系统工程导论作业五——主成分分析

彭程 2020011075

1. 使用 PCA 和线性回归对附件的数据进行建模

1.1 算法思路

1. 样本数据规范化，消除单位影响。

$$\bar{x}_i(t)=\frac{x_i(t)-e\left(x_i\right)}{\sqrt{\delta^2\left(x_i\right)}} \quad \forall i, t$$

其中, $e\left(x_i\right)=\frac{1}{N} \sum_{t=1}^N x_i(t)$ 为样本均值, $\delta^2\left(x_i\right)=\frac{1}{N-1} \sum_{t=1}^N\left(x_i(t)-e\left(x_i\right)\right)^2$ 为样本方差

$$\bar{y}(t)=\frac{y(t)-e(y)}{\sqrt{\delta^2(y)}} \quad \forall t$$

2. 根据阈值确定最优维度 m

对归一化的协方差矩阵 $\Sigma=\frac{1}{N-1} X X^T$ 进行特征值分解。按照特征值从大到小选取特征向量，直到相对逼近误差（未选取的特征值占特征值的比例）小于选取的阈值。

3. 求出降维后的矩阵。

将特征向量组成矩阵 $Q_m=[q(1) q(2) \cdots q(m)]$，从而降维后的矩阵为 $Z=Q_m^T X$。

4. 计算降维后的回归系数 d，从而确定降维前的回归系数 c, 恢复得到归一化前的回归系数

$$\hat{d}=\left(Z Z^T\right)^{-1} Z Y^T$$
$$\hat{\beta}=Q_m \hat{d}$$

5. 进行显著性检验

对计算得到的回归系数进行显著性检验，注意其中 n 为降维之后的维度 m:

$$F=\frac{(N-n-1) E S S}{n R S S}$$

7. 求取置信区间

给定显著性水平 α, 对某一 x_0 , 相应的 y_0 将以 $1-\alpha$ 的概率落在置信区间:

$$\left(\hat{y}_0-Z_{\alpha / 2} S_\delta, \hat{y}_0+Z_{\alpha / 2} S_\delta\right)$$

其中 $Z_{\alpha / 2}$ 是标准正态分布上 $\alpha / 2$ 百分位点的值, 剩余均方差 $S_\delta=\sqrt{\frac{R S S}{N-n-1}}$。

1.2 实验结果

```
PCA+线性回归结果如下
归一化后的回归系数:
beta:
 [-7.14372789e-02 -7.51337839e-02 -3.99718610e-03  1.72879389e-01
  2.11106945e-01 -1.27942808e-01  2.84949246e-01  2.33785569e-01
  1.55614234e-01 -2.37453228e-04 -1.08535378e-01  1.40831051e-01
  1.12213217e-01 -5.75733293e-02]
归一化前的回归系数
beta:
  [-3.78816653e-04 -2.15781284e-06 -1.49610831e-03  6.07704585e-01
  6.80461857e-01 -4.20616476e-04  3.29745055e-01  2.52818439e-04
  1.61117621e-01 -1.67626174e-04 -9.61458256e-02  1.55669417e-01
  5.51412104e-02 -3.04527729e-02]
offset:
  19.58906443461141
F检验结果为: F = 292.2108 > F_alpha = 1.8337   说明x与y存在线性关系
置信区间为: (y - 10.7167, y + 10.7167)
```

```
回归方程为:
  y =
 19.5891
 -0.0004X1
 -0.0000X2
 -0.0015X3
 +0.6077X4
 +0.6805X5
 -0.0004X6
 +0.3297X7
 +0.0003X8
 +0.1611X9
 -0.0002X10
 -0.0961X11
 +0.1557X12
 +0.0551X13
 -0.0305X14
```

根据上述结果，回归方程为:

$$
\begin{aligned}
y = 19.5891 &- 0.0004X_1 - 0.0000X_2 - 0.0015X_3 + 0.6077X_4 + 0.6805X_5 \\
&- 0.0004X_6 + 0.3297 \times 7 + 0.0003X_8 + 0.1611X_9 - 0.0002X_{10} \\
&- 0.0961X_{11} + 0.1557X_{12} + 0.0551X_{13} - 0.0305X_14
\end{aligned}
$$

