

Machine learning algorithm(by using C language)

I . Simple linear regression algorithm

- 1、 Algorithm description (from book)
- 2、 Function implementation
- 3、 Part of the main function
- 4、 Results and estimate

II. Multivariate Linear Regression

- 1、 Algorithm description (from book):
- 2、 Related function
- 3、 Part of the main function

III. K-Nearest Neighbors

- 1、 Algorithm description (from book)
- 2、 Related function
- 3、 Part of the main function

IV. Naive Bayes

- 1、 Introduction of the algorithm (from book)
- 2、 Related function
- 3、 Part of the main function
- 4、 The results of the algorithm

Machine learning algorithm(by using C language)

The algorithm prototype is referenced from 《machine learning algorithms from scratch》 by Jason Brownlee.

Language: python to C language

Software : Codeblocks

Algorithm: SLR, MLR, KNN, Navie Bayes

I . Simple linear regression algorithm

Dataset: Swedish Auto Insurance Dataset (<https://goo.gl/qPSNqY>)

1、 Algorithm description (from book)

In simple linear regression we can use statistics on the training data to estimate the coefficients required by the model to make predictions on new data. The line for a simple linear regression model can be written as:

$$y = b_0 + b_1 \times x$$

Where b_0 and b_1 are the coefficients we must estimate from the training data. Once the coefficients are known, we can use this equation to estimate output values for y given new input examples of x . It requires that you calculate statistical properties from the data such as mean, variance and covariance.

2、 Function implementation

- Create a two-dimensional array

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h> //random number

typedef float type;
#define split 0.6

// Create a two-dimensional array
type **createarray(int n,int m)
{
    int i;
    type **array;
    array=(type **)malloc(n*sizeof(type *));
    array[0]=(type *)malloc(n*m*sizeof(type));
    for(i=1;i<n;i++) array[i]=array[i-1]+m;
    return array;
}
```

- Mean function

```
double mean_x(type **array,int N)
{
    int i;
    double sum=0;
    double meanx;
    for(i = 0; i < N; i++) {
        sum=sum+array[i][0];
    }
    meanx=sum/N;
    return meanx;
}
```

- Variance function

```
double variance_x(double m,type **array,int N) //import the mean and
training parameters
{
    int i;
    double varx=0.0;
    for(i=0;i<N;i++){
        varx+=pow((array[i][0]-m),2);
    }
    return varx;
}
```

- Covariance function

```
// covariance function,import mean
double covariance(double a,double b,type **array,int N) //a and b are
the mean
{
    int i;
    double covar=0.0;
    for(i=0;i<N;i++)
    {
        covar+=(array[i][0]-a)*(array[i][1]-b);
    }
    return covar;
}
```

- Importing datasets

The imported data set is a data set with n rows and D columns. The file format is TXT, and each row of data is separated by spaces.

```
void loadfile(int *n,int *d,type ***array) //Datasets with n rows d
columns
{
    int i,j;
    FILE *fp;
    if((fp=fopen("SLR.txt","r"))==NULL){
        fprintf(stderr,"can not open data.txt!\n");
    }
    if(fscanf(fp,"N=%d,D=%d",n,d)!=2){
        fprintf(stderr,"reading error!\n");
    }

    *array=createarray(*n,*d); //create a matrix with n rows and d
arrays

    for(i=0;i<*n;i++)
        for(j=0;j<*d;j++)
            fscanf(fp,"%f",&(*array)[i][j]); //read data

    if(fclose(fp)){
        fprintf(stderr,"can not close data.txt");
    }
}
```

3、 Part of the main function

- The data set is divided into training set and test set in proportion

```
int i,j;

