Weka[32] PCA 源代码分析

作者：Koala++/屈伟

推荐阅读的 PCA 资料有 A Tutorial on Principal Component Analysis，两篇都是这名字，Jonathon Shlens 写的评价很高，他的介绍也比较直接，证明也比较容易看懂。Lindsay I Smith 的 tutorial 是给统计 Muggles 和矩阵 Muggles 写的，直白到了极点，没什么道理看不懂。Jonathon Shlens 的论文有中文版：主元分析(PCA)理论分析及应用，不过翻译我大概看了一下，有错误，其中一个比较明显的是把 square 方阵，翻译成了平方。最后推荐的是 Ng 的 lecture，他的特点是他没有用矩阵方法，而是用 lagrange 直接得到了结果，有空可以看一下。

从 buildAttributeContructor 开始，前面不是核心的代码不再列出来：

```
m_numInstances = m_trainInstances.numInstances();
m_numAttribs = m_trainInstances.numAttributes();

fillCorrelation();

double[] d = new double[m_numAttribs];
double[][] v = new double[m_numAttribs][m_numAttribs];

Matrix corr = new Matrix(m_correlation);
corr.eigenvalueDecomposition(v, d);
```

fillCorrelation 是计算协方差矩阵的函数：

```
private void fillCorrelation() {
    m_correlation = new double[m_numAttribs][m_numAttribs];
    double[] att1 = new double[m_numInstances];
    double[] att2 = new double[m_numInstances];
    double corr;

    for (int i = 0; i < m_numAttribs; i++) {
        for (int j = 0; j < m_numAttribs; j++) {
            if (i == j) {
                m_correlation[i][j] = 1.0;
            } else {
                for (int k = 0; k < m_numInstances; k++) {
                    att1[k] = m_trainInstances.instance(k).value(i);
                    att2[k] = m_trainInstances.instance(k).value(j);
                }
                corr = Utils.correlation(att1, att2, m_numInstances);
                m_correlation[i][j] = corr;
                m_correlation[j][i] = corr;
            }
        }
    }
}
```

这里的协方差的计算与论文中有点不同，它进行了类似归一化的操作，对角线上的元素全部是 1，然后在 correlation 的计算中：

```
public static final double correlation(double y1[], double y2[], int n)
{

    int i;
    double av1 = 0.0, av2 = 0.0, y11 = 0.0, y22 = 0.0, y12 = 0.0, c;
```

```
    if (n <= 1) {
        return 1.0;
    }
    for (i = 0; i < n; i++) {
        av1 += y1[i];
        av2 += y2[i];
    }
    av1 /= (double) n;
    av2 /= (double) n;
    for (i = 0; i < n; i++) {
        y11 += (y1[i] - av1) * (y1[i] - av1);
        y22 += (y2[i] - av2) * (y2[i] - av2);
        y12 += (y1[i] - av1) * (y2[i] - av2);
    }
    if (y11 * y22 == 0.0) {
        c = 1.0;
    } else {
        c = y12 / Math.sqrt(Math.abs(y11 * y22));
    }

    return c;
}
```

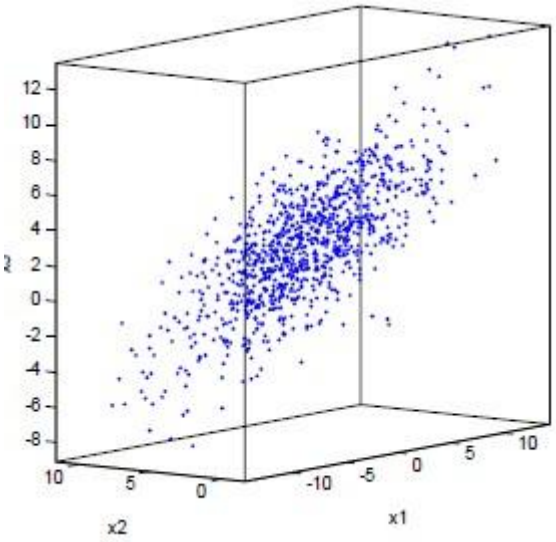
Av1 和 av2 就是 y1[]和 y2[]的平均值，而 y11 就是 y1 的方差*(n-1)，y22 就是 y2 的方差*(n-1)，y12 就是 y1，y2 协方差，这里的计算协方差的时候等于分子分母把(n-1)*(n-1)消了，所以不用除(n-1)，这里还是正规化的原因。回到 fillCorrelation 的函数中，因为协方差矩阵是对称矩阵，所以元素 ij 和 ji 是相等的。buildAttributeConstructor 中 corr.eigenvalueDecomposition 是计算特征值和特征向量。

```
m_eigenvectors = (double[][]) v.clone();
m_eigenvalues = (double[]) d.clone();

// any eigenvalues less than 0 are not worth anything --- change to 0
for (int i = 0; i < m_eigenvalues.length; i++) {
    if (m_eigenvalues[i] < 0) {
        m_eigenvalues[i] = 0.0;
    }
}
m_sortedEigens = Utils.sort(m_eigenvalues);
m_sumOfEigenValues = Utils.sum(m_eigenvalues);

m_transformedFormat = setOutputFormat();
if (m_transBackToOriginal) {
```

得到特征向量和特征值，如果特征值小于 0，无意义，就把它设为 0。然后对特征值进行排序，再得到特征值的和。引用一句：It turns out that the eigenvector with highest eigenvalue is the principle component of the data set.下一段：In general, once eigenvector are found from the covariance matrix, the next step is to order them by eigenvalue, highest to lowest. This gives you the components in order of significance. Now, if you like, you can decide to ignore the components of lesser significance. You do lose some information, but if the eigenvalues are small, you don't lose much. If you leave out some components, the final data set will have less dimensions than the original. To be precise, if you originally have n dimensions in your data, and so you calculate n eigenvectors and eingenvalues, and then you choose only the first p eigenvectors, then the final data set has only p dimensions.

