TP1 - Computação Natural

João Lucas Lage Gonçalves - 2020054552

1 Introdução

O objetivo deste trabalho é explorar o uso da Programação Genética (GP) para a criação de funções de distância personalizadas aplicadas ao agrupamento hierárquico (clustering aglomerativo). A GP permite a evolução de expressões simbólicas que se adaptam às características específicas dos dados, oferecendo maior flexibilidade e potencial para a atingir o objetivo.

Neste contexto, empregamos GP para encontrar uma função $d(e_i, e_j)$ que melhor discrimina entre instâncias de dados com base em suas características, otimizando a qualidade dos clusters gerados, calculando a similaridade intra-cluster e a dissimilaridade inter-cluster.

Para avaliar a função de distância evoluída, utilizamos um algoritmo de clustering aglomerativo, que organiza os dados hierarquicamente, utilizando como entrada a matriz de distâncias gerada pela função de GP. O desempenho é avaliado pela medida V, que considera a homogeneidade e completude dos clusters em relação aos rótulos verdadeiros, que são utilizados apenas para avaliação.

2 Implementação

A implementação foi realizada em *Python*, utilizando um notebook *Jupyter*. O código foi estruturado para oferecer flexibilidade na configuração dos parâmetros e na execução paralela dos experimentos. A seguir, detalham-se os principais componentes da implementação.

2.1 Representação dos Indivíduos e da População

Cada indivíduo é representado como uma árvore simbólica, composta por operadores aritméticos básicos $(+, -, *, /)$ e variáveis que correspondem aos atributos do conjunto de dados. A profundidade das árvores é limitada para controlar a complexidade computacional e evitar o overfitting.

No código, a representação dos indivíduos foi implementada através da classe **Node**, que possui os seguintes atributos:

- *value:* contém o valor do nó (operador ou variável)
- *left:* referência ao nó filho à esquerda
- *right:* referência ao nó filho à direita

Além disso, a classe possui os métodos:

- *is_leaf:* verifica se o nó é uma folha.
- *get_all_nodes:* retorna todos os nós descendentes
- *view_expression:* converte a árvore em uma expressão matemática legível
- *depth:* calcula a profundidade total da árvore
- *get_subtree_depth:* calcula a profundidade da subárvore a partir de um nó
- *protected_division:* realiza divisão entre dois valores, retornando 0 em caso de divisão por zero.

A inicialização de cada indivíduo ocorre com a geração de uma árvore aleatória de **Node**, respeitando o limite de profundidade máxima (definido como 7). A população inicial é gerada aleatoriamente com um número máximo de indivíduos, usando o método de crescimento (grow) para garantir diversidade estrutural.

2.2 Função Fitness

A avaliação da fitness de cada indivíduo utiliza a métrica V, que é a média harmônica entre homogeneidade e completude dos clusters gerados. Essa métrica mede a capacidade da função de distância em formar clusters que respeitam os rótulos verdadeiros (usados apenas para avaliação).

O cálculo da aptidão envolve:

1. **Pré-computação das diferenças:** otimização realizada antes da avaliação da função fitness, responsável por calcular as diferenças entre todos os pares de valores de cada atributo no conjunto de dados.
2. **Construção da matriz de distâncias:** cada indivíduo gera uma matriz com base na sua função $d(e_i, e_J)$
3. **Clustering hierárquico:** utiliza-se a função *AgglomerativeClustering* da biblioteca *scikit-learn* [1], configurada para usar a matriz de distâncias gerada.
4. **Cálculo da métrica V:** os clusters resultantes são avaliados em relação aos rótulos reais para obter a aptidão, resultando em um valor no intervalo [0,1], onde valores mais altos indicam melhor desempenho.

