniciamos uma variável *ans* de resposta igual a 0. Em geral, dada uma célula de nível i não vazia σ, distinguimos 3 casos:

- Se σ é disjunto com $B(q, \epsilon)$, ignore-a.
- Se σ for totalmente coberta por $B(q, \epsilon(1+\rho))$ adicione a quantidade de pontos em σ em *ans*.
- Se nenhuma das opções acima ocorrer verifique se σ é uma folha. Caso não seja uma folha, basta fazer a consulta às células filhas de σ e acumular as quantidades retornadas na variável *ans*. Caso σ seja uma folha, adicione a quantidade de pontos de σ em *ans*. O comprimento da célula folha ($\frac{\epsilon\rho}{\sqrt{d}}$) garante que todos os pontos em σ estão cobertos por $B(q, \epsilon(1+\rho))$.

Algoritmo: O algoritmo ρ-aproximado difere do algoritmo exato na determinação do grafo G. A definição do grafo G, contudo, é a mesma.

Para encontrar o grafo G, o algoritmo constrói, para cada célula central, uma estrutura para consulta de intervalo aproximada no conjunto de pontos centrais. Para gerar arestas a partir de uma célula central σ_1, basta examinar cada célula ϵ-vizinha σ_2 de σ_1. Para cada ponto central p em σ_1, faça uma consulta de contagem de intervalo aproximada no conjunto de pontos centrais em σ_2. Se a consulta retornar uma quantidade positiva, adicione aresta entre σ_1 e σ_2. Caso contrário, não há aresta entre σ_1 e σ_2.

Análise de tempo: O tempo esperado para construir a estrutura de dados para consulta de intervalo aproximada é $O(n)$, uma vez que o número de grades hierárquicas é $\log_2(1/\rho)$, que é uma constante no problema. O tempo para a consulta de intervalo aproximada em si é proporcional ao número de grades hierárquicas, que é uma constante. Pode-se modificar ligeiramente o algoritmo para parar assim que encontrar um ponto, uma vez vez que o propósito é apenas saber se há ou não pontos alcancáveis na célula vizinha para a criação do grafo. Já o tempo esperado para a determinação do grafo G é proporcional ao número de consultas de contagem de intervalo aproximada. Para cada ponto central de uma célula σ_1, são necessárias $O(1)$ consultas no total. Logo, o número total de consultas é em tempo $O(n)$. Em resumo, o tempo do algoritmo exato anteriormente apresentado é de $\Omega(n \log n)$. Já no caso do algoritmo ρ-aproximativo, pelo fato de contentar-se com uma solução aproximada para o problema de agrupamento por densidade, possui complexidade de tempo esperada de $O(n)$.

2.2 **Árvore de Decisão**

Nesta seção, apresentam-se algumas variantes de árvore de decisão. As árvores de decisão podem ser referidas como classificadores hierárquicos porque precisam de discriminação multinível para determinar a qual classe um padrão específico pertence. A árvore é representada inicialmente por um nó raiz. A partir do nó raiz, associados a todos os dados de treinamento, nós filhos são criados, particionando os dados do nó anterior ao considerar um determinado predicado. Um nó filho está associado aos dados que satisfazem ao predicado, e o outro aos que não satisfazem. O processo continua recursivamente e se encerra quando não é mais necessário particionar os dados para fazer a predição de classe. Esses nós tornam-se nós folhas e classificarão todos os dados de consulta que chegarem até esse nó da árvore. Veja a Seção 1.1.1.1 para uma descrição detalhada de uma árvore de decisão.

As árvores de decisão são capazes de lidar de forma natural com a classificação multiclasse. Há inúmeras maneiras de se dividir um nó e, com base nessas divisões, as árvores de decisão podem ser categorizadas em decisão univariada, decisão multivariada e decisão omnivariada.

2.2.1 Árvore de Decisão Univariada

Árvores de *decisão univariadas* são aquelas em que um único atributo participa da divisão dos nós, chamado de *divisão ortogonal*. As árvores de decisão univariadas mais representativas são ID3, CART (BREIMAN, 1984) e C4.5 (QUINLAN, 1993). Estas árvores utilizam respectivamente, ganho de informação, índice de Gini e taxa de ganho como critérios de divisão que são medidas heurísticas baseadas em frequência. A seguir, comenta-se a árvore ID3 como exemplo de árvores de decisão univariada.

2.2.1.1 ID3

A ID3 (QUINLAN, 1993), projetada em 1986, se caracteriza por tratar os dados contínuos como se fossem categóricos. Além disso, ao invés de dividir cada nó em dois filhos, usa um número de filhos igual à quantidade de valores distintos no atributo escolhido para o predicado associado ao nó. Cada filho está associado aos dados que possuem um dos valores existentes do atributo escolhido.

O algoritmo inicia com o conjunto de treinamento representando o nó raiz. Em seguida calcula-se qual atributo possui maior ganho de informação e o escolhe para divisão do nó raiz. Divide-se então o conjunto de treinamento pelo atributo selecionado. Um

atributo não poderá ser selecionado novamente para partição. A recursão nas subárvores ocorrerá da mesma maneira, com as seguintes condições de parada:

- Se em determinado nó só restar elementos da mesma classe, este nó torna-se folha e é rotulado com a classe dos respectivos exemplos;
- não há mais atributos a serem selecionados. Neste caso, o nó torna-se folha e será rotulado com a classe majoritária.

2.2.2 Árvore de Decisão Multivariada

A divisão ortogonal das árvores de decisão univariadas pode ser inapropriada quando conjuntos de dados possuem variáveis numericamente correlacionadas. Nestes casos, um limite oblíquo poderia ter separado melhor as instâncias. Além disso, a seleção de pontos de corte nessas árvores de decisão univariadas envolve uma busca gulosa dos candidatos classificados, que é demorada quando todos os atributos são contínuos.

As árvores de decisão *multivariadas* são aquelas em que mais de um atributo participa da divisão de nós. Esses atributos constantemente constroem uma combinação linear ou não linear que apresenta um limite oblíquo em cada nó dividido nas árvores. No entanto, o problema de determinar o hiperplano que minimiza o número dos exemplos de treinamento incorretamente classificados é NP-difícil (HEATH, 1993). Dessa maneira, árvores de decisão multivariadas devem se basear em boas heurísticas para a determinação de hiperplanos.

Exemplos de árvore de decisão multivariada são: FACT (LOH; VANICHSETAKUL, 1988), OC1 (MURTHY; KASIF; SALZBERG, 1994), QUEST (LOH; SHIH, 1997) e LDT (YILDIZ; ALPAYDIN, 2005).

2.2.3 Árvore de Decisão Omnivariada ou Híbrida

Além da árvore de decisão univariada e multivariada, tem-se também árvores de decisão omnivariadas ou híbridas. As árvores de decisão omnivariadas são aquelas nas quais a divisão em cada nó é univariada, multivariada linear ou multivariada não linear. A árvore SVMDT (Support Vector Machine Decision Tree) (KUMAR; GOPAL, 2010) é uma árvore de decisão híbrida baseada em SVM para classificação binária, que possui nós univariados e multivariados. A árvore de decisão proposta no escopo dessa tese também se enquadrada nessa categoria.

2.2.3.1 SVMDT

A árvore de decisão híbrida baseada no SVM (KUMAR; GOPAL, 2010) é proposta com o intuito de acelerar o SVM em sua fase de teste (isto é, na fase de classificação de dados fora dos exemplos de treinamento) para tarefas de classificação binária. O SVM é consideravelmente mais custoso em tempo na fase de teste do que outras técnicas, por conta da sua função de decisão estar diretamente relacionada ao número de vetores de suporte. Então, se o número de vetores de suporte for muito grande, o SVM levará mais tempo para classificar um novo dado. Por outro lado, árvores de decisão executam a fase de teste com mais eficiência, mas podem possuir uma acurácia menor quando comparada ao SVM.

A abordagem do SVMDT é reduzir o número de dados de teste que requerem a decisão do SVM. Ele usa SVM e árvore de decisão para obter uma classificação rápida sem comprometer a precisão da classificação.

O algoritmo divide o espaço dos dados em três regiões: Duas regiões mais distantes do hiperplano associado ao SVM e uma região mais próxima a ele. Todos os pontos das regiões mais distantes ao limite de decisão deverão ser classificados pela decisão univariada da árvore de decisão. Por outro lado, os pontos na região próxima ao limite de decisão serão separados através da aplicação do SVM. A motivação da estratégia é baseado na premissa de que o SVM possui um maior custo de teste que uma árvore de decisão, mas possui melhor acurácia. Assim, dados mais longe do hiperplano separador poderiam ser deixados a cargo de serem classificados pelas árvores de decisão, uma vez que mais dificilmente haverá uma classificação incorreta. Pontos mais próximos do hiperplano separador são mais sensíveis a erros de classificação, e portanto, continuam sendo classificados pelo SVM.

2.2.4 Árvores de Decisão Baseadas em Métodos Não Supervisionados

Pode-se afirmar que a maioria dos modelos existentes de árvore de decisão aplicam métodos de divisão de nós supervisionados (WANG et al., 2020), ou seja, tais métodos utilizam-se das classes de cada elemento para decidir qual atributo ou quais atributos serão utilizados na partição de um nó. No entanto, recentemente, há na literatura algumas propostas de utilização de métodos de agrupamento (não supevisionados) para aplicação junto às árvores de decisão.

Como exemplos desta metodologia, tem-se a aplicação do agrupamento k-médias, através dos algoritmos propostos por Gadam et al. (GADDAM; PHOHA; BALAGANI, 2007) e Wang et al. (WANG et al., 2020).

2.2.4.1 GADAM

Em seu estudo, Gadam et al. (GADDAM; PHOHA; BALAGANI, 2007) propõem uma combinação da árvore ID3 e do método de agrupamento não supervisionado k-médias. No primeiro estágio, o agrupamento k-médias é realizado nas instâncias de treinamento para obter k grupos disjuntos. Cada grupo representa uma região de instâncias semelhantes em termos de distâncias euclidianas, entre as instâncias e seus centroides de grupo.

O método k-médias garante que cada instância de treinamento esteja associada a apenas um grupo. No entanto, se houver quaisquer subgrupos ou sobreposições dentro de um grupo, a árvore de decisão ID3 treinada nesse grupo refina os limites de decisão particionando as instâncias com um conjunto de regras sobre o espaço de atributos. Mais especificamente, a árvore de decisão ID3, construída em cada instância, aprende as classes dentro do grupo e particiona o espaço de decisão em regiões de classificação mais precisas, melhorando assim o desempenho geral da classificação.

