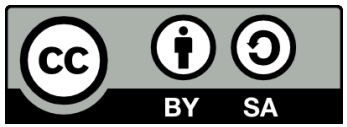
Latent Variable Methods Course

Learning from data

Instructor: Kevin Dunn
kevin.dunn@connectmv.com
http://connectmv.com

Revision: 268:adfd compiled on 15-12-2011

Copyright, sharing, and attribution notice



This license allows you:

- **to share** - to copy, distribute and transmit the work
- **to adapt** - but you must distribute the new result under the same or similar license to this one
- **commercialize** - you *are allowed* to create commercial applications based on this work
- **attribution** - you must attribute the work as follows:
  - "Portions of this work are the copyright of ConnectMV", *or*
  - "This work is the copyright of ConnectMV"

We appreciate:

- if you let us know about **any errors** in the slides
- **any suggestions to improve the notes**
- telling us if you use the slides, especially commercially, so we can inform you of major updates
- emailing us to ask about different licensing terms

All of the above can be done by writing us at

`courses@connectmv.com`

If reporting errors/updates, please quote the current revision number: `268:adfd`

Overview

Last class: lots of questions on:

- how is the PCA model calculated: *loading directions*
- outliers and their effect on the model
- how many components

You will be able to **answer these questions yourself** after today's class.

Food texture example

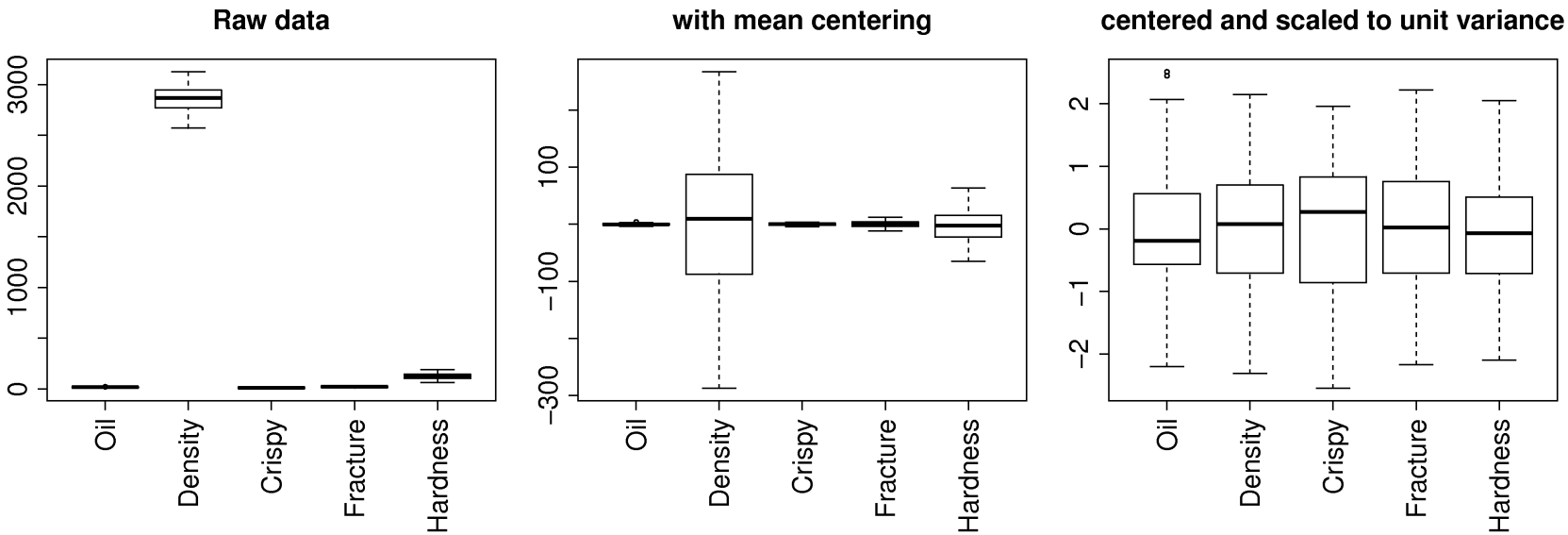
5 quality attributes are measured from pastries:

1. Percentage oil
2. Density
3. Crispiness measurement: from 7 (soft) to 15 (crispy)
4. Fracture angle
5. Hardness: force required before it breaks

