Classificação de amostras por meio de estratégia baseada em grafos: Optimum-Path Forest (OPF).

Antonio Rola Neto
Instituto de Cinência e Tecnologia
Universidade Federal de Itajubá
Itabira, MG
RA: 2019002925
Email: antoniorolaneto@unifei.edu.br

Levi Medeiros Magny
Instituto de Cinência e Tecnologia
Universidade Federal de Itajubá
Itabira, MG
RA: 2019008642
Email: levi_mgy@unifei.edu.br

Rhuan Lucas Alves Soares
Instituto de Cinência e Tecnologia
Universidade Federal de Itajubá
Itabira, MG
RA: 2019006156
Email: rhuanlucas123@unifei.edu.br

***Resumo*—Este relatório apresenta o processo de desenvolvimento de um algoritmo de reconhecimento de padrões supervisionado dentro da área de Inteligência Artificial que utiliza uma estratégia baseada em grafos, isto é, uma floresta de caminhos ótimos. O objetivo principal é utilizar os conhecimentos em análise de algoritmos e teoria de grafos obtidos durante as aulas a fim de modelar e analisar o algoritmo proposto. Ao final da execução do projeto, o classificador foi implementado e classificou com sucesso as amostras fornecidas, duas das quais são apresentadas neste artigo.**

***Índice de Termos*—Reconhecimento de Padrões, Grafos, Inteligência Artificial.**

***Abstract*—This report presents the development process of a pattern recognition algorithm within the Artificial Intelligence area that uses a graph-based strategy called optimum-path forest. The main objective is to use the knowledge in algorithm analysis and graph theory obtained during classes in order to model and analyze the proposed algorithm. At the end of project execution, the classifier was implemented and successfully classified the samples provided, two of which are presented in this article.**

***Index Terms*—Pattern Recognition, Graphs, Artificial Intelligence.**

I. Introdução

SEGUNDO Papa *et al.* (2008) [1], os problemas de reconhecimento de padrões podem ser resumidos em duas partes fundamentais: A primeira consiste na identificação de grupos naturais (clusters) composto por amostras que possuem o mesmo padrão. A segunda é a classificação de uma determinada amostra em uma das classes existentes (labels). O processo de treinamento desses algoritmos pode ser executado a partir de amostras não classificadas (Aprendizado não Supervisionado), amostras classificadas (Aprendizado Supervisionado), ou parte das amostras classificada e outra parte não classificada (Aprendizado Semi-Supervisionado) [1]. O algoritmo implementado nesse projeto é focado no aprendizado supervisionado.

O Classificador Floresta de Caminhos Ótimos (OPF) apresenta uma proposta para classificação de padrões supervisionada utilizando conceitos da teoria dos grafos [2]. Essa solução é rápida, simples, independente de parâmetros, não faz nenhuma suposição sobre a forma das classes e consegue lidar com classes sobrepostas até certo grau [1]. Nessa estratégia, cada árvore da floresta é composta por amostras de mesma classe, sendo que, para gerar o conjunto de árvores, é aplicado um processo de maximização do mapa de conectividade entre amostras [2], o que significa que cada vértice recebe um valor que representa sua conexão com seu grupo. No caso deste trabalho, o custo aplicado a cada amostra será o valor da maior aresta ao longo do menor caminho entre a amostra e seu respectivo protótipo.

A. Estratégia de treinamento

Como foi já foi explicado, e estratégia do OPF pode ser dividida em duas partes: o treinamento e o teste/classificação de novas amostras. Para fins de compreensão do algoritmo implementado, essas duas etapas serão discutidas de forma suscinta nessa subseção, a começar pelo treinamento.

Para começar, o algoritmo recebe como entrada um conjunto de amostras (Z_1) que serão os vértices do grafo, cada uma delas é associada com uma classe e um vetor de características ($v(s)$) utilizado para o cálculo de distância ($d(s,t)$), sendo s e t diferentes amostras de Z_1 [1]. A partir dessas entradas é criado um grafo completo com arestas ponderadas de acordo com a distância entre os vértices como apresentado na Fig. 1. Nesse projeto será usada a distância euclidiana, que também é a mais comum, no entanto outras funções de distância são aceitas.

