## 2.1 读取CSV

在训练模型之前，应该先获得要用的数据集，数据集通常需要保存在文本文件中，这就要求我们要对数据集进行读取，以下get_row()、get_col()函数可以实现对csv文件的行列数的读取，它们需要字符串类型的文件名作为参数输入，而get_two_dimension()可以对csv文件的内容进行读取，它需要字符串类型的文件名作为输入参数。

- 功能——读取cvs文件
- 最终返回行数、列数及数据的二维数组
- 字符串作为参数

```
double **dataset;
int row,col;
int get_row(char *filename)//获取行数
{
    char line[1024];
    int i = 0;
    FILE* stream = fopen(filename, "r");
    while(fgets(line, 1024, stream)){
        i++;
    }
    fclose(stream);
    return i;
}

int get_col(char *filename)//获取列数
{
    char line[1024];
    int i = 0;
    FILE* stream = fopen(filename, "r");
    fgets(line, 1024, stream);
    char* token = strtok(line, ",");
    while(token){
        token = strtok(NULL, ",");
        i++;
    }
    fclose(stream);
    return i;
}

void get_two_dimension(char* line, double** data, char *filename)
{
    FILE* stream = fopen(filename, "r");
    int i = 0;
    while (fgets(line, 1024, stream))//逐行读取
    {
        int j = 0;
        char *tok;
        char* tmp = strdup(line);
        for (tok = strtok(line, ","); tok && *tok; j++, tok = strtok(NULL, ",\n")){
            data[i][j] = atof(tok);//转换成浮点数
        }//字符串拆分操作
        i++;
```

```
        free(tmp);
    }
    fclose(stream);//文件打开后要进行关闭操作
}

```

我们使用data.csv文件，内容如下：

```
1   12.2    12.5    11.1
2.5 555.2   121.4   2.1
```

调用函数:

```
int main()
{
    char filename[] = "data.csv";
    char line[1024];
    double **data;
    int row, col;
    row = get_row(filename);
    col = get_col(filename);
    data = (double **)malloc(row * sizeof(int *));
    for (int i = 0; i < row; ++i){
        data[i] = (double *)malloc(col * sizeof(double));
    }//动态申请二维数组
    get_two_dimension(line, data, filename);
    printf("row = %d\n", row);
    printf("col = %d\n", col);

    int i, j;
    for(i=0; i<row; i++){
        for(j=0; j<col; j++){
            printf("%f\t", data[i][j]);
        }
        printf("\n");
    }
}
```

得到结果如下:

```
row = 2
col = 4
1.000   12.2.000    12.5.000    11.1.000
2.5.000 555.2.000   121.4.000   2.1.000
```

## 2.2 计算RMSE

衡量观测值与真实值之间的偏差。常用来作为机器学习模型预测结果衡量的标准。可以由以下公式计算：

$$RMSE(X,h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(h(x_i) - y_i)^2} \tag{1.8}$$

以下rmse_metric()实现了对测试集RMSE的计算，它需要真实值数组、预测值数组及交叉验证fold的长度作为输入参数。

- 功能——计算按规定划分的测试集的RMSE
- 最终输出RMSE值
- ```c
  float rmse_metric(float *actual, float *predicted, int fold_size)
  {
      float sum_err = 0.0;
      int i;
      int len = sizeof(actual)/sizeof(float);
      for (i = 0; i < fold_size; i++)
      {
          float err = predicted[i] - actual[i];
          sum_err += err * err;
      }
      float mean_err = sum_err / len;
      return sqrt(mean_err);
  }
  ```

我们利用如下数据：

```c
float actual[]= {1,2,3,4};
float pre[] = {4,3,2,1};
printf("%f",rmse_metric(actual,pre,3));
```

得到结果如下：

```
3.316625
```

## 2.3 score.c

该函数用于计算预测所得到的结果的准确率，其基本原理为：将预测正确的结果记为1，错误为0，最终求和得到正确结果个数，利用此个数除以总个数，从而得到正确率。

功能——计算准确率

输入真实值一维数组，预测值一维数组，交叉验证fold的长度

输出准确率值

```c
float accuracy_metric(float *actual, float *predicted, int fold_size)
{
    int correct = 0;
    int i;
    int len = sizeof(actual);
    for (i = 0; i < fold_size; i++)
    {
        if (actual[i] == predicted[i])
            correct += 1;
    }
    return (correct / (float)len)*100.0;
}
```

我们利用如下数据集：
```

```
float actual[]= {1.0,2.0,3.0,4.0};
float pre[] = {1.0,2.0,3.0,3.0};
printf("%f",(accuracy_metric(actual,pre,4)));
```

得到如下结果:

```
75.0000
```

## 2.4 划分数据为k折

K折交叉验证,将数据集等比例划分成K份，以其中的一份作为测试数据，其他的K-1份数据作为训练数据。然后，这样算是一次实验，而K折交叉验证只有实验K次才算完成完整的一次，也就是说交叉验证实际是把实验重复做了K次，每次实验都是从K个部分选取一份不同的数据部分作为测试数据（保证K个部分的数据都分别做过测试数据），剩下的K-1个当作训练数据，最后把得到的K个实验结果进行平分。

此函数则用于将原始数据划分为k等份，以用于k折交叉验证。

- 功能——划分数据为k折
- 输入数据集的二维数组，行数，交叉验证折数，交叉验证fold的长度
- 输出划分后的三维数组
-

