Aluno: Felipe Colombelli Cartão: 00287698

Implementação 1

Codificação dos cromossomos

Os cromossomos do algoritmo genético são representados por uma lista em que cada elemento representa um gene. Cada gene do cromossomo é descrito por um booleano, True ou False, assim, tem-se uma lista com 7128 elementos, onde cada elemento representa se a expressão gênica (coluna do dataset de entrada) é ou não considerada na clusterização. A seguinte lista de exemplo considera a primeira, segunda e penúltima expressão gênica para realizar a clusterização:

[True, True, False, False …… (7121 elementos False) …… , False, True, False]

Função de Fitness

Para o cálculo da função de fitness são utilizadas as medidas de acurácia (extraídas do k-means) e o número de genes utilizados na clusterização. O que queremos é aumentar a acurácia e diminuir o número de genes, logo, nosso ponto de partida é a seguinte equação:

- acc: Acurácia, número real de 0 a 100
- ngs: Número de Genes Selecionados, número inteiro de 0 a 7128

$$Fitness = acc - ngs$$

Como o número de genes varia de 0 a 7128, tal número dominaria em questão de importância, seria melhor escolher 0 genes com 0 de acurácia do que 3000 genes com 100 de acurácia. Para evitar que isso aconteça, a primeira medida a se fazer é normalizar a quantidade de genes para o intervalo [0, 100]. Faremos isso da seguinte forma:

$$ngsNorm = \frac{100 * (ngs - min)}{max - min}$$

$$ngsNorm = \frac{100 * (ngs - 0)}{7128 - 0}$$

$$ngsNorm = \frac{100 * ngs}{7128}$$

$$Fitness = acc - ngsNorm$$

Mas só isso não é o suficiente. Queremos que nosso modelo priorize a acurácia, não adianta nada selecionarmos poucos genes se a acurácia do modelo é o equivalente a um cara-ou-coroa, i. e., um chute aleatório.

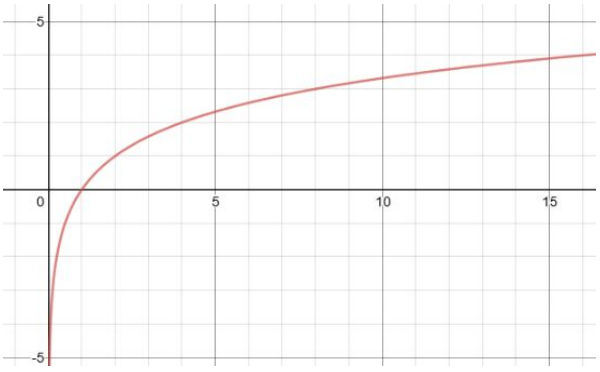
Vamos assumir aqui que qualquer acurácia igual ou menor que 90% seja inaceitável. De fato este valor foi baseado em evidências empíricas após rodar o K-Means sobre os 7128 genes e acabar a execução constantemente com o 98.61% de acurácia.

Mesmo 89% sendo considerado, portanto, inaceitável, ainda continua sendo melhor que 88%. Desta forma, trataremos os valores indesejados de acurácia da seguinte maneira:

$$If\ acc\ \leq\ 90:$$
$$acc\ =\ acc\ -\ 100$$

Isto faz com que tais valores variem dentro do intervalo [-100, -10].

Finalmente, daremos mais um boost para os ganhos de acurácia do modelo, para indicar que acurácia é mais importante que número de genes selecionados. Faremos isso utilizando arbitrariamente uma escala logarítmica, cujos gráficos estão representados a seguir.



$Log_2(x)$

$-Log_2(x)$

A escala $-Log_2$ será utilizada para modificar as acurácias menores ou iguais a 90% que, lembrando, estão deslocadas de [-100, -10]. Multiplicando essa escala por -1, temos uma nova escala de [10, 100], onde, quanto mais próximos os números estiverem do 10, melhores serão os resultados e, portanto, menor deve ser o coeficiente multiplicativo logarítmico que lhes modificará. Portanto, a seguinte fórmula será utilizada:

$$If\ acc\ \leq\ 90:$$
$$acc\ =\ acc\ -\ 100$$
$$accNew\ =\ acc\ *\ -log_2(-acc)$$

$$Else:$$
$$accNew\ =\ acc\ *\ log_2(acc)$$

E, por fim, o fitness é calculado pela equação:

$$Fitness = accNew - ngsNorm$$

Geração da população inicial

Nesta primeira implementação, a geração da população é feita simplesmente pela geração de 7128 booleanos aleatórios, 50 vezes.

```
160     def __generatePopulation(self):
161
162         self.population = np.array([
163             [bool(random.getrandbits(1)) for i in range(NUM_OF_GENES)]
164              for i in range(self.populationSize)])
165         return
```