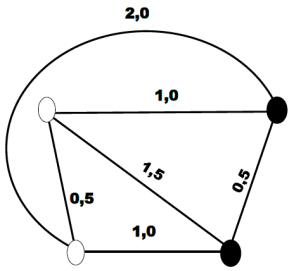

Figura 1. Ilustração de um grafo completo composto por amostras de duas classes diferentes. Fonte: Retirado de [2]

Criado o grafo, o próximo passo é obter a sua árvore geradora mínima (AGM). Esse passo serve para que sejam mantidas apenas as menores arestas, de modo a deixar o cálculo da conectividade entre os vértices mais preciso. A Fig. 2 ilustra o grafo após a execução do AGM. Os detalhes dessa implementação serão apresentados na próxima seção.

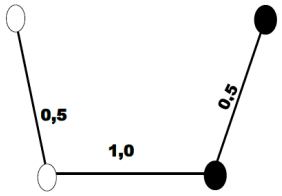

Figura 2. Ilustração do grafo após a execução do algoritmo AGM. Fonte: Retirado de [2]

Com o grafo reduzido à AGM, é possível realizar uma busca pelas arestas que ligam vértices de classes diferentes, e estes vértices serão protótipos em suas respectivas classes. Cada protótipo P irá tentar conquistar os vértices da árvore em que está inserido. O custo $C(s)$ de cada vértice s será definido como o peso da maior aresta contida no caminho entre s e o protótipo que o conquistou ($P(s)$). A Fig. 3 ilustra o resultado desses passos. Quanto à inicialização das estruturas no algoritmo, tem-se que o custo C_1 de cada vértice protótipo é inicializado em zero (0), enquanto cada vértice não protótipo é inicializado em infinito (∞) [1], [2].

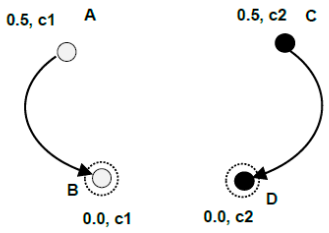

Figura 3. Grafo transformado em uma floresta de caminhos ótimos após o treinamento. Fonte: Retirado de [2]

B. Estratégia de classificação

O algoritmo de classificação recebe como entrada a OPF e um conjunto Z_2 de amostras para serem classificadas. Nesse processo, uma amostra t é um vértice de Z_2 que deve ser conquistado. Para isso o algoritmo utiliza os vértices da OPF e calcula a distância $d(s,t)$ entre t e cada s [1]. O cálculo utilizado para obtenção do custo de classificação da nova amostra é baseado na 1. A figura Fig. 4 Ilustra um processo de classificação baseado no algoritmo OPF.

$$C_2(t) = minimo\{maximo\{C_1(s), d(s,t)\}\}, \forall s \in Z_1' \quad (1)$$

Onde Z_1' é o conjunto de amostra que passou pelo processo de treinamento e possui o valor $C(s)$ definido.

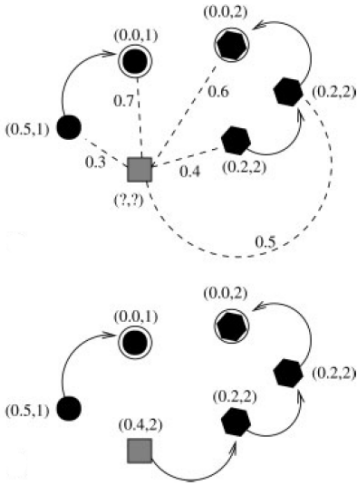

Figura 4. Classificação de uma nova amostra utilizando o OPF. Fonte: Retirado de [1]

II. Implementação

Este artigo apresenta a modelagem do classificador OPF utilizando a linguagem *C++* e o paradigma de programação orientado a objetos. Nessa seção serão detalhados detalhes da implementação, como recursos da linguagem que foram utilizados e algumas estratégias elegidas pelos desenvolvedores.