A aplicação do método em testes realizados no próprio estudo (GADDAM; PHOHA; BALAGANI, 2007) mostraram ganho de precisão quando comparado aos métodos k-médias e ID3 aplicados isoladamente.

2.2.4.2 BDTKS

Em 2020, Wang et al. (WANG et al., 2020) desenvolveram uma nova árvore de decisão binária multivariada linear chamada *Árvore de Decisão Binária baseada em k-médias* (BDTKS). Neste modelo, contorna-se a construção direta da função linear com os parâmetros correspondentes, e utiliza-se o agrupamento não supervisionado k-médias (MACQUEEN, 1967).

O agrupamento k-médias é utilizado para a partição de cada nó e considera todos os atributos de cada elemento para o cálculo. Neste trabalho proposto por Wang et al., a árvore ao final do algoritmo é uma árvore binária, ou seja, o agrupamento é utilizado sempre com $k = 2$.

A condição de parada utilizada, que transformará um nó em folha, é baseada na pureza dos dados, com alguns ajustes considerando o possível desbalanceamento de classes que pode ser oriundo dos dados de treinamento.

Em relação a um novo dado de teste, o algoritmo calcula, através dos centróides dos nós, qual dos dois nós filhos aquele dado está mais perto. Em seguida, o dado é atribuído àquele nó. O processo segue recursivamente até o dado alcançar um nó folha e, então, é classificado. Wang et al., ao considerar o tempo de execução do algoritmo, propõem uma mudança nesse cálculo dos centróides. A cada divisão, utiliza-se dos dois

centróides para calcular um hiperplano que será usado para a divisão do espaço e definir a qual subárvore o novo dado deve pertencer.

Mais precisamente, pode-se dizer que o algoritmo BDTKS possui as seguintes contribuições: primeiro, BDTKS é uma nova árvore de decisão binária multivariada linear, ou seja, enriquece ainda mais os métodos de árvore de decisão multivariada; em segundo lugar, o BDTKS incorpora agrupamento k-médias não supervisionado na estrutura da árvore binária e também melhora o desempenho da generalização. Por fim, uma conversão do modelo BDTKS baseado no centróide k-médias na árvore de decisão baseada no hiperplano acelera o processo de classificação.

3 MODELO PROPOSTO - ÁRVORE DE DECISÃO COM DBSCAN APROXIMATIVO

Neste Capítulo, apresenta-se o modelo proposto com a inserção do DBSCAN aproximativo junto as Árvores de Decisão. Chamaremos esse modelo de DDT – DBSCAN Decision Tree.

3.1 Árvore de Decisão com DBSCAN ρ-aproximado (DDT)

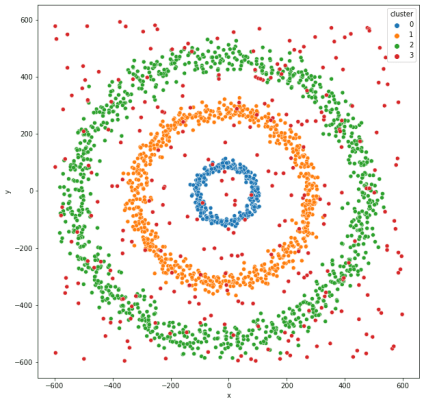
As árvores de decisão, assim como outros métodos de aprendizado de máquina, tanto supervisionados quanto não supervisionados, possuem em seus fundamentos conceitos geométricos. Em geral, o espaço d-dimensional em que os dados estão dispostos é particionado em regiões. Nas árvores de decisão, essas regiões são representadas pelos nós. O método busca definir essas regiões geometricamente e classifica novos dados ao considerar a que região fazem parte. No entanto, nem sempre definir essas regiões é uma tarefa fácil. A escolha de que tipo de árvore de decisão utilizar, bem como do critério de divisão utilizado (univariada ou multivariada) estão diretamente ligados à disposição dos dados de entrada.

Recorde da Seção 1.1.1.1 que uma árvore de decisão emprega um conjunto de possíveis predicados, cada um questionando os dados de maneira distinta. A eficicácia em classificar os dados por certo predicado varia conforme as disposições dos dados no espaço d dimensional de atributos. Sendo assim, a metodologia que melhor divida os dados depende dessa disposição dos mesmos. Más escolhas na sequência dos predicados a aplicar podem gerar árvores cuja capacidade de generalizar seja pequena, além de produzir árvores maiores e de difícil compreensão. Porém, nem sempre as decisões univariada e multivariada podem resultar em árvores concisas, mesmo tomando-se o cuidado de analisar todas as divisões possíveis, tanto univariadas quanto multivariadas lineares. Se considerarmos a disposição dos dados da Figura 9, percebe-se que não há uma maneira de se construir uma árvore pequena utilizando somente as decisões univariadas e multivariadas lineares. Por exemplo, na Figura 10, vê-se uma árvore de decisão resultante.

Em 2020, Wang et al. (WANG et al., 2020) desenvolve a árvore de decisão baseada em k-médias (BDTKS). Nesta árvore, a busca pela melhor divisão univariada é substituída pelo método k-médias. No entanto, o k-médias possui importantes limitações, principalmente ao considerarmos que os agrupamentos sempre retornam regiões esféricas disjuntas. Seria interessante que outros tipos de regiões pudessem ser consideradas por um método de agrupamento que auxilie as árvores de decisão.

O agrupamento não supervisionado DSBCAN possui a qualidade de detectar agru-

Figura 9 - Exemplo de agrupamentos em que as metodologias de árvore de decisão univariada e multivariada linear podem não encontrar uma árvore pequena.



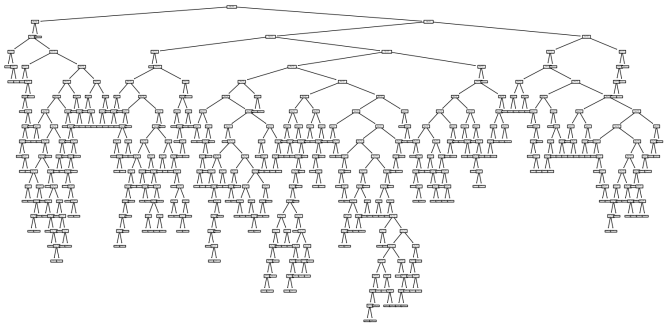
Fonte: https://anderfernandez.com/blog/dbscan-python/

pamentos de diversas formas, considerando o conceito de densidade, em que pontos próximos no espaço devem necessariamente ser colocados em um mesmo grupo. Os grupos coloridos da Figura 9 poderiam ser facilmente classificados pelo DBSCAN com uma escolha apropriada de seus parâmetros, com a vantagem de ignorar os pontos vermelhos, considerados ruídos pelo método. A árvore de decisão não possui em geral essa capacidade, e tais ruídos seriam uma possível fonte de sobreajuste (*overfitting*). Por outro lado, sua complexidade mais elevada pode ser um fator de inviabilidade ao considerar sua aplicação em um conjunto de dados com muitos elementos. Essa deficiência pode ser amenizada empregando-se a versão aproximativa do DBSCAN. O DBSCAN ρ-aproximado possui complexidade de tempo esperado $O(n)$. A desvantagem é que, escondido sob a notação assintótica, há uma constante multiplicativa exponencial no número de características dos dados. A alta dimensão, embora não possa ser de todo negligenciada, potencialmente permite estender na prática o uso do DBSCAN a conjuntos de dados de maior número de elementos, desde que o número de características seja pequeno.

Sendo assim, essa tese aplica o DBSCAN aproximativo para a divisão dos nós de uma árvore de decisão considerando em conjunto as divisões univariadas ordinárias da árvore de decisão. Cabe ressaltar que a árvore proposta é uma árvore de decisão híbrida. Acrescentamos à árvore existente, além das decisões baseadas em um atributo por vez, a possibilidade de utilizar todos atributos numéricos para aplicação do DBSCAN. Para isso, foi criado um predicado baseado no DBSCAN que chamaremos de *predicado por agrupamento*.

O algoritmo da árvore de decisão é uma extensão do Algoritmo 2, acrescentado do emprego do predicado por agrupamento. Originalmente, na Linha 7 do Algoritmo 2, considera-se o conjunto de predicados $\pi_1, \pi_2, \pi_3, ..., \pi_d$ que representa o melhor predicado associado a cada uma das características de índice $1, 2, 3, ..., d$. A DTT considera adicionalmente, o predicado π_0 representando o predicado que considera todas as características

Figura 10 - Árvore de Decisão resultante após aplicação nos dados da Figura 9.



Fonte: O próprio autor

numéricas. Tal predicado questiona se determinado dado pertence a certo grupo retornado pelo o uso do DBSCAN. Exceto pela criação do predicado por agrupamento, os algoritmos apresentados na Seção 1.1.1.1 seguem inalterados, exceto pelas mudanças abaixo:

- No Algoritmo 2, linha 7, com a inclusão do predicado π_0, a iteração incremental de j passa a variar de 0 a d.
- No Algoritmo 4, na definição da função, com a inclusão do predicado π_0, a iteração incremental de j passa a variar de 0 a d.

Algoritmo 11 - Construção da DDT.

```
FUNÇÃO CONSTRUIR (X = {X_1, X_2, ..., X_n})
    | seja T um novo nó
    | y ← ObterClassificaçãoÚnica(X)
    | se y ≠ ∅ então
    |   | associar a classe y ao nó T
    | senão
    |   | melhorPontuação ← −∞
    |   | para j ∈ {0, ..., d} fazer
    |   |   | se X não possui todos os valores iguais característica j então
    |   |   |   | (π, pontuação) ← EscolherPredicado(X, j)
    |   |   |   | se pontuação > melhorPontuação então
    |   |   |   |   | melhorPontuação ← pontuação
    |   |   |   |   | π* ← π
    |   | associar o predicado π* ao nó T
    |   | X_S ← {X_i ∈ X | X_i satisfaz π*}
    |   | X_N ← X \X_S
    |   | T.Esq ← Construir(X_S)
    |   | T.Dir ← Construir(X_N)
    | retornar T
```

Por conveniência, o Algoritmo 11 e o Algoritmo 12 apresentam o Algoritmo 2 e o Algoritmo 4 com as modificações descritas acima.

