Multiple Linear Regression: Predicting House Prices

by PRASHANTB1984

Problem | Submissions | Leaderboard | Discussions | Editorial

Submitted 4 minutes ago • Score: 10.00 Status: Accepted

Test Case #0: ✔ 0s | Test Case #1: ✔ 0s | Test Case #2: ✔ 0s

Submitted Code

Language: C++ Open in editor

```
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>
#include <string.h>
#include <sstream>
#include <fstream>
#include <math.h>
using namespace std;

//-------------------------------------
// This class handles Matrix operations
// including solving  a linear system
//-------------------------------------
class CMatrix
{
private:
    int m_rows;
    int m_cols;
    char m_name[128];
    CMatrix();
public:
    double **m_pData;

    //----------------------
    //Constructor
    //----------------------
    CMatrix(const char *name, int rows, int cols) : m_rows(rows), m_cols(cols)
    {
        m_pData = new double*[m_rows];
        for(int i = 0; i < m_rows; i++)
            m_pData[i] = new double[m_cols];
        for(int i = 0; i < m_rows; i++)
        {
            for(int j = 0; j < m_cols; j++)
            {
                m_pData[i][j] = 0.0;
            }
        }
    }

    //------------------
    //Copy constructor
    //------------------
    CMatrix(const CMatrix &other)
    {
        //strcpy(m_name, other.m_name);
        m_rows = other.m_rows;
        m_cols = other.m_cols;
        m_pData = new double*[m_rows];
        for(int i = 0; i < m_rows; i++)
            m_pData[i] = new double[m_cols];
        for(int i = 0; i < m_rows; i++)
        {
            for(int j = 0; j < m_cols; j++)
            {
                m_pData[i][j] = other.m_pData[i][j];
```

```
            }
        }
    }

    //------------
    //Destructor
    //------------
    ~CMatrix()
    {
        for(int i = 0; i < m_rows; i++)
            delete [] m_pData[i];
        delete [] m_pData;
        m_rows = m_cols = 0;
    }


    //--------------------
    //Overloaded operator
    //--------------------
    CMatrix& operator = (const CMatrix &other)
    {
        if( this->m_rows != other.m_rows ||
            this->m_cols != other.m_cols)
        {
            std::cout << "WARNING: Assignment is taking place with by changing the number of rows and columns of the matrix";
        }
        for(int i = 0; i < m_rows; i++)
            delete [] m_pData[i];
        delete [] m_pData;
        m_rows = m_cols = 0;
        //strcpy(m_name, other.m_name);
        m_rows = other.m_rows;
        m_cols = other.m_cols;
        m_pData = new double*[m_rows];
        for(int i = 0; i < m_rows; i++)
            m_pData[i] = new double[m_cols];
        for(int i = 0; i < m_rows; i++)
        {
            for(int j = 0; j < m_cols; j++)
            {
                m_pData[i][j] = other.m_pData[i][j];
            }
        }
        return *this;
    }

    //-----------------------
    //Matrix transpose
    //-----------------------
    CMatrix Transpose()
    {
        CMatrix trans("TR", m_cols, m_rows);
        for(int i = 0; i < m_rows; i++)
        {
            for(int j = 0; j < m_cols; j++)
            {
                trans.m_pData[j][i] = m_pData[i][j];
            }
        }
        return trans;
    }


    //-------------------------------
    // Multiplication operator
    //-------------------------------
    CMatrix operator * (const CMatrix &other)
    {
        if( this->m_cols != other.m_rows)
        {
            std::cout << "Multiplication could not take place because number of columns of 1st Matrix and number of rows in 2nd Matrix are different";
            return *this;
        }
        CMatrix result("", this->m_rows, other.m_cols);
        for(int i = 0; i < this->m_rows; i++)
        {
            for(int j = 0; j < other.m_cols; j++)
            {
                for(int k = 0; k < this->m_cols; k++)
                {
                    result.m_pData[i][j] += this->m_pData[i][k] * other.m_pData[k][j];
                }
            }
        }
        return result;
    }

    //-----------------------------------
```

```
147     // Merges two matrices columnwise
148     // 'in' Matrix will be placed to the tight of this Matrix
149     //---------------------------------------
150     CMatrix merge(CMatrix& in){
151         CMatrix ret("MER", m_rows, m_cols+ in.m_cols);
152         for (int i=0;i< m_rows;i++){
153             for (int j=0;j< m_cols+in.m_cols;j++){
154                 if (j<m_cols)
155                    ret.m_pData[i][j] = m_pData[i][j];
156                  else