A. Estruturas Iniciais

O algoritmo implementado consiste de uma estratégia baseada em grafos. Por isso, neste projeto os vértices foram implementados como objetos de uma classe chamada *Vertice(x, y, C)* cujo construtor recebe dois valores referentes ao vetor de característica (x, y) e um valore referente à classe do vértice. Esses dados são obtidos no *dataset*. Conforme os vértices vão sendo criados, são armazenados em uma estrutura chamada *vector* que consiste de uma estrutura dinâmica cujo custo de inserção ao final (*push_back()*) é $O(1)$.

Para a representação do grafo, foi escolhida a matriz de adjacências. Essa escolha foi feita devido à OPF ser inicializada como um grafo completo, e como pode ser visto em [3], a representação por matrizes é preferível quando o grafo é denso. A matriz utilizada é alocada dinamicamente logo após da leitura dos vértices.

B. Características do Treinamento

O primeiro passo no processo de treinamento é a execução do algoritmo da AGM. O algoritmo de Kruskal também poderia ser utilizado, visto que segundo [3] a complexidade de ambos os algoritmos são assintoticamente iguais. Nesse projeto optou-se pelo algoritmo de Prim implementado com o auxílio de uma fila de prioridades que, no C++, é obtida com a estrutura *priority_queue* implementada originalmente como uma *max heap*, mas que pode ser adaptada para uma *min heap* com o auxílio de uma função passada para o construtor.

Os tempos dos métodos inserção e remoção da *priority_queue* são $log_2(n)$ [4], sendo n a quantidade de itens dentro

da estrutura, no caso deste projeto, n será a quantidade de vértices.

Em seguida é realizada uma busca no grafo para identificar os protótipos. Esse passo é realizado por uma simples busca por profundidade (DFS) recursiva que, como pode ser verificado em [3], possui complexidade $O(V + E)$. Quando os protótipos dão encontrados, a aresta que os liga é eliminada. Dessa forma o grafo se transforma em uma floresta cujas árvores são os componentes conexos do grafo. Os índices desses protótipos são guardados e serão utilizados no processo de conquista.

Após a definição dos protótipos é realizado o processo de conquista. Esse passo foi implementado também utilizando DFS a partir de cada protótipo. Sendo que a cada instância da busca o algoritmo guarda qual o valor da maior aresta entre o vértice atual e o protótipo, valor o qual será o custo $C(s)$ dos vértices conquistados. O término desta etapa é também a conclusão do treinamento.

C. Características da Classificação

A estratégia de classificação relativamente simples. Ela é composta apenas de um passo que já foi explicado na seção anterior. O processo se a encontrar um custo para a amostra t inserida, e para esse fim a equação 1 é utilizada. Neste projeto, o algoritmo de classificação foi adaptado do pseudocódigo visto na Fig. 5 a seguir.

```
1  void Classificar(novaAmostra){
2    double dist, maior
3    par<double, int> menorEclasse
4    for i in vertices:
5      dist = calcEuclDist(vertices[i], novaAmostra)
6      maior = max{C(vertices[i]), dist}
7      if(novaAmostra.custo > maior):
8        novaAmostra.custo = maior
9        novaAmostra.indicePai = i
10 }
```

Figura 5. Pseudo código da estratégia de classificação do OPF baseada na equação 1. Fonte: Criado pelos autores.

Vale apontar que o algoritmo da Fig. 5 apenas classifica uma amostra por vez. Caso seja necessário classificar um conjunto com *n* amostras, será necessário executar este algoritmo *n* vezes.

III. Complexidade do algoritmo OPF

Agora que foram apresentados a estrutura da OPF e as estratégias de implementação utilizadas ao longo do desenvolvimento deste projeto, será realizada uma análise da complexidade dos algoritmos desenvolvidos. Na etapa de treinamento, serão analisados os custos para:

1. A construção da AGM;
2. Determinação dos protótipos;
3. Competição dos protótipos.

E na etapa de classificação será analisado o algoritmo apresentado na seção anterior na Fig. 5, o qual é uma implementação da equação 1. Nas subseções seguintes serão realizados os cálculos referentes à complexidade do algoritmo implementado. Para tanto, algumas definições precisam ser feitas para garantir a compreensão do que será apresentado: A variável n que será muito utilizada nos cálculos se refere à quantidade de amostras fornecidas para o treinamento. E se refere às arestas presentes no grafo. E por fim, quando o operador logaritmo for apresentado apenas com o logaritmando, como em $\log a$, deve-se considerar base 2.

