Computação Natural - TP1
Programação Genética

Guilherme Torres
Departamento de Ciência da Computação - UFMG

1 Introdução

O objetivo deste trabalho foi formular e escolher os melhores parâmetros para um algoritmo de aprendizado para regressão simbólica. Dada uma função $f : \mathbb{R}^n \to \mathbb{R}$, a regressão simbólica consiste em tentar aproximar essa função usando uma árvore constituída por terminais nas folhas (constantes ou variáveis) e operadores.

A heurística usada foi a programação genética, que usa de conceitos evolucionários para evoluir as soluções para o problema e obter indivíduos mais adaptados. É um algoritmo estocástico, ou seja, com vários componentes de aleatoriedade como geração inicial dos indivíduos e mutações, o que requer que muitos experimentos sejam realizados para tirar as conclusões sobre o trabalho, que serão discutidos nesse documento.

O algoritmo foi implementado em Python 3, na pasta /src. Instruções para executar o teste estão explicitadas no arquivo README.md. O programa também pode ser encontrado no repositório:
`https://github.com/torresguilherme/cn-tp1-gp`.

2 Implementação

2.1 Fundamentação

O funcionamento do algoritmo pode ser descrito, no plano geral, pelos seguintes passos:

1. Geração aleatória dos primeiros indivíduos;
2. Mostra a melhor, pior e média das fitness na população;
3. Adiciona o grupo elitista à nova população;

4. Repete até que o número de indivíduos na população possa ser reposto:
   (a) Seleciona dois pais através do torneio;
   (b) Aplica a probabilidade de cruzamento;
   (c) Ao se obter um cruzamento, aplica a chance de mutação;
   (d) Adiciona os filhos gerados à nova população;
5. Repete desde o passo 2, até que o número máximo de gerações seja alcançado.

Como é sugerido, há muitas variáveis envolvidas e deve-se encontrar o valor ótimo para elas por meio de experimentos.

2.2 Decisões de implementação

Indivíduo

Um indivíduo é representado por uma árvore, que tem um valor de uma função. Cada uma das folhas da árvore faz o papel de uma constante ou uma variável. As constantes são geradas aleatoriamente com números de ponto flutuante entre -10 e 10 (distribuição uniforme).

Os operadores escolhidos foram '+', '*' e '/', respectivamente, para adição, multiplicação e divisão. Como a subtração é praticamente uma adição por um número negativo, não foi necessário adicionar um operador '-'.

Geração aleatória de indivíduos

Cada indivíduo começa com um operador a priori, e sua árvore é estendida até que todos os caminhos acabem em um terminal. Como a profundidade máxima do indivídio é 7, apenas terminais serão escolhidos aleatoriamente quando algum nó chegar na profundidade 6.

Cálculo da Fitness

A fitness de um indivíduo é dada pelo seu erro quadrático em relação ao resultado dado para um conjunto de variáveis da função que se quer aprender. O erro de um indivíduo é obtido por

$$\sqrt{1/N \sum_{i=0}^{n-1} (Ev_i - y_i)^2}$$

sendo N o número de avaliações dadas no arquivo de teste, Ev_i a avaliação de um indivíduo para cada conjunto de variáveis, y_i o valor real dado no arquivo para aquele conjunto.

Como a fitness é uma medida de erro, quanto menor o erro, melhor é o indivíduo.

Tamanho da população e número de gerações

O tamanho da população para um algoritmo de programação genética é tipicamente menor do que para um algoritmo genético, onde o objetivo é achar uma solução para um problema em vez de desenvolver um método para ele.

É razoável que a população varie, neste caso, entre 50 e 500. Por motivos de limitações em questão de tempo e recursos computacionais, o tamanho da população foi fixado em 50, assim como o número de gerações. Sabendo disso, é provável que seja necessária uma taxa de mutações e cruzamento maior para obter-se mais diversidade na população reduzida.

Seleção natural

A seleção é realizada por torneiro. Dado um tamanho de torneio T, uma amostra de tamanho T é selecionada também de forma aleatória entre os indivíduos e o vencedor é o que tiver a melhor fitness, no caso, o menor erro quadrático.

A variável T é o principal fator que define a intensidade da pressão seletiva na população. Quanto maior esse número é, mais difícil é a permanência de indivíduos com fitness pior do que a média na população. Por isso, o valor adequado para esse parâmetro também foi definido através de experimentos, dentro de um intervalo de valores.

Cruzamento

O cruzamento também faz parte da seleção natural. Ele ocorre para definir a nova geração a ser avaliada no GP. Dados dois indivíduo que venceram seus respectivos torneios, a probabilidade de ocorrer um cruzamento entre eles subsequentemente é de p_c (entre 60% e 90%).

