Documentação
INF2102 – Projeto Final de Programação

Ambiente de Simulação para *Iterative Machine Teaching*

Francisco Sergio de Freitas Filho
ffilho@inf.puc-rio.br

Julho de 2020

1 Introdução

Neste projeto buscamos desenvolver um ambiente de simulações em C++ [1] que nos apoie em nossa pesquisa na área de *Iterative Machine Teaching*. *Machine Teaching* estuda quão eficientemente um *Teacher* ensina uma hipótese desejada para um *Learner*. Nossa pesquisa é voltada para o modelo de *Machine Teaching* introduzido em [2], onde o ensino é feito de forma iterativa e não há nenhum conhecimento sobre o algoritmo e conjunto de hipóteses do *Learner*, com exceção que esse conjunto tem a hipótese perfeita. Nesse cenário, o *Learner* é denominado *black box*. A grosso modo, o problema consiste em um *Teacher* que envia iterativamente amostras de um conjunto de dados até que o *Learner* consiga generalizar tal conjunto. O objetivo é atingir essa generalização com o menor número de exemplos possível, encontrando o denominado *teaching set* ou uma boa aproximação desse conjunto. Para se aprofundar mais no problema e seus desafios, consultar [2, 3, 1, 4, 5, 6, 7].

Os objetivos deste projeto são :

1. gerar cenários sintéticos reprodutíveis;
2. simular a interação entre *Teacher* e *Learner*;
3. criar relatórios com informações de cada iteração;
4. padronizar a criação de novos teachers;
5. padronizar a criação de novos learners.

2 Projeto

2.1 Diagrama de Classes

O diagrama de classes do sistema é apresentado na Figura 1.

- **HypothesesGenerator**: Classe abstrata de um gerador de hipóteses;
- **SimpleRandomHypotheses**: Classe responsável por definir um conjunto de hipóteses disjuntas de forma aleatória;
- **Learner**: classe abstrata para Learner;
- **RandomLearner**: Classe responsável interagir com o *Teacher* recebendo os exemplos selecionados pelo *Teacher* e escolhendo aleatoriamente uma hipótese entre as que acertam todos os exemplos recebidos. Apesar da escolha aleatória, em particular, a hipótese perfeita só é retornada em última ocasião;
- **AdversaryLearner**: Classe responsável interagir com o *Teacher* recebendo os exemplos selecionados pelo *Teacher* e escolhendo a hipótese de pior acurácia entre as que acertam todos os exemplos recebidos;
- **Teacher**: Classe abstrata de Teacher. O Teacher é responsável por ensinar o *Learner* iterativamente conduzindo-o para a hipótese desejada;