A. Complexidade do algoritmo de Prim

O algoritmo de Prim é utilizado para obter a Árvore Geradora Mínima (AGM) do grafo completo. Por isso, é necessário definir o número de arestas em um grafo completo (E_c). Como visto em [5], a equação 2 define o valor de E_c conforme desejado, e com essa definição pode-se dar início à análise do algoritmo de Prim.

$$E_c = \frac{n^2 - n}{2} \tag{2}$$

As linhas anteriores ao laço *while* são as inicializações necessárias para o funcionamento do algoritmo. Considerando que a fila de prioridades está vazia inicialmente, a equação 3 representa a complexidade dessa primeira parte.

$$T^{'}(n) = 4n + 7 \tag{3}$$

O laço *while* irá repetir n vezes. Por isso, a complexidade dentro do *while* e fora do *for* é dada pela equação 4.

$$T^{''}(n) = n \log n + 9n + 1 \tag{4}$$

A estrutura dentro do laço *for* será executada E_c vezes, já que o *if()* procura apenas as arestas adjacentes do vértices. A equação 5 apresenta complexidade desse trecho do código.

$$T^{'''}(n) = 5E_c + E_c \log n \tag{5}$$

A fim de representar $T(n)$ apenas em função de n, substitui-se a equação 2 na equação 5 e o resultado será a equação 6 mostrada abaixo.

$$T^{''''}(n) = \left(\frac{5 + \log n}{2}\right) n^2 + \left(\frac{5 + \log n}{2}\right) n \tag{6}$$

Desse modo, a função de complexidade do algoritmo de Prim implementada é dada pela soma das equações 2, 3, 4 e 6, como é mostrado a seguir na equação 7.

$$T_1(n) = \left(\tfrac{5+\log n}{2}\right) n^2 - \left(\tfrac{\log n+5}{2}\right) n + n \log n + 13n + 8$$

$$T_1(n) = \left(2,5 + \frac{\log n}{2}\right) n^2 + \left(10,5 + \frac{\log n}{2}\right) n + 8 \tag{7}$$

B. Complexidade da busca por protótipos

Após a execução do algoritmo de Prim o grafo consistirá de uma AGM com a mesma quantidade de vértices, mas um número menor de arestas. De acordo com Cormen *et al.* (2012) [3], a quantidade de arestas em uma AGM é dada de acordo com a equação 8.

$$E_{AGM} = n - 1 \tag{8}$$

Com essa definição, pode-se dar início à análise da complexidade do algoritmo. Essa implementação consiste em uma busca em profundidade no grafo utilizando programação dinâmica para deixar o algoritmo linear. A condicional no início do método recursivo garante que o bloco inteiro seja executado n vezes. O laço de repetição procura as arestas adjacentes ao vértice de cada instância, consequentemente o bloco interno a ele é executado $(n + E)$ vezes. Isso resulta na equação 9.

$$\begin{gathered} T_2(n) = n + 8(n + E_{AGM}) \\ T_2(n) = n + 8(n + n - 1) \\ T_2(n) = 17n - 8 \end{gathered} \tag{9}$$

C. Complexidade da geração de custos (competição)

Essa etapa do algoritmo, assim como a busca por protótipos, foi implementada como um algoritmo DFS. Ele será executado para cada protótipo existente na OPF o que garante que percorrerá todas as árvores da floresta pelo menos uma vez. Logo o cálculo da função de complexidade será semelhante ao feito na subseção anterior, resultando na equação 10.

$$\begin{gathered} T_3(n) = 3n + 3(n + E) \\ T_3(n) = 3n + 3(n + n - 1) \\ T_3(n) = 9n - 3 \end{gathered} \tag{10}$$

