Trabalho 1
Aprendizado de Máquina - 2018.1
Universidade Federal do Rio de Janeiro

Raphael Barros Parreira

20 de Julho de 2018

Índice

1 Introdução

O trabalho consiste na participação de uma competição do Kaggle. A competição prevê a predição de preços de casas baseada em suas características. Para isso, é necessário que seja aplicado o conhecimento adquirido no curso para fazer com que um programa de computador seja capaz de aprender como fazer essa predição baseado numa base de dados conhecida previamente.

Além de fornecer a base de dados de treino (que contém os preços das casas) e a base de teste (base usada para fazer a predição final), o Kaggle também fornece materiais de outros participantes, que auxiliam o processo de aprendizagem e desenvolvimento do programa.

Uma outra informação presente na competição é a descrição de cada uma das características possíveis das casas, para que auxilie o desenvolvedor a escolher o que fazer com cada uma das colunas da base de dados.

1.1 Arquivos fornecidos pelo Kaggle

- *train.csv* - dataset de treino
- *test.csv* - dataset de teste
- $data_description.txt$ - descrição de cada uma das características da casa
- $sample_submission.csv$ - exemplo de arquivo para ser submetido ao Kaggle

1.2 Como funciona a submissão à competição

Para submeter a reposta ao Kaggle, o competidor deve criar um arquivo *.csv* (valores separados por vírgulas) contendo duas colunas: Id e SalePrice. Além disso, os preços e os Ids devem corresponder, para que o sistema do Kaggle consiga validar o preço para o seu respectivo Id.

Ao enviar o arquivo para a validação, o Kaggle verifica a pontuação da tentativa e informa a colocação.

1.3 Premissas

O trabalho apresenta, como premissa, o fato de se usar apenas algoritmos de regressão para realizar a predição dos preços. Além disso, deixou livre a escolha dos algoritmos por parte do participante. Sendo assim, os algoritmos escolhidos foram: Ridge, Lasso, ElasticNet e KernelRidge.

2 Modelagem do Sistema

O PCP(Priority Ceiling Protocol) tem como princípio estender o PIP(Priority Inheritance Protocol) com a adição de uma regra de concessão para um pedido de bloqueio de um semáforo livre.

Com isso, o PCP impede que uma tarefa entre na região crítica se existe(m) semáforo(s), que possivelmente pode(m) bloqueá-la, esteja(m) trancado(s), evitando assim que ocorra um bloqueamento múltiplo. Ou seja, quando uma tarefa entrar numa seção crítica, ela nunca poderá ser bloqueada por uma tarefa de prioridade mais baixa até que esteja completa.

Com esse propósito, cada semáforo recebe uma "*priorityceiling*" igual à prioridade mais alta das tarefas que podem trancar este semáforo. Sendo assim, só é permitido a uma tarefa τ_i entrar na região crítica se sua prioridade for maior do que todas as prioridades dadas aos semáforos trancados, por outras tarefas, naquele instante.

O algoritmo funciona da seguinte forma:

Inicialmente, cada semáforo S_k recebe uma prioridade $C(S_k)$ igual a maior prioridade das tarefas que podem trancá-lo.Em seguida analisa-se qual a tarefa ativa com a maior prioridade. Feito isso, designa-se esta tarefa para o processador.

Para entrar numa seção crítica guardada por um semáforo S^*, τ_i precisa ter uma prioridade mais alta do que $C(S^*)$, caso contrário, o travamento de S^* é negadp e τ_i é bloqueada no semáforo S^* pela tarefa responsável pelo travamento de S^*. Nesse caso, a tarefa τ_i transmite sua prioridade para a tarefa τ_j que está travando o semáforo, isto é τ_j herda a prioridade de τ_i, e assim sendo a tarefa τ_j retorna a ser executada até terminar sua região crítica.

Terminada a região crítica de τ_j ela destrava o semáforo S^* e atualiza sua prioridade, caso τ_j não esteja bloqueando mais nenhuma tarefa ela retorna à sua prioridade normal, caso contrário ele herda a maior prioridade das tarefas que ainda está bloqueando.

Observação:

A herança de prioridades é transitória, ou seja caso a tarefa τ_3 esteja bloqueando a tarefa τ_2 que por sua vez bloqueia τ_1, a tarefa τ_3 herda a prioridade de τ_1.