F 检验: $F = 292.2108 > F_\alpha = 1.8337$, 因此 x, y 存在线性关系。

置信区间: $(y - 10.7167, y + 10.7167)$

1.3 协方差矩阵系数选取

在编程中协方差矩阵的分母为 N - 1。根据概统的知识，当分母为 N - 1 时可以得到样本协方差的无偏估计。但其实在本次作业中，使用 N 和 N - 1 对实验结果并没有影响, 因为我们是通过相对误差上限来选出最大的 r 个特征向量，与前面的这个系数没有关系。

2. 使用病态线性回归对附件的数据进行建模

2.1 算法思路

和上次作业中的思路完全相同，此处仅展示结果。

2.1 实验结果

```
病态线性回归结果如下
特征值从大到小依次为： [12006.70239794  9428.39070262  4971.98636993  3666.18412224
  2796.58963751  2304.23366953  1977.87651152  1902.33429379
  1746.46521324  1205.62495167   845.43271903   366.74985793
   349.85669219    13.57286087]
降维后的维度为： 10
降维后的系数为： [ 0.20013795  0.20374793 -0.24534054  0.0444032  -0.05773427  0.18024992
  0.27955154 -0.12137926  0.19162341 -0.00420912]
规范化后的系数为： [-7.14372789e-02 -7.51337839e-02 -3.99718610e-03  1.72879389e-01
  2.11106945e-01 -1.27942808e-01  2.84949246e-01  2.33785569e-01
  1.55614234e-01 -2.37453228e-04 -1.08535378e-01  1.40831051e-01
  1.12213217e-01 -5.75733293e-02]
原始系数为： [-3.78816653e-04 -2.15781284e-06 -1.49610831e-03  6.07704585e-01
  6.80461857e-01 -4.20616476e-04  3.29745055e-01  2.52818439e-04
  1.61117621e-01 -1.67626174e-04 -9.61458256e-02  1.55669417e-01
  5.51412104e-02 -3.04527729e-02]
原始偏移为：19.5891
F检验结果为：F = 292.2108 > F_alpha = 1.8337  说明x与y存在线性关系
置信区间为: (y - 10.7167, y + 10.7167)
```

```
回归方程为:
  y =
 19.5891
 -0.0004X1
 -0.0000X2
 -0.0015X3
 +0.6077X4
 +0.6805X5
 -0.0004X6
 +0.3297X7
 +0.0003X8
 +0.1611X9
 -0.0002X10
 -0.0961X11
 +0.1557X12
 +0.0551X13
 -0.0305X14
```

