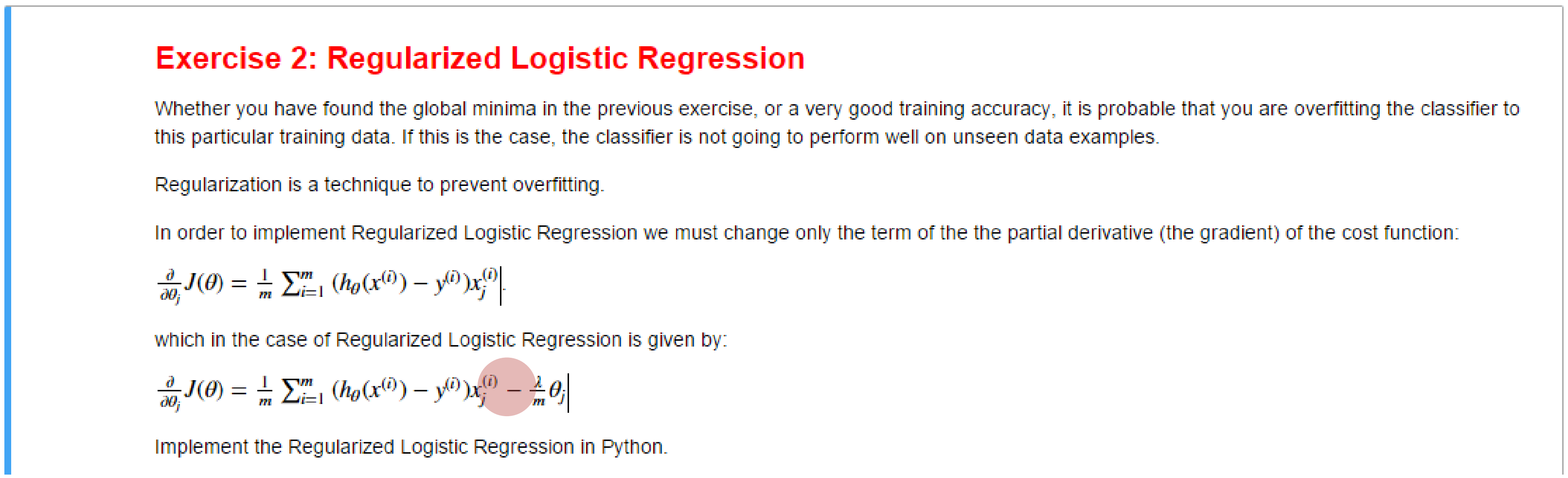
Pattern Analysis and Recognition

Lecture 4: Subspace methods, PCA, whitening, ZCA, LDA

Issue in practical text



Exercise 2: Regularized Logistic Regression

Whether you have found the global minima in the previous exercise, or a very good training accuracy, it is probable that you are overfitting the classifier to this particular training data. If this is the case, the classifier is not going to perform well on unseen data examples.

Regularization is a technique to prevent overfitting.

