Documentação - 1º Trabalho Prático - Computação Natural (DCC831)

Aluno: Victor Hugo Silva Moura
Matrícula: 2018054958

Introdução

A documentação a seguir visa discutir a implementação do primeiro trabalho prático da disciplina de Computação Natural, onde se deseja, por meio de Programação Genética, encontrar uma função que descreva consistentemente uma base de dados, tal que, ao ser aplicada em uma função de clusterização, o resultado se aproxime do esperado. Além disso, serão apresentadas as dificuldades encontradas durante a implementação e uma pequena análise experimental.

O trabalho prático foi inteiramente desenvolvido na Python, utilizando algumas bibliotecas extras como, por exemplo, Pandas, Numpy, PyClustering e Scikit-Learn (para utilizar a métrica V).

Implementação

Começaremos a discussão sobre a implementação falando sobre a representação dos indivíduos.

Os indivíduos foram implementados por meio de árvores binárias, que simbolizam as expressões matemáticas que serão aplicadas posteriormente para indicar a distância entre pontos do dataset. Nesta árvore, os terminais são os parâmetros de cada dataset e as funções são um conjunto simples de funções matemáticas (soma, subtração, multiplicação e divisão). Abaixo temos a representação de um indivíduo:

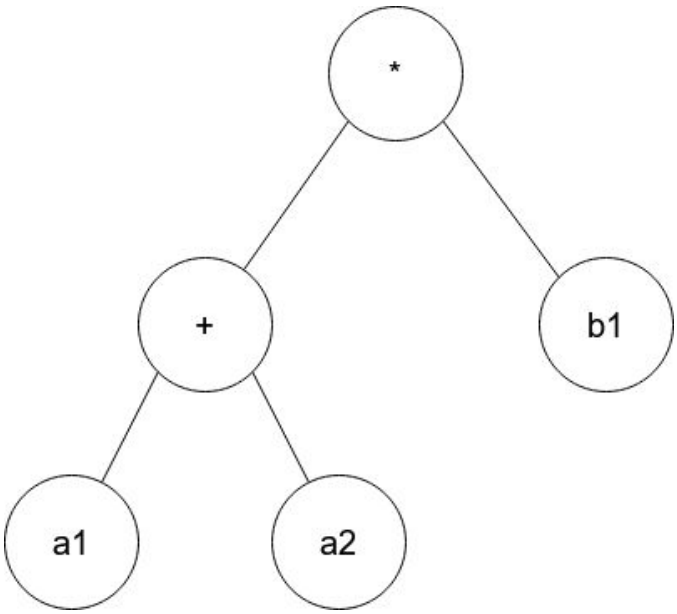

*Figura 1: Árvore de um indivíduo que representa a função (a1 + a2) * b1*

Como os parâmetros do indivíduo variam de acordo com o ponto do dataset que está sendo considerado, os terminais da árvore foram definidos como *[a1, ..., a9]* e *[b1, …, b9]*, representando, respectivamente, os 9 atributos do primeiro e segundo ponto que serão enviados para a função de distância posteriormente.

Após isso, foi feita a inicialização da população utilizando o método *Ramped Half-and-Half*. As alturas das árvores foram definidas numa escala começando em *2* e indo até *7* (profundidade máxima das árvores). Sendo assim, como temos *6* alturas diferentes e *2* tipos de inicialização (*grow* e *full*), a população fica bem distribuída quanto o seu total é múltiplo de *12*.

Tendo a representação do indivíduo definida e a inicialização da população, a próxima etapa foi decidir como seria a implementação do cálculo da fitness. Para implementar esse cálculo, primeiramente temos que conseguir extrair a expressão matemática da árvore, depois aplicar essa expressão em uma função de distância que, mais à frente, será utilizada em um algoritmo de clusterização. O resultado desse clustering, por sua vez, será comparado com o resultado esperado utilizando a medida *V*.

Para extrair a função, inicialmente fazemos um caminhamento pré-ordem na árvore (*Nó -> Filho da Esquerda -> Filho da Direita*), e armazenamos o dado do nó em uma lista. O resultado deste caminhamento e armazenamento dos dados é uma expressão na notação prefixa, onde os operadores vêm antes dos operandos. Com isso, podemos aplicar um algoritmo de avaliação de expressões nessa notação, onde a expressão é invertida e cada elemento dela é adicionada em uma pilha seguindo a seguinte regra:

1. Se o elemento for um número, basta adicioná-lo no topo da pilha

2. Se o elemento for uma operação, os dois primeiros elementos da pilha são extraídos e a operação é executada. O resultado dessa operação é colocado no topo da pilha.