Algoritmo 12 - Determinando predicado π^* de maior ganho associado aos dados X, considerando a DDT.

```
FUNÇÃO ESCOLHERPREDICADO (X = {X_1, X_2, ..., X_n}, j ∈ {0, ..., d})
    | Seja Π_j o conjunto de predicados considerados para a característica j.
    | MelhorPontuação ← −∞
    | para π ∈ Π_j fazer
    |   | pontuação ← Ganho(X, π)
    |   | se pontuação > melhorPontuação então
    |   |   | melhorPontuação ← pontuação
    |   |   | π* ← π
    | retornar π*, melhorPontuação
```

Para finalizar, é necessário definir o predicado Π_0, que representa o conjunto de predicados considerados pela DDT associados ao agrupamento do DBSCAN.

Apresenta-se então, a definição formal do predicado por agrupamento.

3.1.1 Predicado por agrupamento

O predicado por agrupamento tem por objetivo particionar o domínio de valores naqueles que pertencem e não pertencem a um grupo g_i após a execução do DBSCAN.

Seja D o conjunto dos elementos da base de treino em um determinado nó considerando apenas as características numéricas, e $G = \{g_1, g_2, ..., g_m\}$ o agrupamento resultante da aplicação do DBSCAN ρ-aproximativo. O predicado por agrupamento, que considera o uso de todos os atributos numéricos, será da forma

$$\text{“}x \in g_i\text{”}$$

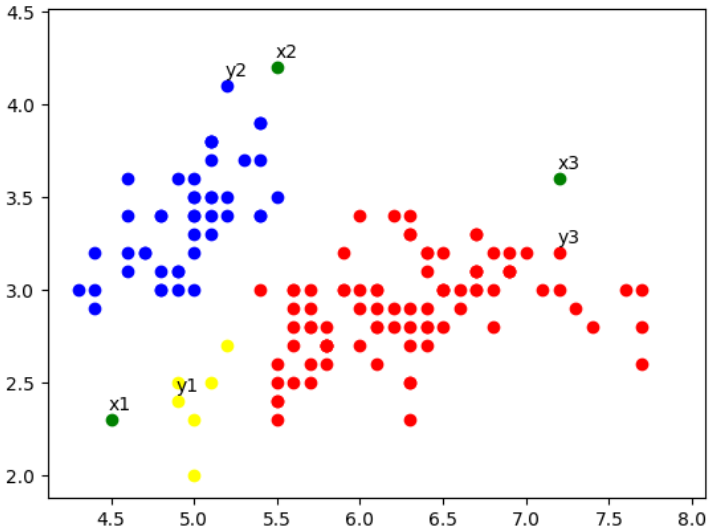
para todo $1 \leq i \leq m$, com $m = |G|$. No entanto, note que o predicado anterior ainda não está bem definido, pois cada grupo g_i do DBSCAN consiste de um subconjunto dos pontos, e portanto a fronteira desse grupo encontra-se indefinida, de modo que não se pode decidir se “$x \in g_i$”.

Há inúmeras maneiras para classificar um ponto como pertencente a um grupo ou não. Um exemplo seria a aplicação do fecho convexo nos grupos determinados pelo DBSCAN e assim, encontrar qual fecho está mais perto do ponto x. No entanto, o custo computacional e a possibilidade do grupo resultante no DBSCAN possuir forma côncava mostram que tal escolha seria inadequada. Dessa forma, optou-se por usar a distância Euclidiana. Busca-se, neste caso, o ponto $q \in D$ que possui a menor distância ao ponto

de teste x e define-se que x pertence ao grupo que contém q. Claramente, esta busca tem custo de tempo linear e o número de predicados a serem considerados é exatamente m. Formalmente, o conjunto de todos os predicados de agrupamento analisados pela DDT é

$$\Pi_0 = \{\text{“}x \in g_i\text{”} \mid 1 \leq i \leq m\}$$

Figura 11 - Exemplo do Predicado por Agrupamento.



Fonte: O próprio autor

Como ilustração, suponha que D seja o conjunto de pontos da Figura 11. Adicionalmente, suponha que o DBSCAN seja executado e retorne o agrupamento destacado na figura. Note que tal agrupamento consiste de $m = 3$ grupos, g_1, g_2 e g_3. Assim, os predicados por agrupamento que serão considerados pela DDT são:

- “$x \in g_1$”
- “$x \in g_2$”
- “$x \in g_3$”

Note, nessa mesma figura, que o ponto $x_1 \in g_1$, pois o ponto y_1 é o ponto mais próximo de x_1, e $y_1 \in g_1$. Analogamente, $x_2 \in g_2$ pois $y_2 \in g_2$ é o ponto mais próximo de x_2, e $x_3 \in g_3$ pois $y_3 \in g_3$ é o ponto mais próximo de x_3.

Apresenta-se no próximo capítulo, o experimento e os resultados do teste comparativo entre a árvore de decisão sem e com o uso do predicado por agrupamento.

4 RESULTADOS EXPERIMENTAIS

4.1 Projeto do Experimento

Nesta seção, apresentaremos como o experimento foi planejado e os resultados. O estudo baseia-se em comparar a árvore de decisão tradicional com decisões univariadas e a árvore de decisão DDT proposta nesta tese. Sendo assim, escolheu-se fazer uso da validação cruzada, do erro médio e do F1-score e o tempo de execução como processos e métricas para comparação das duas metodologias de classificação.

Para a aplicação da validação cruzada, selecionaram-se algumas variações dos hiperparâmetros associados a árvores de decisão e ao DBSCAN, são eles: $Pmax$ (grau de pureza), $Nmin$ (número mínimo de pontos que não permite uma nova divisão de um nó), ϵ (raio da bola centrada em um ponto central no DBSCAN), $MinPts$ (mínimo de pontos na cobertura de uma bola centrada em um ponto, para que esse ponto seja central) e ρ (folga no raio da bola aplicada no DBSCAN aproximativo). Os valores escolhidos foram:

- $Pmax$**:** 0,90 e 1,0;
- $Nmin$**:** 1 e 10;
- ϵ**:** Calcula-se inicialmente a menor distância entre 2 pontos de classes diferentes na base de treino. Então utiliza-se dois valores para ϵ. Optou-se escolher os dois valores, um ligeiramente abaixo de ϵ e um ligeiramente acima de ϵ, da seguinte forma. O valor mais abaixo é calculado subtraindo a distância encontrada por 10^{-z}, onde z é o número de casas decimais até o primeiro dígito significativo da menor distância. O outro valor é encontrado de forma análoga, dessa vez, somando 10^{-z}.
- $MinPts$**:** 1 e 5. Por definição do parâmetro, quando $MinPts = 1$, não há ruídos no agrupamento resultante após a aplicação do DBSCAN;
- ρ: 0,1. Verificou-se em alguns testes iniciais que a variação de ρ não gerou diferença, sendo assim optamos por utilizar apenas um único valor nos experimentos comparativos.

Utilizaram-se duas métricas, erro e F1-score, para comparação dos dois modelos. Utilizou-se o F1-score por ser uma métrica que tem bom desempenho para bases de dados desbalanceadas.

Para o cálculo médio dessas métricas, optou-se por aplicar a validação cruzada 100 vezes e, finalmente, apresentar os resultados. Além disso, optou-se também por selecionar as mesmas sementes para o processo aleatório de definição das partes de cada validação

cruzada. Ou seja, as partes das 100 validações cruzadas para gerar a árvore de decisão são as mesmas para gerar a DDT.

Outro ponto importante foi o pré-processamento transformando cada característica numérica das bases de dados em uma distribuição normal padrão com média 0 e desvio padrão 1. Este processo foi aplicado para que características com dados com maior magnitude não tenham maior influência durante o processamento do DBSCAN do que outros.

A seguir, apresentam-se as bases de dados utilizadas para estudo.

4.2 **Bases de Dados**

Nesta seção, apresenta-se as bases de dados utilizadas para a comparação entre a árvore de decisão e a DDT.

Foram escolhidas 9 bases de dados para teste, conforme a Tabela 1. A maioria das bases podem ser encontradas nos repositórios UCI *Machine Learning Repository* e OpenML (UCI Machine Learning Repository, 2023; OpenML, 2023). As duas primeiras bases escolhidas são bases sintéticas. A primeira base representa um conjunto de duas classes sobrepostas no espaço. Já a segunda base representa uma disposição que, intuitivamente, acredita-se que o DBSCAN possua alguma vantagem na detecção das respectivas classes. As outras bases escolhidas estão disponíveis publicamente, e são frequentemente empregadas como comparação de diferentes métodos de classificação.

Tabela 1 - Base de dados de teste.

Bases	**Exemplos**	**Características**	**Classes**	**Repositório**
Sintética 1	250	2	2	OpenML
Sintética 2	400	2	4	Próprio autor
Íris	150	4	3	UCI
Câncer de Mama	306	3	2	OpenML
Doação de Sangue	748	4	2	OpenML
Perfil Ambiental	111	3	2	OpenML
Estado de Sono	1024	2	4	OpenML
Notas Bancárias	1372	4	2	OpenML
Terremoto	2178	3	2	OpenML

Fonte: O próprio autor

Em seguida, na próxima seção, apresenta-se os resultados dos experimentos.

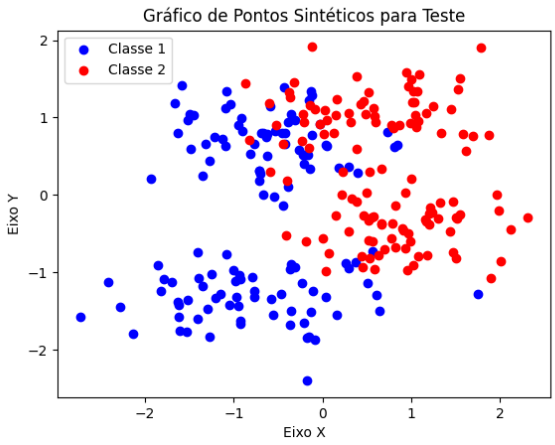
4.3 **Resultados**

Apresentam-se, nesta seção, os resultados encontrados na comparação entre a árvore de decisão e a DDT. Nas árvores de decisão apresentadas nas figuras a seguir, há a seguinte convenção: se o nó estiver na cor verde, o predicado associado é associado a uma característica numérica (Seção 1.1.1.1); se o nó estiver na cor azul, o predicado é sobre uma característica categórica (Seção 1.1.1.1); se o nó estiver na cor cinza, o predicado associado é um predicado de agrupamento (Seção 3.1.1). Os nós em cor branca representam folhas, e a legenda associada é a classe associada àquele nó. Todos os algoritmos foram implementados em Python 3.10, e executados em um computador de processador Core $i7-1165G7$ de 11^a geração, com uma frequência base de 2.8 GHz, de 4 núcleos reais e 8 virtuais, com 16 GB de memória RAM.