3 Trechos importantes do código

Primeiramente é importante ressaltar que os trechos de códigos vistos abaixo foram escritos na linguagem do Matlab.

3.1 Variáveis importantes

```
handles.ns = 0;
handles.Soriginal = [];

.
.
.

chegada = str2num(get(handles.edit1, 'String'));

tempComp = str2num(get(handles.edit2, 'String'));

deadline = str2num(get(handles.edit3, 'String'));

prioridade = str2num(get(handles.edit4, 'String'));
```

No trecho acima pode-se observar algumas variáveis importantes para o bom funcionamento do protocolo estudado, em que:

- handles.ns é o número de semáforos utilizados por um determinado conjunto de tarefas;
- handles.Soriginal são os parâmetros de cada semáforo e sua relação com cada tarefa, ou seja, quanto tempo a tarefa irá utilizá-lo e apartir de que instante ela necessitará desse semáforo.

São as variáveis globais do código, isto é, podem ser acessadas por qualquer função do programa. Já as variáveis abaixo, são as variaveis locais da função principal do programa.

- chegada é um array que contém os tempo de chegada de cada tarefa;
- tempComp é um array que armazena os tempos de execução para cada tarefa;
- deadline array para armazenar os tempos limites(deadlines) de cada tarefa a ser executada;
- prioridade é um array que armazena as prioridades "normal"de cada tarefa;

É importante observar a função $get(handle.,'')$ vistas na atribuição de cada uma dessas últimas variáveis listadas, isso uma vez que esta função é responsável por adquirir o valor do campo correspondente àquela variável na interface.

3.2 Botão de inserção

Nesta seção analisa-se o funcionamento do botão *Inserir*.

```
function pushbutton2_Callback(hObject, eventdata, handles)
w = handles.ns + 1;
initial_temp = cellstr(get(handles.listbox1,'String'))
a = get(handles.edit6, 'String');
temp = [initial_temp; a];
set(handles.listbox1, 'String', temp);
b =  str2num(a);
handles.ns = w;
handles.Soriginal(:,:,w) = str2num(get(handles.edit6, 'String'));
guidata(hObject, handles);
```

Este botão, como o nome sugere serve para a inserção de um semáforo na execução das tarefas, para isso é necessário que se escreva os parâmetros do semáforo, isto é, tempo em que cada tarefafaz o requerimento para utilizá-lo; duração da execução de cada tarefa utilizando este semáforo e por fim o "status"da tarefa em relação ao semáforo sendo:

1. 0 → para tarefas inativas;
2. 1 → para tarefas ativas;
3. 2 → para tarefas finalizadas ou inexistentes em relação ao semáforo.

3.3 Loop principal

Nesta seção é mostrado o trecho do código onde "a mágica acontece", isso tendo em vista que é nesse trecho que todas as comparações e definições de prioridades ocorrem como pode ser visto abaixo:

```
if length(tam)< 3;
    tam(3) = 1;
end
for i=1:tam(3)
    prioridadeBloqueada(i)=101;

    [cores(i,1),cores(i,2),cores(i,3)]=HSVtoRGB(1+(i-1)*(358/(tam(3))),.41,.86);

    for j=1:tam(1)
        if S(j,2,i)>0
            C(i)=j;
            break;
        end
    end
end

C(length(C)+1)=100;

tempo=0;

for i=1:length(chegada)
    prioridadeAtiva(i)=100;
end


ativos=zeros(length(chegada));


largura=max((sum(sum(S(:,2,:)))+sum(tempComp))*1.2,(max(deadline)+2));


tic;


cla
hold on
grid on
set(gca,'xtick',[0:1:largura])

xlim([0,largura])
ylim([0,20*(1+length(chegada))])
set(gca,'YTick',[0:20:20*length(chegada)])

for i=1:length(chegada)
    plot([chegada(i),chegada(i)],[20*(length(chegada)+1-i),20*(length(chegada)+1-i)
     +15],'-.b','LineWidth',2)
    plot([deadline(i),deadline(i)],[20*(length(chegada)+1-i),20*(length(chegada)+1-i
     )+15],'-.r','LineWidth',2)
end

drawnow


SmaiorCeiling=length(C);


while (true)

    paraContador=toc;


    tic;


    tempo=tempo+paraContador;