Dessa forma, já temos como avaliar uma expressão. Porém, ainda temos que substituir os terminais da árvore pelos valores dos pontos. Para fazer isso, basta passar esses valores para a função de avaliação da expressão e, caso o elemento a ser inserido na pilha for um terminal, o valor correspondente é colocado na pilha no lugar desse terminal.

Com isso, temos como avaliar a expressão e retornar valores de distância entre pontos. Esses valores serão utilizados por um algoritmo de clustering (no caso deste trabalho, um K Means) e o resultado desse clustering é comparado com o resultado esperado, utilizando a medida *V*. O resultado dessa comparação é definido como a fitness do indivíduo. Esse processo é feito para todos os indivíduos da população.

A próxima etapa a ser definida são os operadores genéticos. Para esse trabalho prático, foram utilizados o Cruzamento de Um Ponto e a Mutação de Um Ponto. A discussão da implementação deles será feita em conjunto, pois a lógica utilizada é semelhante.

Para implementar esses operadores, foi utilizada uma função em comum responsável por escolher um nó com uma profundidade máxima (a justificativa dessa profundidade será explicada ao falar sobre o operador de cruzamento). Essa função percorre aleatoriamente a árvore e, retorna um nó e os passos para encontrar esse nó na árvore. No caso do operador de mutação, o nó é escolhido e um novo dado é escolhido de acordo com o dado anterior do nó (se era um terminal, é trocado por outro terminal. Se era uma função, é trocado por outra função). Esse dado é então inserido na em uma cópia da árvore, utilizando as instruções obtidas no passo anterior.

Para o operador de cruzamento, um nó aleatório do primeiro pai é escolhido, utilizando a função anteriormente citada. Já para o segundo pai, um nó aleatório é escolhido dele desde que a profundidade máxima desse nó, somada à profundidade do nó escolhido no primeiro pai, não ultrapasse a profundidade máxima da árvore. Isso previne um crescimento descontrolado dos indivíduos. Após isso, o nó do primeiro pai é inserido em uma cópia do segundo pai, utilizando as instruções do segundo pai e o processo análogo é feito para este mesmo.

Após definir os operadores genéticos e sua implementação, temos que definir os indivíduos que serão os operandos. Para isso, foi feita a implementação de uma seleção por torneio. A implementação dessa seleção é bem simples. Inicialmente é feito um shuffle na população, para garantir que os indivíduos serão escolhidos de forma aleatória. Após isso, são selecionados os *k* primeiros indivíduos da população após o shuffle e, o indivíduo com a maior fitness entre eles é retornado. No caso de um cruzamento, que é um operador que necessita de dois indivíduos, o mesmo processo é

aplicado para os k últimos indivíduos, de forma a evitar que um indivíduo faça cruzamento com ele mesmo.

Por fim, temos que juntar todas essas peças para formar o algoritmo completo. Inicialmente uma população é gerada e a fitness de seus indivíduos é calculada. Após isso, enquanto a nova geração não tiver o mesmo tamanho da geração anterior, os operadores genéticos são aplicados de acordo com as suas probabilidades, sendo que os indivíduos são escolhidos por meio da seleção por torneio. Ao final de uma geração, é calculada a fitness dessa geração e o processo se repete até que o número de gerações atinja seu valor máximo. Porém, além dessas operações, há também a aplicação de elitismo na população. Sendo assim, ao começo de cada nova geração, o melhor indivíduo da população anterior é copiado para a próxima geração.

Ao final das gerações, o melhor indivíduo é testado contra os dados de teste dos datasets fornecidos e o resultado da fitness é reportado.

Dificuldades

As maiores dificuldades ocorreram ao implementar os operadores genéticos e ao rodar os experimentos que serão discutidos na próxima seção.

Para os operadores, foi difícil alterar os dados das árvores, pois a linguagem Python não trabalha com passagem por referência. Sendo assim, para modificar o nó de uma árvore, era necessário percorrer todo o caminho até esse nó e, só então modificá-lo. Dessa forma, muito tempo foi perdido durante o trabalho até que soluções viáveis fossem descobertas.