[1] https://www.cplusplus.com/

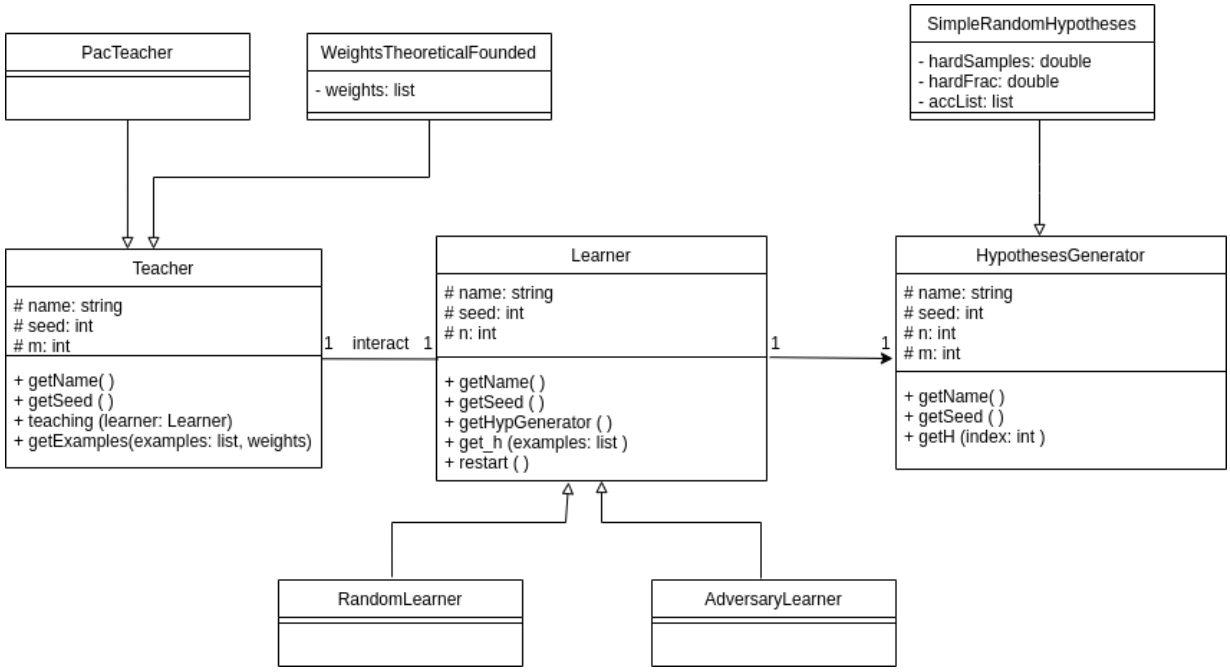

Figura 1: Diagrama de classes.

- **PacTeacher**: Classe responsável por ensinar o *Learner* iterativamente enviando exemplos aleatórios (distribuição uniforme) e desconsiderando seu *feedback*;
- **WeightsTheoreticalFounded (WTF)**: Classe responsável por ensinar o *Learner* iterativamente enviando exemplos em que o *Learner* erra. Esses exemplos são escolhidos de forma aleatária cuja probabilidade de escolha de um exemplo é baseada em seu peso.

2.2 Casos de Uso

Em nosso sistema sistema de simulação temos dois atores, (1) Usuário e (2) Sistema:

1. Usuário: O ator pode interagir com o sistema de três maneiras. Ele pode i) instanciar os objetos e executar a simulação manualmente, ii) definir um arquivo de configuração para automatizar esse processo e iii) criar novas classes de forma padronizada.

2. Sistema: Esse ator interage com a aplicação realizando múltiplas simulações de forma automática a partir de um arquivo de configuração de cenários.

2.2.1 Simular interação entre Teacher e Learner

Objetivo: simular a interação entre Teacher e Learner e salvar informações de cada iteração.

Atores: usuário.

Requisitos: instanciar um objeto que herda da classe Learner e um objeto que herda da classe Teacher.

Fluxo principal: i) instanciar um objeto que herda de HypothesesGenerator; ii) instanciar um objeto que herda de Learner; iii) instanciar um objeto que herda de Teacher; iv) Executar o método teaching de Teacher passando o Learner; v) Salvar o *log* retornado.

2.2.2 Executar simulações entre Teachers e Learners a partir de um arquivo de configuração de cenários

Objetivo: executar, para combinações de parâmetros, diversas simulações entre Teachers e Learners e salvar informações de todas simulações.

Atores: sistema.

Requisitos: escrever arquivo de configuração.

Fluxo principal: i) ler o arquivo de configuração de cenários; ii) instanciar objetos que herdam de HypothesesGenerator; iii) instanciar objetos que herdam de Learner; iv) instanciar um objetos herdam de Teacher; v) Executar o método teaching para cada par (Teacher, Learner) definido; vi) Salvar o *log* de todas as execuções.

2.2.3 Criar um novo Teacher para o ambiente

Objetivo: criar um novo Teacher para o ambiente de simulações.

Atores: usuário.

Requisitos: escrever uma classe que herda da classe Teacher e implementar o método *teaching*.

Fluxo principal: i) criar uma classe que herda de Learner; ii) implementar o método *teaching*; iii) adicionar a função na classe DataHandler para instanciar o novo Teacher a partir do arquivo de configuração de cenários.

2.2.4 Criar um novo Learner para o ambiente

Objetivo: criar um novo Learner para o ambiente de simulações.

Atores: usuário.

Requisitos: escrever uma classe que herda da classe Learner e implementar o método *get_h* e *restart*.

Fluxo principal: i) criar uma classe que herda de Learner; ii) implementar os métodos *get_h* e *restart*; iii) adicionar a função na classe DataHandler para instanciar o novo Learner a partir do arquivo de configuração de cenários.

2.2.5 Criar um novo gerador de hipóteses para o ambiente

Objetivo: criar um novo gerador de hipóteses para o ambiente de simulações.

Atores: usuário.

Requisitos: escrever uma classe que herda da HypothesesGenerator.

Fluxo principal: i) criar uma classe que herda de HypothesesGenerator; ii) adicionar a função na classe DataHandler para instanciar o novo gerador de hipóteses a partir do arquivo de configuração de cenários.