```

```
    for i=1:length(chegada)
        if ~ativos(i) && chegada(i)<=tempo
            ativos(i)=1;
            prioridadeAtiva(i)=prioridade(i);
        end
    end


    [minimo,proximoProcesso]=min(prioridadeAtiva);

    for i=1:tam(3)
        if S(proximoProcesso,3,i)==0 & compFeita(proximoProcesso) >= S(proximoProcesso,1,i)
            for j=1:tam(3)
                if any(S(:,3,j)==1) & C(SmaiorCeiling)>C(j)
                    SmaiorCeiling=j;
                end
            end

            if prioridadeAtiva(proximoProcesso)<C(SmaiorCeiling) | any(S(proximoProcesso,3,:)==1)
                S(proximoProcesso,3,i)=1;

            else

                prioridadeAtiva(find((S(:,3,SmaiorCeiling))==1))=proximoProcesso-0.1;

                prioridadeBloqueada(SmaiorCeiling)=proximoProcesso-0.1;
            end
        end
    end

    if minimo<100

        if ~(any(S(proximoProcesso,3,:)==1))

            compRestante(proximoProcesso)=compRestante(proximoProcesso)-paraContador;
            if compRestante(proximoProcesso)<0
                compRestante(proximoProcesso)=0;
                tempo=tempo+compRestante(proximoProcesso);
                paraContador=paraContador-compRestante(proximoProcesso);
            end

            compFeita(proximoProcesso)=compFeita(proximoProcesso)+paraContador;

            rectangle('Position',[tempo-paraContador,20*(length(chegada)+1-proximoProcesso),paraContador,10],'FaceColor',[0.6 0.6 0.9],'LineStyle','None');
            drawnow

        else

            zonaAtiva=[0 0];
            for i=1:tam(3)
                if S(proximoProcesso,3,i)==1 & S(proximoProcesso,1,i)>=zonaAtiva(2)
                    zonaAtiva=[i S(proximoProcesso,1,i)];
                end
            end

            S(proximoProcesso,2,zonaAtiva(1))=S(proximoProcesso,2,zonaAtiva(1))-paraContador;
            compFeita(proximoProcesso)=compFeita(proximoProcesso)+paraContador;

            if S(proximoProcesso,2,zonaAtiva(1))<0
                S(proximoProcesso,2,zonaAtiva(1))=0;
                tempo=tempo+S(proximoProcesso,2,zonaAtiva(1));
                paraContador=paraContador-S(proximoProcesso,2,zonaAtiva(1));
            end

            rectangle('Position',[tempo-paraContador,20*(length(chegada)+1-proximoProcesso),paraContador,10],'FaceColor',cores(zonaAtiva(1),:),'LineStyle','None');
            xlim([0,largura])
            ylim([0,20*(1+length(chegada))])
            set(gca,'YTick',[0:20:20*length(chegada)])
            drawnow

            if S(proximoProcesso,2,zonaAtiva(1))<=0
                %Marca fim da zona
                S(proximoProcesso,3,zonaAtiva(1))=2;

                prioridadeBloqueada(zonaAtiva(1))=101;

                prioridadeAtiva(proximoProcesso)=proximoProcesso;
                for i=1:tam(3)
                    if S(proximoProcesso,3,i)==1
                        prioridadeAtiva(proximoProcesso)=min(prioridadeAtiva(proximoProcesso),prioridadeBloqueada(i));
                    end
                end

                SmaiorCeiling=length(C);

                for j=1:tam(3)
                    if any(S(:,3,j)==1) & C(SmaiorCeiling)>C(j)
                        SmaiorCeiling=j;
                    end
```

```
154                     end
155
156                 end
157
158
159             end
160         end
161
162         if compRestante(proximoProcesso)<=0
163
164             prioridadeAtiva(proximoProcesso)=105;
165
166             if max(compRestante)<=0
167                 break;
168             end
169         end
```

3.4 Explicação do Programa

Para um melhor entendimento do programa uma breve explicação do mesmo se faz necessária.

Para a simulação, o usuário deve fornecer vetores com as informações dos tempos de chegada, tempo de computação, deadline e as informações sobre as zonas críticas.

