华夏35度

Data Mining,NLP,Search Engine

特征向量的数值求法

-对于正定的对称矩阵，奇异值等于特征值，奇异向量等于特征向量。在这种情况下用奇异值分解就把特征值和特征向量求出来了。但是只要是方阵，它就有特征值和特征向量，对于一般的方阵，特征值和特征向量怎么求呢（当然我指的是数值求法）？这就要用本文即将介绍的"幂法"。

Power Method幂法

Definition 如果λ_1是矩阵A的所有特征值中绝对值最大的那一个，则称λ_1是A的主特征值；与λ_1对应的特征向量v_1是A的主特征向量。

幂法是用来计算方阵的主特征值（即绝对值最大的特征值）和主特征向量的。由此延伸出来的反幂法用来计算在给定点附近的特征值和特征向量（下文把"特征值和特征向量"简称为"特征对"）。

Definition 特征向量V的归一化是指：V的每一个元素除以V中绝对值最大的那个元素。

Theorem(Power Method) 设A是n×n的方阵，有n个不同的特征值，且|λ1| > |λ2| >= |λ3| >= ... >= |λn|。选择一个合适的X_0，序列$\{X_k = (x_1^{(k)}, x_2^{(k)}, \ldots, x_n^{(k)})^T\}$和$\{c_k\}$由下列递归式产生：

$$Y_k = AX_k \qquad (1)$$

$$X_{k+1} = \frac{1}{c_{k+1}} Y_k \qquad (2)$$

其中$c_{k+1} = \max_{1 \le i \le n} \{|x_i^{(k)}|\}$

经过多次迭代后X_k趋于主特征向量V_1，c_k趋于主特征值λ_1。

Remark 如果X_0选取的刚好是一个特征向量，且X_0又不是主特征向量，则X_0需要重新选取。

Speed of Convergence收敛加速

幂法的收敛速度取决于$\frac{|\lambda_1|}{|\lambda_2|}$，也就是说它的收敛速度是线性的。Aitken $\triangle^2$加速法可用于任何线性收敛的序列当中，它采用的加速方式是：

$$\hat{p}_k = p_k - \frac{(\Delta p_k)^2}{\Delta^2 p_k} = p_k - \frac{(p_{k+1} - p_k)^2}{p_{k+2} - 2p_{k+1} + p_k}$$

用Aitken来加速我们的幂法，X_k的调整公式为：

$$\hat{x_j^{(k)}} = x_j^{(k)} - \frac{(x_j^{(k)} - x_j^{(k-1)})^2}{x_j^{(k)} - 2x_j^{(k-1)} + x_j^{(k-2)}}, x_j^{(k)} \neq 1$$

Shifted-Inverse Power Method平移反幂法

使用位移反幂法首先需要提供一个好的起始点，这个点要接近一个特征向量，然后我们的位移反幂法才能够以更高的精度算出这个特征向量。QM和Given's method可以用来获得这种初始点，这里不介绍了。在实际情况中，特征值可能是复数，有多个特征相同或很接近，这都会使得计算变得很复杂需要更高级的算法。我们只考虑简单的情况，所有的特征值各不相同。

Theorem (Shifting Eigenvalues) 假设λ和V是A的一个特征对，a是任何常量，那么λ-a，V就是矩阵$A - aI$的一个特征对。

Theorem (Inverse Eigenvalues) 假设λ和V是A的一个特征对，$\lambda \neq 0$，那么$\frac{1}{\lambda}$，V是A^{-1}的一个特征对。

Theorem (Shifted-Inverse Eigenvalues) 假设λ和V是A的一个特征对，$a \neq \lambda$，那么$\frac{1}{\lambda - a}$，V是$(A - aI)^{-1}$的一个特征对。

请教下，输出误差这里
error/=2;
为什么要除以2？

--我是谁啊

3. Re:聚类算法之DBScan(Java实现)

(8,3),(8,6)都是核心点 为什么（8,3）不加入到旁边那个大的社区里呢？

--藕的梦想

4. Re:FP-Tree算法的实现

求问楼主wordsegservice来自何处，非常感谢！

--smilefacee

5. Re:子进程复制了父进程的什么

写得真的很精彩

--laijiang

Theorem (Shifted-Inverse Power Method) 假设A是一个n×n的矩阵，有n个互不相同的特征值λ_1，λ_2，…，λ_n。对于其中之一个特征值λ_j，可以选择一个常数a，使得$u_1 = \frac{1}{\lambda_j - a}$是$(A-aI)^{-1}$的主特征值。进一步地，如果选择一个合适的$X_0$，序列$\{X_k = (x_1^{(k)}, x_2^{(k)}, \ldots, x_n^{(k)})^T\}$和$\{c_k\}$可由下列递归式产生：