4.3.1 Base Sintética 1

Esta primeira base de teste está disponível no repositório OpenML. A base consiste de duas características, e duas possíveis classes. Esta base foi escolhida por representar uma disposição dos dados que pode ser comumente encontrada em problemas de classificação, conforme Figura 12.

Figura 12 - Disposição dos dados na base sintética 1.

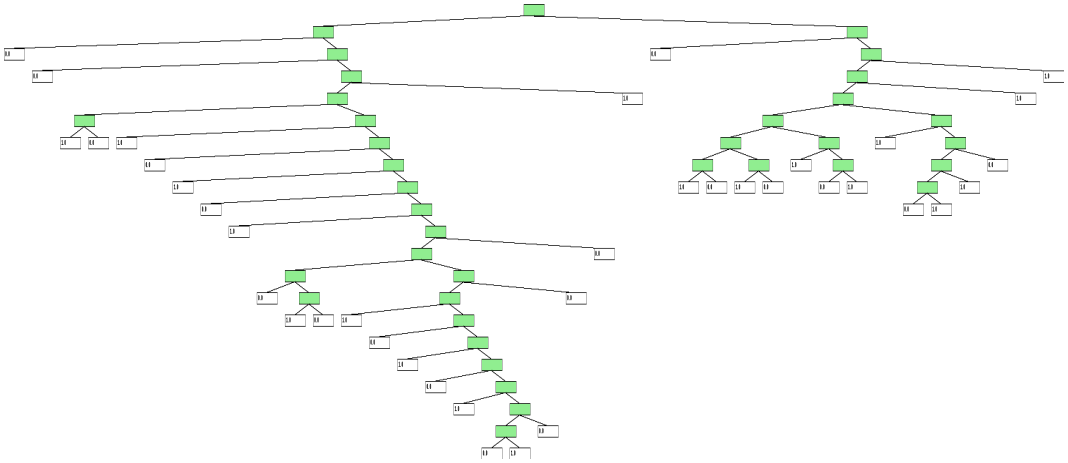


Fonte: O próprio autor

O modelo de árvore de decisão obteve a árvore resultante representada pela Figura 13, enquanto a DDT obteve a a árvore resultante representada pela Figura 14.

Na árvore de decisão, sem considerarmos os nós folhas (classificação), a árvore obtida encontrou o total de 37 nós. Por outro lado, a DDT obteve um total de 33 nós, dos quais 24 estão associados a uma única característica numérica e 9 estão associados a

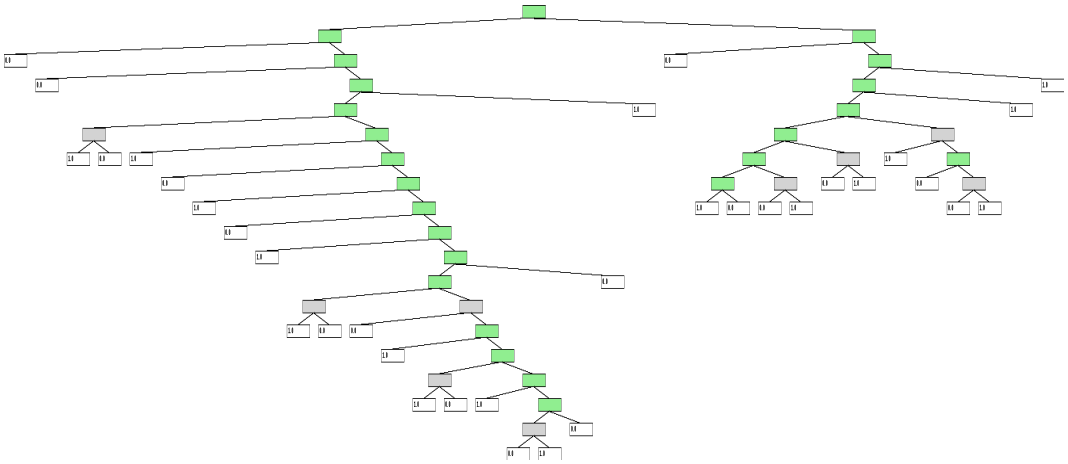
Figura 13 - Árvore de decisão na base sintética 1.



Fonte: O próprio autor

predicados de agrupamento.

Figura 14 - DDT na base sintética 1.



Fonte: O próprio autor

Além disso, podemos analisar a Tabela 2 para comparar os dois métodos através das métricas escolhidas de comparação. Além da construção de uma árvore menor, a árvore DDT apresentou um menor erro médio e um maior valor para F1-score, ou seja, apresentando uma maior capacidade de generalização. Em termos de tempo médio de classificação para um novo dado, ambos os modelos apresentaram tempos de execução similares. Por fim, o tempo médio de treino de uma DDT foi praticamente 5 vezes maior.

Tabela 2 - Tabela comparativa entre as árvores obtidas na base sintética 1.

Métricas aplicadas	Árvore de Decisão	DDT
Erro médio (%)	18,30	16,54
F1-score (%)	80,91	81,76
Erro médio - base de treino (%)	0	4,35
F1-score - base de treino (%)	96,36	96,78
Tempo médio de classificação (seg)	0,00000001500	0,00000001499
Tempo médio de treino (seg)	0,0566	0,2442

Fonte: O próprio autor

4.3.2 Base Sintética 2

A base sintética 2 é outra base com apenas dois atributos. No entanto, foi escolhida por apresentar dados que, intuitivamente, se adequam bem ao DBSCAN. A base é composta por quatro classes. As três primeiras classes apresentam formas geométricas no espaço que favorecem o DBSCAN. Já a quarta classe, é representada por ruídos para fins de teste da técnica. Sendo assim, apresenta-se primeiro a base sem a classe de ruídos.

4.3.2.1 Base Sintética 2 - Sem Ruído

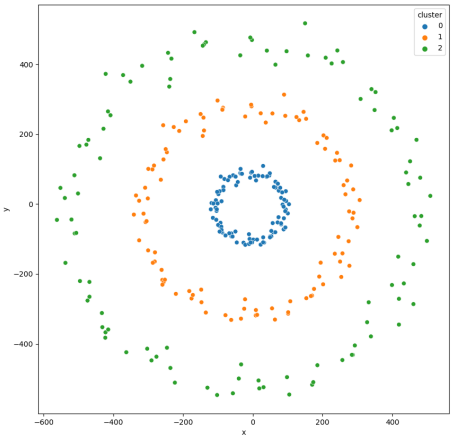
Apresentamos então a base sintética 2 sem ruído, através da Figura 15. Intuitivamente, espera-se que a árvore de decisão não encontre em algumas poucas divisões a árvore resultante. No entanto, espera-se que a DDT consiga detectar facilmente as três classes.

Em seguida, apresenta-se a árvore de decisão para a base sintética 2 sem ruído, conforme Figura 16. A árvore resultante utilizou 12 nós.

Em seguida, treinamos a DDT, e obtivemos a árvore resultante representada pela Figura 17. Como era de se esperar, o DBSCAN reconheceu bem as formas geométricas das classes resultando em uma árvore com apenas dois nós e, em todos eles, o processo optou pela escolha do DBSCAN para dividí-los.

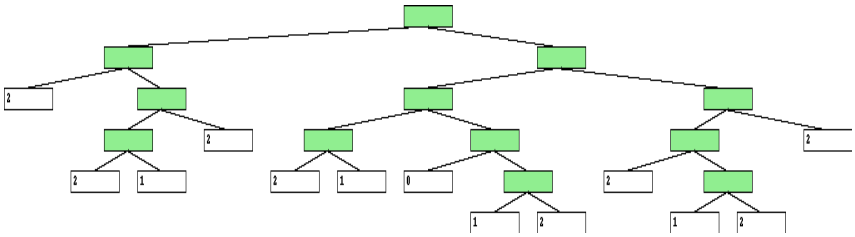
Para comparação dos modelos tem-se a Tabela 3. As duas árvores obtiveram erro médio de classificação na base de treino igual à 0. Por outro lado, ao considerarmos a base de teste, a DDT continou apresentando erro 0, enquanto a árvore de decisão apresentou 1,31% de erro. No entanto, a árvore de decisão apresentou melhor desempenho, tanto na base de treino quanto na de teste, se considerarmos o F1-score. Em relação ao tempo médio de treino, a DDT foi obtida em aproximadamente o dobro de tempo.

Figura 15 - Disposição dos dados na base sintética 2 - sem ruído.



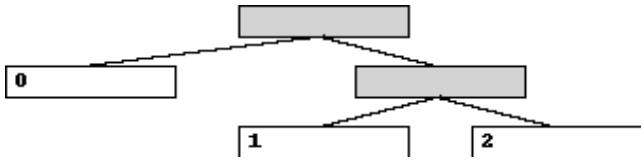
Fonte: O próprio autor

Figura 16 - Árvore de decisão na base sintética 2 sem ruído.



Fonte: O próprio autor

Figura 17 - DDT na base sintética 2 sem ruído.



Fonte: O próprio autor

Tabela 3 - Tabela comparativa entre as árvores obtidas na base sintética 2 sem ruído.

Métricas aplicadas	**Árvore de Decisão**	**DDT**
Erro médio (%)	1,31	0
F1-score (%)	98,28	96,58
Erro médio - base de treino (%)	0	0
F1-score - base de treino (%)	99,97	98,55
Tempo médio de classificação (seg)	0,00000002117	0,000004782
Tempo médio de treino (seg)	0,0627	0,1428

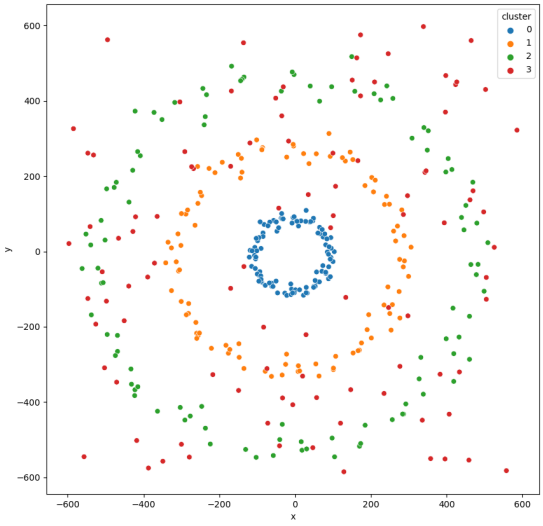
Fonte: O próprio autor

4.3.2.2 Base Sintética - Com Ruído

Apresentamos então, a base sintética 2 com ruído, através da Figura 18. Ainda espera-se que a árvore de decisão não encontre em algumas poucas divisões a árvore resultante e a DDT apresente-se melhor neste caso.

