TP 1 - Regressão Simbólica com Programação Genética

Gabriel Henrique Souto Pires {gabrielpires@dcc.ufmg.br}

Computação Natural - 2/2017
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais

Resumo

Regressão simbólica é um dos problemas mais conhecidos em Programação Genética. É comumente usado como um problema de ajuste para novos algoritmos, mas também é amplamente usado em casos onde outros métodos de regressão podem não ser ideais. É conceptualmente um problema simples, e portanto é uma boa forma de se introduzir programação genética. Este documento tem como objetivo analisar, discutir e apresentar este problema, assim como formas de resolvê-lo através de programação genética. Serão apresentados detalhes da implementação, assim como experimentos feitos nas bases de dados fornecidas à fim de concluir como a programação genética é útil na resolução desse tipo de problema.

1 Introdução

O problema de regressão simbólica consiste em encontrar uma expressão matemática que melhor descreve uma relação dada por um conjunto de valores. Começamos construindo uma população de fórmulas geradas de forma aleatória para representar uma relação entre variáveis independentes conhecidas e uma parte dependente destas variáveis. Cada geração de "funções" é então evoluída da geração que veio antes dela selecionando a *fitness* dos indivíduos da população para que operações genéticas sejam feitas sobre eles.

À medida que indivíduos bons são encontrados, esses indivíduos são mantidos na população seguinte para que sua "qualidade" seja passada adiante. Este procedimento é chamado de *elitismo*. Novos indivíduos são gerados através de cruzamento ou mutação. No caso do cruzamento 2 "pais" são selecionados para dar origem a dois novos indivíduos. Essa seleção é feita por meio de torneio, onde um número de indivíduos da população é selecionado ao acaso e estes indivíduos competem entre si, o indivíduo com o melhor fitness é então selecionado como vencedor e é usado no cruzamento. Na mutação, um dos filhos gerados pelo cruzamento tem uma chance de ter sua árvore alterada por redução, expansão ou então ter apenas um nó alterado, o que gera mais diversidade na população, uma vez que as possíveis soluções não ficam presas à diversidade criada na geração da população inicial.

Depois que uma nova população é criada, o processo é aplicado à ela e às gerações seguintes até que se chegue em um critério de parada.

2 Implementação

Os indivíduos são representados como árvores, sendo que os nós internos são operadores (funções) e as folhas são constantes ou variáveis. Os operadores disponíveis para a criação dos indivíduos são $\{+, -, *, /, cos, sin\}$ sendo que os operadores que representam funções trigonométricas tem apenas um filho à direita e os demais operadores tem sempre dois filhos, que podem ser variáveis, terminais, ou outros operadores. Cada operador da como saída um número, e pode receber como entrada a saída de outro operador, obedecendo a propriedade de *fechamento*.

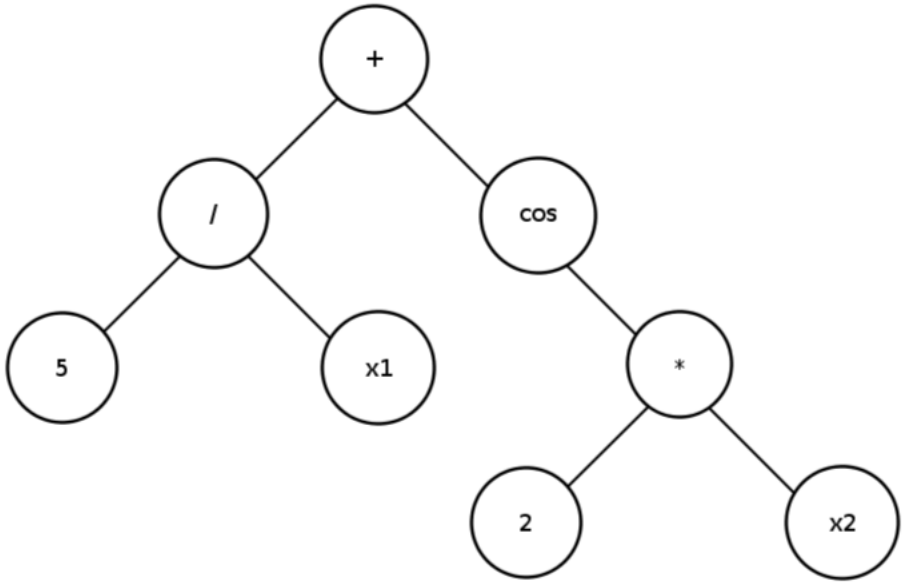

Figura 1: Indivíduo que representa a expressão $(\frac{5}{x1} + cos(2 * x2))$

Os indivíduos e as árvores usadas para representar os indivíduos foram implementados em duas classes que contém os atributos necessários para rodar o algoritmo. Alguns destes atributos são:

Classe Indivíduo

- genotype: Ponteiro para a raiz da árvore de expressão (representação do indivíduo).
- ind_size: Tamanho da árvore que o indivíduo tem (usado para limitar o crescimento do indivíduo).
- num_of_var: Quantas variáveis existem no arquivo de entrada (x1, x2, ..., x8 por exemplo).
- node_number: Usado para dar um número identificador para cada nó, o que ajuda na hora de sortear um nó na mutação.
- fitness: A fitness do indivíduo.