Já sobre os experimentos, o problema principal foi o tempo de execução. À medida que a população de indivíduos cresce, o tempo para executar o código também aumenta, principalmente devido à função de fitness, onde é executado um K Means para cada um dos indivíduos. Com isso, os experimentos ficaram limitados pois, certos tamanhos de população se tornaram inviáveis de serem executados.

Experimentos

Para a parte de experimentos, foi utilizada a base de dados *‘breast_cancer_coimbra_train.csv’*. A escolha dessa base se deve ao fato de ela ser menor e, portanto, mais rápida de se executar. Como dito na seção anterior, um dos maiores problemas durante o desenvolvimento do trabalho foi a parte de experimentos, especialmente o tempo gasto para rodá-los. Sendo assim, o número de gerações e indivíduos foi escolhido de forma a obter algum resultado em tempo viável. Dessa forma, a variação de indivíduos de gerações se deu entre os seguintes valores: 12/12, 24/24 e 36/36, sendo o primeiro número a quantidade de indivíduos e o segundo número o total de gerações. Além disso, foram variadas as probabilidades dos operadores genéticos entre [0.9, 0.05] e [0.6, 0.3], sendo o primeiro número a probabilidade de cruzamento e o segundo número a probabilidade de mutação, e o

valor *k* do torneio. O valor k variou entre 2 e 5 para populações de até 24 indivíduos e entre 3 e 7 para a população de 36 indivíduos. Para completar, cada experimento foi rodado 10x de forma a obtermos o desvio padrão e média das melhores fitness.

A seguir temos um gráfico representando a variação da melhor fitness por geração em cada uma das execuções, para os seguintes parâmetros:

- População e Gerações: 24
- Valor k do Torneio: 2
- Probabilidades: [0.6, 0.3]

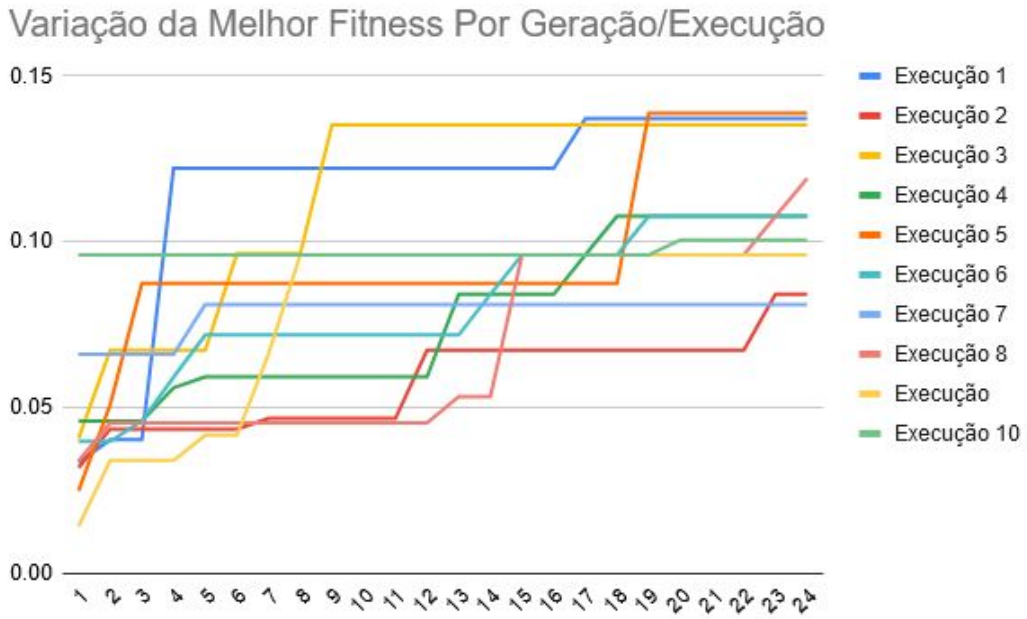

Figura 2: Variação da melhor fitness por geração em cada uma das execuções de população 24

O gráfico acima foi escolhido justamente pois o melhor resultado do teste foi obtido utilizando esses parâmetros. Abaixo temos a distribuição de média e desvio padrão para o treino e teste com uma população de 24 indivíduos:

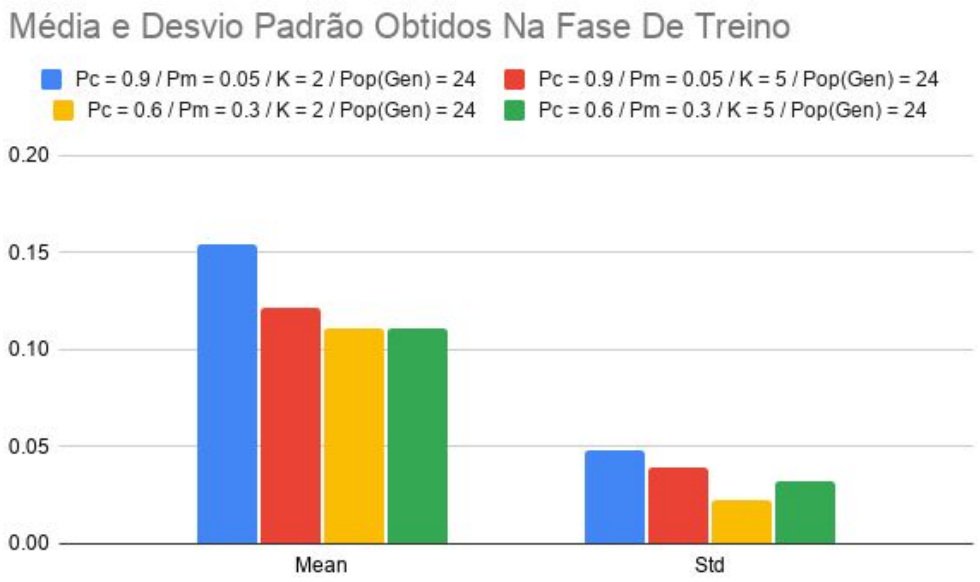

Figura 3: Média e desvio padrão no treino para 24 indivíduos

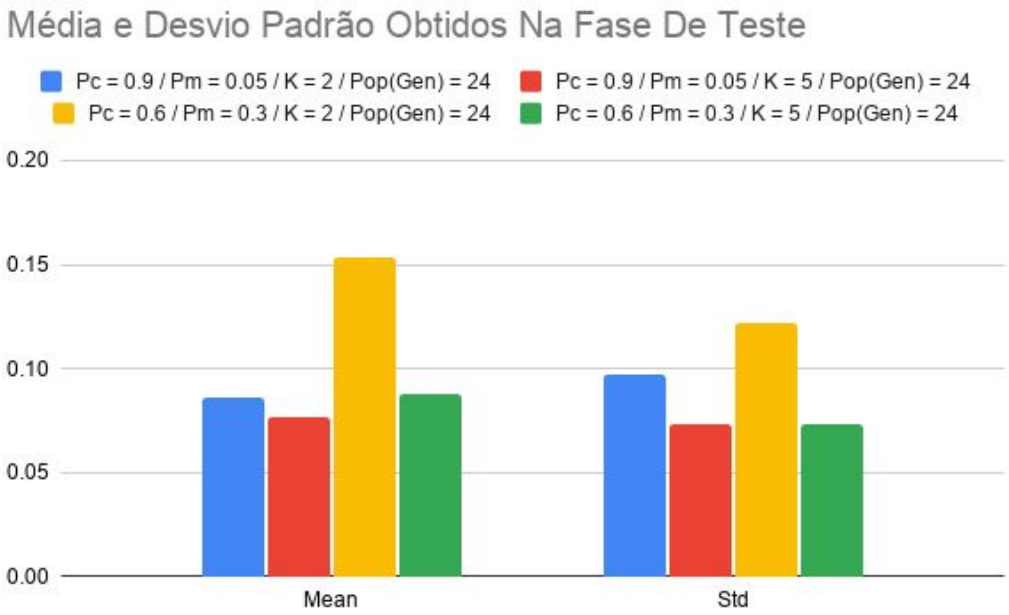

Figura 4: Média e desvio padrão no teste para 24 indivíduos

Assim como podemos ver, a média obtida no teste para os parâmetros citados acima foi por volta de 0.15, o que foi a melhor média obtida ao longo de todos os

experimentos. Abaixo temos os gráficos de melhor fitness, média e desvio padrão para as demais populações:

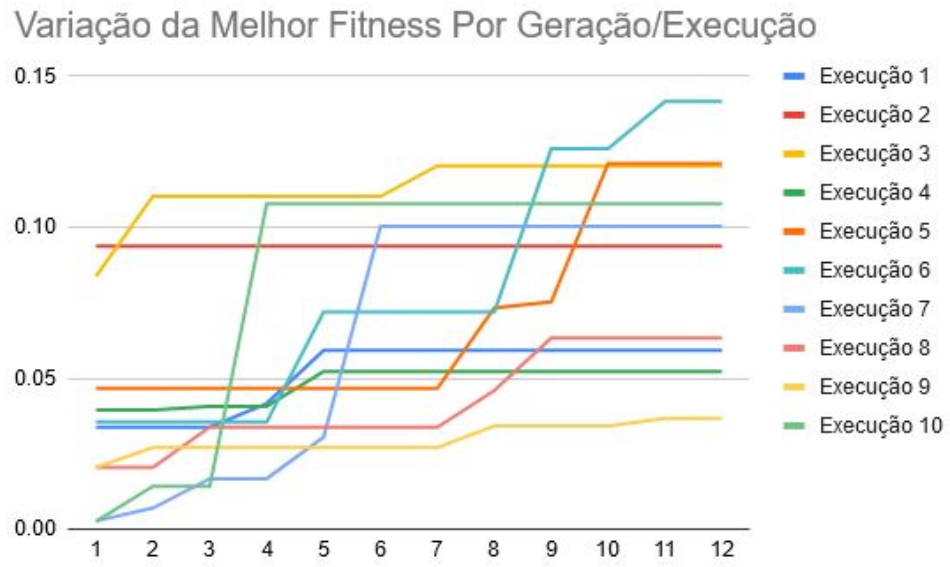

Figura 5: Variação da melhor fitness por geração em cada uma das execuções de população 12