Computação da função de fitness

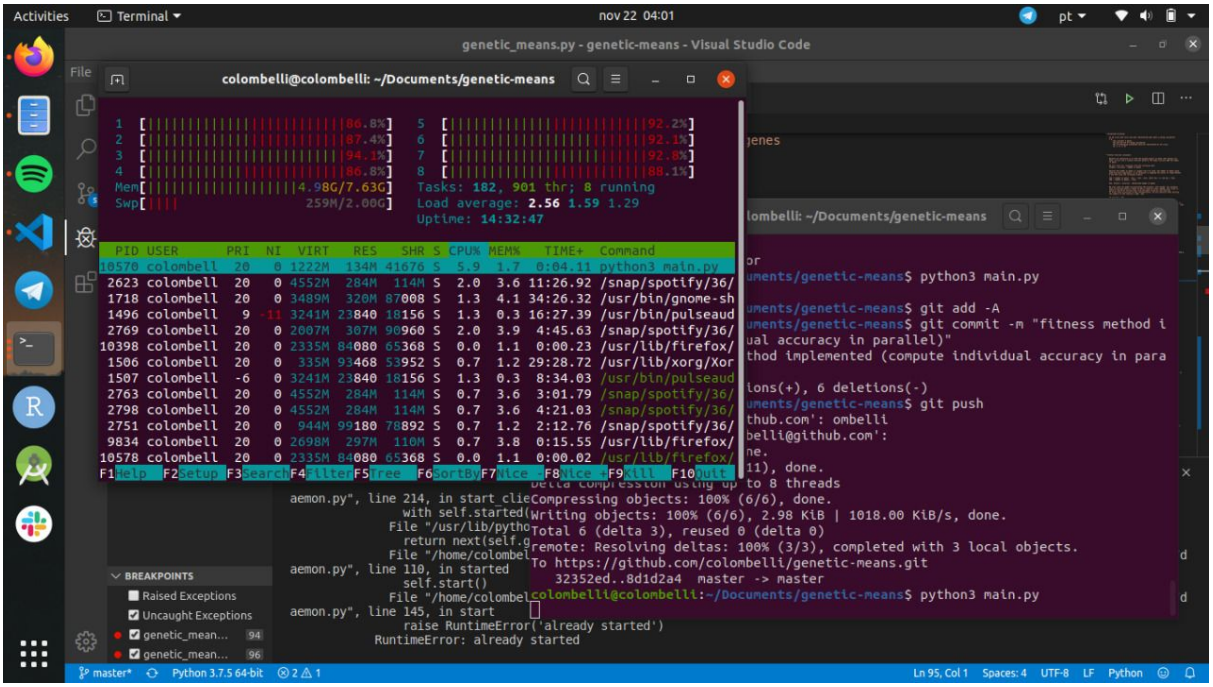
Como a computação da função de fitness é o bottleneck desta implementação, foi feito um esforço para se paralelizar tal operação. O código aloca o poder de processamento das CPUs disponíveis e, para cada indivíduo, computa o seu score associado.

O K-Means utilizado no cálculo de acurácia foi implementado com 25 iterações.

```
169     def __computeFitness(self):
170
171         self.fitness = [None] * len(self.population)
172         pool = mp.Pool(mp.cpu_count())
173
174         self.fitness = np.array([pool.apply(self.computeIndividualFitness, args=(individual, ))
175                 for individual in self.population])
176
177         pool.close()
178         return
```

Com a ajuda do programa Htop, foi possível detectar claramente a utilização praticamente completa dos recursos computacionais disponíveis no momento do cálculo de fitness.

