

Chapter 13 Reinforcement Learning



Principal component analysis

Pros:

- Reduces complexity of data
- Identifies most important features

Cons:

- May not be needed
- Could throw away useful information

Work with : Numerical values

Moving the co-ordinate axes

- In PCA, Rotate axes of the data
- The rotation is determined by data itself
- The first axis is rotated to cover the largest variation in the data
- After choosing the axis covering the most variability. Choose the next axis, which has the second most variability, provided it's perpendicular to the first axis. The real term used is *orthogonal*.
- Rotating the axes hasn't reduced the number of dimensions.

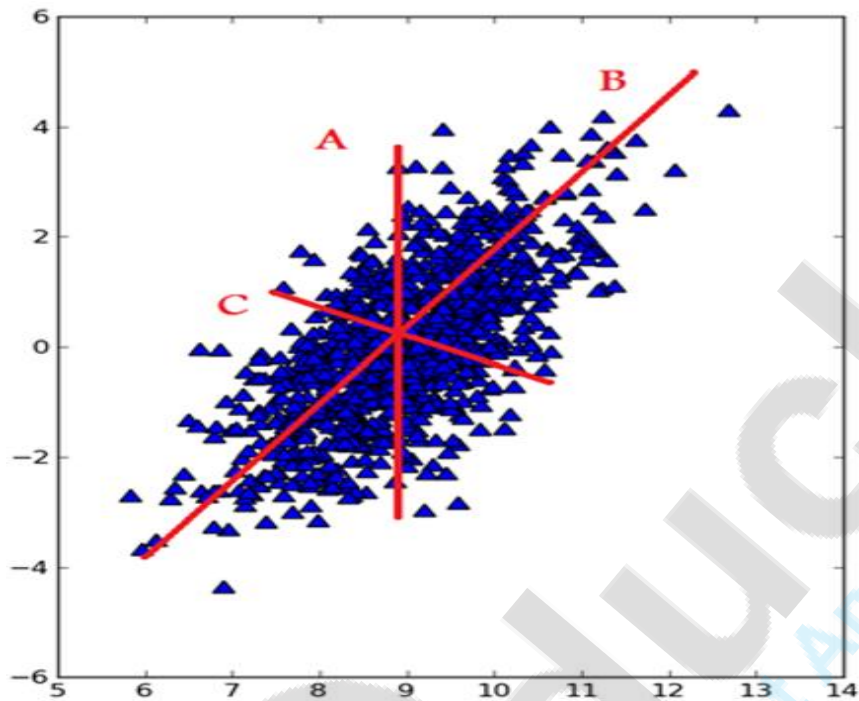
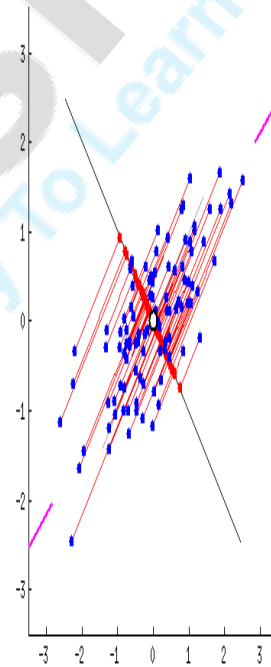


Figure 13.1 Three choices for lines that span the entire dataset. Line B is the longest and accounts for the most variability in the dataset.



Decision trees:

If we want to separate the classes, we could use a decision tree. Decision trees make a decision based on one feature at a time.

Support vector machine:

The support vector machine may give us better margin than the decision tree, but the hyperplane is harder to interpret.

- We can get these values by taking the covariance matrix of the data- set and doing eigenvalue analysis on the covariance matrix.
- Once we have the eigenvectors of the covariance matrix, we can take the top N eigenvectors. The top N eigenvectors will give us the true structure of the N most important features. We can then multiply the data by the top N eigenvectors to transform our data into the new space.