```
double***  cross_validation_split(double **dataset, int row, int n_folds, int fold_size)
{
    //printf("%f",dataset[1][1]);
    srand(10);//种子
    double ***split;
    int i,j=0,k=0;
    int index;
    int num;
    num = row/n_folds;
    double **fold;
    split=(double***)malloc(n_folds*sizeof(double**));
    for(i=0;i<n_folds;i++)
    {

        fold = (double**)malloc(num*sizeof(double *));

        while(j<num)
        {
            fold[j]=(double*)malloc(fold_size*sizeof(double));
            index=rand()%row;
            //printf("%d",index);
            //printf("%f",dataset[index][1]);
            fold[j]=dataset[index];
            //printf("%f",fold[j][1]);
            for(k=index;k<row-1;k++)//for循环删除这个数组中被rand取到的元素
            {
                dataset[k]=dataset[k+1];
            }
            row--;//每次随机取出一个后总行数-1，保证不会重复取某一行
            j++;
        }
        j=0;//清零j
        //printf("%f",fold[0][1]);
```

```
        split[i]=fold;
    }
    return split;
}

```

我们运行以下代码：

```
int main()
{
    double data[6][2];
    double *data_ptr[6];
    double *** split;
    for(int i=0;i<6;i++)
    {

        for(int j=0;j<2;j++)
        {
            data[i][j]=i+j;
            //printf("%f",actual[i][j]);
        }
        data_ptr[i] = data[i];
    };

    split = cross_validation_split(data_ptr,6,3,2);
    printf("%f",split[0][0][1]);
}

```

结果如下：

```
6.0000
```

## 2.5数据标准化

数据标准化（归一化）处理是数据挖掘的一项基础工作，**不同评价指标往往具有不同的量纲和量纲单位，这样的情况会影响到数据分析的结果，为了消除指标之间的量纲影响，需要进行数据标准化处理**，以解决数据指标之间的可比性。原始数据经过数据标准化处理后，各指标处于同一数量级，适合进行综合对比评价。

### min-max Normalization

也称为离差标准化，是对原始数据的线性变换，**使结果值映射到[0 - 1]之间。**转换函数如下：

$$x^* = \frac{x-min(x)}{max(x)-min(x)}$$

其中max为样本数据的最大值，min为样本数据的最小值。这种方法有个缺陷就是当有新数据加入时，可能导致max和min的变化，需要重新定义。

```
void normalize_dataset(float **dataset,int row, int col)
{
// 先 对列循环
float maximum, minimum;
for (int i = 0; i < col; i++)
```

```
{
// 第一行为标题，值为0，不能参与计算最大最小值
maximum = dataset[0][i];
minimum = dataset[0][i];
//再 对行循环
for (int j = 0; j < row; j++)
{
maximum = (dataset[j][i]>maximum)?dataset[j][i]:maximum;
minimum = (dataset[j][i]<minimum)?dataset[j][i]:minimum;
}
// 归一化处理
for (int j = 0; j < row; j++)
{
dataset[j][i] = (dataset[j][i] - minimum) / (maximum - minimum);
}
}
}
```

使用以下数据

```
0.000000    1.000000    2.000000    3.000000    4.000000    5.000000
 10.000000    11.000000    12.000000    13.000000    14.000000    15.000000
 20.000000    21.000000    22.000000    23.000000    24.000000    25.000000
 30.000000    31.000000    32.000000    33.000000    34.000000    35.000000
 40.000000    41.000000    42.000000    43.000000    44.000000    45.000000
 50.000000    51.000000    52.000000    53.000000    54.000000    55.000000
 60.000000    61.000000    62.000000    63.000000    64.000000    65.000000
 70.000000    71.000000    72.000000    73.000000    74.000000    75.000000
 80.000000    81.000000    82.000000    83.000000    84.000000    85.000000
 90.000000    91.000000    92.000000    93.000000    94.000000    95.000000
```

归一化：

```
0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
 0.111111    0.111111    0.111111    0.111111    0.111111    0.111111
 0.222222    0.222222    0.222222    0.222222    0.222222    0.222222
 0.333333    0.333333    0.333333    0.333333    0.333333    0.333333
 0.444444    0.444444    0.444444    0.444444    0.444444    0.444444
 0.555556    0.555556    0.555556    0.555556    0.555556    0.555556
 0.666667    0.666667    0.666667    0.666667    0.666667    0.666667
 0.777778    0.777778    0.777778    0.777778    0.777778    0.777778
 0.888889    0.888889    0.888889    0.888889    0.888889    0.888889
 1.000000    1.000000    1.000000    1.000000    1.000000    1.000000
```

## 2.6测试预测函数

将训练集、测试集、学习率，epoch数，交叉验证fold的长度输入函数，并用函数中的模型框架进行对训练集训练后得到模型后，对预测集进行预测分类，并返回预测结果。

函数如下：

```
float get_test_prediction(float **train, float **test, float l_rate, int n_epoch, int fold_size)
{
double *weights=(double*)malloc(col*sizeof(double));//weights数组的长度就是列数（少一个结果位，多一个bias）
double *predictions=(double*)malloc(fold_size*sizeof(double));//预测集的行数就是数组prediction的长度
weights=train_weights(train,l_rate,n_epoch);
int i;
for(i=0;i<fold_size;i++)
{
predictions[i]=predict(test[i],weights);
}
return predictions;//返回对test的预测数组
}
```

## 2.7利用k-fold交叉验证计算预测准确率

输入训练集、测试集，学习率，epoch数，交叉验证折数，交叉验证fold的长度等，并用函数中的模型进行预测，输出最终准确率。

函数实例：