157                     ret.m_pData[i][j] = in.m_pData[i][j-m_cols];
158             }
159         }
160         return ret;
161     }
162 
163     //---------------------------------------------------------------
164     // Gauss elimination routine for solving a linear system Ax=b
165     // this Matrix holds the [A | b], where A is nxn Matrix, b is nx1 vector
166     // Gauss elimination returns x which satisfies Ax=b
167     //---------------------------------------------------------------------
168     CMatrix  gauss() {
169         double **A= m_pData;
170         int n = m_rows;
171         CMatrix x ("soln", n,1);
172         for (int i=0; i<n; i++) {
173             // Search for the maximum value in the ith column
174             double maxEl = abs(A[i][i]);
175             int maxRow = i;
176             for (int k=i+1; k<n; k++) {
177                 if (abs(A[k][i]) > maxEl) {
178                     maxEl = abs(A[k][i]);
179                     maxRow = k;
180                 }
181             }
182             // Swap the row containing the maximum value with the current row
183             for (int k=i; k<n+1;k++) {
184                 double tmp = A[maxRow][k];
185                 A[maxRow][k] = A[i][k];
186                 A[i][k] = tmp;
187             }
188             // Make all rows below this one has zero value in current column
189             for (int k=i+1; k<n; k++) {
190                 if (A[i][i] == 0){
191                     cout << "This system of equations does not have a unique solution" << endl;
192                     exit(0);
193                 }
194                 double c = -A[k][i]/A[i][i];
195                 for (int j=i; j<n+1; j++) {
196                     if (i==j) {
197                         A[k][j] = 0;
198                     } else {
199                         A[k][j] += c * A[i][j];
200                     }
201                 }
202             }
203         }
204         // Solve equation Ax=b for x using back substitution
205         for (int i=n-1; i>=0; i--) {
206             if (A[i][i] == 0){
207                     cout << "This system of equations does not have a unique solution" << endl;
208                     exit(0);
209             }
210             x.m_pData[i][0] = A[i][n]/A[i][i];
211             for (int k=i-1;k>=0; k--) {
212                 A[k][n] -= A[k][i] * x.m_pData[i][0];
213             }
214         }
215 
216         return x;
217     }
218 
219 
220     friend std::istream& operator >> (std::istream &is, CMatrix &m);
221     friend std::ostream& operator << (std::ostream &os, const CMatrix &m);
222 };
223 
224 //-----------------------------------------------------------
225 // Overloaded output operator
226 //-----------------------------------------------------------
227 std::ostream& operator << (std::ostream &os,const CMatrix &m)
228 {
229     for(int i = 0; i < m.m_rows; i++)
230     {
231         for(int j = 0; j < m.m_cols; j++)
232         {
233             char buf[32];
234             double data = m.m_pData[i][j];
235             if( m.m_pData[i][j] > -0.00001 &&
236                 m.m_pData[i][j] < 0.00001)
```

```
            data = 0;
            sprintf(buf, "%10.2lf ", data);
            os <<  buf;
        }
        os << "\n";
    }
    os << "\n\n";
    return os;
};



//---------------------------------------------------------------------
// This is the class which handles Multiple Linear Regression.
// It basically reads input data, calculates the regression parameters
// based on the training data,computes the target values for the test data
// and finally outputs them.
//---------------------------------------------------------------------
class mlr{

private:

    CMatrix *fmat; //feature matrix, training data
    CMatrix *tmat; // corresponding output vector for training data
    CMatrix *testMat; // feature matrix, test data

public:

    //We are satisfied with the default constructor, hence no constructor

    //------------------------------
    // Destructor
    //------------------------------
    ~mlr(){
        delete fmat;
        delete tmat;
        delete testMat;
    }

    //------------------
    //Allocate fmat
    //------------------
    void create_fmat(const char * name, int r, int c){
      fmat = new CMatrix(name,r,c);
    }

    //-----------------------
    //Allocate tmat
    //-----------------------
    void create_tmat(const char * name, int r, int c){
      tmat = new CMatrix(name,r,c);
    }

    //-----------------------
    // Allocate testMat
    //-----------------------