Example: Pre-processing

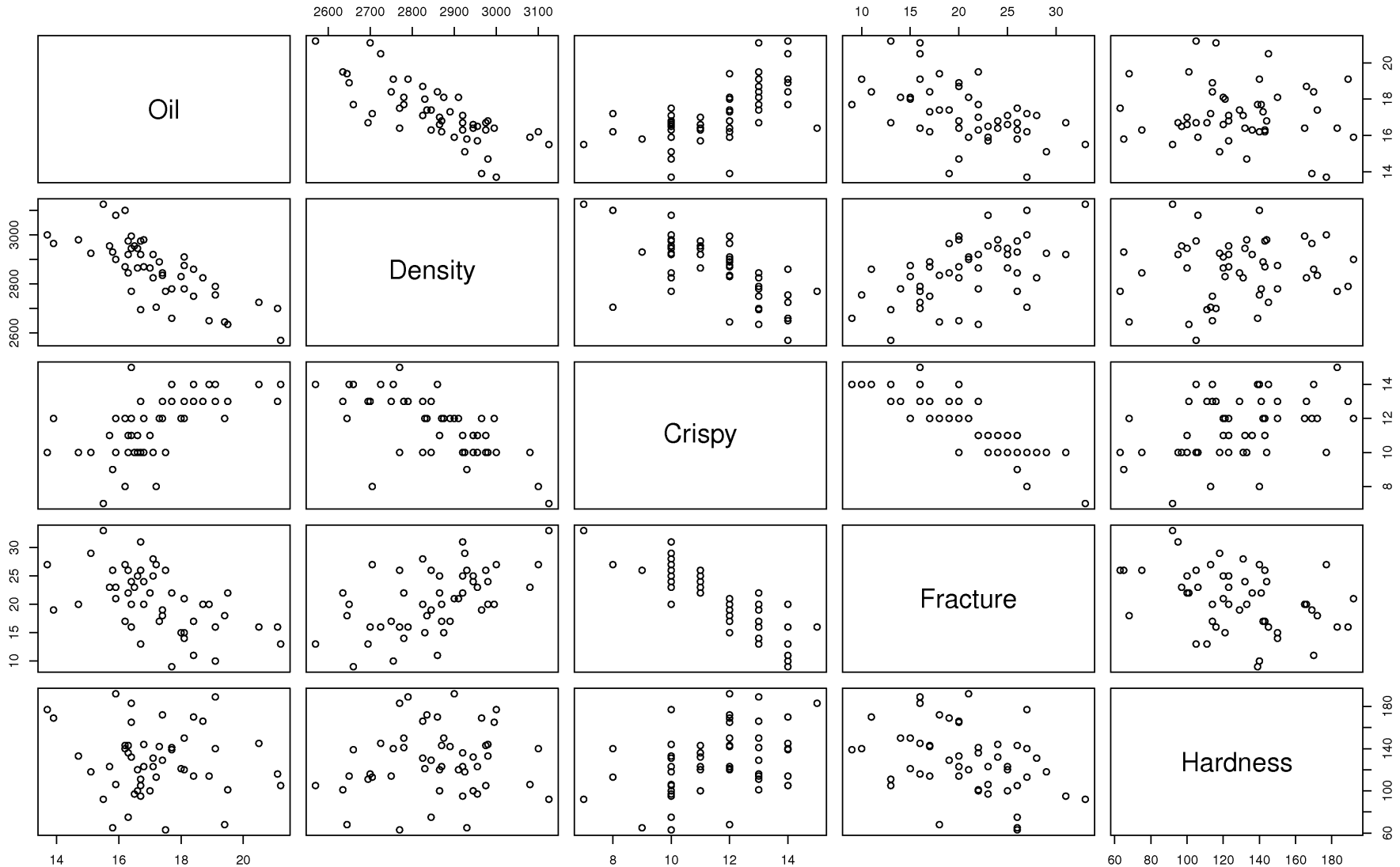
Pre-processing the data: center and scale

- Centering vector: [17.2, 2857.6, 11.5, 20.9, 128.2]
- Scaling vector (divide by standard deviation): [1.6, 124.5, 1.78, 5.47, 31.1]

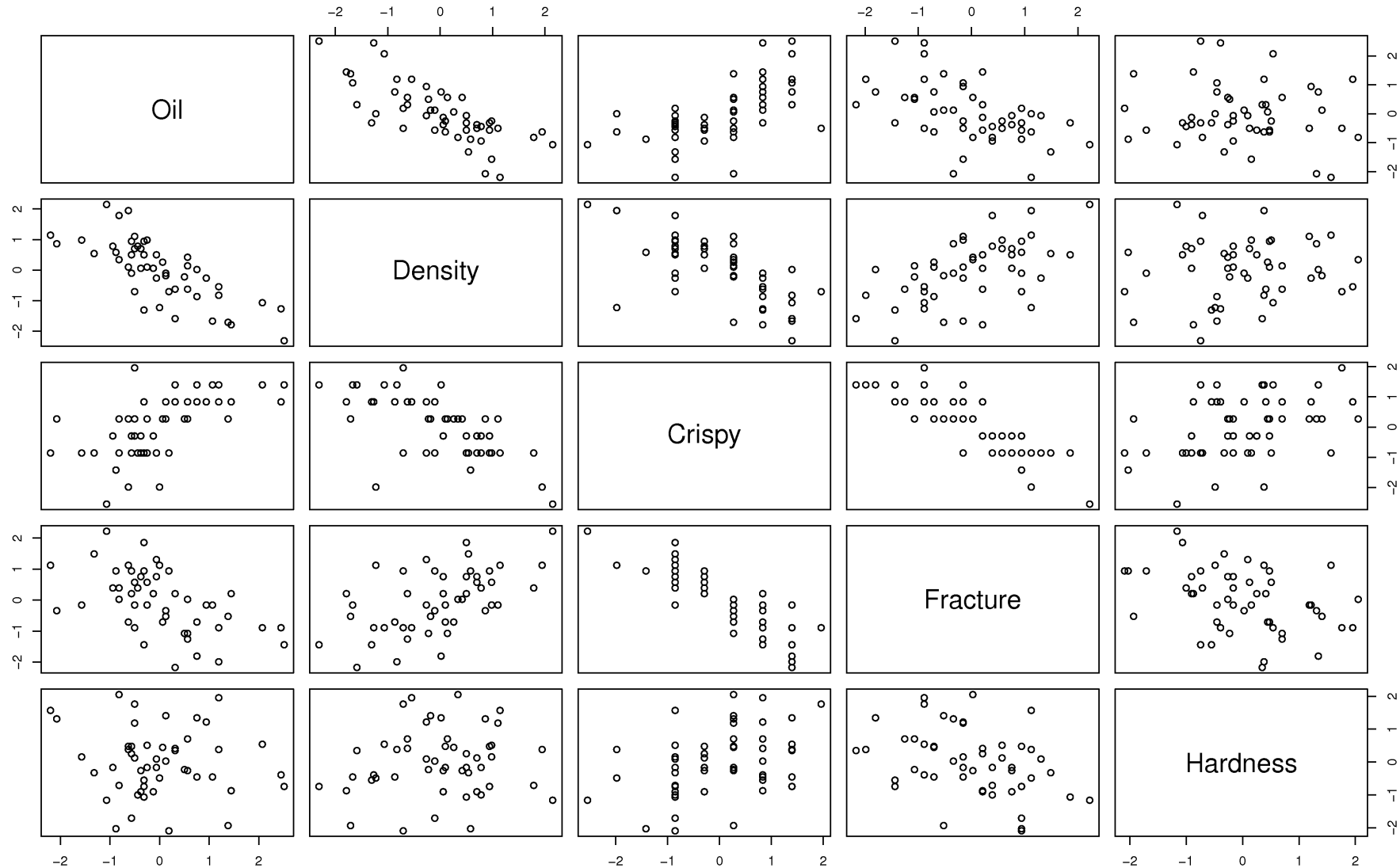


- Centering: $\mathbf{x}_{k,\text{center}} = \mathbf{x}_{k,\text{raw}} - \text{mean}\left(\mathbf{x}_{k,\text{raw}}\right)$
- Scaling: $\mathbf{x}_k = \dfrac{\mathbf{x}_{k,\text{center}}}{\text{standard deviation}\left(\mathbf{x}_{k,\text{center}}\right)}$
- Does not change relationships between variables.