- **Eigenvalue analysis**

Eigenvalue analysis is an area of linear algebra that allows us to uncover the underlying “true” structure of the data by putting it in a common format. In eigenvalue analysis, we usually talk about eigenvectors and eigenvalues. In the following equation, $\mathbf{Av} = \lambda\mathbf{v}$, eigenvectors are \mathbf{v} and eigenvalues are λ . Eigenvalues are simply scalar values, so $\mathbf{Av} = \lambda\mathbf{v}$ says when we multiply the eigenvectors by some matrix, \mathbf{A} , we get the eigenvectors (\mathbf{v}), again multiplied by some scalar values λ . Luckily, NumPy comes with some modules for finding the eigenvectors and eigenvalues. The NumPy linalg module has the `eig()` method, which we can use to find the eigenvectors and eigenvalues.

we can have a classifier as simple as a decision tree, while having margin as good as the support vector machine.

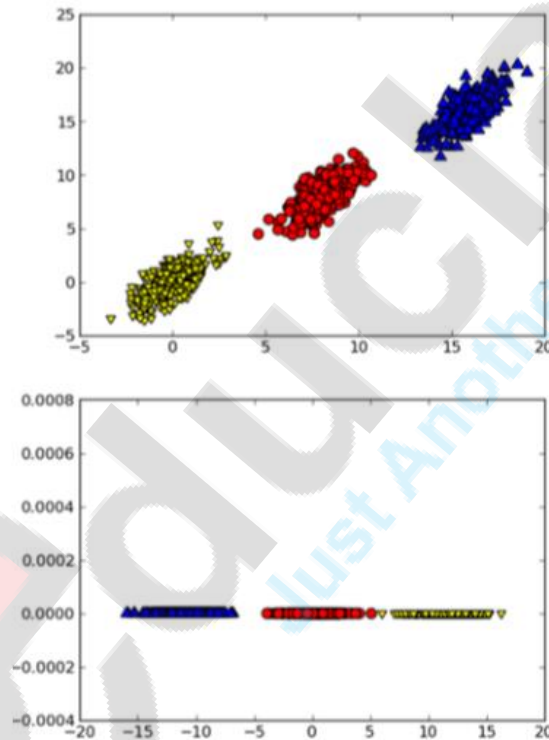


Figure 13.2 Three classes in two dimensions. When the PCA is applied to this dataset, we can throw out one dimension, and the classification problem becomes easier.

Performing PCA in NumPy

Steps:

1. Collect the data
2. Normalize the data
3. Calculate the covariance matrix

Listing 13.1 The PCA algorithm

```
from numpy import *

def loadDataSet(fileName, delim='\t'):
    fr = open(fileName)
    stringArr = [line.strip().split(delim) for line in fr.readlines()]
    datArr = [map(float,line) for line in stringArr]
    return mat(datArr)

def pca(dataMat, topNfeat=9999999):
    meanVals = mean(dataMat, axis=0)
    meanRemoved = dataMat - meanVals
    covMat = cov(meanRemoved, rowvar=0)
    eigVals, eigVects = linalg.eig(mat(covMat))
    eigValInd = argsort(eigVals)
    eigValInd = eigValInd[:-(topNfeat+1):-1]
    redEigVects = eigVects[:, eigValInd]
    lowDDataMat = meanRemoved * redEigVects
    reconMat = (lowDDataMat * redEigVects.T) + meanVals
    return lowDDataMat, reconMat
```

1 Remove mean

2 Sort top N smallest to largest

3 Transform data into new dimensions

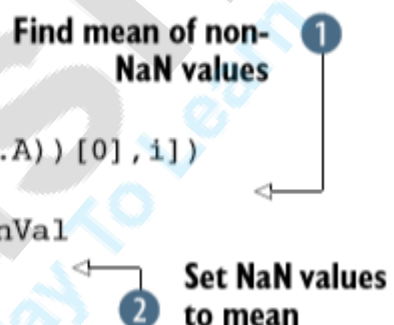
Example: : *using PCA to reduce the dimensionality of semiconductor manufacturing data*

Listing 13.2 Function to replace missing values with mean

```
def replaceNanWithMean():
    datMat = loadDataSet('secom.data', ' ')
    numFeat = shape(datMat)[1]
    for i in range(numFeat):
        meanVal = mean(datMat[nonzero(~isnan(datMat[:, i].A))[0], i])

        datMat[nonzero(isnan(datMat[:, i].A))[0], i] = meanVal

    return datMat
```



Let's see how to do this. First, replace the NaN values in the dataset with mean values using the code we just wrote:

```
dataMat = pca.replaceNanWithMean()
```