Abaixo segue um print screen da execução citada.



Seleção dos indivíduos para a próxima geração

O método de seleção implementado foi o elitismo. Decidiu-se manter 30% da população mais apta a cada geração.

```
235     def __selectPopulation(self):
236
237         numElite = round(self.elitism * self.populationSize)
238         # Get the index of the N greatest scores:
239         eliteIdx = np.argpartition(self.fitness, -numElite)[-numElite:]
240         elite = self.population[eliteIdx]
241
242         self.population = elite
243         return
```

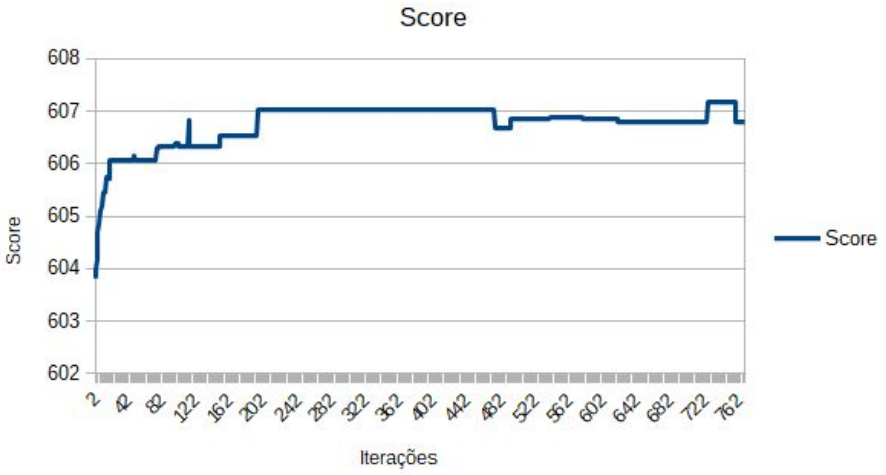
Cruzamento da população

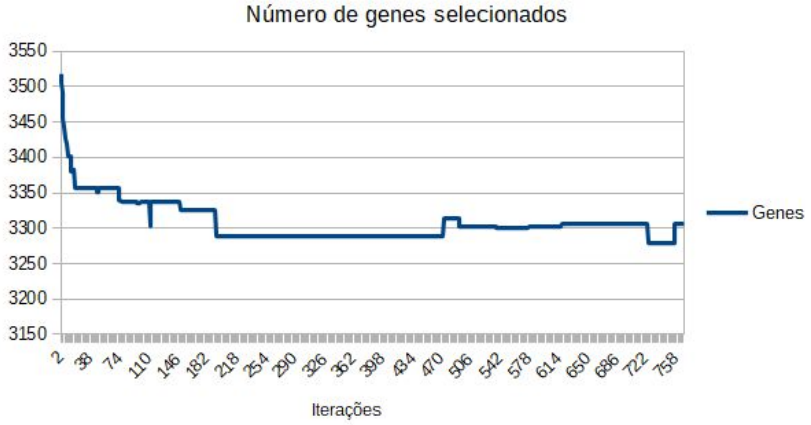
Para realizar o cruzamento da população, utilizou-se uma máscara gerada aleatoriamente para selecionar que gene herdar de qual pai na produção do indivíduo filho. Além disso, a mutação implementada é feita por gene dentre todos os genes de todos os indivíduos filhos. Escolheu-se, nesta primeira implementação, uma taxa de mutação de 20%.

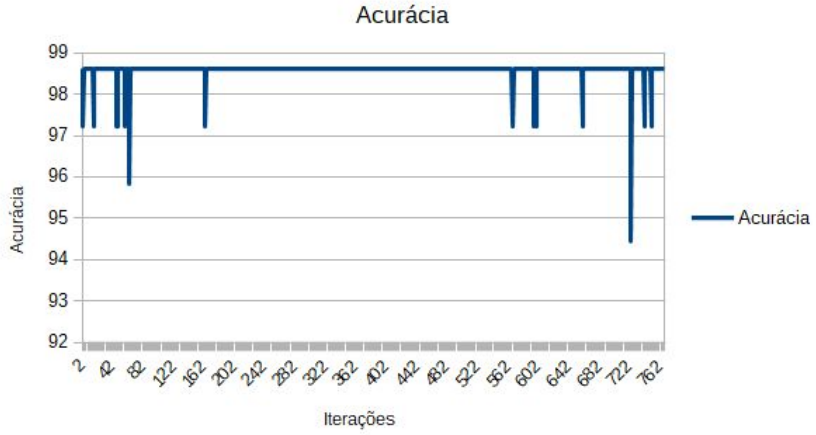
Resultados

Para análise dos resultados, escolheu-se um plot ao longo das iterações para as acurácias, número de genes utilizados e score (pontuação obtida segundo a função de fitness definida).