Uma das principais questões do cruzamento é a possibilidade de se manter a localidade ao cruzar duas árvores (se uma grande parte das árvores for cruzada, a variabilidade gerada pode levar a um indivíduo completamente diferente dos dois pais, o que aumenta drasticamente a aleatoriedade da busca).

Para evitar que um crossover provoque uma variedade maior do que deveria e destrua bons indivíduos, o modelo de crossover adotado é um método mais simples, porém ainda amplamente usado [1], que é a troca de duas folhas aleatoriamente escolhidas entre os pais.

Mutação

Como o cruzamento é escolhido de forma bem conservadora, é papel da mutação provocar uma variabiidade mais drástica e promover a diversidade entre a população. A estratégia usada consiste em escolher um nó aleatório da árvore, podar os seus filhos e re-extender a árvore aleatoriamente a partir desse ponto [1].

Para cada indivíduo que surge de um cruzamento, a chance de ele sofrer uma mutação é p_m (entre 5% e 30%).

Elitismo

Passar um pequeno grupo dos melhores indivíduos da população atual para a próxima geração sem submetê-los ao processo de seleção é um processo muito utilizado para fazer algoritmos convergir mais rápido e preservar bons genótipos. O elitismo é adotado neste projeto também.

Como o número de indivíduos da população é relativamente pequeno, julgou-se razoável manter o tamanho do grupo elitista em apenas um, para que a pressão seletiva não fosse desproporcional e assim, evitando que todos os indivíduos ficassem iguais antes do número de iterações desejado.

3 Experimentos

Os experimentos foram realizados em um desktop com sistema operacional Debian 9, Python 3.5.3, processador Pentium Dual Core (3.3 GHz) e 8GB de memória RAM. Foram usadas três bases de dados diferentes (house, keijzer-10 e keijzer-7, cada um com um arquivo de treino e teste, todos disponíveis na pasta */datasets*) Cada experimento foi repetido 30 vezes antes de se tirar as conclusões.

Os atributos do algoritmo foram testados na seguinte ordem:

1. Variação de p_m de 0.05 a 0.3
2. Variação de p_c de 0.9 a 0.6
3. Variação de T de 2 a 5

Os gráficos plotados a seguir incluem a média aritmética dos dados dos experimentos obtidos para cada configuração, no *arquivo de treino*. As tabelas apresentam a média dos melhores valores obtidos para a função, levando em consideração o *arquivo de teste*.

3.1 Mutação

As figuras de 1 a 3 demonstram a média do erro das melhores soluções para os arquivos de teste nos experimentos realizados. Os plots azul, vermelho, marrom e preto equivalem, respectivamente, às taxas de mutação de 0.05, 0.1, 0.15 e 0.2.

Os dados plotados dão uma boa ideia de como a solução converge. Podemos reparar que, como era esperado, valores maiores de mutação são os que promovem a diversidade necessária para que a convergência a longo prazo seja mais favorável. Este parece ser o caso para todos os conjuntos de dados.

Usar um valor de mutação mais baixo, por sua vez, promove menos diversidade, porém facilita a convergência. Pode-se perceber que o menor valor de mutação (linha azul) no geral converge relativamente rapidamente, porém nos dois últimos testes (que são bases de dados menores e com menos variáveis), ele fica estagnado. Isso pode ser um sinal de que os indivíduos da população ficam iguais em certo ponto, e a chance de se conseguir sair dessa situação é baixa.