Sendo assim, apresenta-se a árvore de decisão, através da Figura 19. Já a DDT pode ser consultada através da Figura 20.

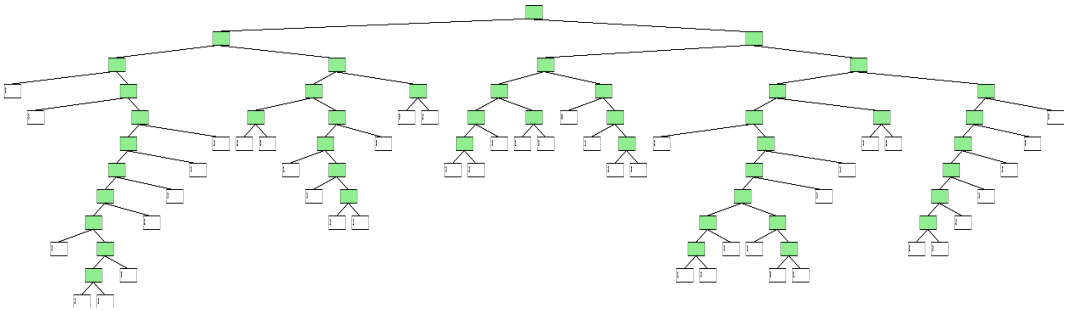
Figura 18 - Disposição dos dados na base sintética 2 - com ruído.



Fonte: O próprio autor

A árvore de decisão foi gerada com 45 nós, enquanto a DDT foi gerada com 48 nós, onde 5 nós foram com a aplicação do DBSCAN e 43 através da decisão por um único atributo numérico.

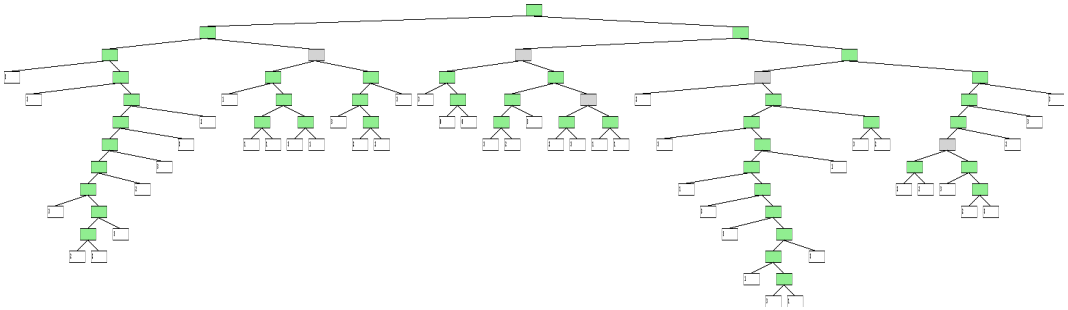
Figura 19 - Árvore de decisão na base sintética 2 com ruído.



Fonte: O próprio autor

Para comparação dos modelos tem-se a Tabela 4. Ao considerarmos a base de teste,

Figura 20 - DDT na base sintética 2 com ruído.



Fonte: O próprio autor

a DDT apresentou erro 23,93%, enquanto a árvore de decisão apresentou 25,80% de erro. O resultado considerando a métrica F1-score também demonstrou melhor desempenho com a DDT, 72,72% comparado com 72,61% da árvore de decisão. A DDT também apresentou melhor desempenho na base de treino se considerarmos as duas estatísticas. Em relação ao tempo médio de treino, a árvore DDT foi obtida em aproximadamente o dobro de tempo.

Tabela 4 - Tabela comparativa entre as árvores obtidas na base sintética 2 com ruído.

Métricas aplicadas	**Árvore de Decisão**	**DDT**
Erro médio (%)	25,80	23,93
F1-score (%)	72,61	72,72
Erro médio - base de treino (%)	10,69	8,97
F1-score - base de treino (%)	94,08	93,90
Tempo médio de classificação (seg)	0,00000001877	0,0000033736
Tempo médio de treino (seg)	0,1311	0,3382

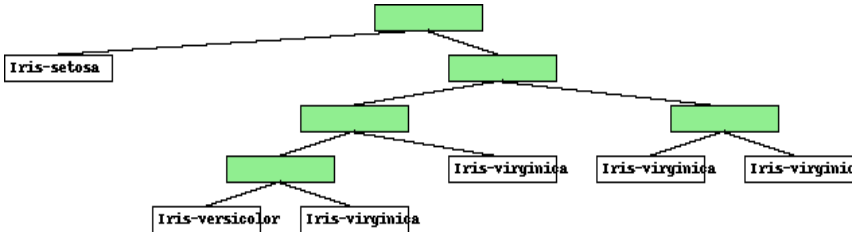
Fonte: O próprio autor

4.3.3 Base Íris

A base íris é a primeira base além das sintéticas que foi escolhida para aplicação do modelo. Esta base representa um conjunto clássico na área de aprendizado de máquina e estatística, servindo como um recurso fundamental para a compreensão e desenvolvimento de algoritmos de classificação. A base é composta por medidas de quatro características morfológicas de três espécies de flores iris: Setosa, Versicolor e Virgínica. A base de dados contém 150 amostras, cada uma com medições do comprimento e largura das sépalas e pétalas.

A árvore de decisão é apresentada pela Figura 21, enquanto a DDT é apresentada

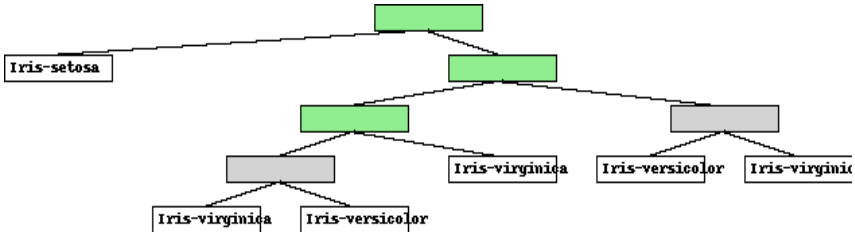
Figura 21 - Árvore de decisão na base iris.



Fonte: O próprio autor

pela Figura 22. As duas árvores foram construídas com 5 nós, no entanto, com a possibilidade da utilização do predicado por agrupamento, a DDT apresentou em 2 nós a divisão com DBSCAN.

Figura 22 - DDT na base iris.



Fonte: O próprio autor

Tabela 5 - Tabela comparativa entre as árvores obtidas na base íris.

Métricas aplicadas	**Árvore de Decisão**	**DDT**
Erro médio (%)	5,17	4,35
F1-score (%)	94,00	94,11
Erro médio - base de treino (%)	2,16	1,49
F1-score - base de treino (%)	98,02	97,97
Tempo médio de classificação (seg)	0,00000000845	0,00000000844
Tempo médio de treino (seg)	0,00367	0,13933

Fonte: O próprio autor

Em termos das métricas de comparação dos dois modelos, apresentamos a Tabela 5. A árvore de decisão, na base de teste, apresentou erro médio de 5,17% e F1-score de 94,00%, enquanto a DDT apresentou um desempenho melhor, com erro médio de 4,35% e F1-score de 94,11%. Também obervou-se melhor desempenho da DDT na base de treino considerando o erro médio de 1,49%, em comparação com a árvore de decisão com erro médio 2,16%. Já no F1-score a árvore de decisão foi sensivelmente melhor com F1-score de 98,02, em comparação a DDT com F1-score de 97,97%.

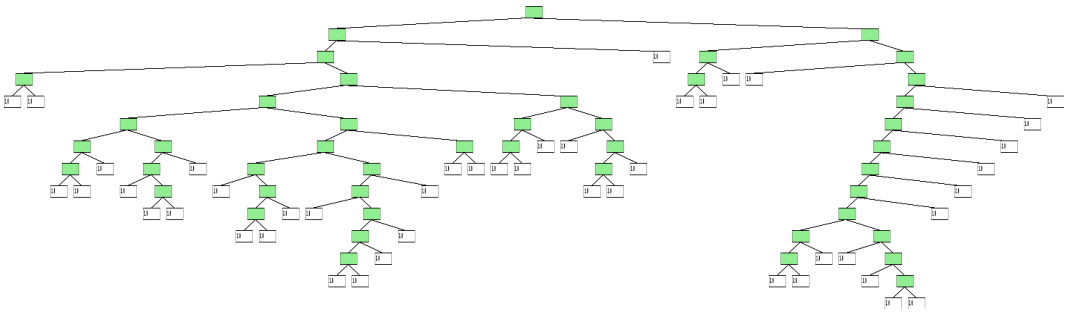
Em relação ao tempo médio de classificação, os dois modelos apresentaram valores próximos. Por outro lado, considerando o tempo médio de treino, a DDT foi aproximadamente 37 vezes mais lenta.

4.3.4 Base Câncer de Mama

A base de dados de câncer de mama é um conjunto de dados clínicos e características de tumores de câncer de mama. A base é composta por três variáveis numéricas: idade do paciente na data de operação, ano de operação e número de nódulos axilares positivos detectados.

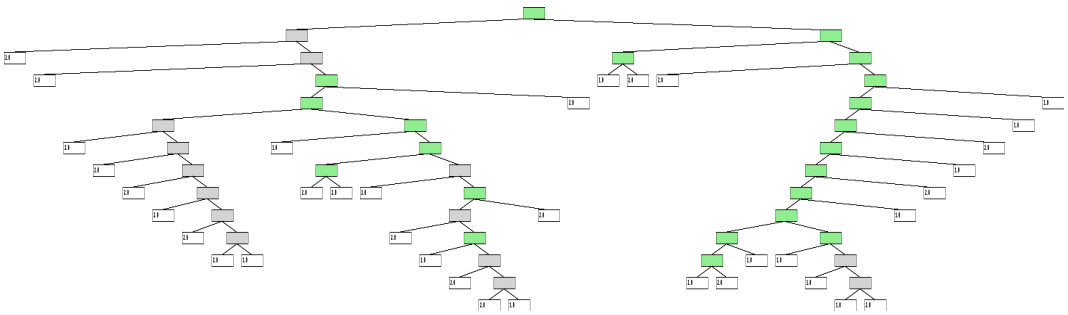
As árvores de decisão geradas pelo modelo, árvore de decisão (Figura 23) e DDT (Figura 24) são apresentadas a seguir. A árvore de decisão foi gerada através de 45 nós. Já a DDT foi gerada com 35 nós, sendo 14 nós gerados através do predicado por agrupamento.

