6/12/2013

Lucas Aoki Chao – 6ºBCC

Aprendizado de Máquina

Regressão linear utilizando

Gradiente Descendente

**Introdução**

Programa para avaliar preço de casas, utilizando Gradiente Descendente com regularização.

**Base de Dados**

1. Title: Boston Housing Data

2. Sources:

(a) Origin: This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University.

(b) Creator: Harrison, D. and Rubinfeld, D.L. 'Hedonic prices and the demand for clean air', J. Environ. Economics & Management, vol.5, 81-102, 1978.

(c) Date: July 7, 1993

3. Past Usage:

- Used in Belsley, Kuh & Welsch, 'Regression diagnostics ...', Wiley, 1980. N.B. Various transformations are used in the table on pages 244-261.

- Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning.

In Proceedings on the Tenth International Conference of Machine Learning, 236-243, University of Massachusetts, Amherst. Morgan Kaufmann.

4. Relevant Information:

Concerns housing values in suburbs of Boston.

5. Number of Instances: 506

6. Number of Attributes: 13 continuous attributes (including "class" attribute "MEDV"), 1 binary-valued attribute.

Attribute Information:

1. CRIM per capita crime rate by town

2. ZN proportion of residential land zoned for lots over 25,000 sq.ft.

3. INDUS proportion of non-retail business acres per town

4. CHAS Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)

5. NOX nitric oxides concentration (parts per 10 million)

6. RM average number of rooms per dwelling

7. AGE proportion of owner-occupied units built prior to 1940

8. DIS weighted distances to five Boston employment centres

9. RAD index of accessibility to radial highways

10. TAX full-value property-tax rate per $10,000

11. PTRATIO pupil-teacher ratio by town

12. B 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town

13. LSTAT % lower status of the population

14. MEDV Median value of owner-occupied homes in $1000's

7. Missing Attribute Values: None.

*Housing* foi a base de dados escolhida por conter apenas atributos numéricos, conter um número adequado de amostras e por ser um exemplo o qual permite o uso regressão utilizando o algoritmo do Gradiente Descendente.

Utilizar regressão com o uso do Gradiente Descendente foi o método de aprendizado escolhido por dois motivos:

O algoritmo em C++ já estava pronto para uso (necessitando apenas de pequenas modificações);

Possibilidade de traçar um paralelo com o uso do mesmo algoritmo na base de dados Wine (para reconhecimento do tipo de vinho a partir de suas características).

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

**Programa**

**Modelo**

h(x) = θ13 + θ0\*x0 + θ1\*x1 + ... + θ12\*x12

Valores esperados de h(x): valor médio da moradia \*1000 (em dólares).

double hx(int i)

{

//bias

double resultado = teta[13];

//teta\*x para cada teta

for(int j = 0; j < 13; 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[13][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[13][i]);

temp[13] = teta[13] - aprendizado \* 1/(max - min) \* erro;

//para cada teta

for(int j = 0; j < 13; j++)

{

erro = 0.0;

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

erro += (hx(i)-x[13][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) < 0.00000005 && (\*ultimo\_erro - erro) > 0.0)

return 2; //erro não esta diminuindo significativamente

\*ultimo\_erro = erro;

if(erro > 5)

return 1; //erro inaceitavel

else

return 0;//erro aceitável

Casas separadas em 3 grupos:

1 – 300 valores de treinamento

301 – 402: valores de teste1

402 – 506: valores de teste2

Valores iniciais de

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

Valor de

0.0000065 – 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 foi embaralhada para que as ocorrências tivessem uma distribuição mais homogênea nos três grupos.

**Testes**

Os testes a seguir não conseguiram chegar em um custo suficientemente baixo (J(x) < 5), então a condição de parada do Gradiente Descendente foi a diferencia entre J(x) (intercalados por uma execução do Gradiente) menor que 0.00000005. Ou seja, o programa considera que os valores de estão adequados quando o uso do gradiente resulta em uma diminuição muito pequena no custo.

= 1.0

: -0.089081;0.051842;-0.090238;1.817980;0.966193;5.549730;-0.011695;-0.876691;0.247280;-0.014090;-0.331242;0.011450;-0.353695;1.663398;

J(x) de valores de treinamento = 12.069446

J(x) de valores de teste 1 = 12.733371

J(x) de valores de teste 2 = 13.367982

Número de execuções: 1145775

Valores com erro inaceitável:

Erros acima de 10k = 20 Erros acima de 20k = 4

= 5.0

: -0.088483;0.053580;-0.089725;1.696247;0.905832;5.414555;-0.008738;-0.852748;0.244938;-0.013936;-0.303001;0.011829;-0.363733;1.659299;

J(x) de valores de treinamento = 12.331913

J(x) de valores de teste 1 = 13.431958

J(x) de valores de teste 2 = 14.151538

Número de execuções: 1065372

Valores com erro inaceitável:

Erros acima de 10k = 19 Erros acima de 20k = 4

= 7.0

: -0.088189;0.054401;-0.089538;1.644014;0.878894;5.350060;-0.007338;-0.841291;0.243803;-0.013861;-0.289594;0.012008;-0.368498;1.657972;

J(x) de valores de treinamento = 12.457648

J(x) de valores de teste 1 = 13.762606

J(x) de valores de teste 2 = 14.517910

Número de execuções: 1033373

Valores com erro inaceitável:

Erros acima de 10k = 19 Erros acima de 20k = 4

= 10.0

: -0.087753;0.055584;-0.089319;1.573346;0.842073;5.256492;-0.005315;-0.824619;0.242137;-0.013749;-0.270201;0.012268;-0.375388;1.656029;

J(x) de valores de treinamento = 12.640187

J(x) de valores de teste 1 = 14.236783

J(x) de valores de teste 2 = 15.040410

Número de execuções: 991168

Valores com erro inaceitável:

Erros acima de 10k = 19 Erros acima de 20k = 4

= 15.0

: -0.087042;0.057448;-0.089078;1.471328;0.789034;5.107779;-0.002119;-0.797979;0.239446;-0.013568;-0.239484;0.012678;-0.386289;1.651949;

J(x) de valores de treinamento = 12.929892

J(x) de valores de teste 1 = 14.976965

J(x) de valores de teste 2 = 15.846481

Número de execuções: 930567

Valores com erro inaceitável:

Erros acima de 10k = 20 Erros acima de 20k = 4

= 20.0

: -0.086351;0.059199;-0.088954;1.384418;0.744631;4.966889;0.000892;-0.772562;0.236850;-0.013393;-0.210489;0.013066;-0.396567;1.646318;

J(x) de valores de treinamento = 13.203428

J(x) de valores de teste 1 = 15.661029

J(x) de valores de teste 2 = 16.582172

Número de execuções: 877739

Valores com erro inaceitável:

Erros acima de 10k = 23 Erros acima de 20k = 4

A partir dos resultados obtidos, foi possível notar que = 7.0 teve um bom resultado. Para verificar se esse resultado poderia ser melhorado, o programa foi executado novamente, porém com a condição de parada modificada para (ultimo\_erro – erro\_atual < 0.0000000005).

|  |  |
| --- | --- |
| < 0.0000000005 | < 0.00000005 |
| = 7.0  : -0.088814;0.052884;-0.089174;1.986480;0.120252;5.375849;-0.008884;-0.872685;0.249376;-0.013888;-0.324537;0.011558;-0.364556;2.845260;  J(x) de valores de treinamento = 12.417722  J(x) de valores de teste 1 = 13.931862  J(x) de valores de teste 2 = 14.531400  Número de execuções: 4356229  Valores com erro inaceitável:  Erros acima de 10k = 19 Erros acima de 20k = 4 | = 7.0  : -0.088189;0.054401;-0.089538;1.644014;0.878894;5.350060;-0.007338;-0.841291;0.243803;-0.013861;-0.289594;0.012008;-0.368498;1.657972;  J(x) de valores de treinamento = 12.457648  J(x) de valores de teste 1 = 13.762606  J(x) de valores de teste 2 = 14.517910  Número de execuções: 1033373  Valores com erro inaceitável:  Erros acima de 10k = 19 Erros acima de 20k = 4 |

Com uma condição de parada mais rígida, o programa teve um número de execuções quatro vezes maior. Entretanto, o número de erros permaneceu o mesmo. É possível notar também que, embora no grupo de aprendizado, uma condição de parada mais rígida levou a um valor menor de J(x), o mesmo não ocorreu com os dois grupos de testes.

**Conclusões**

Os valores de casas variam entre 5 000 e 50 000 dólares, sendo a média 22 500 dólares por casa.

Com = 7.0, foram encontrados 23 casos em que h(x) retornou um valor com erro acima de 10 000 dólares, sendo 4 casos em que h(x) retornou um valor com erro acima de 20 000.

A partir dessas informações, é possível avaliar que o programa teve uma taxa de 4.5% de erros (considerando erros abaixo de 10 000 dólares como valores corretos) e 0.8% de erros absurdos.

Considerando casos com erro acima de 5 000 dólares, o erro cresce para 93 casos (sendo 70 casos com erro entre 0 e 5 000 e 23 casos com erro acima de 10 000), o que representa 18.3% das amostras.

Tal resultado foi abaixo do esperado, visto que esse mesmo algoritmo teve uma eficiência maior para classificar vinhos (mesmo sendo um algoritmo de regressão). Uma mudança no modelo, assim como a exclusão de casos que apresentam dificuldade de se encaixar em h(x), podem ser fatores que melhorem a eficiência do programa.