Classe Tree

- node_value: O que o nó representa (constante, operador ou variável).
- left, right: Os filhos desse nó (lembrando que funções trigonométricas só tem um filho).
- number: O número identificador que foi dado para esse nó específico da árvore.

Algumas constantes também foram criadas para controlar os parâmetros, que são:

- ind_max_size: Tamanho máximo do indivíduo.
- max_gen: Número máximo de gerações.
- init_pop_size: Número inicial de indivíduos.
- max_pop_size: Número máximo de indivíduos que a população pode atingir (após isso começam a ser descartados os piores indivíduos).
- tourn_size: Tamanho do torneio (quanto maior o torneio maior a pressão seletiva).
- cross_rate: Frequência em que cruzamentos ocorrem.
- mut_rate: Frequência em que mutações ocorrem.

Para gerar a população inicial, a função *generate_population* é chamada. Dentro dela um loop roda *init_pop_size* vezes criando indivíduos da classe *Individual*. No construtor da classe *Individual* alguns valores padrão são atribuídos ao objeto e o genótipo do indivíduo é gerado.

Na função *generateGenotype* uma árvore aleatória é criada a partir da raiz. A raiz da árvore é criada de forma que seja sempre um operador, dessa forma evitamos que a raiz da árvore seja uma constante ou variável, o que nos deixaria com uma árvore de um nó só. A partir do segundo nó da árvore, cada nó tem uma probabilidade de 60% de virar um operador, 20% de virar uma variável e 20% de virar uma constante. Essa decisão de implementação foi tomada para evitar que indivíduos muito pequenos fossem maioria na população, o que mais tarde iria prejudicar a evolução. Quando um nó vira uma constante ou variável, ele não gera mais filhos e se torna uma folha da árvore.

Depois de gerar toda a população inicial, a fitness de cada indivíduo é calculada na função *calc_fitness* de acordo com a fórmula da raiz quadrada do erro quadrático médio (RMSE)

$$f(Ind) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(Eval(Ind,x) - y)^2}$$

Com a fitness da população calculada, usamos a função *evolve* para criar uma nova população a partir da atual. Nessa função verificamos se a população atingiu o limite de indivíduos, se sim a função *remove_worst* é chamada para remover o indivíduo com a pior fitness da população. Após isso a função *keep_the_elite* é chamada e faz o inverso da função anterior, salvando e enviando para a próxima população o indivíduo com a melhor fitness da população atual.

Depois de manter o elite, rodamos um loop até que a população esteja vazia. Nesse loop temos uma chance de chamar o crossover e uma chance de simplesmente reproduzir o indivíduo, No crossover, 2 indivíduos são selecionados por um torneio de tamanho 2 e são usados como pais. Com os pais selecionados, nós aleatórios das árvores de cada um são selecionados e trocados entre eles. Após fazer o cruzamento, existe uma probabilidade de que os filhos sofram uma mutação, nessa mutação um ou mais nós da árvore do indivíduo são alterados aleatoriamente, gerando mais diversidade na população. Caso o crossover não seja feito, algum indivíduo da população é jogado para a próxima população sem ser alterado.41

3 Experimentos

Para realizar os experimentos, foram testadas várias configurações dos parâmetros do PG como os listados abaixo:

- Máximo tamanho do indivíduo: 7
- Tamanho da população inicial: 50
- Tamanho máximo da população: 350
- Tamanho do torneio: 2
- Número máximo de gerações: 300
- Taxa de crossover: 0.95
- Taxa de mutação: 0.05

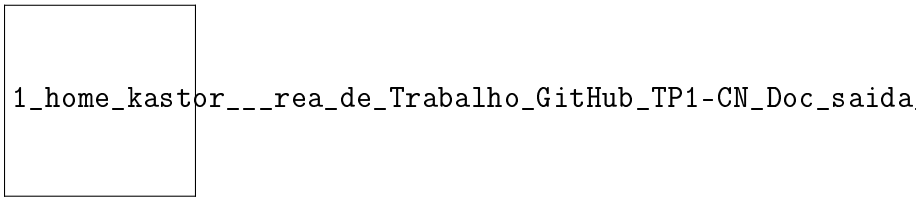

Figura 2: Keijzer 7 - Média de 30 execuções

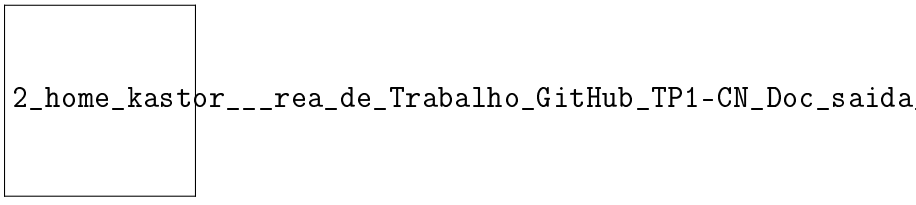

Figura 3: Keijzer 10- Média de 30 execuções

Figura 4: saida_house_med

4 Conclusões

5 Bibliografia