Food texture: raw data



Food texture: centered and scaled data



Food texture: Variance explained

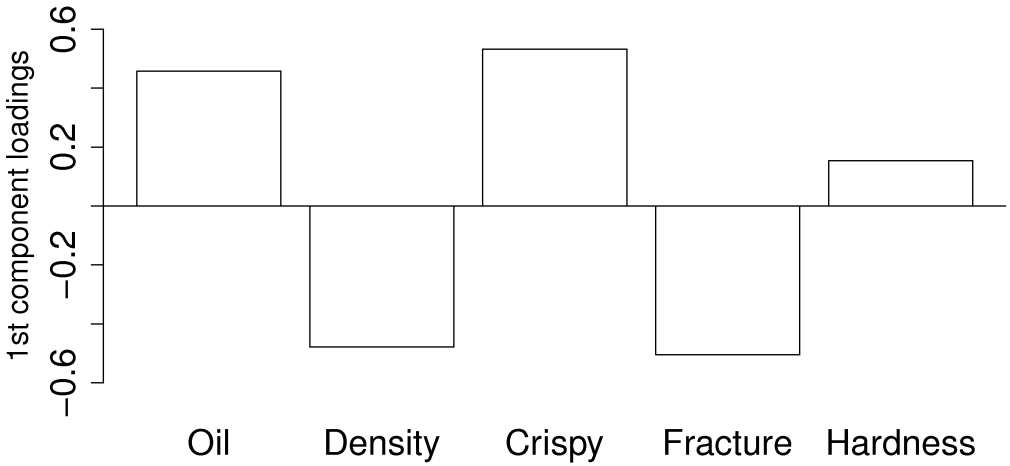
- PC1: explains 60.6%
- PC2: explains an additional 25.9% for a total of 86.5%

Each variable has an R^2 value. After 2 components they are:

1. Percentage oil: 81.2%
2. Density: 86.0%
3. Crispy: 90.9%
4. Fracture: 83.4%
5. Hardness: 91.0%

Food texture: loading p_1

Loadings = direction vector



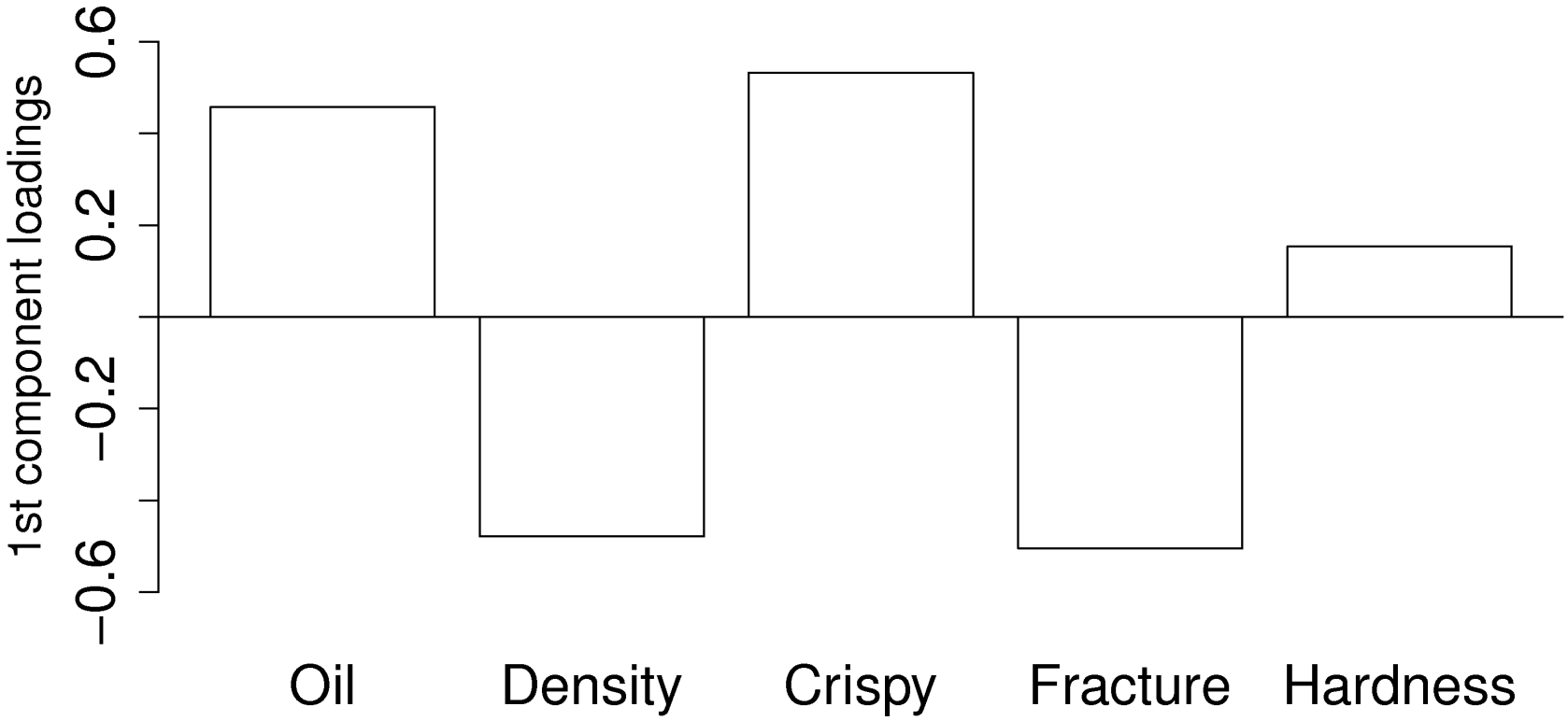
$$\mathbf{p}_1^T = \begin{bmatrix} 0.46, & -0.48, & 0.53, & -0.50, & 0.15 \end{bmatrix}$$

$$t_{1,i} = 0.46 x_{\text{oil}} - 0.48 x_{\text{density}} + 0.53 x_{\text{crispy}} - 0.50 x_{\text{fract}} + 0.15 x_{\text{hard}}$$

where:

- $x_{\text{oil}} = \dfrac{x_{\text{oil, raw}} - \text{mean}(x_{\text{oil, raw}})}{\text{standard deviation}(x_{\text{oil,raw}})}$
- *etc* for the other variables

Example: loading p_1

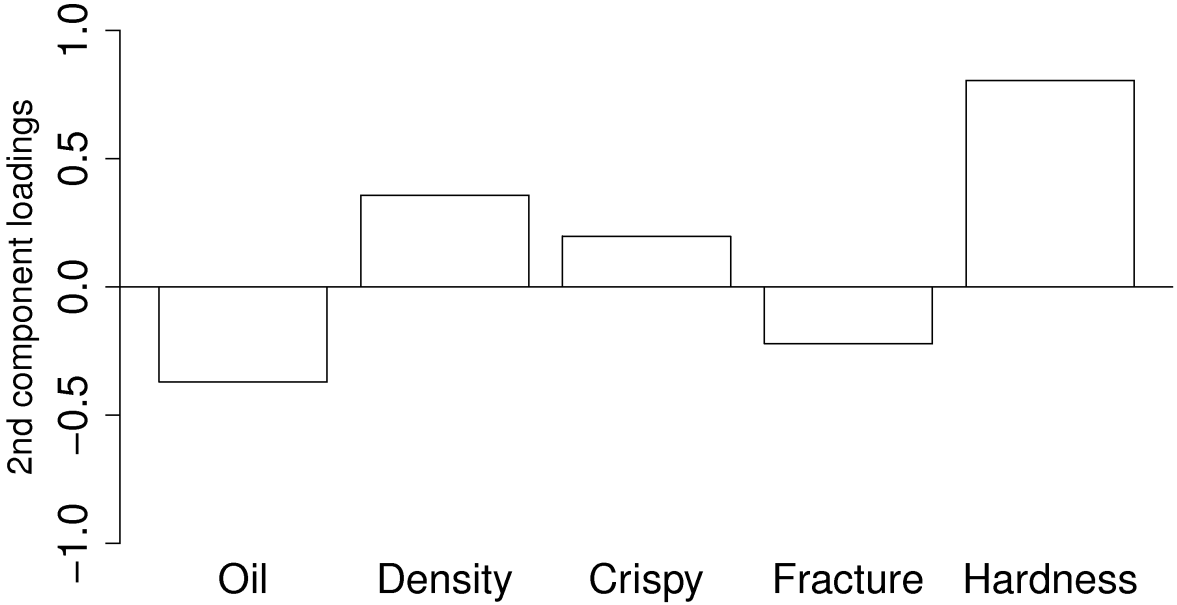


$$t_{1,i} = 0.46x_{\text{oil}} - 0.48x_{\text{density}} + 0.53x_{\text{crispy}} - 0.50x_{\text{fract}} + 0.15x_{\text{hard}}$$

- A high t_1 value:
- A low t_1 value:

Example: loadings

The second loading vector:



- Interpretation?
- Explains 26% of additional variability
- Is orthogonal (independent) to p_1. This means ...
  - can adjust process conditions for hardness without affecting other pastry properties

Replicate t_1 score for pastry B758

Observation B758 (row 36): $t_1 = 3.61$ (value from software)

Raw data:

- Oil = 21.2%
- Density = 2570
- Crisp = 14
- Fracture = 13
- Hardness = 105

Replicate t_1 score for pastry B758

- $t_1 = 0.46x_{\text{oil}} - 0.47x_{\text{density}} + 0.53x_{\text{crispy}} - 0.50x_{\text{fract}} + 0.15x_{\text{hard}}$
  - $x_{\text{oil}} = (21.2 - 17.2)/1.59 = 2.516$
  - $x_{\text{density}} = (2570 - 2857)/124.5 = -2.305$
  - $x_{\text{crisp}} = (14 - 11.52)/1.78 = 1.393$
  - $x_{\text{fracture}} = (13 - 20.9)/5.47 = -1.44$
  - $x_{\text{hardness}} = (105 - 128)/31.1 = -0.740$

$$
\begin{aligned}
t_1 \quad = \quad & +0.46(2.516) \\
& -0.47(-2.305) \\
& +0.53(1.393) \\
& -0.50(-1.44) \\
& +0.15(-0.740) = \mathbf{3.59}
\end{aligned}
$$

$$
t_1 \quad = \quad 1.16 + 1.08 + 0.738 + 0.72 - 0.11 = \mathbf{3.59}
$$

Overview: how is a PCA model calculated?

We will look at 3 ways today:

- Eigenvalue decomposition
- Singular value decomposition
- Non-linear iterative partial least-squares (NIPALS) algorithm
  - Used by most software packages

Why look at all the algorithms?

Each method highlights interesting properties of PCA

Optimization recap

Optimization problems are written in standard form:

$$\max_{\square} \quad \varphi$$

$$\textit{subject to:}$$

$$\square$$

PCA: optimization point of view

$$\max_{\square} \quad \varphi$$

$$\textit{subject to:}$$

$$\square$$

For PCA:

- What is a reasonable objective function?
- What are we searching for?
- Any constraints?

PCA: optimization derivation

Will be completed on the board. Brace yourselves for some math ...

- First component derivation
- Second component derivation

So what have we learnt?