$$Y_k = (A-aI)^{-1}X_k \qquad (3)$$

$$X_{k+1} = \frac{1}{c_{k+1}}Y_k \qquad (4)$$

其中$c_{k+1} = \max_{1\le i\le n}\{|x_i^{(k)}|\}$

经过多次迭代后X_k趋于$(A-aI)^{-1}$的主特征向量V_j，X_k同时也是A的主特征向量，c_k趋于$(A-aI)^{-1}$的主特征值u_1。最终我们可以求出A的主特征值：

$$\lambda_j = \frac{1}{u_1} + a$$

在求解(3)式时需要解一个线性方程组$(A-aI)Y_k = X_k$，常用的方法是雅可比迭代和高斯-赛德尔迭代。当然你也可以用$A^{-1}(A|I) = (I|A^{-1})$的方法进行初等行变换来求得矩阵的逆，那样就不用解线性方程组。不过你要衡量哪种方式快一些，而且矩阵的逆不存在怎么办。

高斯-赛德尔迭代公式为：

$$x_i^{k+1} = \frac{b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij}x_j^{k}}{a_{ii}} \qquad (1 \le i \le n)$$

注意a_{ii}即系数矩阵主对角线上的元素不能有0，否则需要事先进行行变换，把0移走。

Exercise

用幂法求一个矩阵的主特征对。

$$A=\begin{pmatrix} 87 & 270 & -12 & -49 & -276 & 40 \\ -14 & -45 & 6 & 10 & 46 & -4 \\ -50 & -156 & 4 & 25 & 162 & -25 \\ 94 & 294 & -5 & -47 & -306 & 49 \\ 1 & 1 & 3 & 1 & 0 & 2 \\ 16 & 48 & 1 & -6 & -48 & 8 \end{pmatrix}$$

取最大迭代次数为50，收敛时误差为0.000001。

matrix.h

```
#ifndef _MATRIX_H
#define _MATRIX_H

#include<assert.h>
#include<stdlib.h>
#include<stdio.h>

//初始化一个二维矩阵
double** getMatrix(int rows,int col
    double **rect=
(double**)calloc(rows,sizeof(double
    int i;
    for(i=0;i<rows;++i)
        rect[i]=
(double*)calloc(columns,sizeof(douk
    return rect;
}

//返回一个单位矩阵
double** getIndentityMatrix(int row
    double** IM=getMatrix(rows,rows
    int i;
    for(i=0;i<rows;++i)
        IM[i][i]=1.0;
    return IM;
}

//返回一个矩阵的副本
double** copyMatrix(double** matrix
rows,int columns){
    double** rect=getMatrix(rows,cc
    int i,j;
    for(i=0;i<rows;++i)
```

```
        for(j=0;j<columns;++j)
            rect[i][j]=matrix[i][j]
    return rect;
}

//从一个一维矩阵得到一个二维矩阵
void getFromArray(double** matrix,i
rows,int columns,double *arr){
    int i,j,k=0;
    for(i=0;i<rows;++i){
        for(j=0;j<columns;++j){
            matrix[i][j]=arr[k++];
        }
    }
}

//打印二维矩阵
void printMatrix(double** matrix,ir
columns){
    int i,j;
    for(i=0;i<rows;++i){
        for(j=0;j<columns;++j){
            printf("%-10f\t",matri
        }
        printf("\n");
    }
}

//释放二维矩阵
void freeMatrix(double** matrix,int
    int i;
    for(i=0;i<rows;++i)
        free(matrix[i]);
    free(matrix);
}

//获取二维矩阵的某一行
double* getRow(double **matrix,int
columns,int index){
    assert(index<rows);
    double *rect=
(double*)calloc(columns,sizeof(doub
    int i;
    for(i=0;i<columns;++i)
        rect[i]=matrix[index][i];
    return rect;
```

```
}

//获取二维矩阵的某一列
double* getColumn(double **matrix,i
rows,int columns,int index){
    assert(index<columns);
    double *rect=
(double*)calloc(rows,sizeof(double)
    int i;
    for(i=0;i<rows;++i)
        rect[i]=matrix[i][index];
    return rect;
}

//设置二维矩阵的某一列
void setColumn(double **matrix,int
columns,int index,double *arr){
    assert(index<columns);
    int i;
    for(i=0;i<rows;++i)
        matrix[i][index]=arr[i];
}

//交换矩阵的某两列
void exchangeColumn(double **matrix
rows,int columns,int i,int j){
    assert(i<columns);
    assert(j<columns);
    int row;
    for(row=0;row<rows;++row){
        double tmp=matrix[row][i];
        matrix[row][i]=matrix[row][
        matrix[row][j]=tmp;
    }
}


//得到矩阵的转置
double** getTranspose(double **matr
rows,int columns){
    double **rect=getMatrix(columns
    int i,j;
    for(i=0;i<columns;++i){
        for(j=0;j<rows;++j){
            rect[i][j]=matrix[j][i]
        }
```

```
    }
    return rect;
}

//计算两向量内积
double vectorProduct(double *vector
*vector2,int len){
    double rect=0.0;
    int i;
    for(i=0;i<len;++i)
        rect+=vector1[i]*vector2[i]
    return rect;
}

//两个矩阵相乘
double** matrixProduct(double **mat
rows1,int columns1,double **matrix2
columns2){
    double **rect=getMatrix(rows1,c
    int i,j;
    for(i=0;i<rows1;++i){
        for(j=0;j<columns2;++j){
            double
*vec1=getRow(matrix1,rows1,columns1
            double
*vec2=getColumn(matrix2,columns1,co
            rect[i]
[j]=vectorProduct(vec1,vec2,columns
            free(vec1);
            free(vec2);
        }
    }
    return rect;
}

//矩阵和一个数相乘
double** dotProduct(double **matrix
rows,int columns,double a){
    double **rect=getMatrix(rows,co
    int i,j;
    for(i=0;i<rows;++i){
        for(j=0;j<columns;++j){
            rect[i][j]=matrix[i][j]
        }
    }
    return rect;
```

```
}

//两个矩阵相加
double** matrixAdd(double **matrix1
**matrix2,int rows,int columns){
    double **rect=getMatrix(rows,cc
    int i,j;
    for(i=0;i<rows;++i){
        for(j=0;j<columns;++j){
            rect[i][j]=matrix1[i]
[j]+matrix2[i][j];
        }
    }
    return rect;
}

//得到某一列元素的平方和
double getColumnNorm(double** matri
rows,int columns,int index){
    assert(index<columns);
    double*
vector=getColumn(matrix,rows,columr
    double
norm=vectorProduct(vector,vector,rc
    free(vector);
    return norm;
}

//打印向量
void printVector(double* vector,int
    int i;
    for(i=0;i<len;++i)
        printf("%-15.8f\t",vector[i
    printf("\n");
}

#endif
```