2.3 Operadores Genéticos

Os operadores genéticos implementados no algoritmo GP incluem:

- *Crossover:* O crossover de um ponto tem um probabilidade p_c de ser realizada. Dois indivíduos (pais) são selecionados e têm subárvores trocadas em pontos escolhidos aleatoriamente, garantindo que a profundidade máxima não seja excedida.
- *Mutação:* A mutação de um ponto tem um probabilidade p_m de ser realizada. Um nó da árvore é substituído por outro gerado aleatoriamente (variável, operador ou subárvore), respeitando o limite de profundidade.
- *Seleção:* Realizada por torneio. O tamanho do torneio é recebido por parâmetro, representando o número de indivíduos que serão escolhidos aleatoriamente, e o com melhor fitness é selecionado para reprodução.

2.4 Algoritmo GP

O algoritmo GP é responsável por inicializar a população e iterar pelas gerações, aplicando os operadores genéticos para criar novas populações. Sua execução segue os passos:

1. *Inicialização:* A população é gerada aleatoriamente, com base nos parâmetros recebidos.
2. *Avaliação:* A aptidão de todos os indivíduos é calculada usando a função fitness.
3. *Reprodução:* A nova população é criada utilizando crossover, mutação e elitismo, caso habilitado, até que a nova população atinja o tamanho máximo recebido como parâmetro.
4. *Iteração:* O processo é repetido por um número fixo de gerações, armazenando o histórico das fitness e os melhores indivíduos de cada geração.
5. *Resultados:* O algoritmo retorna o histórico das fitness, o melhor indivíduo, a melhor fitness geral e a contagem de filhos melhores e piores que seus pais.

2.5 Execução

Para melhorar o desempenho e viabilizar a execução de múltiplos experimentos, foram implementadas funções que permitem executar o algoritmo em paralelo. A execução paralela utiliza *ThreadPool* para explorar diferentes configurações de parâmetros, como tamanho da população, número de gerações e probabilidades de operadores, reduzindo significativamente o tempo necessário para os experimentos. A cada experimento, os resultados são salvos como *.json* para avaliação posterior.

3 Experimentos

A parte de experimentos seguiu as intruções na proposta do trabalho, começando por testar o tamaho da população e número de gerações, fixando o tamanho máximo do indivíduo, elitismo, tamanho do torneio e probabilidades de crossover e mutação. Após obter a melhor configuração, fixou-se população e número de gerações para testar as probabilidades de crossover e mutação. Por fim foi testado o melhor valor para o tamanho do torneio na seleção. Essa etapa foi realizada usando o dataset Breast Cancer [2].

Para tamanho a população e número de gerações foram testadas os valores 30,50,100,250, de forma que 15 combinações possíveis foram testadas e repetidas 10 vezes (apenas a combinação 250x250 não foi testada por demandar muitos recursos computacionais e tempo). A Figura 1 mostra os resultados obtidos, onde é possível observar que para todos os valores de população, os melhores resultados foram quando o número de gerações era o maior, tanto no valor máximo da fitness, quanto no valor médio, sempre aumentando progressivamente ao aumentar o número de gerações. Além disso o desvio padrão se manteve relativamente baixo, sempre entre 0.03 e 0.05. Já em teste, o melhor resultado foi para uma população menor (30), mas também no maior número de gerações. Por fim o melhor valor encontrado foi com **Tamanho População 100 e Número de Gerações 250**. Pela Figura 2 é possível ver que a fitness média e máxima de treino segue um crescimento constante, assim como a variação entre experimentos, enquanto a fitness mínima não evolui com o passar do tempo, e nem se altera muito entre as repetições. Já a fitness de teste não apresentou grandes resultados, e uma variação muito grande entre repetições.

		best_train_fitness			test_fitness
		max	mean	std	mean
population_size	num_generations				
30	30	0.14	0.10	0.03	0.05
	50	0.19	0.12	0.03	0.06
	100	0.23	0.14	0.04	0.07
	250	0.24	0.16	0.04	0.11
50	30	0.17	0.11	0.03	0.05
	50	0.23	0.14	0.04	0.06
	100	0.22	0.15	0.04	0.06
	250	0.31	0.18	0.05	0.06
100	30	0.19	0.13	0.03	0.05
	50	0.23	0.14	0.04	0.08
	100	0.27	0.17	0.05	0.07
	250	0.31	0.20	0.04	0.02
250	30	0.21	0.15	0.03	0.06
	50	0.25	0.17	0.03	0.07
	100	0.37	0.21	0.05	0.06

