模式识别第二次作业

- 姓名：潘国盛(本科保送)　　本科学号：3014218157　　院系：计算机系研究生

1.

(a)

```
>> run('/home/ph0en1x/Program_Files/vlfeat/vlfeat-0.9.21/toolbox/vl_setup.m')
>> vl_version
0.9.21
```

(b)使用了两种方法，第二种方法不使用for循环直接用矩阵计算比暴力计算快很多

```
x = rand(5000, 10);
disp('brute-force')
tic
D1 = zeros(5000, 5000);
for i = 1:5000
    for j=1:5000
        D1(i,j)=0;
        for k = 1:10
            D1(i,j) = D1(i,j) + (x(i,k)-x(j, k))^2;
        end
    end
end
[B1, I1] = sort(D1);
% disp(I1(2, :))
toc
```

```
%% OUTPUT
% brute-force
% 时间已过 2.299174 秒。
```

```
square = x .* x;
sumS = sum(square,2);
D2 = repmat((sumS), [1, N]) - 2 * x * x' + repmat((sumS)', [N, 1]);
```

(c)

```
disp('KD-Tree')
newx = x';
kdtree = vl_kdtreebuild(newx, 'NumTrees', 1);
tic
I3 = zeros(50, 5000);
B3 = zeros(50, 5000);
for i = 1:5000
    [I3(:,i), B3(:,i)] = vl_kdtreequery(kdtree, newx, newx(:,i), 'NumNeighbors', 50,
'MaxNumComparisons', 6000);
end
% disp(I2(2,:))
toc
```

```
%% OUTPUT
% KD-Tree
% 时间已过 0.567261 秒。
```

(d)

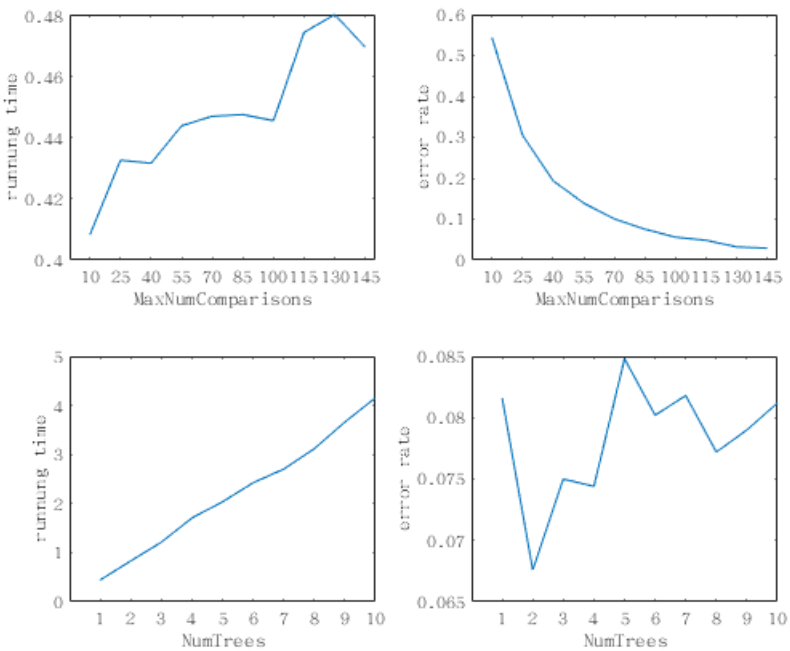
将KD-Tree的结果与直接精确求解的knn的结果进行`~=`布尔运算，得到的布尔矩阵求和占N的比例为error rate

```
error_rate = sum(I1(2,:) ~= I3(2,:)) / N
```

(e)

error rate随着`MaxNumComparisons`的增大而减小，而与`NumTrees`不直接相关

运行时间随着`MaxNumComparisons`的增大而增大，随着`NumTrees`的增大而呈现增大趋势



(f)

当数据集变为500时，运行速度提高，同参数下错误率会降低

当数据集变为10000时，运行速度降低，错误率会有所提高

(g)

Bentley J L. Multidimensional binary search trees used for associative searching[J]. Communications of the ACM, 1975, 18(9): 509-517.