767 iterações foram feitas, 3306 genes foram selecionados com 98.6% de acurácia.







Implementação 2

Observando os resultados obtidos pela primeira implementação e como o algoritmo evolui, algumas modificações foram feitas para buscar melhorar a performance do modelo.

Modificação 1

Taxa de mutação: como as mutações eram realizadas por gene considerando todos os genes de todos os filhos, ou seja, (1 - elitismo) * tamanho da população * 7128, isso significa que uma taxa de 20% de mutação, trocaria o conteúdo de, em média, 0.2 * 249480 genes, ou 49896 genes dentre a população. Como esse é um número considerável de mutações, o valor foi reajustado para 0.1%, ou seja, em média 249 genes seriam afetados por mutações.

Modificação 2

Número de iterações do K-means: o baixo número de iterações para o K-means responsável por calcular a acurácia do modelo, aliado com sua propriedade estocástica de inicialização, faziam com que alguns indivíduos que passavam de uma geração para outra por meio do elitismo fossem prejudicados por uma execução aleatória que acabou em uma acurácia menor do que a que estes indivíduos de fato poderiam proporcionar. A consequência disto é que seu score diminuía e tais indivíduos eram descartados, mantendo soluções piores pelo elitismo. Sendo assim, o número de iterações aqui consideradas para o K-means dobrou, passando de 25 para 50.

Modificação 3

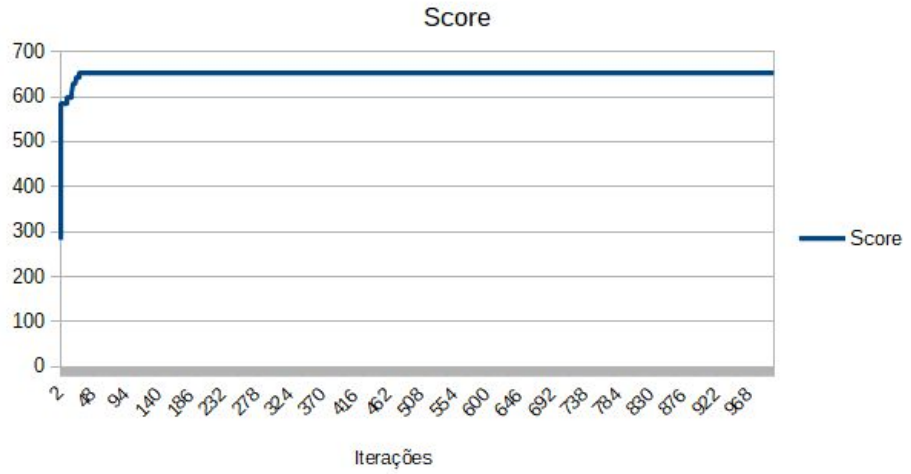
Geração da população inicial: talvez uma das partes mais importantes de uma metaheurística seja a geração da população/soluções inicial. Esta operação era feita randomizando valores booleanos para os genes de cada cromossomo e, em média, metade dos genes do dataset começavam sendo selecionados. Isso aliado ao fato de que a função de fitness priorizava $\log_2$ vezes mais a acurácia, foram fatores determinantes para manter a população com muitos genes sendo selecionados. Pensando nisso, uma nova estratégia para geração da população inicial foi adotada.

