Trabalho Prático 3 - Meta Heurística - Programação Genética para predição de Churn

Leandro Lázaro Araújo Vieira - 3513[1], Mateus Pinto da Silva - 3489[2]

[1]Ciência da Computação – Universidade Federal de Viçosa (UFV-caf) – Florestal – MG – Brasil

(leandro.lazaro[1], mateus.p.silva[2])@ufv.br

Resumo. *Programação genética é bastante sedutora ao programador, pois é agnóstica em relação à entrada e, por conseguinte, permite códigos bastante flexíveis. Resolvemos o problema de predição de Churn proposto utilizando programação genética através do Framework Deap da linguagem Python. Optamos pela implementação mais simples possível, fazendo pré-processamento dos dados para limpeza, utilizando tipagem estática e funções puras. Conseguimos resultados bastante semelhantes nas duas configurações propostas e nos scores de fitness de treinamento e de teste. Em média, obtemos 78% de acurácia.*

1. Pré-processamento dos dados

Optamos por limpar os dados antes de usá-los nos algoritmos de programação genética a fim de poupar processamento da próxima fase. Retiramos as colunas **Unnamed: 0**, **IDCliente** e **Codigo**, que não são úteis para previsão de Churn. Além disso, trocamos o tipo da coluna **MesesComoCliente** de inteiro para ponto flutuante, mantivemos as colunas **ValorMensal** e **TotalGasto** como ponto flutuante também, e convertemos todas as outras colunas em variáveis dummy, ou seja, booleanos. Utilizamos o método $N - 1$ de conversão para dummy, ou seja, colunas com duas categorias geram apenas uma variável dummy, o que gera menos colunas e poupa processamento da fase posterior. Além disso, retiramos todas as linhas que continham alguma coluna com valor ausente.

Dessa forma, geramos um conjunto de dados com 31 colunas, sendo as 3 primeiras de ponto flutuante e todas as outras 28 com valores booleanos, sendo a última delas o resultado do Churn. Além disso, tal conjunto ficou com 5974 linhas, sendo que 26.56% dos usuários efetivamente fizeram Churn.

2. Metodologia

Para realizar o propóstio deste trabalho utilizamos o framework DEAP da linguagem de programação Python. Optamos por utiliza-lo devido a sua vasta aplicação em problemas desse tipo, além de se mostrar uma ferramenta bem estabelecida. Utilizamos o seu modo tipado, pois os tipos das colunas eram conhecidos, o que torna a evolução mais correta (não gerando árvores de decisão defeituosas) e mais performática.

Para utilizar o framework, antes de tudo é necessário definir os tipos das colunas de entrada e da saída. Definimos, seguindo o pré-processamento, 3 variáveis de ponto flutuante e 27 booleanas. O tipo, é claro, foi definido como booleano, sendo o valor verdadeiro como cliente que efetivamente fez o Churn (ou seja, classe Churn), e falso como o cliente que não fez (ou seja, classe não Churn). Depois, é necessário definir quais

serão as possíveis funções (ou nós internos) das árvores de decisão Assim, definimos os operadores AND, OR, XOR, NOT para entrada e saída de tipo booleano. Os ADD (soma), SUB (subtração), MUL (multiplicação), DIV (divisão) para entrada e saída do tipo ponto flutuante. LT (menor que), EQ (igualdade) para entrada com pontos flutuantes e saída em booleano. E, por fim, IF_TE (Se então) que tem como entrada dois pontos flutuantes e um booleano, e retorna um ponto flutuante. Além disso, é importante ressaltar que a divisão é um pouco diferente, pois trata-se de uma divisão protegida, que caso o seu divisor seja 0, o retorno será -1.0 ao invés de uma exceção.