2.

(a)

$$\arg\min_{\beta} \sum_{i=1}^{n} (y_i - x_i^T \beta)^2$$

(b)

$$\arg\min_{\beta} (y - X\beta)^T (y - X\beta)$$

(c)

$$\hat{\beta}^* = (X^T X)^{-1} X^T y$$

(d) 当数据维数多于数据样本数时，$X^T X$不满秩，所以不可逆

(e) 一是解决了不可逆的问题。二是训练样本数量过少容易出现过拟合的情况。正则项将会抑制某些参数，导致某些参数趋近于0,降低模型的拟合能力防止过拟合。λ越大抑制也就越大。

(f)

$$\arg\min_{\beta} (y - X\beta)^T (y - X\beta) + \lambda \beta^T \beta$$

(g) 加入正则项之后相当于实对称矩阵$X^T X$加了一个对角阵可以变成满秩。

(h) $\lambda = 0$ 时与原来的线性回归解相同，$\lambda = \infty$, 则得到的$\beta = 0$

(i) 可以，这是正则化方法，通过正则项来抑制某些参数，然这些参数变为0，降低模型的维度，减少过拟合问题的出现

3.

(a) (b)

index	label	score	precision	recall	AUC-PR	AP
0			1.0000	0.0000	-	-
1	1	1.0	1.0000	0.2000	0.2000	0.2000
2	2	0.9	0.5000	0.2000	0.0000	0.0000
3	1	0.8	0.6667	0.4000	0.1167	0.1333
4	1	0.7	0.7500	0.6000	0.1417	0.1500
5	2	0.6	0.6000	0.6000	0.0000	0.0000
6	1	0.5	0.6667	0.8000	0.1267	0.1333
7	2	0.4	0.5714	0.8000	0.0000	0.0000
8	2	0.3	0.5000	0.8000	0.0000	0.0000
9	1	0.2	0.5556	1.0000	0.1056	0.1111
10	2	0.1	0.5000	1.0000	0.0000	0.0000
					0.6907	0.7277

AP取右端点作为矩形的长来近似PR下方面积，而AUC-PR采用梯形面积。两者都是PR曲线下方面积的近似，但AP总不小于AUC-PR

(c)

index	label	score	precision	recall	AUC-PR	AP
9	2	0.2	0.4444	0.8000	0	0
10	1	0.1	0.5000	1.0000	0.0944	0.1000
					0.6795	0.7166