//parameters i,j,k
int D,N,size1,size2;
type **test=NULL;
```

```

type **Train=NULL;
type **train=NULL; // N and D

//type **Array=NULL;
loadfile(&N,&D,&Train);
size2=N*split;
size1=N-size2;
test=createarray(size1,D);
train=createarray(size2,D);

srand(2); //random seed
int t=0;
while (t < size1)
{
    int i=rand()%11*(N-t-1)/10;
    for( j = 0; j < D; j++){
        test[t][j] = Train[i][j]; //add the ith row of the dataset
to the testing set
    }
    for (j = i; j < (N-t-1); j++) //delete the ith row of the
dataset and (i+1)th row become the ith row of the dataset
    {
        Train[j] = Train[j + 1];
    }
    t++;
}

```

- Calculate the regression coefficients b0 and b1

```

type *predict;
double rmse;
double meanx,meany,covar,varx;
double b0,b1;

predict=(type *)malloc(size1*sizeof(type));

meanx=mean_x(train,size2);
meany=mean_y(train,size2);
varx=variance_x(meanx,train,size2);
covar=covariance(meanx,meany,train,size2);
b1=covar/varx;
b0=meany-b1*meanx;;

```

- Calculating accuracy and standard deviation

```

for(i=0;i<size1;i++)
{
    predict[i]=b0+test[i][0]*b1;
}
//calculate the RMSE
type sumerror=0.0;
for(j=0;j<size1;j++)
{
    sumerror+=pow(predict[j]-test[j][1],2);
}

rmse=sumerror/size1;
rmse=sqrt(rmse);
printf("RMSE: %f \n",rmse);

```

4、Results and estimate

E:\2-three-nextterm\opencv&pcb\机器学习算法C语言重现\SLR_c\bin\Debug\SLRC.exe
RMSE: 34.529773

- Process returned 0 (0x0) execution time : 0.040 s
Press any key to continue.
- All the functions of the algorithm are realized, and the prediction of simple linear regression analysis can be realized.

II. Multivariate Linear Regression

1、Algorithm description (from book):

Dataset: Wine Quality Dataset <https://goo.gl/8jsvFK>

Linear regression is a technique for predicting a real value. Confusingly, these problems where a real value is to be predicted are called regression problems. Linear regression is a technique where a straight line is used to model the relationship between input and output values. In more than two dimensions, this straight line may be thought of as a plane or hyperplane. Predictions are made as a combination of the input values to predict the output value. Each input attribute (x) is weighted using a coefficient (b), and the goal of the learning algorithm is to discover a set of coefficients that results in good predictions (y)

$$y = b_0 + b_1 \times x_1 + b_2 \times x_2 + \dots$$

Coefficients can be found using stochastic gradient descent.

2、Related function

The two functions below are the same as functions that I have referred in the SLR algorithm

- Import datasets
- Create a dynamic two-dimensional array

- Forecast results

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

typedef float type;
#define nfold 5
#define split 0.2

double predict(type *row,type *coefficients,int D){ //input two one-
dimensional arrays with the length of D
    double yhat;
    int i;
    yhat=coefficients[0];
    for(i=0;i<D-1;i++){
        yhat+=coefficients[i+1]*row[i];
    }
    return yhat;
}
```

- Refresh coefficient by iteration

```
type *coefficient_sgd(type **array,type l_rate,type n_epoch,int D,int
size2)//train set, train speed, train cycle
{
    double yhat,error;
    int i,j,k;
    type *coef;
    coef=(type *)malloc(D*sizeof(type));
    for(i=0;i<D;i++){
        coef[i]=0.0;
    } //Initialize the coefficient matrix

    for(i=0;i<n_epoch;i++){
        for(j=0;j<size2;j++){
            yhat=predict(array[j],coef,D);
            error=yhat-array[j][D-1];
            coef[0]=coef[0]-l_rate*error;
            for(k=0;k<D-1;k++){
                coef[k+1]=coef[k+1]-l_rate*error*array[j][k];
            }
        }
    }
    return coef;
}
```