As zonas críticas, associadas cada uma a um recurso exlusivo e um semáforo, devem ser fornecidas com tempos de chegada e de computação. O tempo de chegada é o tempo total de computação da tarefa que deve ter se passado para que ela entre numa zona crítica, esse tempo leva em conta outras zonas críticas pelas quais essa tarefa pode ter passado.

O simulador funciona em loop, que só acaba quando todas as tarefas terminam de ser computadas. A cada ciclo, o programa mede quanto tempo se passou desde o ciclo anterior, para simular o melhor possível um sistema em tempo real.

Ao início de cada loop o programa primeiro checa se alguma tarefa nova chegou, caso sim, ele a marca com uma tarefa ativa. Após isso, há uma checagem de qual é a tarefa ativa que possui maior prioridade (no caso o valor menor entre os dados pelo usuário) e essa é a tarefa que rodará nesse ciclo. A tarefa que roda no ciclo atual começa verificando se ela acabou de entrar em uma zona crítica. Caso tenha entrado, é preciso verificar se há algum ceiling $C(S^*)$ com prioridade maior ou igual à da tarefa atual entre os semáforos ativos, para bloquear e passar a prioridade da tarefa pra S^*, caso sim, ou permitir a entrada na zona crítica, caso não.

Após essa verificação, há a verificação de se há alguma zona crítica ativa ou não. Caso haja, o programa identifica qual é e desconta do seu contador de duração da zona crítica o tempo do ciclo, verificando se a zona crítica chegou ao fim, para poder liberar a tarefa de maior prioridade por ela bloqueada e reduzir a prioridade da tarefa atual de acordo com a lógica do PCP. Caso não haja zona crítica ativa, o tempo do ciclo é descontado do contador de duração da computação, verificando se a computação chegou ao fim, para desativar a tarefa.

Com essa forma de funcionamento, o simulador é capaz de simular o PCP ao longo de tantos loops quanto forem necessários para que todas as tarefas terminem de computar.

4 Exemplos de simulações

Nesta seção é possível observar algumas simulações feitas afim de exemplificar o PCP e assim verificar se a implementação do mesmo está correta.

4.1 Teste 1

4.1.1 Configurações

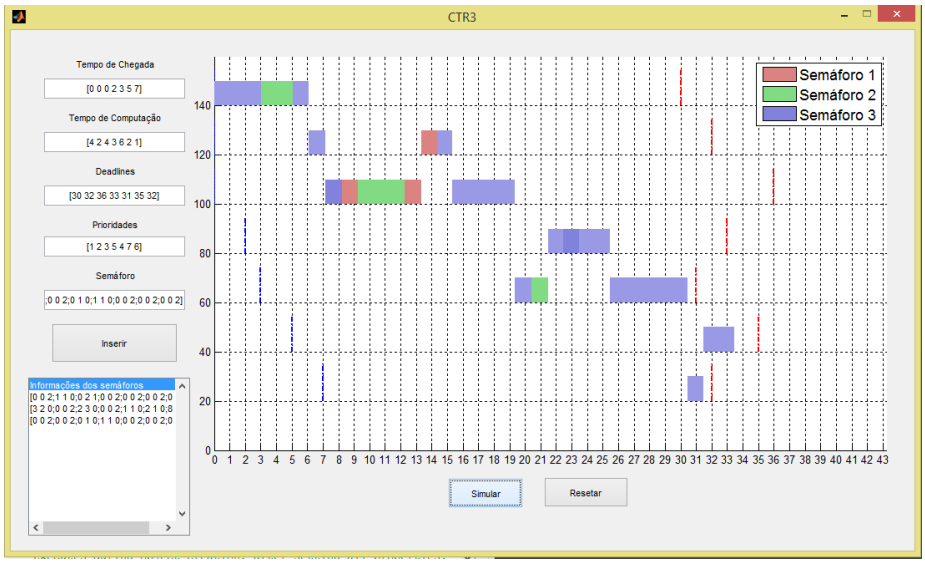
Tabela 1: *Tarefas.*

Parâmetros	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7
a_i	0	0	0	2	3	5	7
C_i	4	2	4	3	6	2	1
D_i	30	32	36	33	30	40	31
P_i	1	2	3	5	4	7	6