PCA: optimization derivation

- PCA is the eigendecomposition of $\mathbf{X}'\mathbf{X}$
- Note that $\mathbf{X}'\mathbf{X}$ is a real, symmetric matrix
- Eigendecomposition of a real, symmetric matrix:
  - can always be calculated
  - the eigenvectors are linearly independent (orthogonal)
    - $p_i \perp p_j$ for $i \neq j$
  - the eigenvalues are all real and nonnegative
  - which is good, because we showed that eigenvalue $\lambda_a = \mathcal{V}(\mathbf{t}_a) >= 0$
  - we forced $\lambda_1 > \lambda_2 > \ldots > \lambda_A$
  - sum of all eigenvalues $= \sum_{a}^{a=K} \lambda_a = \text{trace}(\mathbf{X}'\mathbf{X})$
  - for a centered $\mathbf{X}$ matrix, $\text{trace}(\mathbf{X}'\mathbf{X}) = \text{ssq}(\mathbf{X})$
  - that's the denominator used to calculate $R^2 = 1 - \frac{\text{Var}(\mathbf{E}_a)}{\text{Var}(\mathbf{X})}$

PCA: optimization derivation

These 2 optimization problems are identical for PCA:

$$\max \quad \mathbf{t}_a' \mathbf{t}_a$$

(Maximizing variance)

$$\min \quad \text{ssq}\left(\mathbf{E}_a\right)$$

(Minimizing residual error)

Prove it to yourself.

Eigenvalue summary

For long and thin matrices ($N > K$), compute the PCA model:

- loadings, p_a, are the eigenvectors of $\mathbf{X}'\mathbf{X}$ (a $K \times K$ matrix)
- once you have the eigenvectors, then $\mathbf{t}_a = \mathbf{X}\mathbf{p}_a$
- then calculate the predicted $\widehat{\mathbf{X}}_A = \mathbf{t}_1\mathbf{p}_1' + \mathbf{t}_2\mathbf{p}_2' + \ldots + \mathbf{t}_A\mathbf{p}_A'$
- residuals $= \mathbf{E}_A = \mathbf{X} - \widehat{\mathbf{X}}_A$
- eigenvalues are the variances of the scores, s_a^2
- sum of all eigenvalues $= \text{trace}(\mathbf{X}'\mathbf{X}) = \text{Var}(\mathbf{X})$
- finally, calculate $R^2 = 1 - \dfrac{\text{Var}(\mathbf{E}_A)}{\text{Var}(\mathbf{X})}$

Eigenvalue summary

Alternatively for short and wide matrices where $N < K$:

- scaled version of $\mathbf{t}_a$ = eigenvectors of $\mathbf{X}\mathbf{X}'$ (an $N \times N$ matrix)
- scaled loadings = $\mathbf{p}_a = \mathbf{X}\mathbf{t}_a$
- rescale loadings: $\mathbf{p}_a = \frac{\mathbf{p}_a}{\|\mathbf{p}_a\|}$
- recalculate scores again: $\mathbf{t}_a = \mathbf{X}\mathbf{p}_a$ (or just using the scaling factors above)

Singular Value Decomposition (SVD)

- $\mathbf{X} = \mathbf{U\Sigma V}' = \mathbf{TP}'$
- scores, $\mathbf{T} = \mathbf{U\Sigma}$ and the loadings, $\mathbf{P} = \mathbf{V}$

Disadvantages of Eigendecomposition and SVD

These two approaches suffer the same drawbacks:

- cannot handle missing data
- both methods calculate all components at once, even though we only require $A \ll K$

Further disadvantages of the eigendecomposition for *large* matrices:

- calculating $\mathbf{X}'\mathbf{X}$ can be difficult on large arrays
- also prone to numerical overflow for very large datasets
- we need to keep $\mathbf{X}$ available anyway to calculate the scores
- negates the intended benefit of the eigendecomposition

Any advantages?

- They teach us a lot about what PCA is doing
- All the properties of PCA can be derived from these decompositions
- Are slightly more accurate since calculate error is spread over all components*

* NIPALS algorithm error increases as we add more components.

NIPALS algorithm

- NIPALS: Non-linear *iterative* partial *least squares* algorithm
- NIPALS: Non-linear iterative projections using alternating least squares
- Why study it?
  - insight into what the loadings and scores mean
  - another look at orthogonality between components
  - handles missing data
  - used by all major software packages

NIPALS algorithm

- Start with $\mathbf{X}$: preprocessed matrix of raw data
- More correctly, call it $\mathbf{X}_{a=0}$ or just $\mathbf{X}_0$
- to indicate that no components have been calculated yet

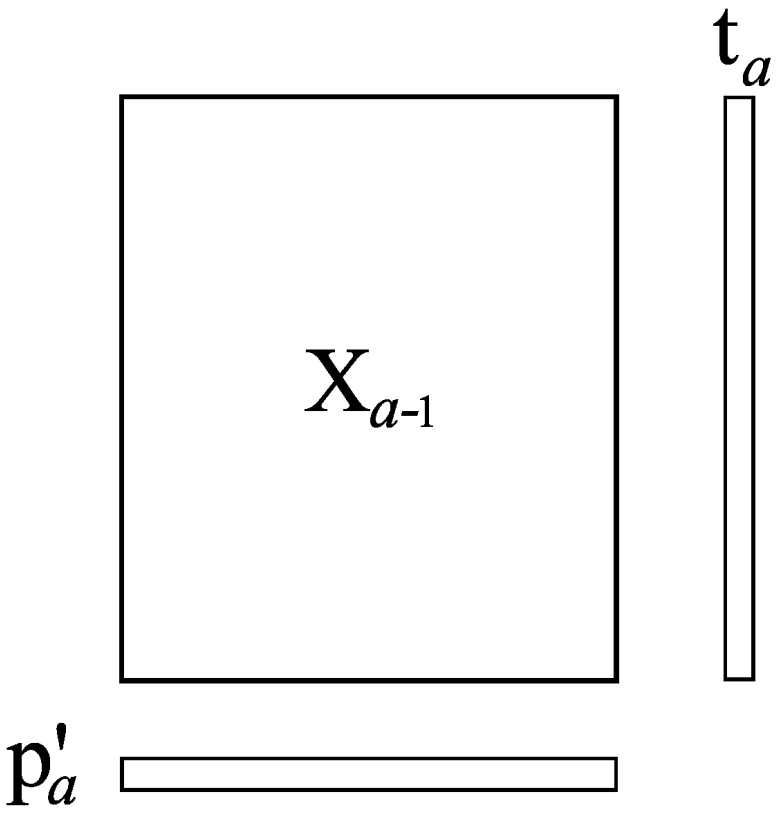
- We will break the algorithm into steps.