Figura 23 - Árvore de decisão na base câncer de mama.



Fonte: O próprio autor

Figura 24 - DDT na base câncer de mama.



Fonte: O próprio autor

Para comparar as métricas de avaliação dos dois modelos temos a Tabela 6. A DDT mostrou-se com melhor desempenho tanto na base de teste quanto na base de treino. A DDT obteve, na base de teste, erro médio 30,45% e F1-score de 65,44% em comparação

a árvore de decisão que obteve, erro médio 33,03% e F1-score de 64,83%. Já na base de treino, os valores da árvore de decisão foram de: erro médio 14,44% e F1-score de 91,04%. Já para a DDT os valores foram de: erro médio 10,52% e de 92,47% para o F1-score.

Tabela 6 - Tabela comparativa entre as árvores obtidas na base câncer de mama.

Métricas aplicadas	**Árvore de Decisão**	**DDT**
Erro médio (%)	33,03	30,45
F1-score (%)	64,83	65,44
Erro médio - base de treino (%)	14,44	10,52
F1-score - base de treino (%)	91,04	92,47
Tempo médio de classificação (seg)	0,000000059908	0,000000082701
Tempo médio de treino (seg)	0,03086	0,65443

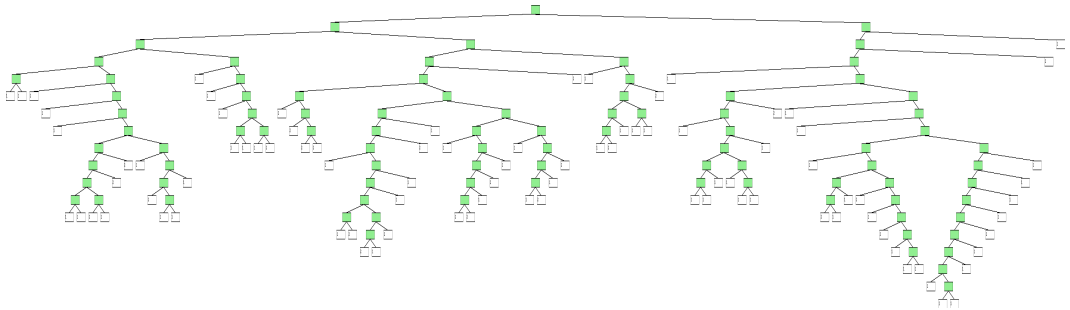
Fonte: O próprio autor

Em termos de tempo médio de classificação, as duas árvores classificam um ponto rapidamente. Já ao considerarmos o tempo médio de treino, a DDT foi aproximadamente 20 vezes mais lenta.

4.3.5 Base Doação de Sangue

A base de dados de doação de sangue é um conjunto de informações que contém dados relacionados a doações de sangue por indivíduos, sendo frequentemente utilizado em contextos de análise de comportamento de doadores. Ela compreende informações de doação, como a frequência de doações, o tempo desde a última doação, o número total de doações e o fato de um doador ter feito uma doação na campanha específica.

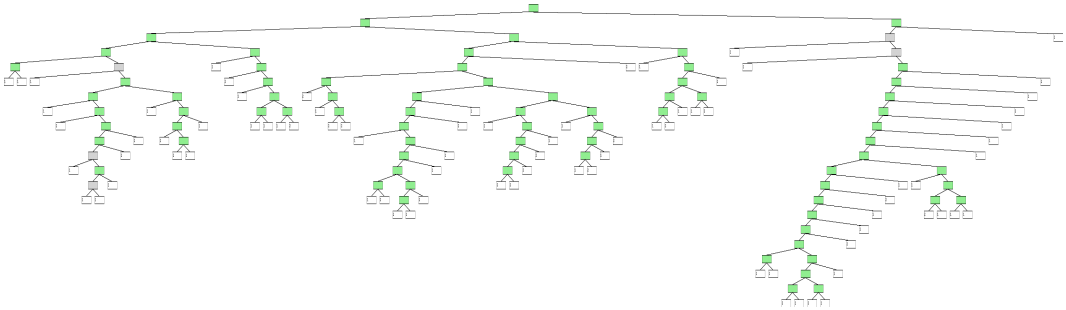
Figura 25 - Árvore de decisão na base doação de sangue.



Fonte: O próprio autor

As árvores de decisão geradas, árvore de decisão e DDT, podem ser encontradas, respectivamente, nas Figuras 25 e 26. A árvore de decisão foi gerada com 89 nós. Já a DDT foi gerada com 81 nós, com 5 nós aplicando o predicado por agrupamento.

Figura 26 - DDT na base doação de sangue.



Fonte: O próprio autor

Apresenta-se então, a Tabela 7 para comparação dos modelos. Assim como na base de dados de câncer de mama, a DDT também apresentou desempenho melhor na base de treino e teste para a base de dados de doação de sangue. Em relação a base de teste, a DDT obteve erro médio de 39,54% enquanto a árvore de decisão obteve erro de 40,99%. Em relação ao F1-score, a DDT obteve 59,39% enquanto a árvore de decisão obteve 59,05%. Na base de treino o comportamento das estatísticas para os dois modelos foi similar e a DDT obteve melhor desempenho.

Tabela 7 - Tabela comparativa entre as árvores obtidas na base doação de sangue.

Métricas aplicadas	**Árvore de Decisão**	**DDT**
Erro médio (%)	40,99	39,54
F1-score (%)	59,05	59,39
Erro médio - base de treino (%)	13,66	11,16
F1-score - base de treino (%)	90,46	90,98
Tempo médio de classificação (seg)	0,0000000203	0,00000006613
Tempo médio de treino (seg)	0,0464	1,812

Fonte: O próprio autor

Em relação ao tempo de classificação, todos os dois modelos classificam rapidamente um único ponto. Já em termos do tempo de treino, a DDT é, aproximadamente, 40 vezes mais lenta.

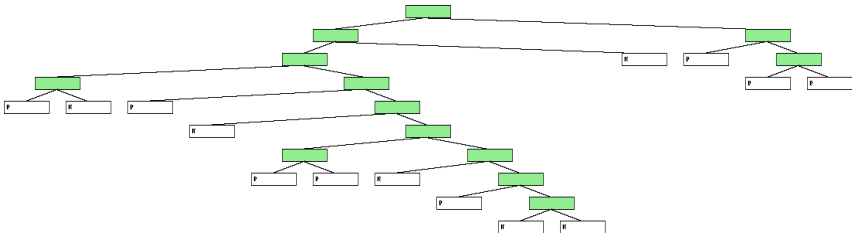
4.3.6 Base Perfil Ambiental

Esta base é composta por três atributos, ozônio, radiação e temperatura, para explicar a variável vento. Em resumo, o ozônio, a temperatura e a radiação estão envolvidos em processos atmosféricos que afetam diretamente a distribuição de pressão e temperatura na atmosfera, influenciando assim a variabilidade do vento. A compreensão desses

fatores é essencial para prever e entender os padrões climáticos e atmosféricos que afetam o comportamento do vento. Nesta base, a variável vento foi binarizada para tratamento de problemas de classificação.

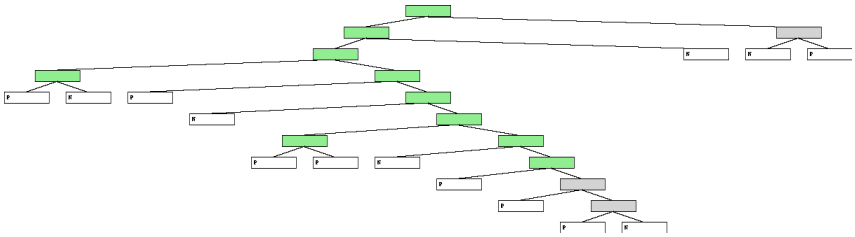
Apresenta-se então as árvores resultantes da aplicação dos dois modelos. A árvore de decisão é apresentada pela Figura 27 e a DDT é apresentada pela Figura 28. A aplicação da árvore de decisão gerou uma árvore com 13 nós. A DDT encontrou a árvore resultante com 13 nós, no entanto, desta vez, 3 nós trocaram a decisão em um único atributo pelo predicado por agrupamento.

Figura 27 - Árvore de decisão na base perfil ambiental.



Fonte: O próprio autor

Figura 28 - DDT na base perfil ambiental.



Fonte: O próprio autor

Tabela 8 - Tabela comparativa entre as árvores obtidas na base perfil ambiental.

Métricas aplicadas	**Árvore de Decisão**	**DDT**
Erro médio (%)	31,92	30,18
F1-score (%)	66,17	65,95
Erro médio - base de treino (%)	0,02	0,07
F1-score - base de treino (%)	93,42	93,68
Tempo médio de classificação (seg)	0,00000001136	0,00000002268
Tempo médio de treino (seg)	0,014	0,0829

Fonte: O próprio autor

Apresenta-se então, a Tabela 8. Comparando-se a aplicação dos modelos na base de treino e na base de teste, a DDT apresentou uma melhor generalização considerando

o erro médio de 30,18%, em comparação a árvore de decisão com erro médio de 31,92%. No entanto, a DDT obteve desempenho pior se considerarmos a métrica F1-score.

Em relação ao tempo de classificação, novamente os dois modelos classificam rapidamente um novo dado. Por outro lado, em relação ao tempo de treino da árvore, a DDT foi aproximadamente 8 vezes mais lenta.

4.3.7 Base Estado de Sono

Esta base de dados descreve o estado de sono de um recém-nascido através da frequência cardíaca e temperatura. A base é composta por dois atributos numéricos e quatro valores (de 1 à 4) representado o estado de sono do recém-nascido.

A árvore de decisão resultante foi construída com 151 nós. Já a DDT foi construída com 161 nós, dos quais, 26 nós aplicaram o predicado por agrupamento. Optou-se por não incluir as figuras referentes as árvores pela quantidade de nós gerada.

Temos então a Tabela 9. A DDT só obteve melhor desempenho na base de teste se considerarmos o erro médio de 67,06% comparado com o erro da árvore de decisão, 69,91%. No entanto, na base de treino a DDT obteve erro médio maior. Além disso, ao observarmos a estatística F1-score, tanto na base de teste quanto na base de treino, a DDT obteve valor menor, logo, com pior desempenho.