Em ambas as configurações, optamos por utilizar 40 gerações de 100 indivíduos, além de usar sempre 1 indivíduo adicional para o elitismo. Como método de seleção, optamos por torneio aleatório de tamanho 3. Utilizamos crossover (ou seja, união de árvore com subárvore) como método de cruzamento e com taxa de 50%. Como mutação, utilizamos uma geração de expressão aleatória válida a ser colocada no lugar de uma subárvore do indivíduo que mantém o seu tamanho, a fim de evitar bloat adicional.

Sobre as duas configurações específicas pedidas, utilizamos dois algoritmos diferentes com parâmetros apropriados. A primeira configuração usa o algoritmo **eaSimple**, que é uma implementação do algoritmo mais simples proposto por Baeck no capítulo 7 do seu livro [Baeck et al. 2000] e utilizamos nele 20% de taxa de mutação. A segunda configuração usa o algoritmo **eaMuCommaLambda** que é uma implementação de seleção comma e utilizamos nele 10% de taxa de mutação.

3. Discussão dos resultados

Obtivemos resultados interessantes, principalmente quando comparamos com uma implementação que apenas chuta que o cliente não fará Churn. Nessa hipótese, a acurácia seria de 73.44%, enquanto a nossa acurácia foi entorno de 78%. As duas configurações geraram resultados muito próximos, o que mostra que conseguimos um resultado estável. Os conjuntos de treino e teste também tiveram resultados próximos, o que mostra que existe um padrão razoavelmente estabelecido entre todos os clientes.

Algoritmo	Mínimo	Máximo	Média	Desvio-padrão
Configuração A	0.735714	0.787723	0.759234	0.013162
Configuração B	0.735714	0.783929	0.756585	0.011561

Tabela 1. Resultados do conjunto de treinamento.

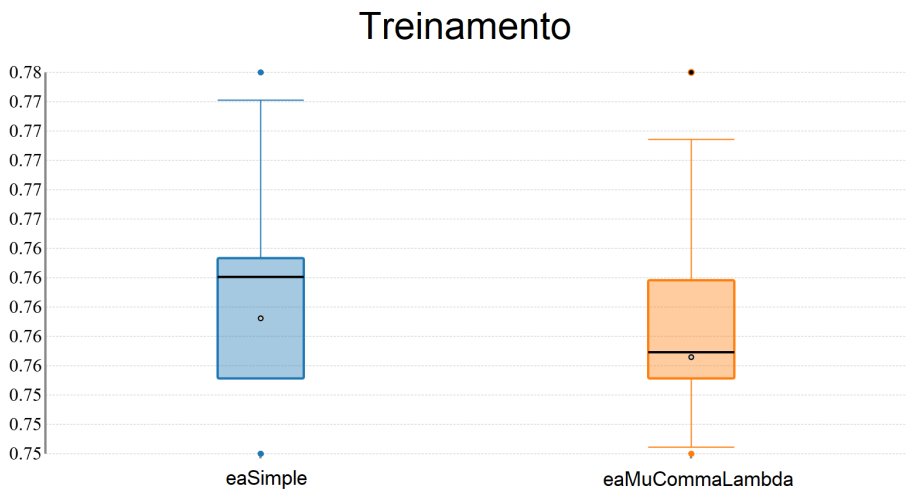

Figura 1. Resultados do conjunto de treinamento.

Algoritmo	Mínimo	Máximo	Média	Desvio-padrão
Configuração A	0.730254	0.783133	0.757006	0.013767
Configuração B	0.730254	0.782463	0.755533	0.013251

Tabela 2. Resultados do conjunto de teste.