我找了一个蛮直观的例子，或者是我发现有人（Texax A &M University PRISM）找了一个直观的例子：

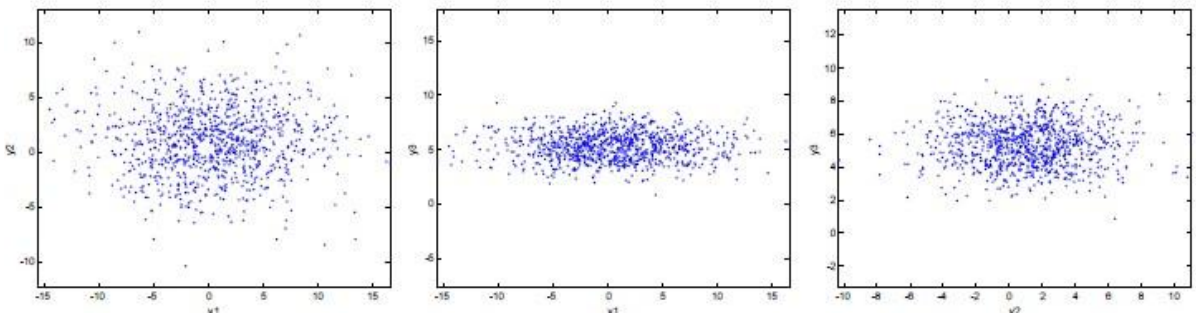
数据集如下：



它的特征值和特征向量如下：

$$\mu = [0\ 5\ 2]^T \text{ and } \Sigma = \begin{bmatrix} 25 & -1 & 7 \\ -1 & 4 & -4 \\ 7 & -4 & 10 \end{bmatrix}$$

The three pairs of principal component projections are shown below:



注意它们的方差，这就是原因。

setOutputFormat 代码不短，但都和 PCA 算法本身没有太大关系，只是设置转换后的数据格式。

```
public Instances transformedData() throws Exception {
    if (m_eigenvalues == null) {
        throw new Exception("Principal components hasn't been built yet");
    }

    Instances output;

    if (m_transBackToOriginal) {
        output = new Instances(m_originalSpaceFormat);
    } else {
        output = new Instances(m_transformedFormat);
    }
    for (int i = 0; i < m_trainCopy.numInstances(); i++) {
        Instance converted = convertInstance(m_trainCopy.instance(i));
        output.add(converted);
```

```
    }

    return output;
}
```

m_transformedFormat 就是刚才用 setOutputFormat 得到的格式。下面是的 convertInstance 就是转换函数：

```
if (m_hasClass) {
    newVals[m_outputNumAtts - 1] = instance
            .value(instance.classIndex());
}

double cumulative = 0;
for (int i = m_numAttribs - 1; i >= 0; i--) {
    double tempval = 0.0;
    for (int j = 0; j < m_numAttribs; j++) {
        tempval += (m_eigenvectors[j][m_sortedEigens[i]] * tempInst
                .value(j));
    }
    newVals[m_numAttribs - i - 1] = tempval;
    cumulative += m_eigenvalues[m_sortedEigens[i]];
    if ((cumulative / m_sumOfEigenValues) >= m_coverVariance) {
        break;
    }
}

if (!m_transBackToOriginal) {
    if (instance instanceof SparseInstance) {
        return new SparseInstance(instance.weight(), newVals);
    } else {
        return new Instance(instance.weight(), newVals);
    }
}
```

如果有类别，类别不参与转换。这里可以看到 m_sortedEigens 是排序后的下标，也就是从大到小的下标，更明白一点就是 m_eigenvectors[j][m_sortedEigens[i]]就是 Eigenvalue 从大到小相应的 eigenvector 的第 j 个元素，两个向量的内积就是新的属性值。

Yoshua Bengio（NIPS 的 General Chair）和 James Bergstra 的 Algorithmes d'apprentissage 中的 cours après cours 的 analyse en composantes principales，我法语学了也忘的差不多了，还好它们长的和英语差不多，alogorithmes d’apprentissage 是学习算法的意思。Cours après cours 直译是课程之后的课程，自己理解吧。可以看一下它的第 7 页最开始的公式，就是 ccumulative / m_sumOfEigenValues >= m_coverVariance 的意思，明白一点就是 m_coverVariance 是牺牲信息的一个阈值，超过它就可以了。

Lindsay I Smith 最后写的就是如何将转换后的数据集再转换回来，这个我想似乎用途不是很大，convertInstanceToOriginal 就是做这个的，但是我想就算了吧，略过。