- Calculate the test set results with the trained parameters

```

type *linear_regression_sgd(type **train,type **test,double
l_rate,double n_epoch,int D,int size1,int size2)
{
    double yyhat;
    int i;
    type *predictions;
    type *coeff;
    predictions=(type *)malloc(size1*sizeof(type));
    coeff=(type *)malloc(D*sizeof(type));
    // Compute coefficients
    coeff=coefficient_sgd(train,l_rate,n_epoch,D,size2);

    for(i=0;i<size1;i++){
        yyhat=predict(test[i],coeff,D);
        predictions[i]=yyhat;
    }

    return predictions;
}

```

- Calculate the final RMSE

```

double rmse_metric(type **test,type **train,double l_rate,double
n_epoch,int D,int size1,int size2) //input prediction set and testing
set
{
    double sumerror=0.0;
    double predictionerror=0.0;
    double meanerror;
    int i;
    type *Array;
    Array=(type *)malloc(size1*sizeof(type));

    Array=linear_regression_sgd(train,test,l_rate,n_epoch,D,size1,size2);

    for(i=0;i<size1;i++){
        predictionerror=Array[i]-test[i][D-1]; //calculate the error
        sumerror+=pow(predictionerror,2);
    }
    meanerror=sumerror/size1;

    return sqrt(meanerror);
}

```

3、 Part of the main function

- Cross validation

```

.....

```

```

loaddata(&N,&D,&Sarray); // load dataset

```

```

//Generate training dataset and testing dataset
size1=split*N; // size of the training set
size2=N-size1;// size of the testing set
train=createarray(size2,D);
test=createarray(size1,D);

// Divide the dataset into five folds for cross validating
// The first fold, the index of the row can divided by 5
for(j=0;j<size1;j++){
    test[j]=Sarray[nfold*j];
}
for(i=0;i<size1;i++){
    for(k=0;k<4;k++){
        train[i*(nfold-1)+k]=Sarray[nfold*i+k+1];
    }
}
}

```

- Calculate and display the results

```

result1=rmse_metric(test,train,l_rate,n_epoch,D,size1,size2);
printf("The first fold rmse: %f \n",result1);

```

4、Result

-

```

2  #include <stdlib.h>
3  #include <time.h>

E:\2-three-nextterm\opencv&pcb\机器学习算法C语言重现\MLR_C\bin\Debug\MLR.exe
The first fold rmse: 0.132942
The second fold rmse: 0.121118
The third fold rmse: 0.127179
The fourth fold rmse: 0.126281
The fifth fold rmse: 0.129207
The mean RMSE:0.127345
Process returned 0 (0x0) execution time : 0.366 s
Press any key to continue.

```

- The results are accurate with small error, and the results of cross validation are reasonable. The related functions of MLR algorithm are fully realized. Function comprehensive, code concise.

III. K-Nearest Neighbors

1、Algorithm description (from book)

The k-Nearest Neighbors algorithm or KNN for short is a very simple technique. The entire training dataset is stored. When a prediction is required, the k-most similar records to a new record from the training dataset are then located. From these neighbors, a summarized prediction is made. Similarity between records can be measured many

different ways. A problem or data-specific method can be used. Generally, with tabular data, a good starting point is the Euclidean distance.

Once the neighbors are discovered, the summary prediction can be made by returning the most common outcome or taking the average. As such, KNN can be used for classification or regression problems. There is no model to speak of other than holding the entire training dataset. Because no work is done until a prediction is required, KNN is often referred to as a lazy learning method.