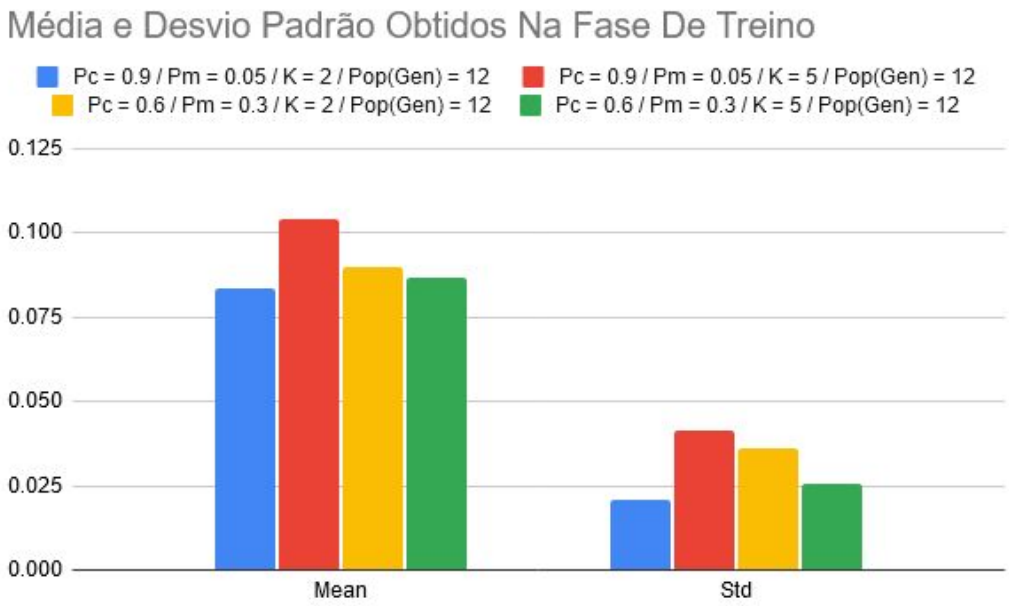

Figura 6: Média e desvio padrão no treino para 12 indivíduos

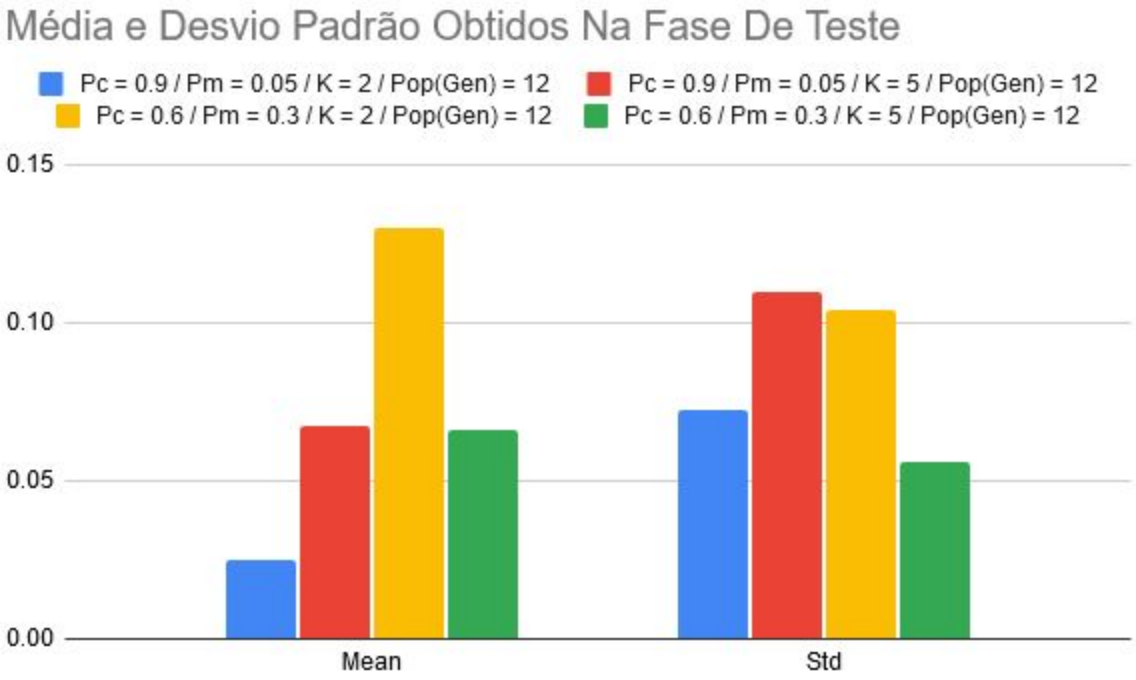

Figura 7: Média e desvio padrão no teste para 12 indivíduos

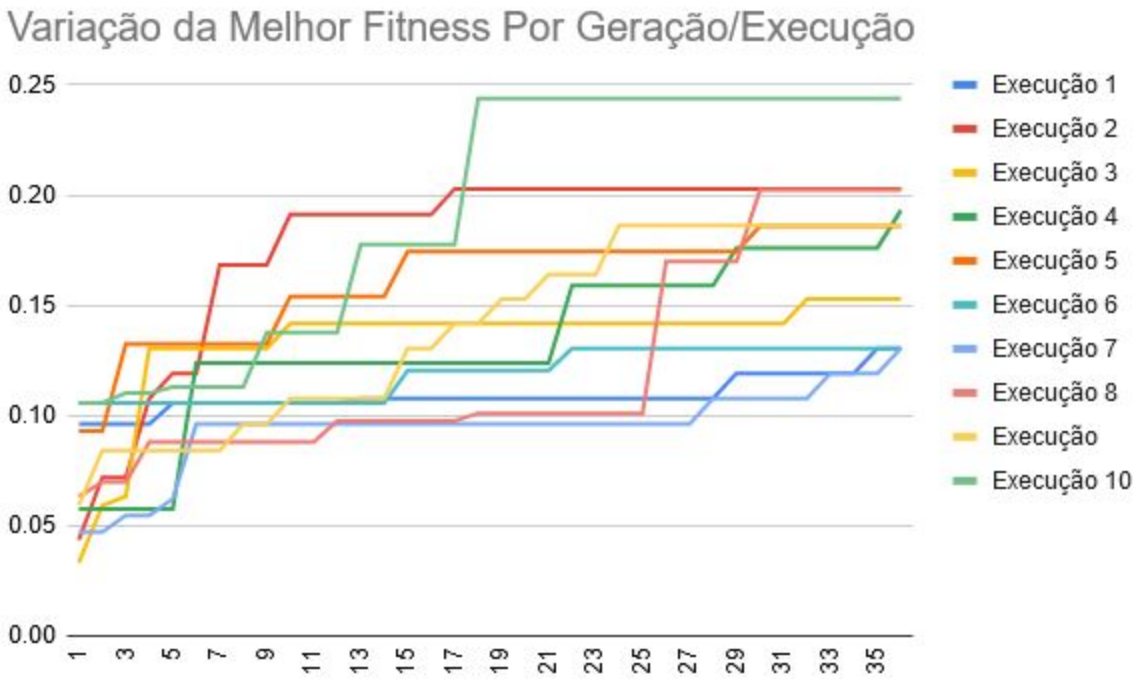

Figura 8: Variação da melhor fitness por geração em cada uma das execuções de população 36