    void create_testMat(const char *name, int r , int c){
        testMat = new CMatrix(name, r,c);
    }

    //---------------------------------------------------------------
    // We do the regression here and compute the output for test data
    //---------------------------------------------------------------
    void fit_data(){
        CMatrix X_t = (fmat->Transpose()); //X' = transpose of fmat
        CMatrix X_t_X = X_t*(*fmat);       //X' * X

        CMatrix X_t_target = X_t*(*tmat);  //X' * t  (t= training outputs)

        //Create [X'*X | X'*t] for gauss elimination routine
        CMatrix merged = X_t_X.merge(X_t_target);

        // run Gauss elimination with the merged Matrix, and get the
        // regression parameters weights
        CMatrix weights = merged.gauss();

        //Finally compute the outputs for the test data, i.e. predicted values
        CMatrix predicted = (*testMat)*weights;

        //output the predicted values
            cout << predicted;
    }


    //---------------------------------------------------------------
    // Routine for reading the input data from the STDIN (or any istream
    //---------------------------------------------------------------
    void read_data(istream &is)
    {
        int dim_features; // feature dimension
```

```
        int n_samples; // number of samples
        int n_test_samples; // number of test samples
        int d1, d2, d3; //temp values

        //read the feature dimension and number of training samples
        // and check whether they are in the correct range
        is >> d1;
        is >> d2;
        dim_features=d1;
        n_samples= d2;
        if ((dim_features > 10) || (dim_features < 1)) {
            cout << "feature dimension =" << dim_features  << "not in the allowed range" << endl;
            exit(0);
        }
        if ((n_samples > 100) || (n_samples < 5)) {
            cout << "number of samples =" << n_samples <<  "not in the allowed range" << endl;
            exit(0);
        }

        //Now we can allocate for fmat and tmat
        create_fmat("F", n_samples, dim_features+1);
        create_tmat("T", n_samples, 1);

        float value=0;

        //account for unread characters in the buffer
        is.clear();
        is.ignore(2056, '\n');

        //Now read the training samples and fill fmat and tmat
        for (int k=0;k<n_samples;k++){

            fmat->m_pData[k][0]=1.0; //first column is 1 corresponding to the bias coefficient
            for (int l=1;l<=dim_features;l++){
                is >> value;
                if ((value <0) || (value > 1.0)){
                    cout << "Feature value = " << value << " not in the allowed range" << endl;
                    exit(0);
                }
                fmat->m_pData[k][l] = value;
            }
            is >> value;
            if ((value <0) || (value > 1.0E6)){
                cout << "Target value = " << value << " not in the allowed range" << endl;
                exit(0);
            }
            tmat->m_pData[k][0]= value;
        }

        // Now read the number of test samples and check whether it is in the correct range
        is >> n_test_samples;
        if ((n_test_samples > 100) || (n_test_samples < 1)) {
            cout << "number of test samples =" << n_test_samples <<  "not in the allowed range" << endl;
            exit(0);
        }

        //Allocate the testMat for testdata
        create_testMat("T", n_test_samples, dim_features+1);
        is.clear();
        is.ignore(2056, '\n');

        //And read the test samples
        for (int k=0;k<n_test_samples;k++){
            testMat->m_pData[k][0]=1.0; //first column is 1 corresponding to the bias coefficient
            for (int l=1;l<=dim_features;l++){
                is >> value;
                if ((value <0) || (value > 1.0)){
                    cout << "Feature value = " << value << " not in the allowed range" << endl;
                    exit(0);
                }
                testMat->m_pData[k][l] = value;
            }
        }

    }
};


//------------------------------------------------------
//  The main program
//------------------------------------------------------
int main() {
    /* Enter your code here. Read input from STDIN. Print output to STDOUT */
    mlr MLR;  //Create the MLR object
/*
    ifstream filein;
    filein.open("data2.txt");
    MLR.read_data(filein);
*/
    MLR.read_data(cin);  // Read input data
```

```
417     MLR.fit_data();      // run regression and generate the output
418
419     return 0;
420 }
```