D. Complexidade total do treinamento

Em posse das funções de complexidade de todas as etapas do treinamento (equações 7, 9 e 10), para calcular a complexidade total, basta somar as três funções. A equação 11 apresenta o resultado dessa soma.

$$\begin{gathered} T(n) = T_1(n) + T_2(n) + T_3(n) \\ T(n) = \left(2,5 + \frac{\log n}{2}\right) n^2 + \left(36,5 + \frac{\log n}{2}\right) n - 3 \end{gathered} \tag{11}$$

E. Complexidade da classificação

A classificação, da forma como foi implementada neste projeto (verificar Fig. 5), recebe apenas uma amostra s_2 como parâmetro e a classifica. A função de complexidade para a classificação de s_2 é simples de ser calculada, já que o laço *for* será executado $(n + 1)$ vezes, e os comandos internos a ele serão executados n vezes. Como todos os demais métodos utilizados nesse algoritmo possuem complexidade $O(1)$, a função $T(n)$ será dada pela equação 12.

$$T(n) = 6n + 5 \tag{12}$$

IV. Resultados

Como observado na seção anterior, o algoritmo implementado possui complexidade polinomial $O(n^2 \log_2 n)$ que consegue classificar novas amostras de forma eficiente e livre de erros através do treinamento supervisionado realizado com um conjunto de amostras classificadas obtida nos arquivos de *dataset* fornecidos como parâmetro para o algoritmo.

Os arquivos de dataset utilizados neste projeto são: *banana.txt* contendo 5300 amostras, e *spirals.txt* contendo 200 amostras. Com o objetivo de testar o algoritmo, o grupo selecionou dois conjuntos de amostras escolhidos ao acaso em cada um dos *datasets* disponíveis e modificou os valores de suas amostras para que ficassem próximas das originais, apenas para uma validação simples. Em um teste mais completo se faz necessária uma variedade maior de amostras para uma estatística mais detalhada.

A. Execução do algoritmo

Foram realizadas duas execuções do algoritmo, uma utilizando o dataset *banana.txt* e outra o *spirals.txt*. Foram definidas duas amostras para cada instância cujos resultados podem ser conferidos na Tabela I. O algoritmo classificou corretamente todas as amostras fornecidas.

TABELA I
RESULTADO DA CLASSIFICAÇÃO DE AMOSTRAS.

Entradas			**Saídas**	
dataset	**x**	**y**	**Custo**	**Classe**
banana.txt	-0.136	-0.953	0.0273607	-1.0
banana.txt	2.60	0.962	0.130292	1.0
spirals.txt	-0.434948	4.41612	0.436891	1
spirals.txt	3.362312	-2.15282	0.376288	2

V. Considerações finais

Com base nos cálculos de complexidade e nos resultados apresentados, conclui-se que o objetivo de implementar o algoritmo de Floresta de Caminhos Ótimos foi alcançado com sucesso. O código escrito em linguagem C++ possui complexidade $O(n^2 \log_2 n)$ e classificou corretamente todas as amostras de teste fornecidas, duas das quais foram apresentadas neste artigo.

Este projeto agregou significativo conhecimento aos integrantes do grupo que precisaram utilizar os conhecimentos obtidos durante toda a disciplina de Projeto e Análise de Algoritmos para implementá-lo.

References

[1] J. P. Papa, A. X. Falcão, C. T. N. Suzuki. *Supervised Pattern Classification Based on Optimum-Path Forest*. Universidade de Campinas, Campinas, 2008.

[2] J. J. da Silva. *Combinação de Classificadores Floresta de Caminhos Ótimos aplicados no Reconhecimento Facial*. Campo Limpo Paulista, 2016.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein. *Algoritmos: Teoria e Prática*, 3ª ed. Rio de Janeiro, Elsevier, 2012.

[4] Cplusplus Reference, *std::priority_queue*, [online], cplusplus.com/reference/queue/priority_queue/.

[5] G. B. Vitor, *Projeto e Analise de Algoritmos: Teoria dos Grafos*. [slide], Universidade Federal de Itajubá, Julho de 2021.