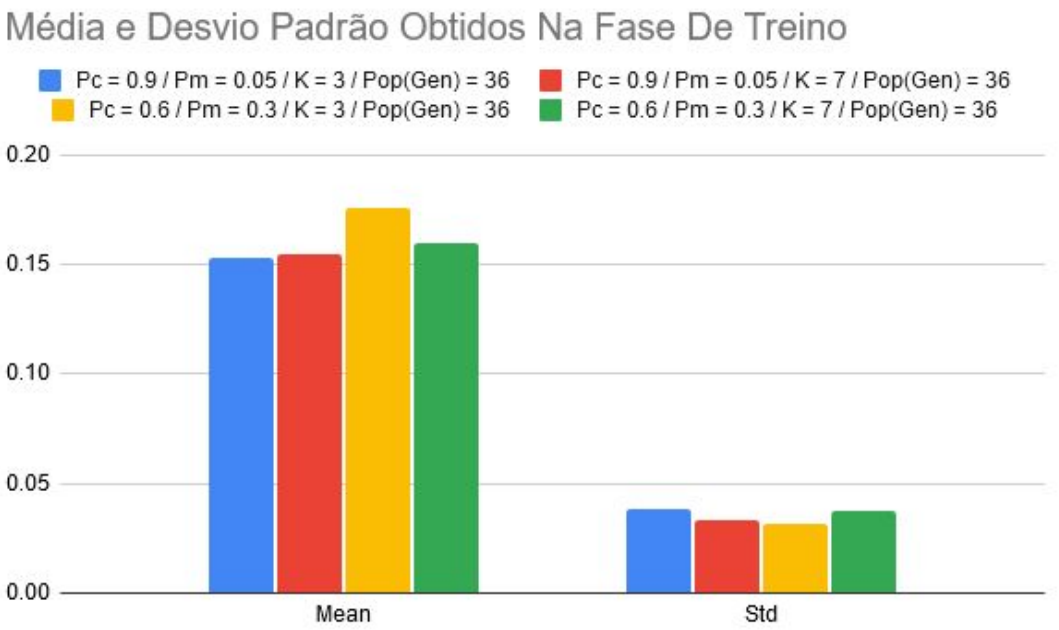

Figura 9: Média e desvio padrão no treino para 36 indivíduos

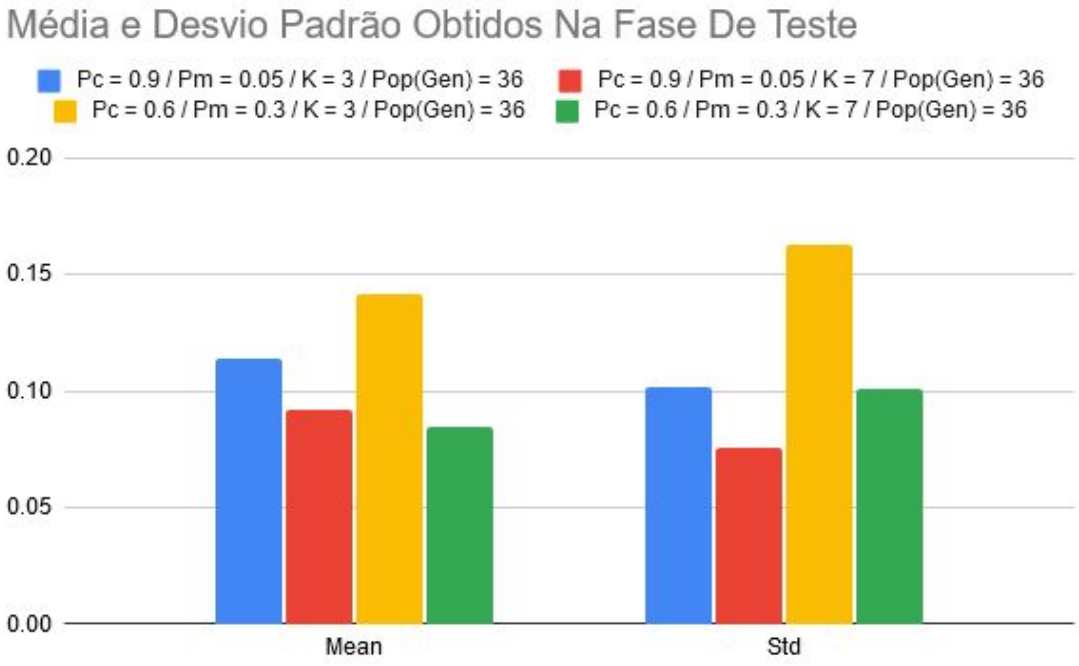

Figura 10: Média e desvio padrão no teste para 36 indivíduos

Podemos observar que apesar da variação de parâmetros, os resultados obtidos não foram consideravelmente diferentes. Uma possível explicação para isso seria o baixo número de indivíduos na população, ou até mesmo o baixo número de gerações.