可以看到病态线性回归结果和 PCA 主成分分析结果完全相同，说明两者的操作是等价的。

3. 附代码

```python
import numpy as np
from scipy import stats
import pandas as pd


def pca_compress(data, rerr):
    """
    输入：
```

```python
        -data:输入的原始数据 N*(n+1)
        -rerr:相对误差界限
    输出:
        -pcs:主成分
        -cprs_data:压缩后的数据
        -cprs_c:压缩时的常数
    功能: 实现主成分分析
    """

    # data
    x = data[:, 0: -1].T # 规模 n*N
    n, N = x.shape

    # 规范化:
    x_mean = np.mean(x, axis=1, keepdims=True)
    x_std = np.std(x, axis=1, keepdims=True, ddof=1)
    x_norm = (x - x_mean) / x_std

    # 存恢复参数
    cprs_c = {}
    cprs_c["x_mean"] = x_mean
    cprs_c["x_std"] = x_std

    # 主成分分析
    eigenvalue, eigenvector = np.linalg.eig(np.dot(x_norm, x_norm.T))
    eigen_index = np.argsort(-eigenvalue) # 返回排序的下标,从大到小
    eigen_sum = np.sum(eigenvalue)
    eigen_error = eigen_sum
    for m in range(0, n):
        eigen_error = 0
        for i in range(0, m):
            eigen_error += eigenvalue[eigen_index[n - 1 - i]]
        eigen_error = eigen_error / eigen_sum
        if eigen_error > rerr:
            break
    m = n - m + 1 # 前m项保留
    pcs = eigenvector[:, eigen_index[0: m]]
    cprs_data = x_norm.T.dot(pcs)
    # print(pcs.shape)
    # print(cprs_data.shape)

    return pcs, cprs_data, cprs_c


def pca_reconstruct(pcs, cprs_data, cprs_c):
    """
    输入:
        -pcs:主成分(14*10)
        -cprs_data:压缩后的数据(3114*10)
        -cprs_c:压缩时的常数
    输出:
        -recon_data: 恢复的数据 N*n (3114*14)
    功能: 实现数据恢复
    """
    recon_data_norm = cprs_data.dot(pcs.T)
    recon_data = recon_data_norm * cprs_c["x_std"].T + cprs_c["x_mean"].T
```

```python
    return recon_data


def pca_linear_regression(data, alpha=0.05, error=0.05):
    """
    输入:
        -PCS:主成分
        -CPRS_DATA:压缩后的数据
        -CPRS_C:压缩时的常数
    输出:
        -RECON_DATA: 恢复的数据
    功能: 实现完整主成分分析+线性回归
    """

    x = data[:, 0: 14].T
    y = data[:, 14]
    y_mean = np.mean(y.T)
    y_std = np.std(y, ddof=1)
    y_norm = (y - y_mean) / y_std
    n = x.shape[0]
    N = x.shape[1]

    # PCA
    # PRINT("初始数据\N", DATA)
    pcs, cprs_data, cprs_c = pca_compress(data, error)
    # PRINT("压缩数据\N", CPRS_DATA)
    recon_data = pca_reconstruct(pcs, cprs_data, cprs_c)
    # PRINT("恢复数据\N", RECON_DATA)
    Z = cprs_data.T

    # REGRESSION COEFFICIENT AFTER NORMALIZATION
    d = np.linalg.inv(np.dot(Z, Z.T)).dot(Z.dot(y_norm.T)) # 最小二乘
    beta = np.dot(pcs, d)
    print("归一化后的回归系数: ")
    print("beta: \n", beta)

    # 恢复得到归一化前的回归系数BETA
    beta = beta / cprs_c["x_std"].reshape(-1) * y_std
    offset = y_mean - np.sum(beta.reshape(-1,1) * cprs_c["x_mean"].reshape(-1, 1))


    print("归一化前的回归系数")
    print("beta:\n ", beta)
    print("offset:\n ", offset)

    # F检验
    r = pcs.shape[1]
    y_estimate = np.dot(beta.T, x) + offset
    ess = np.dot((y_estimate - y_mean).T, (y_estimate - y_mean))
    rss = np.dot((y - y_estimate).T, (y - y_estimate))
    F = ((N - r - 1) * ess) / (r * rss)
    F_alpha = stats.f.isf(alpha, r, N - r - 1)
    if F > F_alpha:
        print("F检验结果为: F = {:.4f} > F_alpha = {:.4f}".format(F, F_alpha), " 说明x与y存在线性关系")
    elif F <= F_alpha:
        print("F检验结果为: ")
```

```
            print("F-value = {:.4f} <= F_alpha = {:.4f}".format(F, F_alpha))
            print("x与y不存在线性关系")
            return 0

        # 打印置信区间
        S_delta = np.sqrt(rss / (N - r - 1))
        Z_alpha_div2 = stats.norm.isf(alpha / 2, 0, 1)