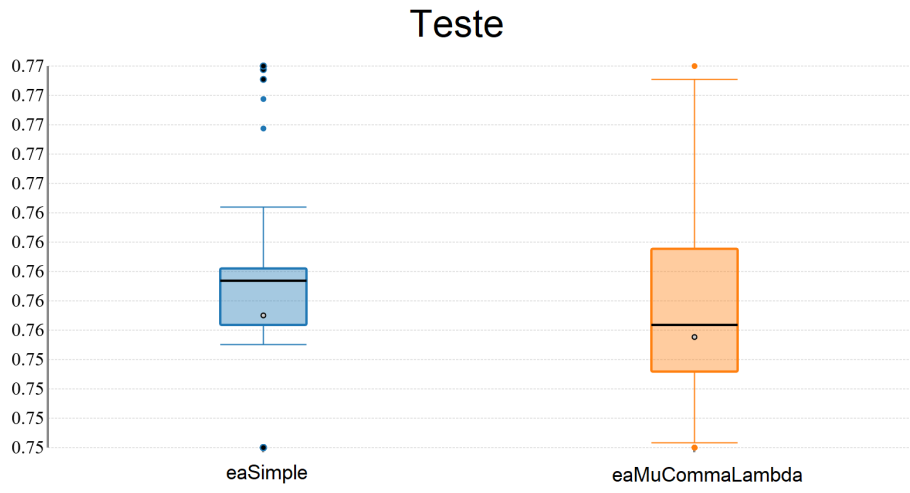

Figura 2. Resultados do conjunto de teste.

3.1. Árvore resultante da configuração 1 (eaSimple)

and_(ServicoInternet_Fibra, lt(p_div(add(ValorMensal, TotalGasto), sub(add(TotalGasto, TotalGasto), add(ValorMensal, TotalGasto))), sub(if_te(ServicoStreamingTV_SemInternet, add(ValorMensal, TotalGasto), sub(ValorMensal, MesesComoCliente)), add(sub(MesesComoCliente, Valor-Mensal), if_te(and_(ServicoInternet_Fibra, ServicoStreamingTV_SemInternet),

sub(MesesComoCliente, TotalGasto), add(MesesComoCliente, if_te(or_(Casado, TipoContrato_Mensal), mul(MesesComoCliente, MesesComoCliente), add(ValorMensal, ValorMensal))))))))

3.2. Árvore resultante da configuração 2 (eaMuCommaLambda)

lt(mul(p_div(p_div(sub(TotalGasto, MesesComoCliente), sub(TotalGasto, add(p_div(ValorMensal, TotalGasto), sub(sub(add(TotalGasto, MesesComoCliente), sub(TotalGasto, TotalGasto)), TotalGasto)))), mul(add(MesesComoCliente, TotalGasto), p_div(MesesComoCliente, TotalGasto))), sub(if_te(not_(lt(MesesComoCliente, mul(if_te(ServicoSegurancaOnline, ValorMensal, MesesComoCliente), sub(MesesComoCliente, ValorMensal)))), p_div(TotalGasto, TotalGasto), if_te(xor(eq(ValorMensal, mul(add(ValorMensal, ValorMensal), if_te(TipoContrato_Mensal, ValorMensal, MesesComoCliente))), Dependentes), ValorMensal, ValorMensal)), p_div(p_div(ValorMensal, MesesComoCliente), if_te(GeneroMasculino, MesesComoCliente, MesesComoCliente)))), sub(p_div(sub(add(TotalGasto, MesesComoCliente), sub(TotalGasto, TotalGasto)), sub(add(mul(mul(ValorMensal, ValorMensal), if_te(ServicoBackupOnline_SemInternet, ValorMensal, MesesComoCliente)), MesesComoCliente), sub(TotalGasto, MesesComoCliente))), p_div(if_te(or_(ServicoSegurancaOnline, ServicoInternet_Nao), mul(TotalGasto, TotalGasto), if_te(GeneroMasculino, MesesComoCliente, MesesComoCliente)), add(TotalGasto, p_div(MesesComoCliente, ValorMensal)))))

Referências

[Baeck et al. 2000] Baeck, T., Fogel, D., and Michalewicz, Z. (2000). *Evolutionary Computation 1: Basic Algorithms and Operators*. Basic algorithms and operators. Taylor & Francis.