2. Related function

Abalone Dataset <https://goo.gl/BDYgh5>

- Two dimensional array function and read data set function are the same as above
- Calculation of Euclidean distance

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#include<time.h>

#define K 5 //number of the neighbours
typedef float type;
#define n folds 5

type computedistance(int n,type *avector,type *bvector)
{
    int i;
    type dist=0.0;
    for(i=0;i<n;i++)
        dist+=pow(avector[i]-bvector[i],2); //calculate the distance
    return sqrt(dist);
}
```

- Bubble sorting

Reference: (<https://www.cnblogs.com/LCcnblogs/p/6031218.html>)

```
void bubblesort(int n,type **a,int choice) // K,karray, 0/1
{
    int i,j;
    type k; //
    for(j=0;j<n;j++) // n-->k
        for(i=0;i<n-j-1;i++){
            if(0==choice){ //sort the first row
                if(a[0][i]>a[0][i+1]){
                    k=a[0][i];
                    a[0][i]=a[0][i+1];
                    a[0][i+1]=k;
                    k=a[1][i];
                    a[1][i]=a[1][i+1];
                    a[1][i+1]=k;
                }
            }
        }
}
```

```

    }
}
else if(1==choice){ //sort the second row
    if(a[1][i]>a[1][i+1]){
        k=a[0][i];
        a[0][i]=a[0][i+1];
        a[0][i+1]=k;
        k=a[1][i];
        a[1][i]=a[1][i+1];
        a[1][i+1]=k;
    }
}
}
}
}

```

- Get neighbor list

Reference: (<https://www.cnblogs.com/LCcnblogs/p/6031218.html>)

```

type orderedlist(int n,type *list) //k,array[1]
{
    int i,count=1,maxcount=1;
    type value; //
    for(i=0;i<(n-1);i++) {
        if(list[i]!=list[i+1]) {

            if(count>maxcount){
                maxcount=count;
                value=list[i];
                count=1;
            }
        }
        else
            count++;
    }
    if(count>maxcount){
        maxcount=count;
        value=list[n-1];
    }

    return value; // return max
}

```

3、Part of the main function

- The core part

```

for(c=0;c<NN;c++){
    for(i=0;i<K;i++){
        if(K>N1) exit(-1);

        // karray
    }
}

```

```

        karray[0][i]=computedistance(D-1,Array[c],train[i]); //
calculate the distance
        karray[1][i]=train[i][D-1]; //store the real result of the
training set
    }

    bubblesort(K,karray,0); // sort the distance

    maxdist=karray[0][K-1]; // record the maximum value after the
first sort

    for(i=K;i<N1;i++){ //i=3--6
        dist=computedistance(D-1,Array[c],train[i]);
        if(dist<maxdist){
            for(j=0;j<K;j++){
                if(dist<karray[0][j]){
                    for(k=K-1;k>j;k--){ // Elements move in
preparation for inserting the new value
                        karray[0][k]=karray[0][k-1];
                        karray[1][k]=karray[1][k-1];
                    }
                    karray[0][j]=dist; //insert the value in the
position j
                    karray[1][j]=train[i][D-1];

                    break;
                }
            }
        }
        maxdist=karray[0][K-1]; //sort again
    }

    bubblesort(K,karray,1); // find the value with the most
occurrences in the neighborhood

    predict[c]=orderedlist(K,karray[1]);

    if(predict[c]==Array[c][D-1]){
        scores1=scores1+1;
    }
    //printf("\n scores:%d \n",scores);
}

```

4、Algorithm evaluation

-

```
E:\2-three-nextterm\opencv&pcb\机器学习算法C语言重现\KNN_c\bin\Debug\knn_c.exe
scores1: 177, Accuracy1:21 %
scores2: 185, Accuracy2:22 %
scores3: 354, Accuracy3:42 %
scores4: 454, Accuracy4:54 %
scores5: 518, Accuracy5:62 %
mean_accuracy:40 %