        interval = Z_alpha_div2 * S_delta
        print("置信区间为: (y - {:.4f}, y + {:.4f})".format(interval, interval))

        # 打印回归方程
        equation = "y =\n %.4f\n" % offset
        for i in range(n):
            equation += " %+.4fX%d\n" % (beta[i], i + 1)
        print("回归方程为:\n ", equation)

        return beta, offset


    def morbid_linear_regression(Y, X, alpha=0.05, error=0.01):
        """
        输入: N×N的矩阵X,1×N的矩阵Y
        输出: N×1的回归系数THETA
        功能: 实现Y=C^T*X+B的多元线性回归, 自适应病态线性回归
        """
        n, N = X.shape

        # 数据规范化
        x_mean = np.mean(X, axis=1, keepdims=True)
        delta_x = np.sqrt(
            np.sum(np.multiply(X - np.mean(X, axis=1, keepdims=True), X - np.mean(X, axis=1, keepdims=True)),
                axis=1,
                 keepdims=True) / (N - 1))
        X_normal = np.divide(X - x_mean, delta_x)
        y_mean = np.mean(Y)
        delta_y = np.sqrt(np.sum(np.multiply(Y - y_mean, Y - y_mean)) / (N - 1))
        Y_normal = (Y - y_mean) / delta_y

        # 根据阈值确定最优维度M
        eigenvalue, eigenvector = np.linalg.eig(np.dot(X_normal, X_normal.T))
        eigen_index = np.argsort(-eigenvalue) # 返回排序的下标
        eigen_sum = np.sum(eigenvalue)
        for m in range(0, n):
            eigen_error = 0
            for i in range(0, m):
                eigen_error += eigenvalue[eigen_index[n - 1 - i]]
            eigen_error = eigen_error / eigen_sum
            if eigen_error > error:
                break
        m = n - m + 1 # 则前M项是要保留的

        # 计算降维后的回归系数D, 从而确定降维前的回归系数C_N,恢复得到归一化前的回归系数C
        Q = eigenvector[:, eigen_index[0: m]]
        Z = Q.T.dot(X_normal)
        d = np.linalg.inv(Z.dot(Z.T)).dot(Z.dot(Y_normal.T))
        c_n = Q.dot(d)
```

```
    c_n=c_n.reshape(-1)
    c = (c_n.T / np.squeeze(delta_x) * delta_y).T
    c=c.reshape(-1)
    b = y_mean - np.sum(c * np.squeeze(x_mean))
    print("特征值从大到小依次为: ", eigenvalue[eigen_index[:]])
    print("降维后的维度为: ", m)
    print("降维后的系数为: ", d)
    print("规范化后的系数为: ", c_n)
    print("原始系数为: ", c)
    print("原始偏移为: {:.4f}".format(b))

    # F检验操作
    Y_estimate = np.dot(c.T, X) + b
    ESS = np.dot((Y_estimate - y_mean), (Y_estimate - y_mean).T)
    RSS = np.dot((Y - Y_estimate), (Y - Y_estimate).T)
    F = ((N - m - 1) * ESS) / (m * RSS) # ESS自由度为 (N-M-1) , RSS自由度为M
    F_alpha = stats.f.isf(alpha, m, N - m - 1)

    if F > F_alpha:
        print("F检验结果为: F = {:.4f} > F_alpha = {:.4f}".format(F, F_alpha), " 说明x与y存在线性关系")
    elif F <= F_alpha:
        print("F检验结果为: ")
        print("F-value = {:.4f} <= F_alpha = {:.4f}".format(F, F_alpha))
        print("x与y不存在线性关系")
        return 0

    # 置信区间
    S_delta = np.sqrt(RSS / (N - m - 1))
    Z_alpha_div2 = stats.norm.isf(alpha / 2, 0, 1)
    interval = Z_alpha_div2 * S_delta
    print("置信区间为: (y - {:.4f}, y + {:.4f})".format(interval, interval))

    # 打印回归方程
    equation = "y = %.4f" % b
    for i in range(n):
        equation += " + %.4fX%d" % (c[i], i + 1)
    print("回归方程为: ", equation)

    return


if __name__ == '__main__':
    raw_data = pd.read_excel('counties.xlsx', usecols='C:Q')
    data = np.array(raw_data)
    x = data[:, 0: 14].T
    y = data[:, 14]

    # PCA+线性回归
    print("PCA+线性回归结果如下")
    pca_linear_regression(data, alpha=0.05, error=0.05)
    # 直接病态线性回归
    print("病态线性回归结果如下")
    morbid_linear_regression(y, x, alpha=0.05, error=0.05)
```