26/10/2013

Lucas Aoki Chao – 6ºBCC

Aprendizado de Máquina

Classificação de Vinho

**Introdução**

Programa para classificar vinhos, utilizando Gradiente Descendente com regularização.

**Dados**

These data are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines.

1) Alcohol

2) Malic acid

3) Ash

4) Alcalinity of ash

5) Magnesium

6) Total phenols

7) Flavanoids

8) Nonflavanoid phenols

9) Proanthocyanins

10)Color intensity

11)Hue

12)OD280/OD315 of diluted wines

13)Proline

Number of Instances

class 1: 59

class 2: 71

class 3: 48

Programa desenvolvido em C++ utilizando MS Visual Studio 2012.

**Programa**

**Modelo**

h(x) = θ0 + θ1\*x1 + θ2\*x2 + ... + θ13\*x13

Valores esperados de h(x) : 1, 2 ou 3.

double hx(int i)

{

//bias

double resultado = teta[0];

//teta\*x para cada teta

for(int j = 1; j < 14; j++)

resultado += teta[j] \* x[j][i];

return resultado;

}

**Custo J(x)**

double regularizacao()

{

double temp = 0.0;

for(int j = 0; j < 14; j++)

temp += pow(teta[j],2);

return (lamb \* temp);//coeficiente de regularizacao

}

double jx()

{

double erro = 0;

//somatoria do quadrado dos erros

for(int i = min; i <max; i++)

erro += pow((hx(i)-x[0][i]),2);

return ((1.0/(2\*(max - min))) \* (erro + regularizacao()));

}

**Gradiente**

void gradiente()

{

double temp[14] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};

double erro = 0.0;

//para o bias

for(int i = min; i < max; i++)

erro += (hx(i)-x[0][i]);

temp[0] = teta[0] - aprendizado \* 1/(max - min) \* erro;

//para cada teta

for(int j = 1; j < 14; j++)

{

erro = 0.0;

for(int i = min; i < max; i++)

erro += (hx(i)-x[0][i])\*x[j][i];

temp[j] = teta[j] - aprendizado \* ((1.0/(max - min))\*erro + (lamb/(max - min))\*teta[j]);

}

//atribui os valores aos tetas

for(int j = 0; j < 14; j++)

teta[j] = temp[j];

}

**Condição de parada do gradiente descendente**

//confere se o erro ainda esta diminuindo

if((\*ultimo\_erro - erro) < 0.0000000000005 && (\*ultimo\_erro - erro) > 0.0)

return 2; //erro não esta diminuindo significativamente

\*ultimo\_erro = erro;

if(erro > 0.02)

return 1; //erro inaceitavel

else

return 0;//erro aceitável

Vinhos separados em 3 grupos:

1 – 105: valores de treinamento

106 – 142: valores de teste1

143 – 178: valores de teste2

Valores iniciais de

{1,1,1,1,1,1,1,1,1,1,1,1,1,1}

Valor de

0.0000025 – máximo valor encontrado que permite o funcionamento adequado do algoritmo do gradiente-descendente

Não foi feito normalização na base de dados.

A base de dados se encontrava ordenada por tipo de vinho. Para que o aprendizado pudesse chegar a resultados mais consistentes, os dados foram reorganizados em três grupos (60%,20%,20%), sendo o grupo maior utilizado para aprendizado e os outros dois para testes. Os vinhos foram divididos de maneira que no grupo maior existe a mesma quantidade de vinhos de cada tipo (35), e nos grupos restantes os vinhos foram divididos de maneira a conter a mesma quantidade de vinhos de um tipo nos dois grupos.

**Análise**

O primeiro resultado não utilizou regularização e teve seu aprendizado em cima de todos os vinhos. Caso deseje ver quais vinhos especificamente estão apresentando resultados errados, coloque os valores de teta no arquivo teta.txt e execute o programa.