2.3 Tecnologias

O modo como esse ambiente de simulações foi idealizado fez com que sua implementação não necessite de tecnologias sofisticadas e bibliotecas especializadas em *Machine Learning*. Por outro lado, velocidade sempre foi algo essencial desde a definição do projeto, isso devido ao grande número de simulações necessárias para consolidar os resultados estatísticos da natureza dos algoritmos implementados. Tais fatores levaram a escolha da linguagem de programação C++ [2] utilizado o ambiente de desenvolvimento integrado (IDE) QT Creator [3] (QMake 3.1, Qt versão 5.9.7) e sistema operacional Ubuntu 18.04.3.

2.4 Testes Realizados

Para testar os módulos desenvolvidos foram implementados testes unitários das principais funções do programa. O módulo de testes foi desenvolvido manualmente. Em particular, em casos em que as funções testadas são funções com aleatoriedade, foram feitas milhares de chamadas e os resultados foram comparados com os valores esperados. Os testes foram realizados nas componentes bases que compõe métodos mais complexos: funções de escolhas aleatórias com base em distribuições dadas, comportamento dos *Teachers*, reprodutibilidade dos experimentos, testes sobre o conjunto de hipóteses gerados, dentre outros. Todos os testes estão reunidos na classe *UnitTests* (./src/Tests/unittests.h).

1. **Módulo SimpleRandomHypotheses:** esse módulo é essencial e responsável por gerar o conjunto de hipóteses. Foram feitos testes para checar a existência de uma hipótese perfeita, se a acurácia das hipóteses seguem uma distribuição uniforme e se a proporção de exemplos mais difíceis estava sendo respeitada.

2. **Módulo PacTeacher:** foi testado se os retornos de duas execuções com a mesma semente eram iguais, se o PacTeacher ignora o Learner e se o *teaching set* retornado não apresenta repetição de exemplos.

3. **Módulo WTF:** foi testado se os retornos de duas execuções com a mesma semente eram iguais, se o *teaching set* retornado não apresenta repetição de exemplos e o método *getExamples* é exemplificado adiante.

4. **Learners:** a parte complexa desses módulos recai no gerador de hipóteses cujos testes já foram relatados. Testes simples também foram feitos indiretamente nos passos anteriores.

Na Figura 2 é apresentado uma parte da função que testa o método *getExamples* do *Teacher WeightsTheoreticalFounded*. Esse método recebe uma lista de exemplos na qual o *Learner* errou e o *Teacher* escolhe um exemplo (nesse teste é escolhido apenas um exemplo) utilizando a distribuição expressa pela lista *deltas*. Nesse teste, a probabilidade de um exemplo de índice par ser escolhido é o dobro da probabilidade de um exemplo ímpar ser escolhido. Além disso, nesse caso, a soma das probabilidades (*sumDeltas*) é menor que 1.0, resultando na possibilidade de nenhum exemplo ser escolhido. A função em questão foi executada 10.000 vezes e foi analisado se o resultado esperado ocorreu com um erro de no máximo 3%.

[2] https://www.cplusplus.com/
[3] https://www.qt.io/

```
int count = 0;// contador de vezes que nenhum exemplo foi selecionado
int even = 0; // contador de vezes que um exemplo de indice par foi selecionado
int odd = 0; // contador de vezes que um exemplo de indice impar foi selecionado

for(int i=0; i<10000; i++){
    //selecionando um exemplo da lista examples cujas probabilidades dos exemplos sao dadas por delta
    std::vector<int> selectedExamples = WTF.getExamples(examples, deltas);
    if(selectedExamples.size() == 0)
        count++; //incrementando se nenhum exemplo foi selecionado
     else if(selectedExamples[0]%2 == 0)
        even++; //incrementando se o exemplo retornado tem indice par
    else
        odd++; //incrementando se o exemplo retornado tem indice impar
}
//testando a probabilidade de nao sortear nenhum exemplo: TESTE 1
double ratio = count/((1-sumDeltas)*10000);
if(ratio < 0.97 || ratio > 1.03)
    return std::pair<bool, std::string> (false, "Falha no metodo getExamples() do WTF: TESTE 1");

//testando a probabilidade de sortear pares e impares: TESTE 2
ratio = even/(2.0*odd); //e esperado que um numero par tenha duas vezes mais chances de ser sorteado do que um impar
if(ratio < 0.97 || ratio > 1.03)
    return std::pair<bool, std::string> (false, "Falha no metodo getExamples() do WTF: TESTE 2");

return std::pair<bool, std::string> (true, "Metodo getExamples() do WeightsTheoreticalFounded testado com sucesso");
```

Figura 2: Exemplo de teste realizado para a função getExamples

3 Documentação

A utilização do simulador pode ser via executável (disponivel em ./src/Framework) ou manualmente via código como exemplificado na Figura 3.