Tabela 2: *Semáforo.*

Tarefas	Sa_1	Sa_C1	Status 1	Sa_2	Sa_C2	Status 2	Sa_3	Sa_C3	Status 3
τ_1	0	0	2	3	2	0	0	0	2
τ_2	1	1	0	0	0	2	0	0	2
τ_3	0	2	1	2	3	0	0	1	0
τ_4	0	0	2	0	0	2	1	1	0
τ_5	0	0	2	1	1	0	0	0	2
τ_6	0	0	2	2	1	0	0	0	2
τ_7	0	0	2	0	0	2	0	0	2

4.1.2 Resultado da Simulação



4.2 Teste 2

4.2.1 Configurações

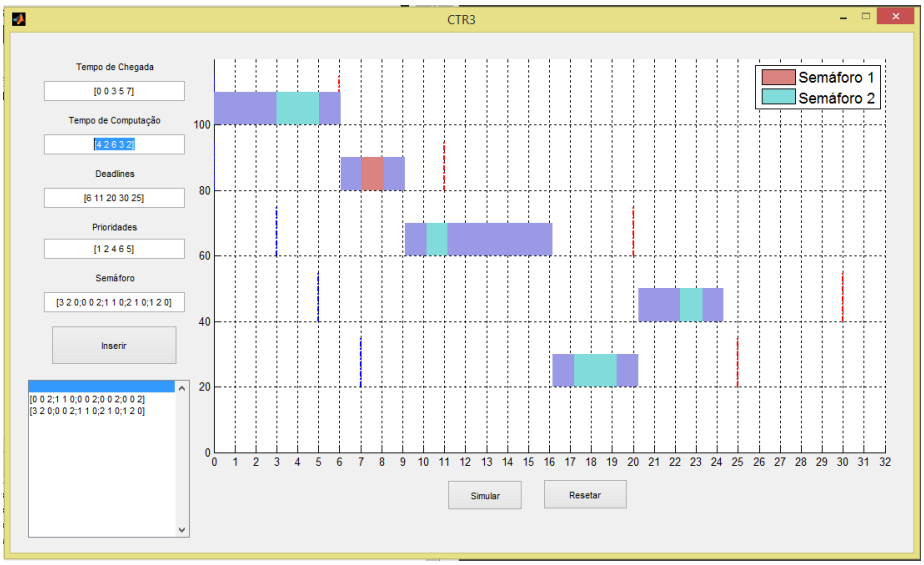
Tabela 3: *Tarefas.*

Parâmetros	τ_1	τ_2	τ_3	τ_4	τ_5
a_i	0	0	3	5	7
C_i	4	2	6	3	2
D_i	6	11	20	30	25
P_i	1	2	4	6	5

Tabela 4: *Semáforo.*

Tarefas	Sa_1	Sa_C1	Status 1	Sa_2	Sa_C2	Status 2
τ_1	0	0	2	3	2	0
τ_2	1	1	0	0	0	2
τ_3	0	0	2	1	1	0
τ_4	0	0	2	2	1	0
τ_5	0	0	2	1	2	0