power.c

```
#include"matrix.h"
#include<math.h>

#define ROW 6
#define ITERATION 50
#define EPSILON 0.000002
```

```
//找出矩阵元素绝对值最大者
double getMaxEle(double **matrix,ir
    double rect=0;
    int i,j;
    for(i=0;i<rows;++i){
        for(j=0;j<cols;++j){
            if(fabs(matrix[i][j])>r
                rect=fabs(matrix[i]
        }
    }
    return rect;
}

int main(){
    //给矩阵A赋值
    double **A=getMatrix(ROW,ROW);
    double A1[ROW*ROW]=
{87,270,-12,-49,-276,40,-14,-45,6,1
    getFromArray(A,ROW,ROW,A1);
    //取初始X
    double **X=getMatrix(ROW,1);
    double X0[ROW]={1,1,1,1,1,1};
    getFromArray(X,ROW,1,X0);
    //初始化c
    double c=0;
    //初始化Y
    double **Y=getMatrix(ROW,1);
    //开始迭代
    int iteration=0;
    while(iteration++<ITERATION){
        Y=matrixProduct(A,ROW,ROW,X
        c=getMaxEle(X,ROW,1);
        assert(c>0);
        double **newX=dotProduct(Y,
        int i;
        //计算前后两次X的差值
        double epsilon=0.0;
        for(i=0;i<ROW;++i){
            epsilon+=(newX[i][0]-X[
        }
        freeMatrix(X,ROW);
        X=copyMatrix(newX,ROW,1);
        freeMatrix(newX,ROW);
        if(sqrt(epsilon)<EPSILON){
            break;
```

```
        }
    }
    printf("Iteration: %d\n",iterat
    printf("Dominant Eigenvalue=%f\
    printf("Dominant Eigenvector=[%
    int i;
    for(i=1;i<ROW;i++)
        printf(",%f",X[i][0]/c);
    printf("]\n");

    freeMatrix(A,ROW);
    freeMatrix(X,ROW);
    freeMatrix(Y,ROW);
    return 0;
}
```

用位移反幂法求下列矩阵在a=4.2附近的特征值及对应的特征向量。

$$A = \begin{pmatrix} 0 & 11 & -5 \\ -2 & 17 & -7 \\ -4 & 26 & -10 \end{pmatrix}$$

取最大迭代次数为50，收敛误差为0.000002。

GS.h （高斯-赛德尔迭代）

+ View Code

InversePower.c

+ View Code

原文来自:博客园（华夏35度）http://www.cnblogs.com/zhangchaoyang 作者:Orisun

分类: Algorithms

« 上一篇：读《成长比成功更重要》
» 下一篇：机器学习问题方法总结

最新知识库文章：
· 开家公司？比你想的难多了
· 金庸笔下的良好代码风格
· 编程语言中一些令人抓狂的规则
· 项目经理应该把30%的时间用在编程上
· 一名IT从业者的英语口语能力成长路径
» 更多知识库文章...