O complemento da documentação com instruções de uso detalhadas estão descritas no arquivo **README.md** disponível juntamente com o executável e código fonte comentado em: `https://github.com/sfilhofreitas/ProjetoFInaldeProgramacao`.

4 Resultados

A implementação deste projeto de programação foi de fundamental importância para o andamento da pesquisa de doutorado. Inclusive, a partir desta ferramenta foram obtidos parte dos experimentos utilizados em uma publicação recente na International Conference on Machine Learning (ICML) [4]. Portanto, fica claro a utilidade da ferramenta e forte relação com a pesquisa desenvolvida no doutorado.

Referências

[1] Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph Naor. The online set cover problem. *SIAM J. Comput*, 39(2):361–370, 2009.

[4] https://icml.cc/

```cpp
#include "HypothesesGenerator/simple_random_hypotheses.h"
#include "Learners/random_learner.h"
#include "Teachers/pac_teacher.h"
#include "Teachers/weights_theoretical_founded.h"
#include <exception>
#include "Learners/adversary_learner.h"
#include "DataHandler/data_handler.h"
#include "Tests/unittests.h"

using namespace std;

int main(int argc, char* argv[]){
    int n=1000; //tamanho do conjunto de hipoteses do Learner
    int m=10000; //tamanho do conjunto de exemplos
    int seed = 0; // semente de aleatoriedade dos objetos
    double hardSamples = 0.2;// existirao hardSamples*m exemplos mais dificeis
    double hardFrac = 0.3; //hardFrac dos erros de cada hipotese vem dos exemplos mais dificeis
    std::vector<double> accList; //lista de acurácia das hipóteses
    accList.push_back(0.9); //algumas hipoteses podem ter acuracia de 0.9
    accList.push_back(0.95);//algumas hipoteses podem ter acuracia de 0.95

    HypothesesGenerator* Hypotheses = new SimpleRandomHypotheses(n, m, accList, hardSamples, hardFrac, seed);
    RandomLearner randomLearner(Hypotheses, seed);
    WeightsTheoreticalFounded WTF(m, seed);
    PacTeacher pacTeacher(m, seed);

    pair<vector<int>, string> solution = WTF.teaching(randomLearner);//a funcao teaching retorna o teaching set e o log
    cout << solution.first.size() << endl; //saida: 89

    pair<vector<int>, string> solution2 = pacTeacher.teaching(randomLearner);//a funcao teaching retorna o teaching set e o log
    cout << solution2.first.size() << endl; //saida: 126

    return 0;
}
```

Figura 3: Exemplo de como instanciar os objetos e executar simulações manualmente.

[2] Sanjoy Dasgupta, Daniel Hsu, Stefanos Poulis, and Xiaojin Zhu. Teaching a black-box learner. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 1547–1555. PMLR, 2019.

[3] Thorsten Doliwa, Gaojian Fan, Hans Ulrich Simon, and Sandra Zilles. Recursive teaching dimension, vc-dimension and sample compression. *J. Mach. Learn. Res*, 15(1):3107–3131, 2014.

[4] Edward Johns, Oisin Mac Aodha, and Gabriel J. Brostow. Becoming the expert - interactive multi-class machine teaching. In *CVPR*, pages 2616–2624. IEEE Computer Society, 2015.

[5] Weiyang Liu, Bo Dai, Ahmad Humayun, Charlen Tay, Chen Yu, Linda B. Smith, James M. Rehg, and Le Song. Iterative machine teaching. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 2149–2158. PMLR, 2017.

[6] Weiyang Liu, Bo Dai, Xingguo Li, Zhen Liu, James M. Rehg, and Le Song. Towards black-box iterative machine teaching. In Jennifer G. Dy and Andreas Krause 0001, editors, *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 3147–3155. PMLR, 2018.

[7] Yao Zhou, Arun Reddy Nelakurthi, and Jingrui He. Unlearn what you have learned: Adaptive crowd teaching with exponentially decayed memory learners. In Yike Guo and Faisal Farooq, editors, *KDD*, pages 2817–2826. ACM, 2018.