Tabela 9 - Tabela comparativa entre as árvores obtidas na base estado de sono.

Métricas aplicadas	**Árvore de Decisão**	**DDT**
Erro médio (%)	69,91	67,06
F1-score (%)	26,12	26,08
Erro médio - base de treino (%)	33,10	35,18
F1-score - base de treino (%)	70,56	70,00
Tempo médio de classificação (seg)	0,000000025707	0,000000017495
Tempo médio de treino (seg)	0,045	0,3424

Fonte: O próprio autor

Em relação ao tempo de treino, a DDT foi aproximadamente 7 vezes mais lenta. Por outro lado, um novo dado é classificado nas duas árvores rapidamente.

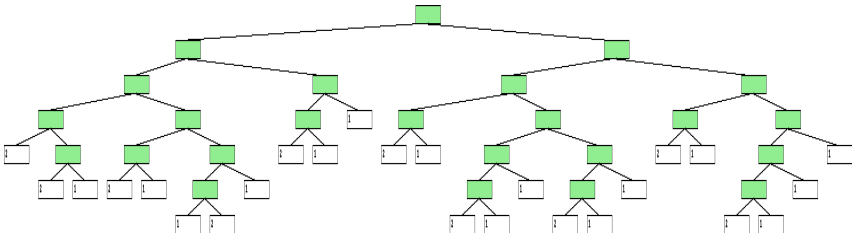
4.3.8 Base Notas Bancárias

A base de dados de notas bancárias é composta de quatro atributos numéricos referentes a uma imagem de uma cédula monetária (nota de dinheiro) e o problema consiste em classificar se a nota é uma falsificação ou genuína. Ou seja, o problema possui

quatro atributos numéricos e duas classes (nota falsa ou verdadeira).

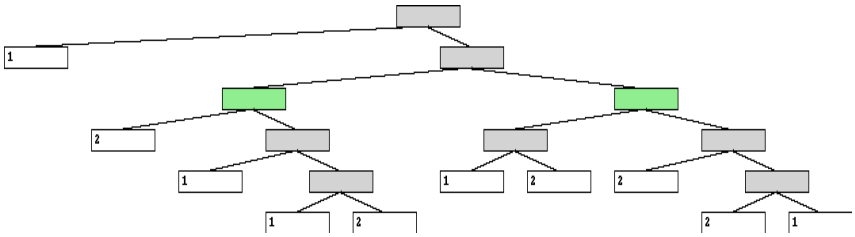
A árvores de decisão e a DDT são, respectivamente, representadas pelas Figuras 29 e 30. A árvore de decisão foi construída com 24 nós. Por outro lado, a DDT gerada foi construída com 9 nós, dos quais 7 utilizaram a decisão pelo predicado por agrupamento. Dentre todas as bases, em relação a quantidade de nós e uso do DBSCAN, esta foi a que utilizou o DBSCAN na maioria de seus nós, além de ter reduzido a quantidade de nós na árvore resultante.

Figura 29 - Árvore de decisão na base notas bancárias.



Fonte: O próprio autor

Figura 30 - DDT na base notas bancárias.



Fonte: O próprio autor

Tabela 10 - Tabela comparativa entre as árvores obtidas na base notas bancárias.

Métricas aplicadas	**Árvore de Decisão**	**DDT**
Erro médio (%)	1,33	0,43
F1-score (%)	95,62	97,10
Erro médio - base de treino (%)	0	0
F1-score - base de treino (%)	96,97	97,97
Tempo médio de classificação (seg)	0,000000005672	0,0000176
Tempo médio de treino (seg)	1,673	3,62

Fonte: O próprio autor

Para comparação dos modelos temos a Tabela 10. Na base de treino, os dois métodos encontraram erro de 0%, no entanto, a DDT encontrou F1-score maior que a árvore de decisão. Já na base de teste, a DDT encontrou erro menor e F1-score maior

(0,43% e 97,10, respectivamente) que a árvore de decisão (1,33% e 95,62, respectivamente). Ou seja, a DDT obteve melhor desempenho. Cabe ressaltar, novamente, que esse foi a primeira base em que o DBSCAN foi utilizado mais vezes para divisão de um nó do que as decisões de apenas um único atributo.

4.3.9 Base Terremoto

A base terremoto é uma base para descrever a gravidade de um abalo sísmico através de três atributos numéricos: profundidade focal, latitude e longitude. A classificação é feita em dois fatores, baseados na escala ritcher e foi binarizada para utilização em testes de problemas de classificação.

A árvore de decisão encontrada foi gerada com 727 nós. Já a DDT foi gerada com 408 nós, sendo 110 nós com a aplicação do predicado por agrupamento. Optou-se por não incluir as figuras referentes as árvores pela quantidade de nós gerada.

Para comparar as duas árvores temos a Tabela 11. Neste caso, considerando a base de teste, a DDT obteve melhor desempenho no erro médio (47,91%). Por outro lado, considerando a métrica F1-score, a árvore de decisão obteve melhor resultado (51,29). Em relação a base de treino, há indícios de que a árvore de decisão sofreu o processo de sobreajuste. Logo, apesar da DDT possuir maior erro na base de treino, resultou em uma melhor generalização na base de teste.

Tabela 11 - Tabela comparativa entre as árvores obtidas na base terremoto.

Métricas aplicadas	**Árvore de Decisão**	**DDT**
Erro médio (%)	48,26	47,91
F1-score (%)	51,29	51,13
Erro médio - base de treino (%)	0,01	15,50
F1-score - base de treino (%)	93,13	92,14
Tempo médio de classificação (seg)	0,00000005026	0,0000001677
Tempo médio de treino (seg)	6,824	11,288

Fonte: O próprio autor

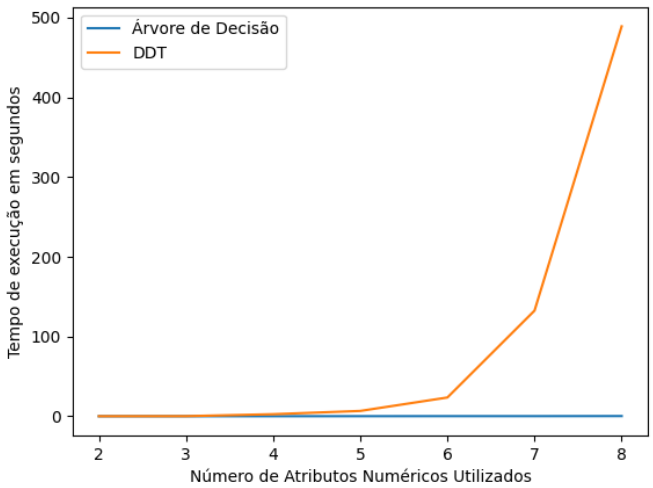
Em relação ao tempo de classificação, novamente as duas árvores classificam um novo ponto rapidamente. Por outro lado, em relação ao tempo médio de treino de uma nova árvore, a DDT foi, aproximadamente, 2 vezes mais lenta.

Em seguida, utiliza-se uma base de dados com oito atributos para uma análise mais ampla em relação ao tempo de treino de uma árvore de decisão e da DDT. Para isso, relacionamos esse tempo com o acréscimo de cada atributo ao gerar uma nova árvore de decisão.

4.3.10 Impacto do Número de Características na DDT

Para análise do custo de tempo do treino da árvore de decisão e da DDT, utilizou uma base com 8 atributos numéricos. O processo de treinamento da árvore de decisão foi aplicado, utilizando dois atributos, depois três, e assim por diante, até utilizar todos os oito atributos. O tempo para os tipos de árvore foi computado e relacionado na Figura 31. Apesar do tempo esperado linear do DBSCAN ρ-aproximativo, a Figura 31 corrobora com a previsão de tempo exponencial com o aumento do número de características. A figura sugere que a DDT possui tempo médio de treinamento similares àqueles das árvores de decisão para até 5 ou 6 características. A partir daí, o tempo de treinamento é percebido como mais custoso.

Figura 31 - Análise de evolução do tempo de treino com o aumento de características numéricas disponíveis.



Fonte: O próprio autor

Por fim, apresenta-se então, a conclusão desta tese.

CONCLUSÃO E PROPOSTA

Estudos relacionados às árvores de decisão seguem gerando discussões e contribuições acerca do modelo inicialmente proposto na década de 1980. As primeiras árvores de decisão, chamadas de univariadas, utilizavam uma única característica para avaliação do predicado de divisão de um nó. Em geral, as diferenças entre as árvores univariadas davam-se pelo critério utilizado para determinar se um nó deveria torna-se uma folha ou ser expandido em uma subárvore.

Posteriormente surgiram as árvores multivariadas e híbridas. A primeira surge motivada pelo uso de mais de um atributo para decisão. Neste caso, divisões lineares ou não lineares podem ser implementadas para divisão de um nó. Nas divisões lineares, um hiperplano envolvendo todos os atributos dos dados é empregado para separar os dados. No entanto, a busca pelo melhor hiperplano que divide os dados é um problema custoso em termos de complexidade. Sendo assim, é proposto na literatura a árvore híbrida que pode ser composta por divisões utilizando mais de um atributo e divisões que contenham somente um atributo.

Na última década novas formas de dividir o nó foram implementadas. Kumar et al. (KUMAR; GOPAL, 2010) propuseram o uso da árvore de decisão para acelerar o processo de classificação de novos dados pelo SVM. Além disso, Gaddam et al. e Wang et al. propuseram o uso da técnica de agrupamento k-médias para a divisão do nó. O primeiro, aplica o k-médias para a determinação de um agrupamento e determina uma árvore de decisão univariada ID3 para os dados associados a cada grupo encontrado. Já o segundo algoritmo, propõe utilizar o método de agrupamento em cada nó da árvore como processo de divisão do nó, ao invés dos critérios univariados usuais. O processo encerra-se quando os nós filhos atingirem o grau de pureza estabelecido. Naquela proposta, novos dados a serem classificados são atribuídos a grupos pela proximidade do centróide de cada grupo. Adicionalmente, os autores propuseram encontrar um hiperplano através das coordenadas dos centróides para acelerar o processo de classificação.

Tanto os algoritmos de árvore de decisão quanto os de agrupamento são fortemente baseados em conceitos geométricos. Há na literatura extenso estudo de algoritmos randomizados e aproximativos para problemas geométricos. Os algoritmos randomizados ou aproximativos, com respeito aos algoritmos correspondentes determinísticos, ora trazem ganhos na complexidade esperada, ora simplificam sua implementação. Por outro lado, como exemplificado durante a apresentação do DBSCAN, os algoritmos de classificação ou agrupamento podem possuir complexidades elevadas, sendo um motivo de preocupação comum com respeito à quantidade de dados na base de treinamento ou a dimensionalidade desses dados (número de características). Esta tese tem então como motivação geral a aplicação de um método randomizado ou aproximativo em um processo de agrupamento

ou classificação.