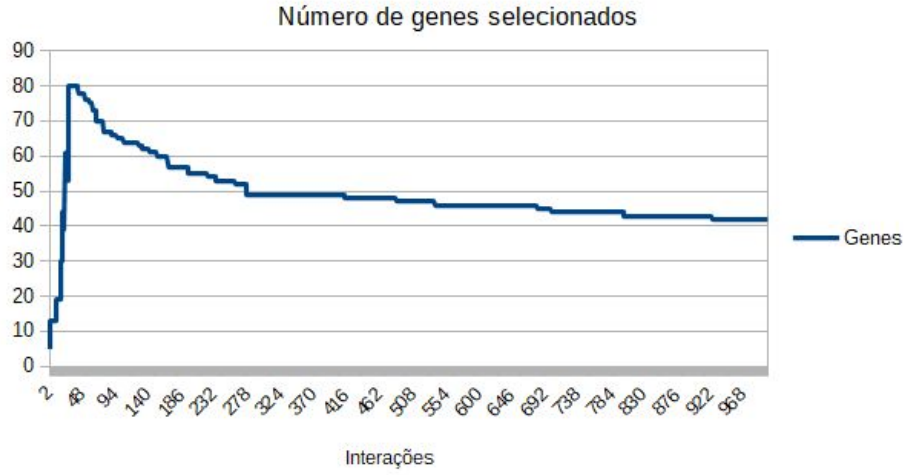
A nova implementação começa criando cromossomos com todos os genes sendo não-selecionados, ou seja, um array com 7128 elementos *False*. Um operação então é definida para gerar 50 (número de indivíduos) números aleatórios no intervalo de [0, 7127], cada número representa o índice do array que vai ter seu elemento trocado de *False* para *True*. Essa operação, então, faz com que cada um dos 50 indivíduos iniciais tenham apenas um gene considerado. É definido, ainda, um loop que realiza tal operação um número x de vezes fazendo com que cada indivíduo tenha (provavelmente) x genes selecionados aleatoriamente. Foi considerado x = 5 nesta implementação.

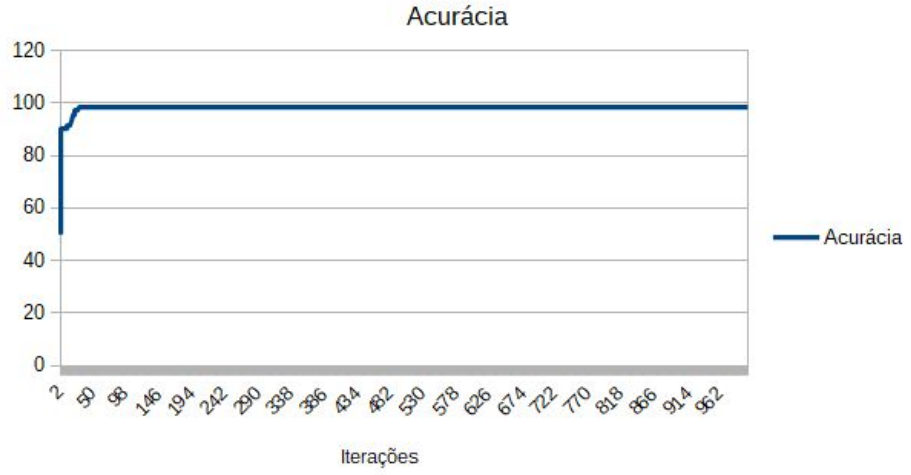
Com esta nova operação, o espaço de busca deixaria de partir de 3500 genes com muita acurácia, para iniciar com 5 genes e baixíssima acurácia.

Resultados

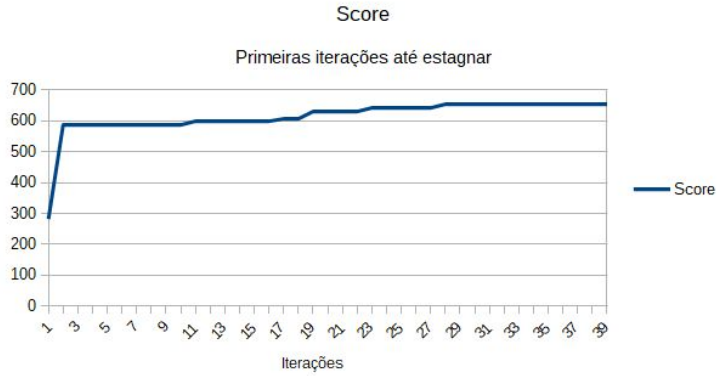
1000 iterações foram feitas, 42 genes foram selecionados com 98.6% de acurácia.