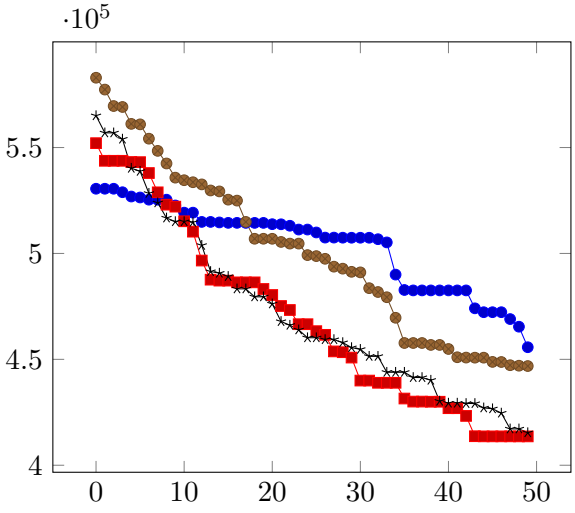

Figure 1: Testes de mutação: melhores soluções, house-train.csv

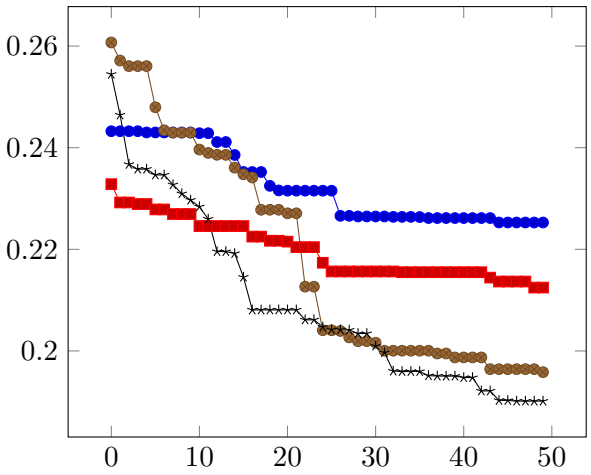

Figure 2: Testes de mutação: melhores soluções, keijzer-10-train.csv

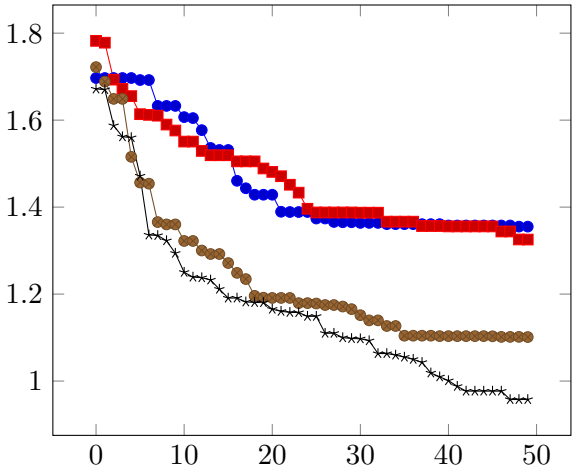

Figure 3: Testes de mutação: melhores soluções, keijzer-7-train.csv

3.2 Cruzamento

3.3 Torneio

4 Conclusões

5 Referências

1. *A Field Guide To Genetic Programming.* - Riccardo Poli, William B. Langdon, and Nicholas F. McPhee, 2008