Figura 1: Resultados da fitness obtida ao rodar os experimentos variando população e geração. Os números em amarelo são os melhores valores enquanto os roxos são os piores

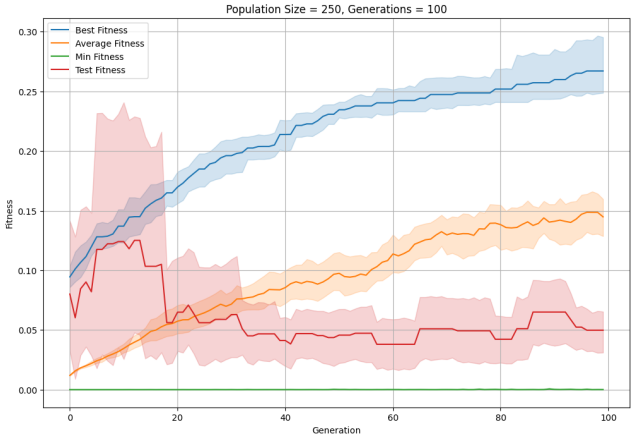

Figura 2: Evolução temporal da fitness no melhor experimento. As áreas sombreadas representam a variação dos resultados nas 10 repetições enquanto a linha representa a média dos 10 experimentos por geração.

Após fixar esses parâmetros, testou-se duas combinações de probabilidades para crossover e mutação, 0.9 e 0.05 respectivamente, e depois 0.6 e 0.3 respectivamente. A Figura 3 mostra a comparação entre os resultados da fitness obtida, destacando claramente a combinação de **Probabilidade Crossover 0.6 e Probabilidade Mutação 0.3** como superior. Podemos ver também na 4 que reforça a superioridade dessa combinação, gerando uma quantidade maior proporcionalmente de filhos melhores que os pais. Por fim, no gráfico 5 de fitness por geração é possível observar um comportamento semelhante ao no experimento da população e geração, porém com maior desvio entre repetições.

		best_train_fitness			test_fitness
		max	mean	std	mean
crossover_prob	mutation_prob				
0.60	0.30	0.38	0.19	0.06	0.07
0.90	0.05	0.30	0.18	0.05	0.05

Figura 3: Resultados da fitness obtida ao rodar os experimentos variando as probabilidades dos operadores.

		count_better	count_worse	percentage_better	percentage_worse
crossover_prob	mutation_prob				
0.60	0.30	9573.80	5414.60	63.87%	36.13%
0.90	0.05	12918.50	9572.50	57.44%	42.56%

Figura 4: Contagem de filhos melhores que os pais gerados pelo crossover para cada probabilidade. A porcentagem mostra proporcionalmente em relação ao número de filhos gerados no total

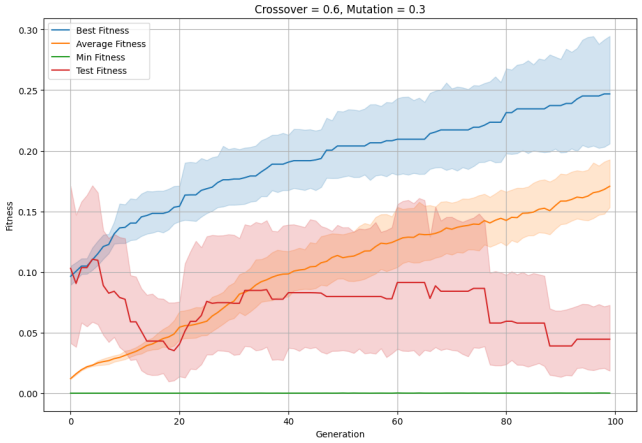

Figura 5: Evolução temporal da fitness no melhor experimento das probabilidades de crossover e mutação.