For $a = 1, 2, \ldots A$:

1. Select an arbitrary initial column for $\mathbf{t}_a$
2. In a while-loop, until convergence:
   - 2.1 Regress columns from $\mathbf{X}_{a-1}$ onto $\mathbf{t}_a$
   - 2.2 Normalize the loadings
   - 2.3 Regress rows from $\mathbf{X}_{a-1}$ onto $\mathbf{p}'_a$
3. Deflate component from $\mathbf{X}_{a-1}$ to calculate $\mathbf{X}_a$

End

NIPALS algorithm



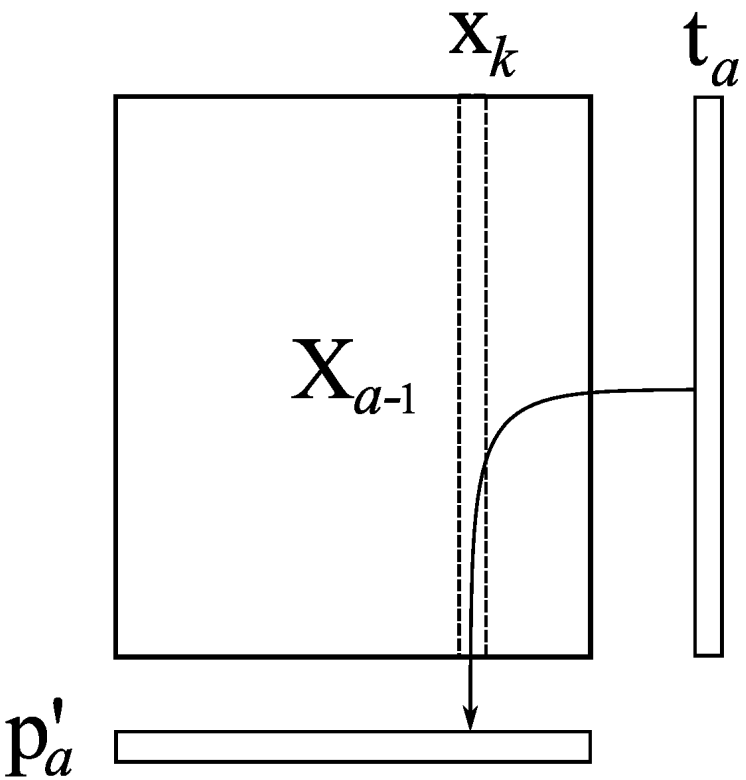
Step 1 Select an arbitrary initial column for $\mathbf{t}_a$

- Any column from $\mathbf{X}_0$
- A column of random numbers will also work
- Actually anything except a column of zeros works

NIPALS algorithm

Step 2.1 Regress every column from $\mathbf{X}_{a-1}$ (called $\mathbf{x}_k$) onto $\mathbf{t}_a$

- regress $\mathbf{x}_k$ onto $\mathbf{t}_a$ (terminology: "*regress a* $\mathbf{y}$ *onto an* $\mathbf{x}$")
- store regression coefficient as entry in $p_{k,a}$



- Recall LS for centered data:
  $\hat{\mathbf{y}} = \beta\mathbf{x}$, and $\widehat{\beta} = \dfrac{\mathbf{x}'\mathbf{y}}{\mathbf{x}'\mathbf{x}}$
- In this case: $p_{k,a} = \dfrac{\mathbf{t}'_a\mathbf{x}_k}{\mathbf{t}'_a\mathbf{t}_a}$

NIPALS algorithm

Step 2.1

- Repeat regression for every column in $\mathbf{X}_{a-1}$
- Can calculate regressions in one go:

$$\mathbf{p}'_a = \frac{1}{\mathbf{t}'_a \mathbf{t}_a} \cdot \mathbf{t}'_a \mathbf{X}_{a-1}$$

  - $\mathbf{t}_a$ is an $N \times 1$ column vector
  - $\mathbf{X}_{a-1}$ is an $N \times K$ matrix
  - $\mathbf{p}_a$ is a $K \times 1$ column vector

NIPALS algorithm

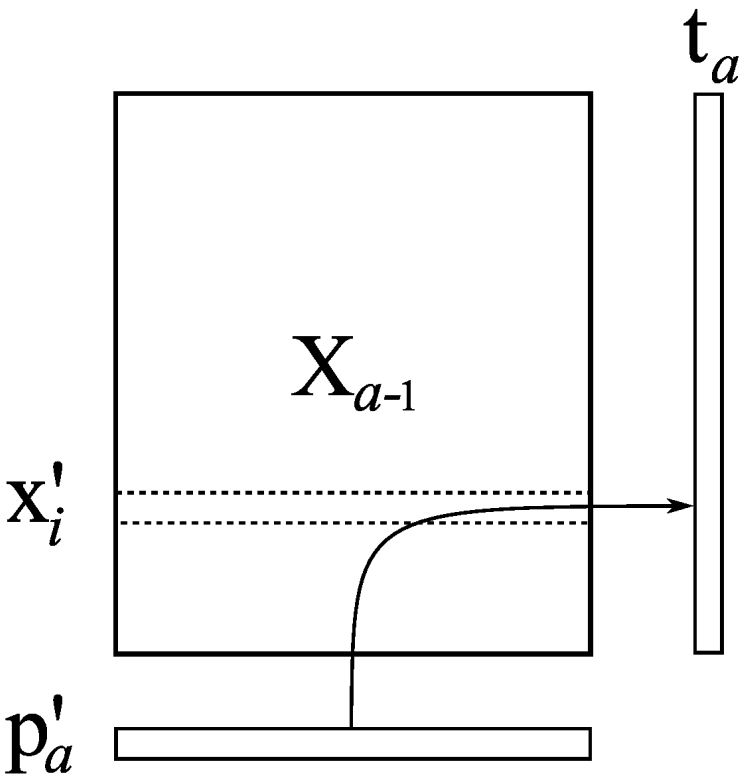
Step 2.2 Normalize the loadings

- $\mathbf{p}'_a$ won't have unit length (magnitude)
- Rescale it to magnitude 1.0
- $\mathbf{p}'_a = \frac{1}{\sqrt{\mathbf{p}'_a \mathbf{p}_a}} \cdot \mathbf{p}'_a = \frac{\mathbf{p}'_a}{\|\mathbf{p}'_a\|}$