Process returned 0 (0x0)   execution time : 9.523 s
Press any key to continue.
```

- The results of cross validation are not stable, and the overall accuracy is high. The algorithm programming is complex, some steps are complicated, there is still room for simplification.

IV. Naive Bayes

1、 Introduction of the algorithm (from book)

Bayes' Theorem provides a way that we can calculate the probability of a piece of data belonging to a given class, given our prior knowledge. Bayes' Theorem is stated as:

$$P(class|data) = \frac{P(data|class) \times P(class)}{P(data)}$$

Where $P(class|data)$ is the probability of class given the provided data. Naive Bayes is a classification algorithm for binary (two-class) and multiclass classification problems. It is called Naive Bayes or idiot Bayes because the calculations of the probabilities for each class are simplified to make their calculations tractable.

Rather than attempting to calculate the probabilities of each attribute value, they are assumed to be conditionally independent given the class value. This is a very strong assumption that is most unlikely in real data, i.e. that the attributes do not interact. Nevertheless, the approach performs surprisingly well on data where this assumption does not hold.

2、 Related function

- The code for array creation and dataset import is the same as above.
- Mean function

```
double meanlist(type *array,int size) // one-dimension matrix and the
length of it
{
    double sum=0.0;
    int i;
    for(i=0; i<size; i++) //
    {
        sum += array[i];
    }
    return sum/size;
}
```

- Calculate the standard deviation

```
double stdev(type *array,int size) //size is the number of the array
{
    double avg;
    int i;
    double variance=0.0;
    avg=meanlist(array,size);

    for(i=0; i<size; i++) //
    {
        variance+=pow(array[i]-avg,2);
    }
    variance=variance/(size-1);
    return sqrt(variance);
}
```

- Calculate Gaussian Probability Density Function

```
double calculate_probability(type x,double mean,double stdev) //
import the mean and std value
{
    double exponent;
    double res;
    double pi=3.14159;
    exponent=exp(-(pow(x-mean,2)/(2*pow(stdev,2))));
    res=(1/(sqrt(2*pi)*stdev))*exponent;
    return res;
}
```

- Extract arrays of the same category

```
type **separate_by_class(type **array,int n,int d) // Separate samples
that have the same label from the dataset
{
    type **separated=NULL;
    separated=createarray(n,d);
    type **separated1=NULL;
    separated1=createarray(n/3,d);
    type **separated2=NULL;
    separated2=createarray(n/3,d);
    type **separated3=NULL;
    separated3=createarray(n/3,d);

    //separated=array;
    int i;
    int t=0;
    int tt=0;
    int ttt=0;
    for(i=0;i<n;i++){
        if(array[i][d-1]==0){
```

```

        separated1[t]=array[i];
        t++;
    }
    if(array[i][d-1]==1){
        separated2[tt]=array[i];
        tt++;
    }
    if(array[i][d-1]==2){
        separated3[ttt]=array[i];
        ttt++;
    }
}
for(i=0;i<n/3;i++){
    separated[nclass*i]=separated1[i];
    separated[nclass*i+1]=separated2[i];
    separated[nclass*i+2]=separated3[i];
}
}
return separated;
}

```

- Calculate the probability that the sample belongs to a certain class

```

type *calculate_class_probabilities(type **array,type *row,int d,int n) //
{
    // use 'cal_p' and 'summarize_dataset' function
    int i;
    //int total_rows=N-N/nfold;
    type *probability;
    probability=(type *)malloc(nclass*sizeof(type));
    type **separated1=NULL;
    separated1=createarray(n/3,d);
    type **separated2=NULL;
    separated2=createarray(n/3,d);
    type **separated3=NULL;
    separated3=createarray(n/3,d);
    for(i=0;i<nclass;i++){
        probability[i]=0.33;
    }
    for(i=0;i<n/3;i++){
        separated1[i]=array[nclass*i];
        separated2[i]=array[nclass*i+1];
        separated3[i]=array[nclass*i+2];
    }
    //After initialing
    type **summary=NULL;
    summary=createarray(d,nclass); //5*3

    // the probability of label one
    summary=summarize_dataset(separated1,d,40);
}

```

```

        for(i=0;i<d-1;i++){
            probability[0]=probability[0]*calculate_probability(row[i],summary[i]
[0],summary[i][1]);
        }