Por fim foi testado a variação do tamanho do torneio para a operação de seleção. Os valores testados foram 2,3,5,7. A figura 6 mostra que os valores 3 e 5 tiveram melhor resultado em relação aos outros. Além disso é possível ver na figura 7 que o tamanho 2 foi quando teve a maior proporção de filhos melhores que os pais, enquanto o tamanho 5 teve a pior, ainda assim um pouco superior a 50%. Por ter os melhores resultados, ainda que com pouca superioridade, o **Tamanho do Torneio 5** foi selecionado e no gráfico 8 é possível ver a evolução por geração, onde nota-se uma estabilização após a geração 40, e uma variação muito grande entre repetições.

	best_train_fitness			test_fitness
	max	mean	std	mean
tournament_size				
2	0.26	0.17	0.03	0.05
3	0.35	0.21	0.06	0.06
5	0.36	0.22	0.06	0.08
7	0.26	0.19	0.04	0.06

Figura 6: Resultados da fitness obtida ao rodar os experimentos variando o tamanho do torneio.

tournament_size	count_better	count_worse	percentage_better	percentage_worse
2.00	9561.20	5445.00	63.71%	36.29%
3.00	8146.40	6795.80	54.52%	45.48%
5.00	7617.22	7418.11	50.66%	49.34%
7.00	8391.89	6692.33	55.63%	44.37%

Figura 7: Contagem de filhos melhores que os pais gerados pelo crossover para cada tamanho de torneio.

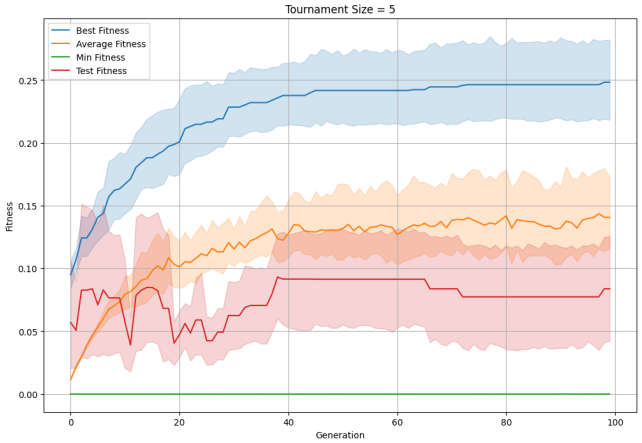

Figura 8: Evolução temporal da fitness no melhor experimento do tamanho de torneio.

Ao final dos experimentos a melhor configuração de parâmetros encontrada foi: **Tamanho da população 250, Número de Gerações 100, Probabilidade Crossover 0.6, Probabilidade Mutação 0.3, Tamanho Torneio 5.**

					best_train_fitness			test_fitness
					max	mean	std	mean
population_size	num_generations	crossover_prob	mutation_prob	tournament_size				
30.00	30.00	0.90	0.05	2.00	0.14	0.10	0.03	0.05
	50.00	0.90	0.05	2.00	0.19	0.12	0.03	0.06
	100.00	0.90	0.05	2.00	0.23	0.14	0.04	0.07
	250.00	0.90	0.05	2.00	0.24	0.16	0.04	0.11
50.00	30.00	0.90	0.05	2.00	0.17	0.11	0.03	0.05
	50.00	0.90	0.05	2.00	0.23	0.14	0.04	0.06
	100.00	0.90	0.05	2.00	0.22	0.15	0.04	0.06
	250.00	0.90	0.05	2.00	0.31	0.18	0.05	0.06
100.00	30.00	0.90	0.05	2.00	0.19	0.13	0.03	0.05
	50.00	0.90	0.05	2.00	0.23	0.14	0.04	0.08
	100.00	0.90	0.05	2.00	0.27	0.17	0.05	0.07
	250.00	0.90	0.05	2.00	0.31	0.20	0.04	0.02
250.00	30.00	0.90	0.05	2.00	0.21	0.15	0.03	0.06
	50.00	0.90	0.05	2.00	0.25	0.17	0.03	0.07
	100.00	0.60	0.30	2.00	0.38	0.18	0.05	0.06
				3.00	0.35	0.21	0.06	0.06
				5.00	0.36	0.22	0.06	0.08
				7.00	0.26	0.19	0.04	0.06
		0.90	0.05	2.00	0.37	0.20	0.06	0.05