NIPALS algorithm

Step 2.3 Regress every row in $\mathbf{X}$ onto $\mathbf{p}'_a$

- regress $\mathbf{x}_i$ onto $\mathbf{p}'_a$
- store regression coefficient as entry in $t_{i,a}$



- Recall LS for centered data:
  $\hat{\mathbf{y}} = \beta\mathbf{x}$, and $\widehat{\beta} = \dfrac{\mathbf{x}'\mathbf{y}}{\mathbf{x}'\mathbf{x}}$
- $t_{i,a} = \dfrac{\mathbf{p}'_a\mathbf{x}_i}{\mathbf{p}'_a\mathbf{p}_a}$

NIPALS algorithm

Step 2.3

- Repeat regression for every row in $\mathbf{X}_{a-1}$
- In practice: $\mathbf{t}_a = \frac{1}{\mathbf{p}'_a \mathbf{p}_a} \cdot \mathbf{X}_{a-1} \mathbf{p}_a$
  - $\mathbf{t}_a$ is an $N \times 1$ column vector
  - $\mathbf{X}_{a-1}$ is an $N \times K$ matrix
  - $\mathbf{p}_a$ is an $K \times 1$ column vector

NIPALS algorithm

Back to **step 2.** Have we converged?

- $\mathbf{t}_a$ compared to $\mathbf{t}_a$ from previous iteration
- Stop if change less than $\sqrt{\text{eps}} \approx 1.5 \times 10^{-8}$
- Could also compare change in $\mathbf{p}_a$ to previous iteration

- Safety net: also stop if number of iterations > 300

At convergence:

- $\mathbf{t}_a$ and $\mathbf{p}_a$ jointly form the a^{th} component
- Store them as columns in matrix $\mathbf{T}$ and $\mathbf{P}$ respectively

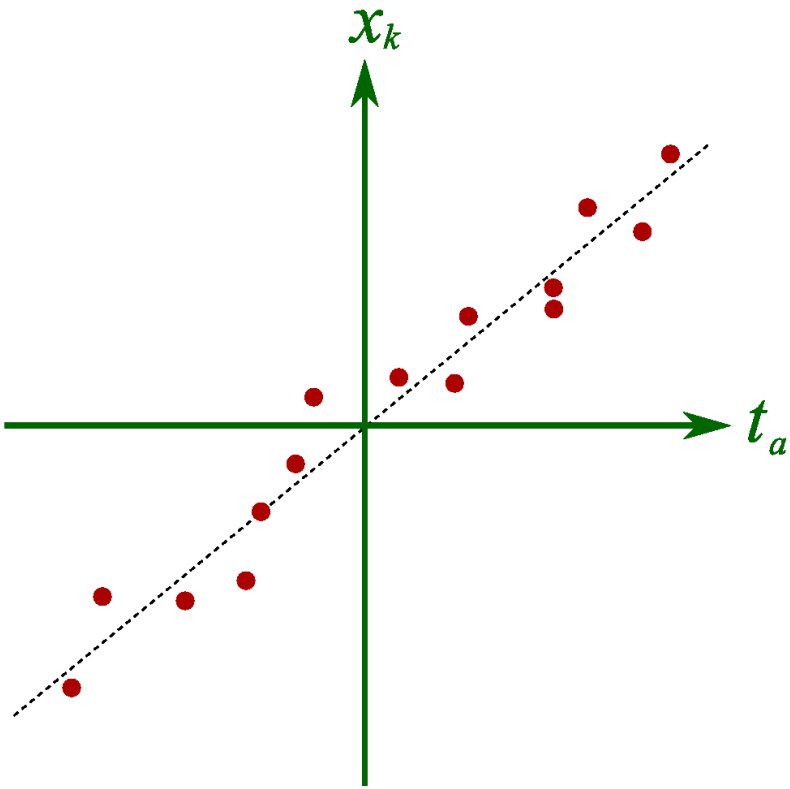
NIPALS algorithm

Finally, **step 3** Deflate the $\mathbf{X}_{a-1}$ matrix

- Deflation removes the part we can explain
- $\mathbf{E}_a = \mathbf{X}_{a-1} - \mathbf{t}_a\mathbf{p}'_a$
- $\mathbf{E}_a$ = residuals *after* fitting the a^{th} component
- Then let $\mathbf{X}_a = \mathbf{E}_a$ and repeat from step 1 for $a + 1$
- e.g. for $a = 1$: $\mathbf{X}_{a-1} = \mathbf{X}_0$ = preprocessed raw data
- e.g. for $a = 2$: $\mathbf{X}_1$ = residuals after 1 component = data matrix used to calculate 2^{nd} component

What happens at convergence?

Let's review the regressions calculated on the last iteration:



- **Step 2.1** Regress every column from $\mathbf{X}_{a-1}$ onto $\mathbf{t}_a$
- $p_{k,a} = \dfrac{\mathbf{t}_a' \mathbf{x}_k}{\mathbf{t}_a' \mathbf{t}_a}$
- What will regression look like for a strong relationship?
- Weak/no relationship?
- Meaning of the loading $p_{k,a}$ should be apparent now
- Regression can be used to predict: $\hat{\mathbf{x}}_k = \mathbf{t}_a' p_{k,a}$

On your own: interpret **step 2.3** when we regress rows in $\mathbf{X}_{a-1}$ onto $\mathbf{p}_a$

What happens after convergence?