(d)

```
#include <iostream>
#include <algorithm>
#include <vector>

using namespace std;

struct node{
    int label;
    double score;
    double precision;

    double recall;
```

```
    double AUC_PR;
    double AP;
    bool operator<(node b) const{
        return score > b.score;
    }
};

int main(){
    vector<node> arr;
    const double init_p = 1.0;
    const double init_r = 0.0;
    int label;
    double score;
    int cnt = 0;
    int Pcnt = 0;
    int Ncnt = 0;
    while(~scanf("%d %lf", &label, &score)){
        node tmp;
        tmp.label = label;
        tmp.score = score;
        arr.push_back(tmp);
        cnt++;
        if(label == 1) Pcnt++;
        else if(label == 2) Ncnt++;
    }
    sort(arr.begin(), arr.end());
    int curPcnt = 0;
    double AUC_PR_sum = 0.0;
    double AP_sum = 0.0;
    for(int i = 0; i < cnt; i++){
        if(arr[i].label == 1) curPcnt++;
        arr[i].precision = curPcnt * 1.0 / (i+1);
        arr[i].recall = curPcnt * 1.0 / (Pcnt);
        if(i == 0){
            arr[i].AUC_PR = (arr[i].recall - init_r) * (arr[i].precision + init_p) / 2.0;
            arr[i].AP = (arr[i].recall - init_r) * arr[i].precision;
        }else{
            arr[i].AUC_PR = (arr[i].recall - arr[i-1].recall) * (arr[i].precision + arr[i-
1].precision) / 2.0;
            arr[i].AP = (arr[i].recall - arr[i-1].recall) * arr[i].precision;
        }
        AUC_PR_sum += arr[i].AUC_PR;
        AP_sum += arr[i].AP;
    }
    for(int i = 0; i < cnt; i++){
        printf("%-3d%-3d%-8.1f%-8.4f%-8.4f%-8.4f%-8.4f\n", i+1, arr[i].label,
               arr[i].score, arr[i].precision, arr[i].recall,
               arr[i].AUC_PR, arr[i].AP);
    }
    printf("Sum: %8.4f%8.4f\n", AUC_PR_sum, AP_sum);
    return 0;
}
```

```
/** OUTPUT
1  1  1.0     1.0000  0.2000  0.2000  0.2000
2  2  0.9     0.5000  0.2000  0.0000  0.0000
3  1  0.8     0.6667  0.4000  0.1167  0.1333
4  1  0.7     0.7500  0.6000  0.1417  0.1500
5  2  0.6     0.6000  0.6000  0.0000  0.0000
6  1  0.5     0.6667  0.8000  0.1267  0.1333
7  2  0.4     0.5714  0.8000  0.0000  0.0000
8  2  0.3     0.5000  0.8000  0.0000  0.0000
9  1  0.2     0.5556  1.0000  0.1056  0.1111
10 2  0.1     0.5000  1.0000  0.0000  0.0000
Sum:   0.6906  0.7278
**/
```

4.

(a)

$$p(x,y)=\begin{cases}N(-0.5,0.0625)\ (y=1)\\N(0.5,0.0625)\quad(y=2)\end{cases}$$

(b)

当决策错误的损失值为1时，无论x的ground truth属于哪个类别，x做决策c的风险为

$$R(c|x)=1-p(c|x)$$

即决策错误获得损失的期望值。要让损失值最小，那么$c=f(x)=\arg\max_y p(y|x)$

在多分类决策错误损失值相等时，这种决策也是最优的

(c)

决策为1的风险为$p(y=2|x)=1-p(y=1|x)$

决策为2的风险为$p(y=1|x)=1-p(y=2|x)$

要做出最优决策，就要选择风险较小的决策，即x>0时$f(x)=2$，x<0时$f(x)=1$

(d)

决策为1的风险为$10*p(y=2|x)$

决策为2的风险为$p(y=1|x)$

这时决策会向y=2有一定的偏移，$x>\sigma^2\ln 10$时$f(x)=2$，$x<\sigma^2\ln 10$时$f(x)=1$

5.

(a) XX^T的特征值是奇异值$\sigma_1,\sigma_2\dots\sigma_{min(m,n)}$的平方，特征向量是U的列向量

(b) X^TX的特征值是奇异值$\sigma_1,\sigma_2\dots\sigma_{min(m,n)}$的平方，特征向量是V的列向量

(c) 它们共享一套特征值

(d) 它们的特征值是X的奇异值的平方

(e) 可以计算XX^T的特征值

6.

(a)当scale = 0或非常接近于0时，first eigenvector与原数据的first eigenvector一致，相关系数为1,不发生偏移。当scale增大时，first eigenvector的方向会逐渐偏向与average vector同向

下图为scale逐渐减小时corr1(与avg之间的相关性)与corr2(与正确eigenvector之间的相关性)的变化

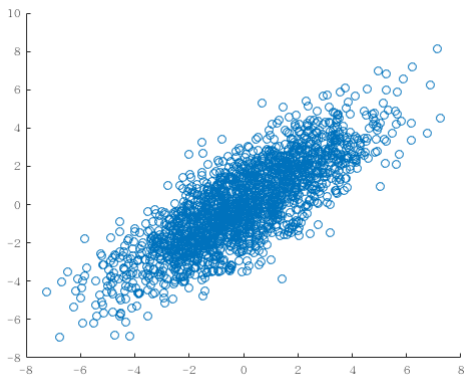
	1		1
1	-1.0000	1	-0.4569
2	0.9707	2	0.6277
3	0.5099	3	0.9955
4	0.4725	4	0.9993
5	0.4584	5	0.9999
6	0.4576	6	1.0000
7	0.4571	7	1.0000
8	0.4570	8	1.0000
9	0.4570	9	1.0000