A técnica de agrupamento não supervisionado DBSCAN possui a capacidade de descobrir grupos com formas arbitrárias. Tal capacidade pode ser vantajosa ao compará-lo com o método k-médias que sempre retornará grupos semelhantes a esferas. Em contrapartida, o DBSCAN é custoso computacionalmente para um número razoável de dados de treinamento. Com a intenção de reduzir a complexidade do DBSCAN, Gan et al. propôs um algoritmo aproximativo para o problema e que possui complexidade de tempo esperada de $O(n)$.

Esta tese integra o método aproximativo DBSCAN ρ-aproximado junto ao modelo da árvore de decisão, modelo que chamamos de DBSCAN Decision Tree (DDT).

No Capítulo 1, apresenta-se o referencial teórico necessário para o entendimento de todos os algoritmos de classificação, agrupamento, randomizados e aproximativos que são apresentados nos capítulos subsequentes. Em especial, apresenta-se alguns algoritmos geométricos acompanhados de suas respectivas soluções randomizadas. Além disso, considerando os métodos de classificação, apresenta-se métricas para avaliação de um modelo treinado.

No Capítulo 2, apresenta-se uma alternativa aproximativa do DBSCAN (DBSCAN ρ-aproximado). Também faz-se uma breve revisão de literatura acerca de alguns trabalhos sobre árvore de decisão e suas variantes.

No Capítulo 3, apresentamos a árvore de decisão DDT proposta por esta tese, com a criação de um novo tipo de predicado, o predicado por agrupamento. Neste predicado, todas as características numéricas são consideradas. A árvore de decisão proposta fará uso de predicados categóricos, númericos e predicados por agrupamento, com o uso do DBSCAN. Optou-se pela utilização do método aproximativo do DBSCAN, por considerarmos que o tempo de execução do algoritmo determinístico é quadrático. Sendo assim, pela complexidade de tempo esperada linear do DBSCAN aproximativo, optou-se por utilizá-lo nesta nova proposta de predicado e de árvore de decisão.

No Capítulo 4 apresentou-se a proposta do experimento, a base de dados utilizada e os resultados experimentais. Os resultados mostraram-se promissores. Em todos os casos o DBSCAN foi utilizado quando foi permitido a árvore fazer uso do predicado por agrupamento. Além disso, na maioria dos experimentos, a árvore com DBSCAN demonstrou melhor em relação a árvore sem DBSCAN, considerando as métricas de erro de classificação e F1-score. Por outro lado, ao aumento de características numéricas utilizadas, a técnica com DBSCAN tem um aumento exponencial no tempo de treino da árvore.

Alguns pontos tornam-se importantes para aprimoramento da técnica e propostas para trabalhos futuros. Em nossa proposta, utilizamos a menor distância entre dois exemplos de treinamento de classes diferentes para definição de qual ϵ utilizar ao longo do processo. No entanto, calculado o valor, ele é utilizado em todas as instâncias da árvore.

Não há, neste caso, uma garantia de que este seja o melhor valor em todas as instâncias. Sendo assim, há espaço para um aprimoramento de encontrar o valor de ϵ que maximize a possibilidade do uso do predicado por agrupamento em um determinado nó.

Outro ponto importante, é a ideia de aplicar uma seleção de características utilizáveis pelo DBSCAN. Ao considerarmos que o aumento do custo de tempo é exponencial ao aumentarmos a quantidade de características numéricas, é possível que a seleção de algumas delas possa também aprimorar a técnica. Além disso, também não há garantia de que o uso de todos os atributos dá ao predicado por agrupamento a melhor versão para competir com as decisões com apenas um atributo. Mais especificamente, a ideia consiste em selecionar apenas um conjunto pequeno de características numéricas, de forma aleatória, para o treinamento do DBSCAN, ao invés de todas as características. O sorteio dessas características é refeito em cada nó. Além de diminuir o tempo de execução, há a vantagem de diminuir a correlação dos resultados do DBSCAN entre os diferentes nós.

Outro fator que pode ter influenciado no aumento do tempo da árvore com DBSCAN é a implementação dos algoritmos na linguagem Python. O uso de linguagem mais eficientes, como C/C++, pode ser conveniente para a avaliação comparativa da DDT com árvores de decisão presentes em pacotes pré-programados de aprendizado de máquina.

Sendo assim, propomos para trabalhos futuros: a implementação da árvore em novo ambiente, a possibilidade da seleção de características para contornar o aumento exponencial do tempo de treino e a melhor seleção de ϵ para aumentar ainda mais a possibilidade da escolha do predicado por agrupamento para particionar o nó.

REFERÊNCIAS

BANYASSADY, B.; MULZER, W. A simple analysis of Rabin's algorithm for finding closest pairs. In: EUROPEAN WORKSHOP ON COMPUTATIONAL GEOMETRY. Malmo, 2007.

BENTLEY, J.; SHAMOS, M. Divide-and-conquer in multidimensional space. In: ACM. *Proceedings of the eighth annual ACM symposium on Theory of computing*. New York, 1976. p. 220–230.

BERG, M.; CHEONG, O.; KREVELD, C. Van; OVERMARS, M. *Computational geometry algorithms and applications*. Primeira edição. Berlin: Spinger, 1998.

BISHOP, C. *Pattern Recognition and Machine Learning*. New York: Springer, 2006.

BREIMAN, L. *Classification and regression trees*. Primeira edição. New York: Routledge, 1984.

BUCHIN, K.; MULZER, W. Delaunay triangulations in $O(\text{sort}(n))$ time and more. *Journal of the ACM (JACM)*, ACM New York, v. 58, n. 2, p. 1–27, 2011.

CHRYSTAL, G. On the problem to construct the minimum circle enclosing n given points in the plane. *Proceedings of the Edinburgh Mathematical Society*, v. 3, p. 30–33, 1885.

ELZINGA, J.; HEARN, D. Geometrical solutions for some minimax location problems. *Transportation Science*, INFORMS, v. 6, n. 4, p. 379–394, 1972.

ESTER, M.; SANDER, J.; KRIEGEL, H.; XU, X. Density-based clustering in spatial databases: The algorithm gdbscan and its applications. *Data mining and knowledge discovery*, Springer, v. 2, p. 169–194, 1998.

FARIA, L.; OLIVEIRA, F.; PINTO, P.; SZWARCFITER, J. *Ciência de dados: algoritmos e aplicações*. Rio de Janeiro: IMPA, 2021.

FIGUEIREDO, C.; FONSECA, G.; LEMOS, C.; Sá, V. *Introdução aos Algoritmos Randomizados*. Rio de Janeiro: IMPA, 2007.

GADDAM, S.; PHOHA, V.; BALAGANI, K. K-means+ id3: A novel method for supervised anomaly detection by cascading k-means clustering and ID3 decision tree learning methods. *IEEE transactions on knowledge and data engineering*, IEEE, v. 19, n. 3, p. 345–354, 2007.

GAN, J.; TAO, Y. Dbscan revisited: Mis-claim, un-fixability, and approximation. In: SIGMOD. *Proceedings of the 2015 ACM SIGMOD international conference on management of data*. Hong Kong, 2015. p. 519–530.

GERON, A. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. [S.l.]: O'Reilly Media, 2019.

HEATH, David George. *A geometric framework for machine learning*. Maryland: The Johns Hopkins University, 1993.

HROMKOVIČ, J. *Design and Analysis of Randomized Algorithms*. Primeira edição. Berlin: Springer, 2004.

JAMES, G.; WITTEN, G.; HASTIE, T.; TIBSHIRANI, R. *Introduction to Statistical Learning*. : Springer, 2013.

KUMAR, M.; GOPAL, M. A hybrid SVM based decision tree. *Pattern Recognition*, Elsevier, v. 43, p. 3977–3987, 2010.

LOH, W.; SHIH, Y. Split selection methods for classification trees. *Statistica sinica*, JSTOR, p. 815–840, 1997.

LOH, W.; VANICHSETAKUL, N. Tree-structured classification via generalized discriminant analysis. *Journal of the American Statistical Association*, Taylor & Francis, v. 83, n. 403, p. 715–725, 1988.

MACQUEEN, J. Some methods for classification and analysis of multivariate observations. In: BERKELEY SYMPOSIUM ON MATHEMATICAL STATISTICS AND PROBABILITY. *Theory of Statistics*. California, 1967. v. 1, p. 281–297.

MONARD, M.; BARANAUSKAS, J. Conceitos sobre aprendizado de máquina. *Sistemas inteligentes-Fundamentos e aplicações*, v. 1, n. 1, p. 32, 2003.

MOTWANI, R.; RAGHAVAN, P. *Randomized algorithms*. Cambridge: Cambridge university press, 2007.

MURTHY, S.; KASIF, S.; SALZBERG, S. A system for induction of oblique decision trees. *Journal of artificial intelligence research*, v. 2, p. 1–32, 1994.

OpenML. *OpenML*. 2023. <https://www.openml.org/>. Accessado em 28 de novembro.

QUINLAN, J. Program for machine learning. *C4.5*, Morgan Kaufmann, California, 1993.

RABIN, M. *Probabilistic Algorithms Algorithms and Complexity: New Directions and Recent Results*. New York: Academic Press New York, 1976.

SHAMOS, M.; HOEY, D. Closest-point problems. In: IEEE. *16th Annual Symposium on Foundations of Computer Science (sfcs 1975)*. California, 1975. p. 151–162.

SHAMOS, M.; PREPARATA, F. *Computational geometry: an introduction*. New York: Springer-Verlag New York, 1985.

UCI Machine Learning Repository. *UCI Machine Learning Repository*. 2023. <https://archive.ics.uci.edu/ml/index.php>. Accessado em 21 de novembro.

WANG, F.; WANG, Q.; NIE, F.; LI, Z.; YU, W.; REN, F. A linear multivariate binary decision tree classifier based on k-means splitting. *Pattern Recognition*, Elsevier, v. 107, p. 107521, 2020.

YILDIZ, O.; ALPAYDIN, E. Linear discriminant trees. *International Journal of Pattern Recognition and Artificial Intelligence*, World Scientific, v. 19, n. 03, p. 323–353, 2005.