Sendo assim, essas populações não conseguem sofrer muitas variações e ficam estagnadas na mesma região de busca.

Após gerar todos os gráficos, os melhores parâmetros foram testados para o outro dataset disponível (o '*glass_train.csv*'). Utilizando esses parâmetros, foi obtido o seguintes gráficos:

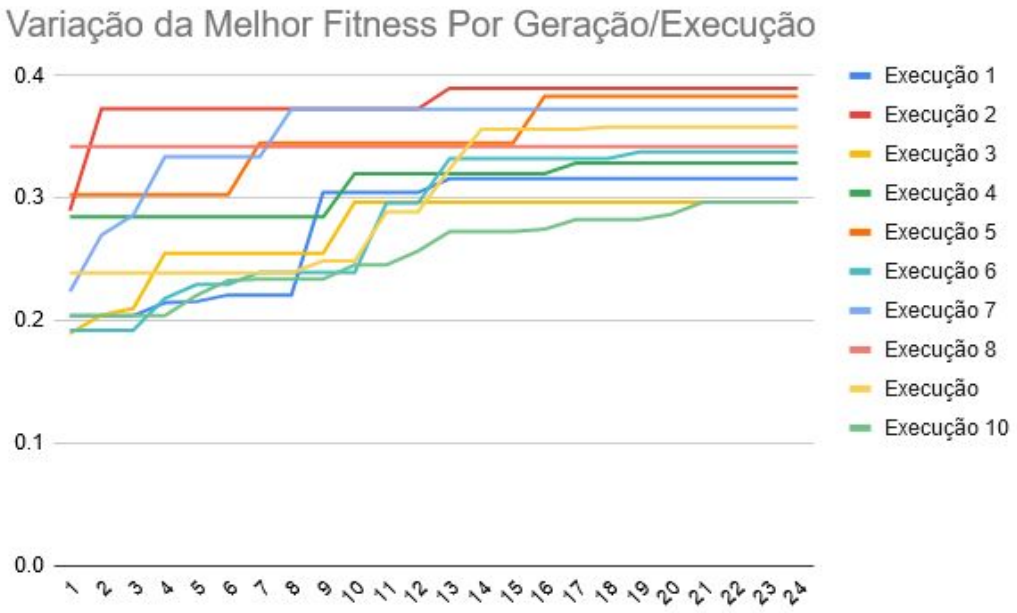

Figura 11: Variação da melhor fitness por geração em cada uma das execuções de população 24 para o dataset 'glass_train.csv'

Observando o gráfico acima, podemos ver que os resultados obtidos no segundo dataset foram ainda melhores do que os resultados do primeiro. A média de treino obtida foi 0.342, com desvio padrão de 0.031, enquanto que a média de teste ficou em 0.351, com desvio padrão de 0.167, o que são valores bem melhores do que os encontrados no dataset de breast cancer.

Os dados de cada uma das execuções se encontram na pasta do projeto.

Considerações Finais

A realização deste trabalho foi bem desafiadora e interessante. Por meio dele foi possível entender um pouco mais sobre como funcionam algoritmos evolucionários, seu poder de expressão e suas complexidades, tanto computacionais quanto temporais, além de poder praticar mais os conhecimentos aprendidos na aula de Computação Natural.

Bibliografia

- Pyclustering Distance Metric Class Reference - https://pyclustering.github.io/docs/0.9.0/html/df/df9/classpyclustering_1_1utils_1_1metric_1_1distance__metric.html
- Riccardo Poli, William B. Langdon, and Nicholas Freitag McPhee. 2008. *A Field Guide to Genetic Programming*. Lulu Enterprises, UK Ltd.
- finding duplicates in a list of lists - https://stackoverflow.com/questions/19811418/finding-duplicates-in-a-list-of-lists