After convergence of $\mathbf{t}_a$ and $\mathbf{p}_a$:

- $\mathbf{p}'_a = \dfrac{\mathbf{t}'_a \mathbf{X}_{a-1}}{\mathbf{t}'_a \mathbf{t}_a}$
- Drop the "a" subscripts for now; transpose the entire equation
- Rewrite step 2.1 as: $\mathbf{p} = \dfrac{\mathbf{X}'\mathbf{t}}{\mathbf{t}'\mathbf{t}}$
- Rewrite step 2.3 as: $\mathbf{t} = \dfrac{\mathbf{X}\mathbf{p}}{\mathbf{p}'\mathbf{p}}$
- Note that $\mathbf{p}'\mathbf{p} = 1.0$
- Substitute $\mathbf{t}$ into equation for $\mathbf{p}$ gives $\mathbf{p} = \dfrac{\mathbf{X}'\mathbf{X}\mathbf{p}}{\mathbf{t}'\mathbf{t}}$
- Rearrange to $(\mathbf{X}'\mathbf{X} - \mathbf{t}'\mathbf{t}\mathbf{I}_K)\,\mathbf{p} = \mathbf{0}$ where $\mathbf{I}_K$ is a $K \times K$ identity matrix

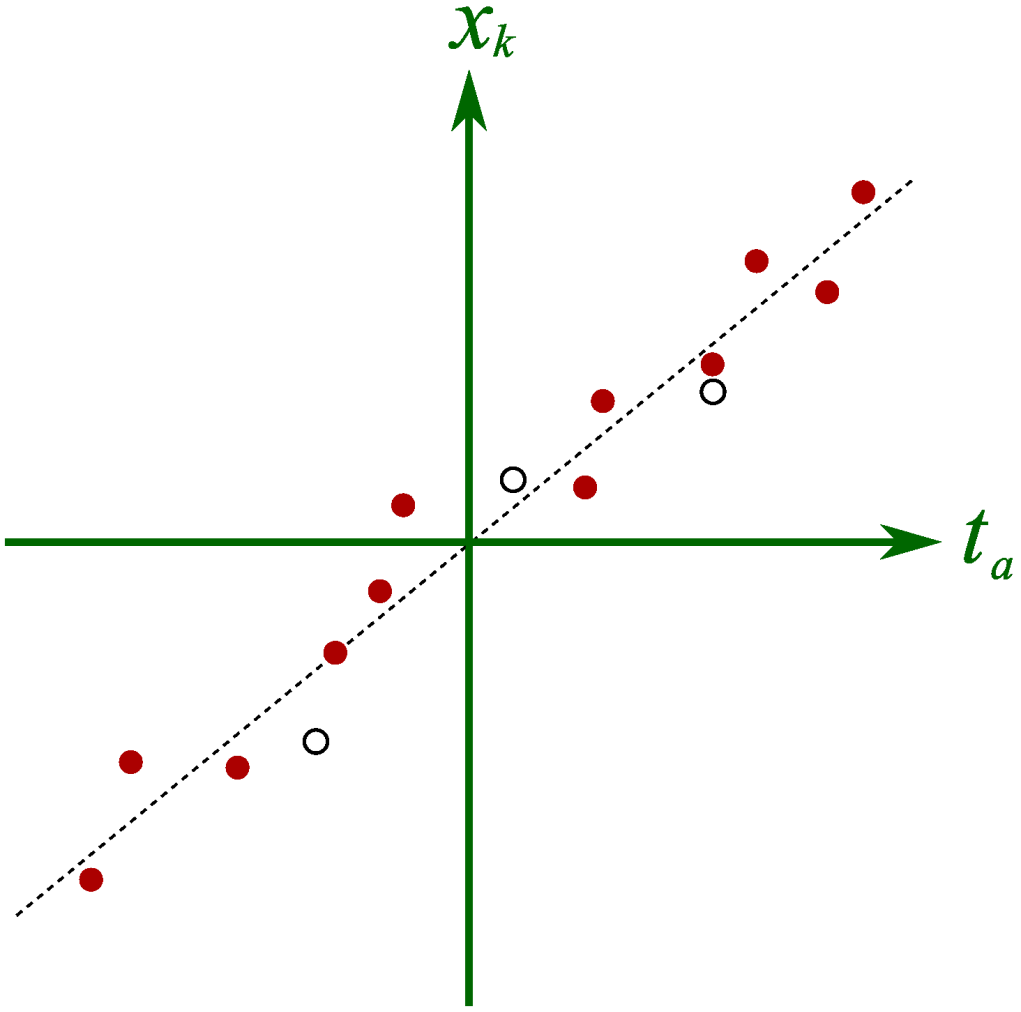
This shows (again) that:

- $\mathbf{p}$ is an eigenvector of $\mathbf{X}'\mathbf{X}$
- The eigenvalue is $\lambda = \mathbf{t}'\mathbf{t}$, which we interpret/know as the variance of $\mathbf{t}$

NIPALS notes

- Convergence is fast if the eigenvalues are well separated
- Two close eigenvalues leads to very slow convergence, followed by very fast convergence for the next one
- The algorithm handles missing data (next)

NIPALS algorithm: concept of handling missing data



Missing values are ignored and do not influence the slope calculation.

More details:

- Nelson, Taylor, MacGregor (paper 68)
- Arteaga and Ferrer (paper 20)

Outliers

Discussion

What will an outlier do to a PCA model?

NIPALS summary

Advantages:

- Calculates one component at a time
- Handles missing data
- It converges (sometimes slowly)

Disadvantages:

- Round off errors may accumulate if you go very far (not usually a problem on modern computers)

Notes:

- Also called the Power algorithm for computing eigenvalues of a square matrix
- Excellent on large data sets (large N *and* large K)
- Google used this algorithm for their first search engine (called PageRank)
  - http://ilpubs.stanford.edu:8090/422/
  - Ipsen, Ilse, and Wills, "*Analysis and Computation of Google's PageRank*", 7th IMACS International Symposium on Iterative Methods in Scientific Computing, Fields Institute, Toronto, Canada, 5-8 May 2005

Flipping signs

In NIPALS, SVD or eigendecompositions:

- $\hat{\mathbf{X}}_1 = \mathbf{t}_1\mathbf{p}_1' = (-\mathbf{t}_1)(-\mathbf{p}_1')$
- Both the scores and loadings may flip sign
- Depends on the computer, initial guesses, algorithm implementation
- Not a problem: model interpretation is still consistent
- Not a problem: model's performance is identical

Just be aware when comparing results from different users/software/computers.

For next class

1. Read the following 2 papers for an overview of process monitoring
   - Kresta, MacGregor and Marlin (paper 9)
   - Kourti and MacGregor (paper 31)
2. Next class will cover
   - using PCA for process monitoring
   - various contribution plots from PCA models
   - how are the *limits* derived for PCA models