(b) 在scale变化的过程中，正确的eigenvector不会发生改变，以正确的first eigenvector为例

	1	2	3	4	5	6	7	8	9
1	-0.3343	-0.3343	-0.3343	-0.3343	-0.3343	-0.3343	-0.3343	-0.3343	-0.3343
2	0.0368	0.0368	0.0368	0.0368	0.0368	0.0368	0.0368	0.0368	0.0368
3	-0.5221	-0.5221	-0.5221	-0.5221	-0.5221	-0.5221	-0.5221	-0.5221	-0.5221
4	-0.0756	-0.0756	-0.0756	-0.0756	-0.0756	-0.0756	-0.0756	-0.0756	-0.0756
5	-0.0454	-0.0454	-0.0454	-0.0454	-0.0454	-0.0454	-0.0454	-0.0454	-0.0454
6	0.2358	0.2358	0.2358	0.2358	0.2358	0.2358	0.2358	0.2358	0.2358
7	0.6841	0.6841	0.6841	0.6841	0.6841	0.6841	0.6841	0.6841	0.6841
8	0.0616	0.0616	0.0616	0.0616	0.0616	0.0616	0.0616	0.0616	0.0616
9	-0.2183	-0.2183	-0.2183	-0.2183	-0.2183	-0.2183	-0.2183	-0.2183	-0.2183
10	0.1773	0.1773	0.1773	0.1773	0.1773	0.1773	0.1773	0.1773	0.1773

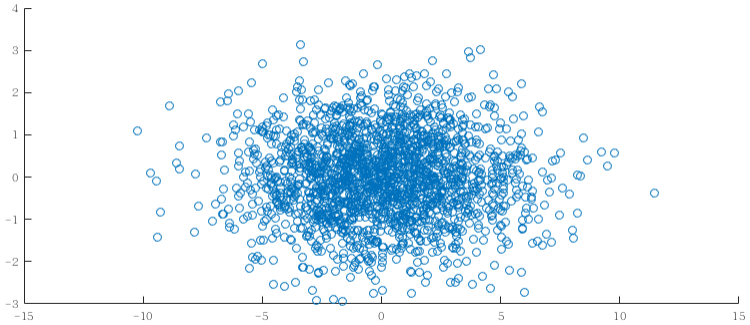
7. (a)

```
x = randn(2000,2)*[2,1;1,2];
```



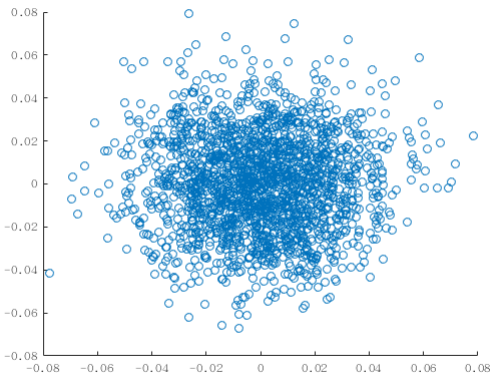
(b)

```
[~ , S , V ] = svd ( x );
m = mean(x);
x = x - repmat(m, 2000, 1);
x = x * V;
```



(c)

```
x = x * diag(diag(S.^(-1)));
scatter(x(:,1), x(:,2));
```



(d)

PCA选取新的一组坐标系进行投影，每个坐标轴都是主成分的方向，如果没有忽略任何一个成分的话，就相当于让原来的数据点做了一个旋转操作，使得主成分对齐于每个坐标轴。这样做的目的是利用主成分间线性无关的性质，让新的坐标系下数据点的表达的每个维度相互独立，这时候进行标准化操作才是真正有意义的操作，让原来空间中的欧氏距离矫正为马氏距离，让离群点的分析更加准确。