Next, borrow some code from the `pca()` function because we want to look at the intermediate values, not the output. We're going to remove the mean:

```
meanVals = mean(dataMat, axis=0)
meanRemoved = dataMat - meanVals
```

Now, calculate the covariance matrix:

```
covMat = cov(meanRemoved, rowvar=0)
```

Finally, do Eigenvalue analysis on the covariance matrix:

```
eigVals, eigVects = linalg.eig(mat(covMat))
```

Now, let's look at the eigenvalues:

```
>>> eigVals
array([[ 5.34151979e+07,  2.17466719e+07,  8.24837662e+06,
        2.07388086e+06,  1.31540439e+06,  4.67693557e+05,
        2.90863555e+05,  2.83668601e+05,  2.37155830e+05,
        2.08513836e+05,  1.96098849e+05,  1.86856549e+05,
        ...,
        0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        0.00000000e+00,  0.00000000e+00,  0.00000000e+00])
```

Over 20% of the eigenvalues are zero. That means that these features are copies of other features in the dataset, and they don't provide any extra information.

15 have magnitudes greater than 105, but after that, the values get really small. This tells you that there are a few important features, but the number of important features drops off quickly.

Negative values are caused by numerical errors and should be rounded to zero.

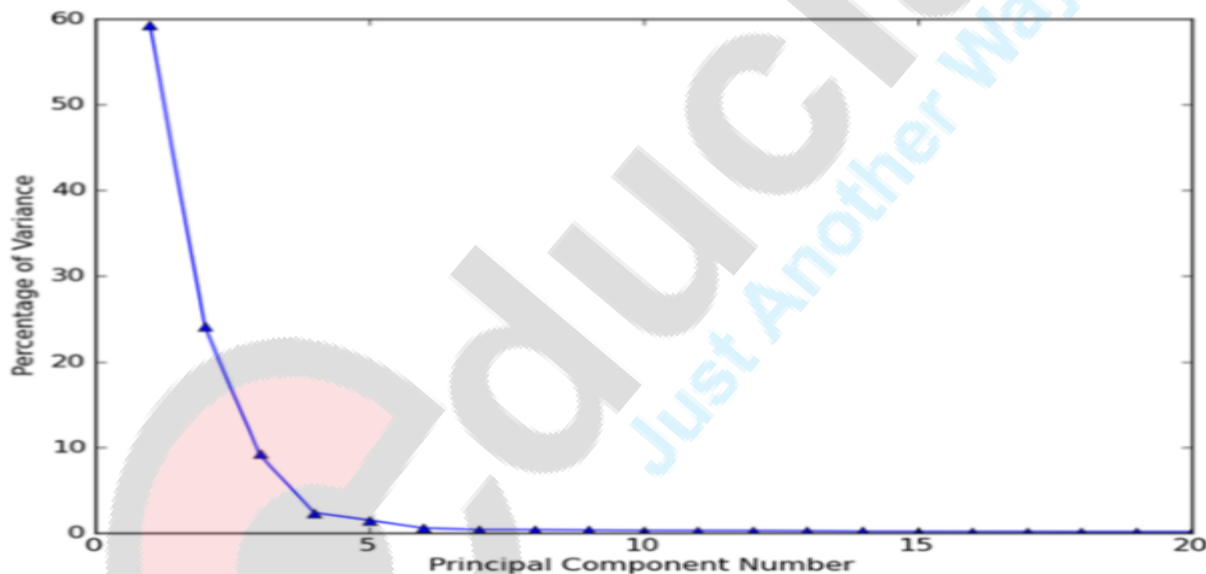


Figure 13.4 Percentage of total variance contained in the first 20 principal components. From this plot, you can see that most of the variance is contained in the first few principal components, and little information would be lost by dropping the higher ones. If we kept only the first six principal components, we'd reduce our dataset from 590 features to 6 features, almost a 100:1 compression.

Table 13.1 % variance for the first 7 principal components of the semiconductor data

Principal component	% Variance	% Cumulative
1	59.2	59.2
2	24.1	83.4
3	9.2	92.5
4	2.3	94.8
5	1.5	96.3
6	0.5	96.8
7	0.3	97.1
20	0.08	99.3

Thank You....!