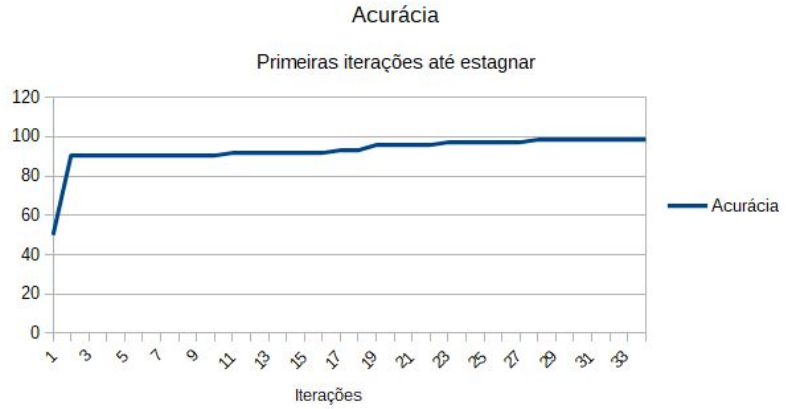


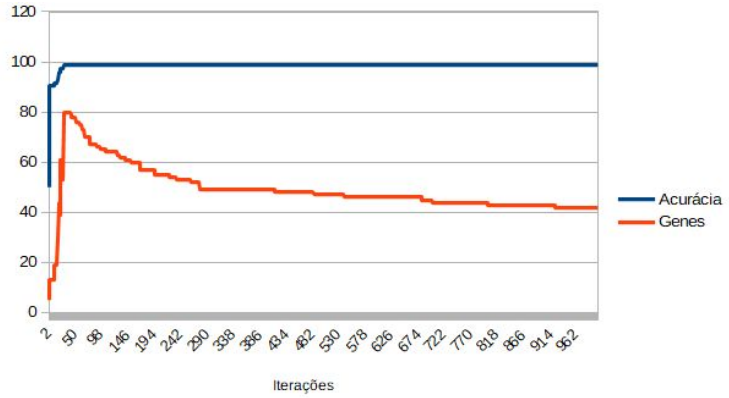


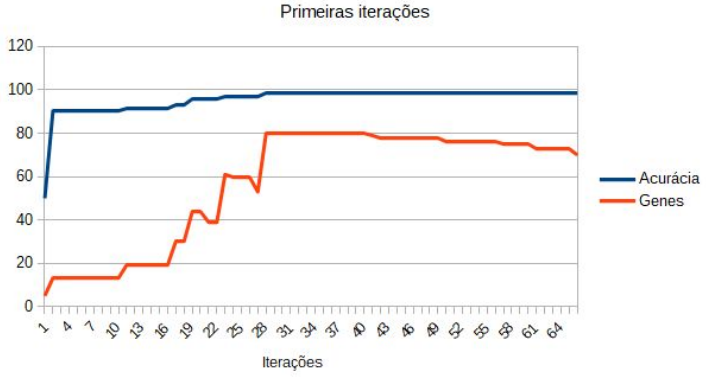


Também é útil observarmos o comportamento da evolução da acurácia e score no início das iterações, antes de estagnarem e da acurácia vs genes ao longo das iterações.









Conclusões

O sucesso das modificações na segunda implementação foi bastante significativo. De mais de 3000 genes selecionados na primeira implementação, com algumas modificações (destacando-se a geração da população inicial), foi possível selecionar apenas 42 genes mantendo a mesma acurácia alcançada pela Implementação 1.

Podemos observar, também, através dos gráficos, que o comportamento segue a idealização original da função de fitness, onde a prioridade da busca deve ser numa acurácia decente para, só então, o foco migrar para a redução de dimensionalidade em si. Exatamente este comportamento pode ser observado nos gráficos, aonde o modelo vai aumentando o número de genes para conseguir melhor acurácia e, depois de atingir o seu maior potencial possível (98.6%), regride retirando genes enquanto mantém esta performance preditiva alcançada. Em outras palavras, vai se refinando.