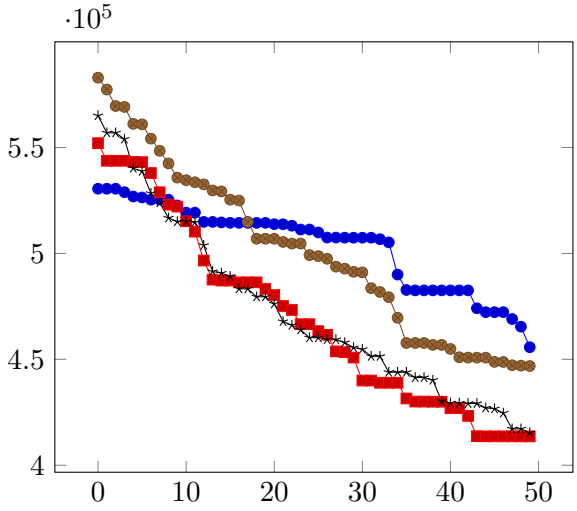

Figure 4: Testes de mutação: melhores soluções, house-train.csv

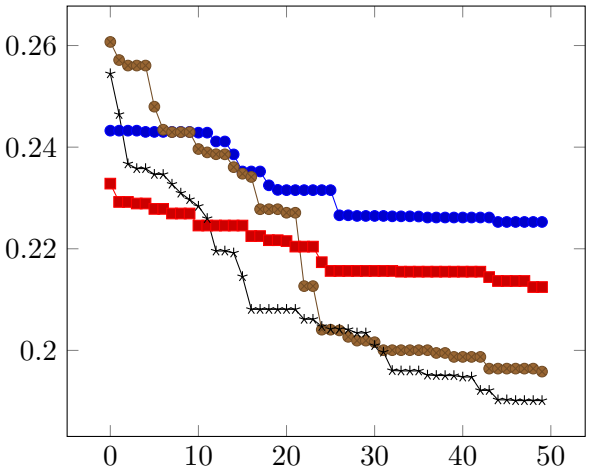

Figure 5: Testes de mutação: melhores soluções, keijzer-10-train.csv

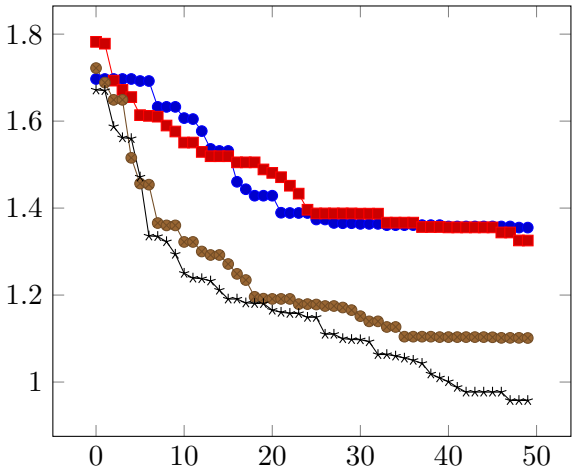

Figure 6: Testes de mutação: melhores soluções, keijzer-7-train.csv

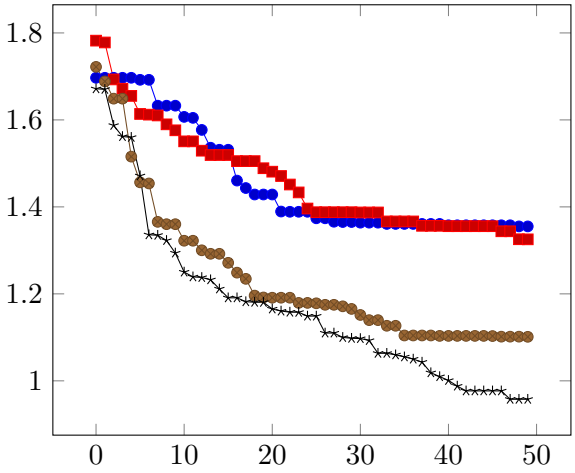

Figure 7: Testes de mutação: melhores soluções, keijzer-7-train.csv

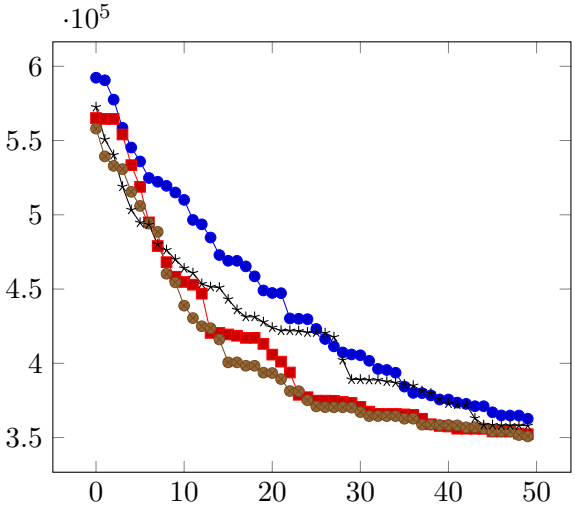

Figure 8: Testes de mutação: melhores soluções, house-train.csv

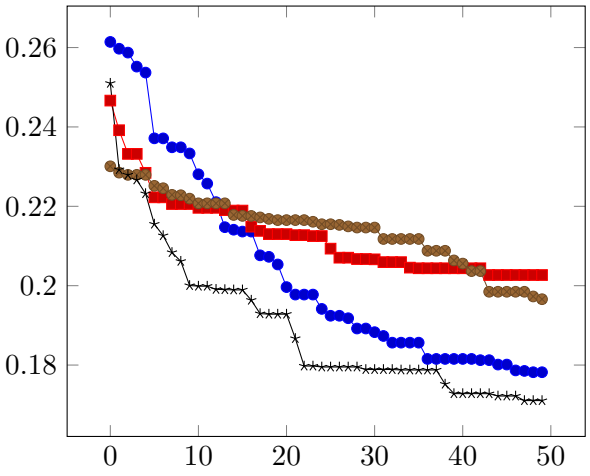

Figure 9: Testes de mutação: melhores soluções, keijzer-10-train.csv

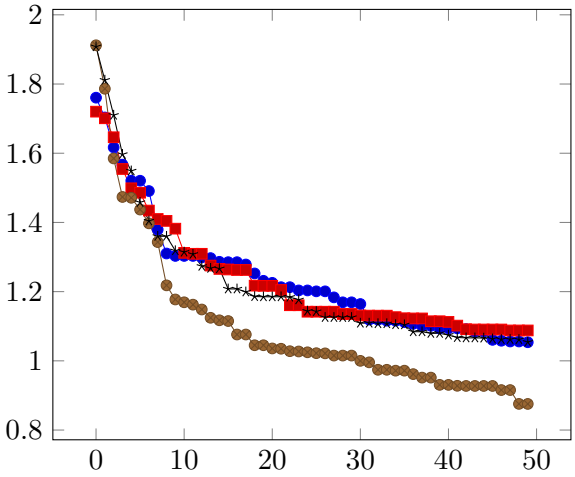

Figure 10: Testes de mutação: melhores soluções, keijzer-7-train.csv