Figura 9: Resultado da fitness obtida em todas as combinações testadas

Por fim, os melhores parâmetros encontrados foram testados no outro dataset Wine Quality [3], mostrando resultados menos satisfatórios do que no dataset anterior. Pelo gráfico 12 é possível perceber que o algoritmo teve uma convergência muio rápida, com a fitness se estabilizando por volta da geração 20. Outra observação é que a fitness de teste manteve uma evolução similar a de treino, diferente do que aconteceu no dataset original.

					best_train_fitness			test_fitness
					max	mean	std	mean
population_size	num_generations	crossover_prob	mutation_prob	tournament_size				
250.00	100.00	0.60	0.30	5.00	0.20	0.17	0.02	0.14

Figura 10: Resultados da fitness obtida ao rodar os experimentos no dataset Wine Quality

tournament_size	count_better	count_worse	percentage_better	percentage_worse
2.00	9561.20	5445.00	63.71%	36.29%
3.00	8146.40	6795.80	54.52%	45.48%
5.00	7617.22	7418.11	50.66%	49.34%
7.00	8391.89	6692.33	55.63%	44.37%

Figura 11: Contagem de filhos melhores que os pais gerados pelo crossover no dataset Wine Quality

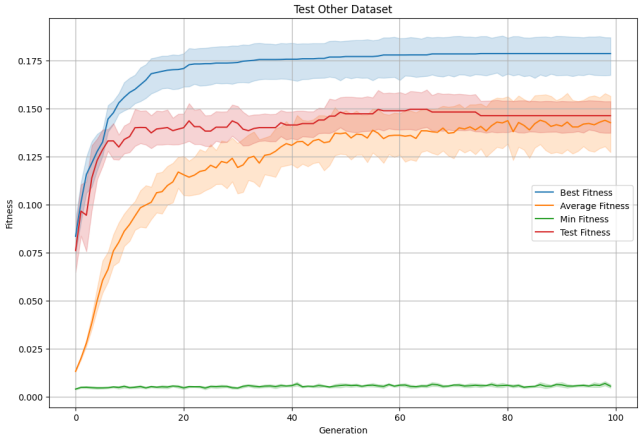

Figura 12: Evolução temporal da fitness com as melhores configurações no dataset Wine Quality.

4 Conclusão

Este trabalho explorou a aplicação da Programação Genética (GP) na construção de uma função de distância personalizada para o algoritmo de clustering aglomerativo. O objetivo principal foi otimizar a qualidade dos clusters gerados, considerando a similaridade intra-cluster e a dissimilaridade inter-cluster.

A implementação em Python utilizou uma representação em árvore para os indivíduos, onde cada nó representava um operador aritmético ou uma variável correspondente a um atributo do conjunto de dados. A função fitness foi baseada na métrica V, que avalia a homogeneidade e a completude dos clusters em relação aos rótulos verdadeiros.

A análise experimental, realizada com diferentes configurações de parâmetros da GP, permitiu identificar os valores mais adequados para o tamanho da população, número de gerações, probabilidades de crossover e mutação, e tamanho do torneio. Os resultados demonstram que a escolha criteriosa desses parâmetros impacta significativamente a qualidade das soluções encontradas.

Observou-se que o aumento do número de gerações e o uso de uma probabilidade de mutação mais alta favoreceram a obtenção de melhores resultados. A escolha do tamanho do torneio também se mostrou relevante, com os melhores resultados obtidos para valores intermediários.

Apesar dos bons resultados, a variação entre as repetições dos experimentos sugere a necessidade de algumas melhorias, como a implementação de técnicas para garantir a diversidade da população.

Referências

[1] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". Em: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[2] Zwitter et al. *Breast Cancer*. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C51P4M. 1988.

[3] Cortez et al. *Wine Quality*. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C56S3T. 2009.