4.2.2 Resultado da Simulação



4.3 Teste 3

4.3.1 Configurações

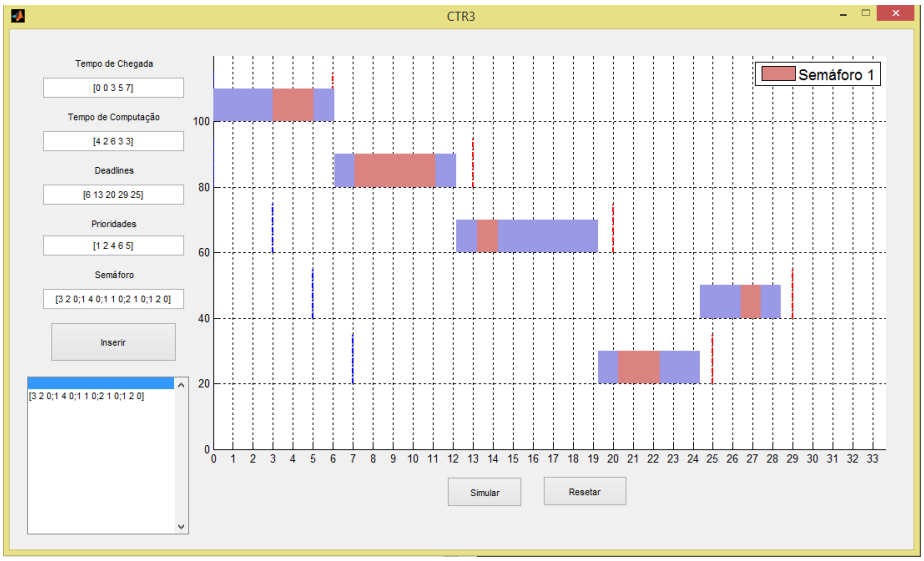
Tabela 5: *Tarefas.*

Parâmetros	τ_1	τ_2	τ_3	τ_4	τ_5
a_i	0	0	3	5	7
C_i	4	2	6	3	3
D_i	6	13	20	29	25
P_i	1	2	4	6	5

Tabela 6: *Semáforo.*

Tarefas	Sa_1	Sa_C1	Status 1
τ_1	3	2	0
τ_2	1	4	0
τ_3	1	1	0
τ_4	2	1	0
τ_5	1	2	0

4.3.2 Resultado da Simulação



4.4 Teste 4

4.4.1 Configurações

Tabela 7: *Tarefas.*

Parâmetros	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8
a_i	1	0	0	0	0	0	0	0
C_i	4	2	7	5	6	3	3	9
D_i	30	32	36	33	37	43	45	55
P_i	1	2	3	5	4	7	6	8

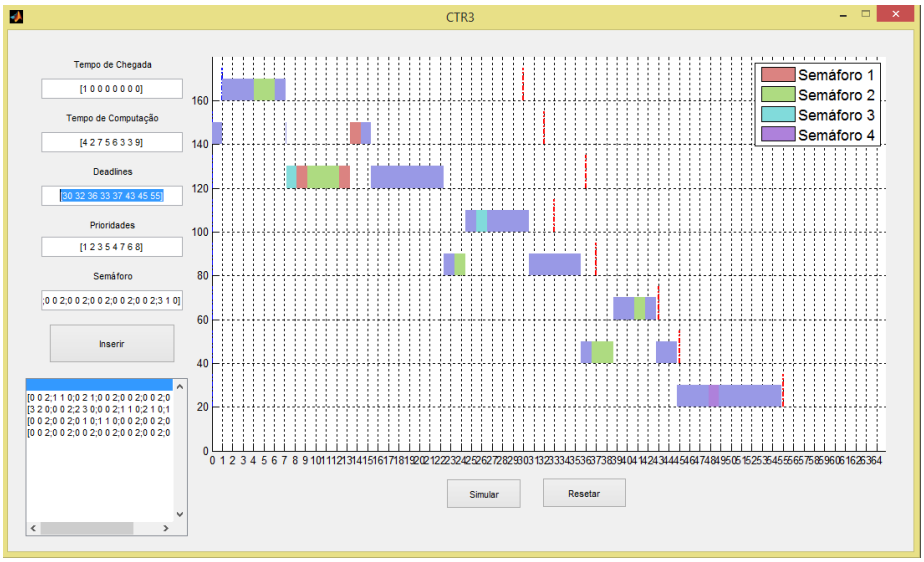
Tabela 8: *Semáforos 1.*

Tarefas	Sa_1	Sa_C1	Status 1	Sa_2	Sa_C2	Status 2
τ_1	0	0	2	3	2	0
τ_2	1	1	0	0	0	2
τ_3	0	2	1	2	3	0
τ_4	0	0	2	0	0	2
τ_5	0	0	2	1	1	0
τ_6	0	0	2	2	1	0
τ_7	0	0	2	1	2	0
τ_8	0	0	2	0	0	2

Tabela 9: *Semáforos 2.*

Tarefas	Sa_3	Sa_C3	Status 3	Sa_4	Sa_C4	Status 4
τ_1	0	0	2	0	0	2
τ_2	0	0	2	0	0	2
τ_3	0	1	0	0	0	2
τ_4	1	1	0	0	0	2
τ_5	0	0	2	0	0	2
τ_6	0	0	2	0	0	2
τ_7	0	0	2	0	0	2
τ_8	0	0	2	3	1	0

4.4.2 Resultado da Simulação



5 Anexo