= 0.0

: 4.460650;-0.116631;0.029776;-0.155090;0.039934;-0.000403;0.143331;-0.369818;-0.267130;0.038708;0.075462;-0.155063;-0.268378;-0.000700;

J(x) de valores de treinamento = 0.030077

J(x) de valores de teste 1 = 0.035292

J(x) de valores de teste 2 = 0.023567

Valores com erro inaceitável:

Erros acima de 0.5 = 7 Erros acima de 1.0 = 0

O número de execuções foi perdido durante os testes. Estima-se acima de 250 milhões de interações.

**COM REGULARIZAÇÃO:**

Os seguintes conjuntos de tetas foram obtidos utilizando uma formula errada de custo durante o gradiente descendente. Entretanto, a condição de parada do algoritmo não foi afetada por esse erro, uma vez que a causa da parada foi a diferença entre a medição de erros (separadas pela execução do gradiente) resultava num número muito baixo, finalizando o loop do gradiente. Os valores de J(x) apresentados abaixo já estão corrigidos.

*= 0.1*

: 0.907409;-0.017983;0.072824;0.482929;0.001070;0.002144;0.035851;-0.468356;0.900155;0.078889;0.112532;0.724191;-0.164007;-0.000975;

J(x) de valores de treinamento = 0.067817

J(x) de valores de teste 1 = 0.082248

J(x) de valores de teste 2 = 0.061483

Valores com erro inaceitável:

Erros acima de 0.5 = 33 Erros acima de 1.0 = 1

Número de interações: 13145955

*= 1.0*

: 0.921441;0.017604;0.046757;0.355889;0.010206;0.002018;0.022951;-0.458524;0.738496;0.054633;0.103498;0.595703;-0.178168;-0.000935;

J(x) de valores de treinamento = 0.067052

J(x) de valores de teste 1 = 0.100393

J(x) de valores de teste 2 = 0.086772

Valores com erro inaceitável:

Erros acima de 0.5 = 26 Erros acima de 1.0 = 1

Número de interações: 17130767

*= 10.0*

: 0.907649;0.039470;0.077811;0.174072;0.029950;0.002772;-0.018429;-0.299376;0.345049;0.003327;0.095231;0.255378;-0.134453;-0.001049;

J(x) de valores de treinamento = 0.105167

J(x) de valores de teste 1 = 0.218970

J(x) de valores de teste 2 = 0.189796

Valores com erro inaceitável:

Erros acima de 0.5 = 16 Erros acima de 1.0 = 0

Número de interações: 10696510

*= 100.0*

: 2.124656;0.000941;0.051406;-0.008361;0.039868;0.000654;-0.038983;-0.096000;0.005405;-0.030807;0.084135;-0.019834;-0.071603;-0.001293;

J(x) de valores de treinamento = 2.229088

J(x) de valores de teste 1 = 6.205475

J(x) de valores de teste 2 = 6.359166

Valores com erro inaceitável:

Erros acima de 0.5 = 30 Erros acima de 1.0 = 0

Número de interações: 229472041

*= 20.0*

: 0.906225;0.051572;0.076868;0.078598;0.040244;0.003091;-0.035023;-0.227613;0.172219;-0.020167;0.091643;0.111677;-0.116505;-0.001108;

J(x) de valores de treinamento = 0.141788

J(x) de valores de teste 1 = 0.319515

J(x) de valores de teste 2 = 0.292552

Valores com erro inaceitável:

Erros acima de 0.5 = 15 Erros acima de 1.0 = 0

Número de interações: 9248874

*= 50.0*

: 0.901795;0.046238;0.072407;0.020954;0.049202;0.003662;-0.038462;-0.142669;0.057368;-0.028734;0.089979;0.020645;-0.084382;-0.001226;

J(x) de valores de treinamento = 0.266732

J(x) de valores de teste 1 = 0.648832

J(x) de valores de teste 2 = 0.638081

Valores com erro inaceitável:

Erros acima de 0.5 = 28 Erros acima de 1.0 = 0

Número de interações: 6307388

*= 30.0*

: 0.903233;0.050699;0.076861;0.046420;0.044747;0.003329;-0.038765;-0.187488;0.109351;-0.026449;0.091148;0.060762;-0.102518;-0.001157;

J(x) de valores de treinamento = 0.182881

J(x) de valores de teste 1 = 0.427372

J(x) de valores de teste 2 = 0.405212

Valores com erro inaceitável:

Erros acima de 0.5 = 21 Erros acima de 1.0 = 0

Número de interações: 7926288

Após esses testes, foi possível determinar que utilizar lambida igual a 20 resultou em bons resultados. Foi feito então um único teste onde os valores esperados de h(x) foram multiplicados por 10.

= 20.0

: 14.781950;0.534226;0.571118;-0.331155;0.442370;0.021449;-0.487428;-2.143621;0.057286;-0.431788;0.844490;-0.287152;-1.327141;-0.010431;

J(x) de valores de treinamento = 26.049792

J(x) de valores de teste 1 = 67.136876

J(x) de valores de teste 2 = 65.734540

Valores com erro inaceitável:

Erros acima de 0.5 = 14 Erros acima de 1.0 = 0

Número de interações: 194286004

**CUSTO CORRIGIDO**

Os seguintes resultados foram obtidos utilizando a fórmula de custo descrita nesse artigo (correta). Os resultados agora não apresentam grande diferença entre os anteriores, porém sua execução se tornou muito mais rápida devido a diversas otimizações no código.

*= 20.0*

: 0.972803;0.076104;0.051510;0.042900;0.038975;0.002556;-0.031934;-0.291822;0.161132;-0.021999;0.084439;0.104773;-0.150746;-0.000999;

J(x) de valores de treinamento = 0.149454

J(x) de valores de teste 1 = 0.360080

J(x) de valores de teste 2 = 0.338794

Valores com erro inaceitável:

Erros acima de 0.5 = 15 Erros acima de 1.0 = 0

Número de interações: 8985322

*= 15.0*

: 0.988001;0.085987;0.046264;0.010968;0.039876;0.002443;-0.022118;-0.313644;0.133573;-0.017270;0.080309;0.086106;-0.161904;-0.000964;

J(x) de valores de treinamento = 0.123220

J(x) de valores de teste 1 = 0.291236

J(x) de valores de teste 2 = 0.271024

Valores com erro inaceitável:

Erros acima de 0.5 = 13 Erros acima de 1.0 = 0

Número de interações: 5012164

*= 13.5*

: 0.998894;0.091041;0.044887;-0.011205;0.040945;0.002411;-0.016437;-0.320429;0.106866;-0.015250;0.077906;0.066417;-0.166074;-0.000950;

J(x) de valores de treinamento = 0.115251

J(x) de valores de teste 1 = 0.271139

J(x) de valores de teste 2 = 0.251585

Valores com erro inaceitável:

Erros acima de 0.5 = 13 Erros acima de 1.0 = 0

Número de interações: 6002662

**Conclusões**

Foi possível notar que em grande parte dos testes, mais tempo rodando implica em super ajuste e portanto em resultados piores.

As análises foram colocadas na ordem que foram executadas. Algumas melhoras no número de execuções ou no próprio resultado em si (conjunto de encontrados) podem ser atribuídas a ajustes no , e a melhorias na lógica do código.

Definir que o resultado esperado deixou de ser {1,2,3} a passou a ser {10,20,30} trouxe resultados semelhantes, porém necessitou de muito mais interações (194milhões contra 9 milhões), o que tornou esse artifício ineficaz.

Com 13 erros em 178 casos, = 13.5 atingiu o melhor resultado (~7.3% erro aproximadamente). Esse resultado poderia ser melhorado caso o grupo de treinamento fosse re-embaralhado, porém para o objetivo desse projeto, uma taxa de acerto acima de 90% já se demonstrou adequada.