        // the probability of label two
        summary=summarize_dataset(separated2,D,40);
        for(i=0;i<d-1;i++){
            probability[1]=probability[1]*calculate_probability(row[i],summary[i]
[0],summary[i][1]);
        }

        // the probability of label three
        summary=summarize_dataset(separated3,D,40);
        for(i=0;i<d-1;i++){
            probability[2]=probability[2]*calculate_probability(row[i],summary[i]
[0],summary[i][1]);
        }

        return probability;
    }

```

- Compare the probability and predict results

```

double predict(type *array)
{
    int i;
    double result;
    double max;
    max=array[0];
    result=0;
    for(i=0;i<nclass;i++){
        if(array[i]>max){
            max=array[i]; // get the maximum probability
            result=i;
        }
    }
    return result;
}

```

3、Part of the main function

- Algorithm score

```

int evaluate_algorithm(type **train,type **test,int size2,int
size1,int d)
{
    int i;
    int scores=0;
    type *ppre;
    ppre=(type *)malloc(size1*sizeof(type));

```

```

ppre=naive_bayes(train,test,size2,size1,d);
for(i=0;i<size1;i++){
    if(ppre[i]==test[i][d-1]){
        scores+=1;
    }
}
//printf("%d",ppre);
accuracy1=scores/size1;
return scores;
}

```

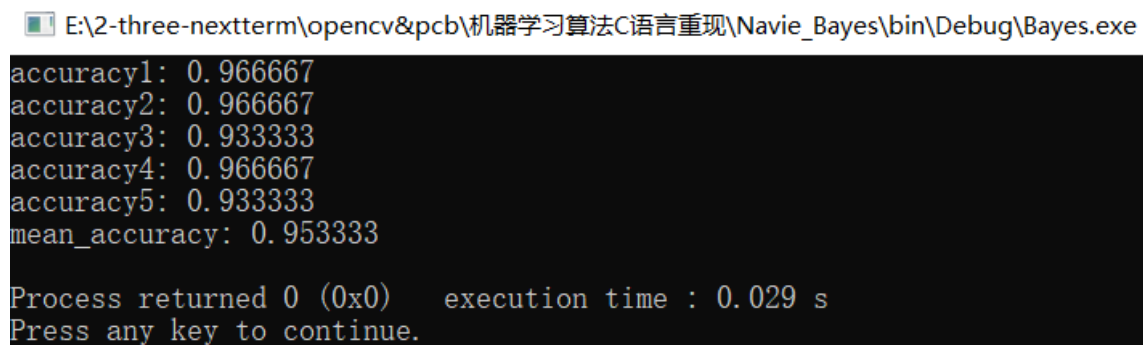
- Cross validation (take the first folder as an example)

```

// The first folder
for(j=0;j<size1;j++){
    test[j]=Sarray[nfold*j]; // store the index of the row that can
    be divided by 5
}
for(i=0;i<size1;i++){
    for(k=0;k<4;k++){
        train[i*(nfold-1)+k]=Sarray[nfold*i+k+1];
    }
}
scores=evaluate_algorithm(train,test,size2,size1,D);
accuracy1=(float)scores/(float)size1;
printf("accuracy1: %f \n",accuracy1);

```

4、The results of the algorithm



```

E:\2-three-nextterm\opencv&pcb\机器学习算法C语言重现\Navie_Bayes\bin\Debug\Bayes.exe
accuracy1: 0.966667
accuracy2: 0.966667
accuracy3: 0.933333
accuracy4: 0.966667
accuracy5: 0.933333
mean_accuracy: 0.953333

Process returned 0 (0x0)   execution time : 0.029 s
Press any key to continue.

```

- Evaluation

Without dict (), we have completed all the functions and steps of the algorithm, and realized cross validation and the algorithm has a high accuracy. The optimization space of the algorithm is small.