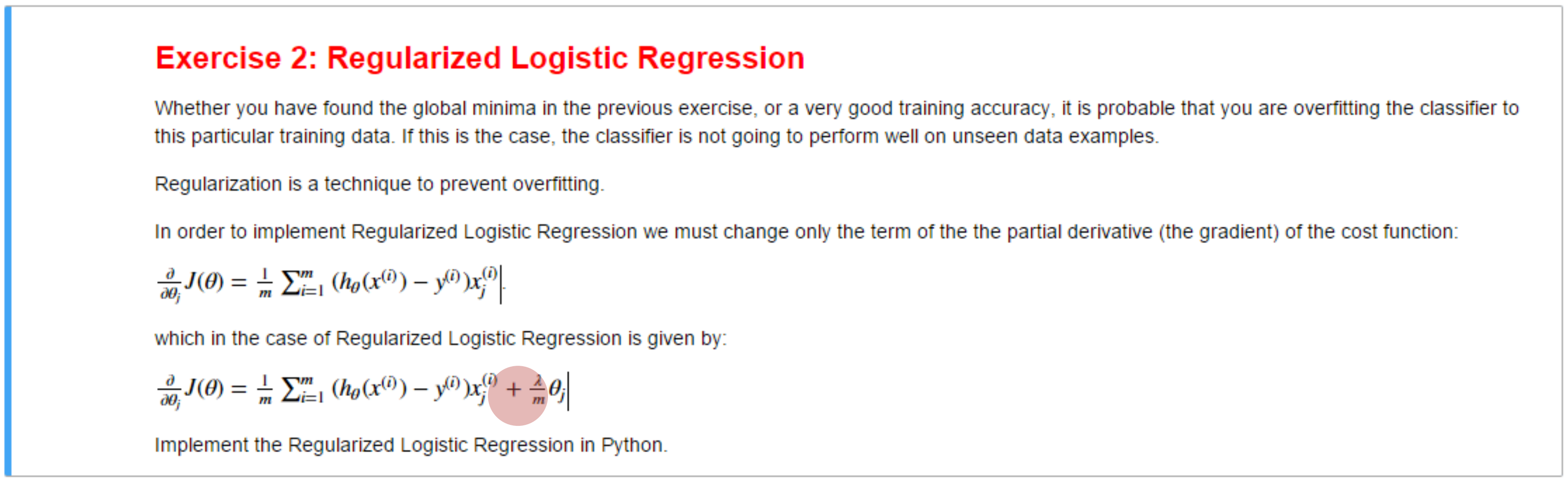
In order to implement Regularized Logistic Regression we must change only the term of the the partial derivative (the gradient) of the cost function:

$$\frac{\partial}{\partial\theta_j}J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

which in the case of Regularized Logistic Regression is given by:

$$\frac{\partial}{\partial\theta_j}J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} - \frac{\lambda}{m}\theta_j$$

Implement the Regularized Logistic Regression in Python.



Exercise 2: Regularized Logistic Regression

Whether you have found the global minima in the previous exercise, or a very good training accuracy, it is probable that you are overfitting the classifier to this particular training data. If this is the case, the classifier is not going to perform well on unseen data examples.

Regularization is a technique to prevent overfitting.

In order to implement Regularized Logistic Regression we must change only the term of the the partial derivative (the gradient) of the cost function:

$$\frac{\partial}{\partial\theta_j}J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

which in the case of Regularized Logistic Regression is given by:

$$\frac{\partial}{\partial\theta_j}J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} + \frac{\lambda}{m}\theta_j$$

Implement the Regularized Logistic Regression in Python.

Last time on Pattern Analysis and Recognition

RECAP

Feature scaling – mean normalisation

Aim: get every feature x_j into approximately a $-1 \leq x_j \leq 1$ range

Mean normalization:

- Subtract from each feature x_j the feature mean (μ_j) to make features have approximately zero mean
- Divide by the feature range or the standard deviation (s_j)
- Do not apply to x_0 !!!

$$x_j^{(i)} = \frac{x_j^{(i)} - \mu_j}{s_j}$$

$$\mu_j = \overline{x_j} = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)}$$

$$s_j = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(x_j^{(i)} - \mu_j\right)^2}$$

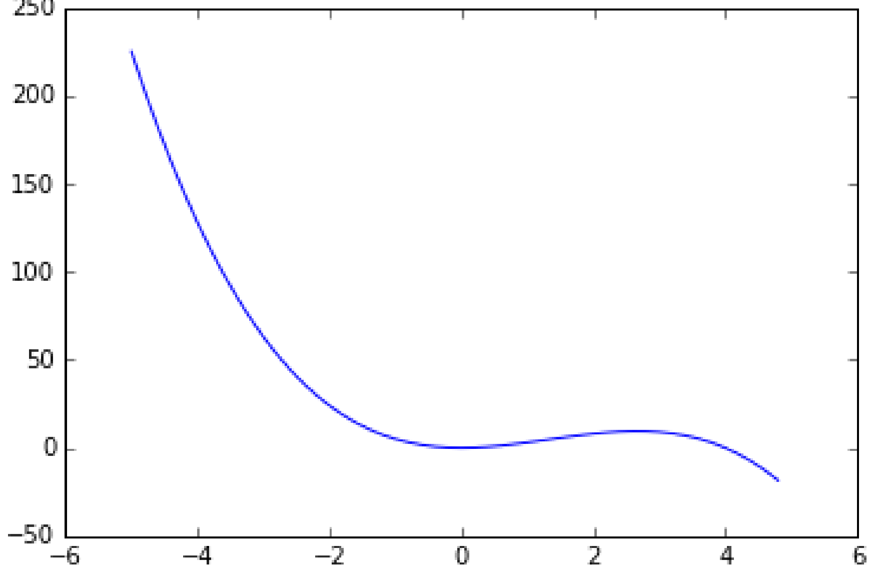
Running Example: a noisy cubic

Suppose our data model is:

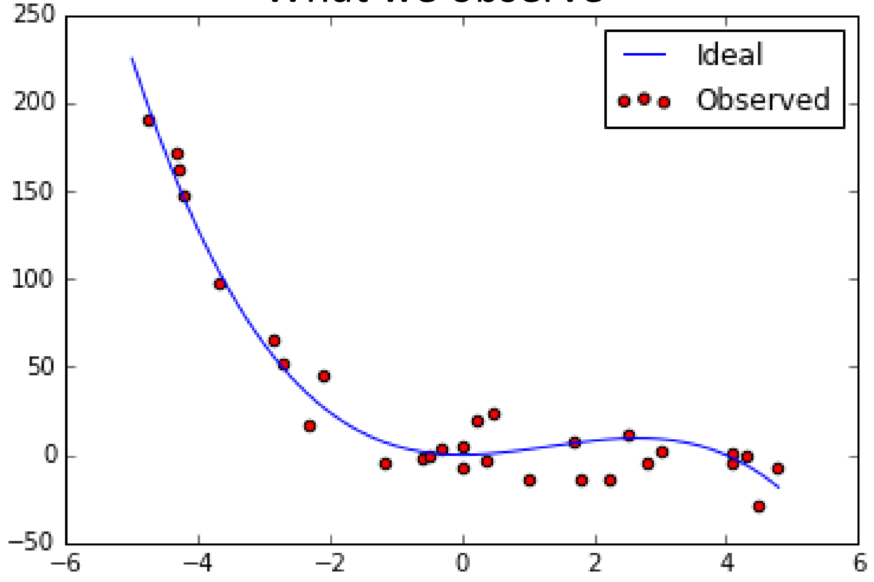
$$y^i = f\left(x^i\right) + N(0, \sigma)$$

$$f(x) = 4x^2 - x^3, \qquad \sigma = 5$$

Behind the scenes



What we observe



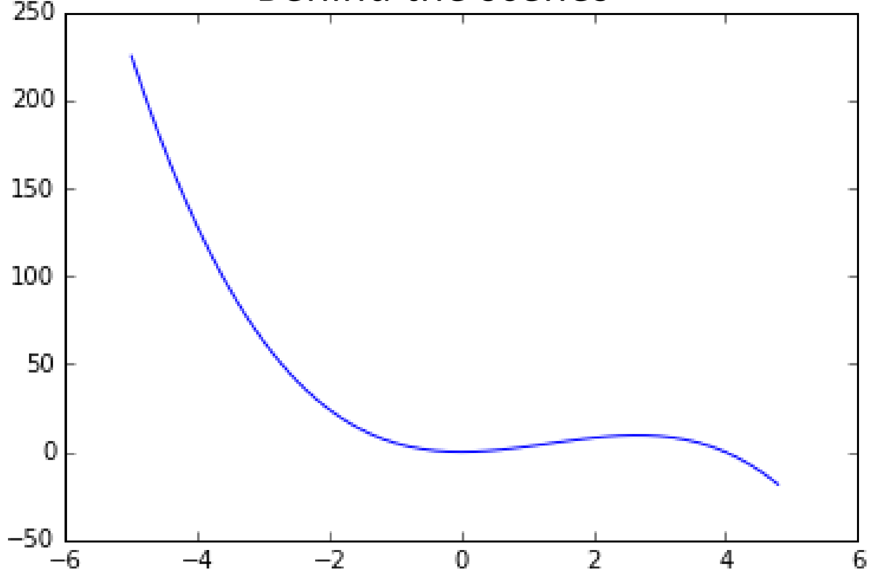
Running Example: a noisy cubic

Suppose our data model is:

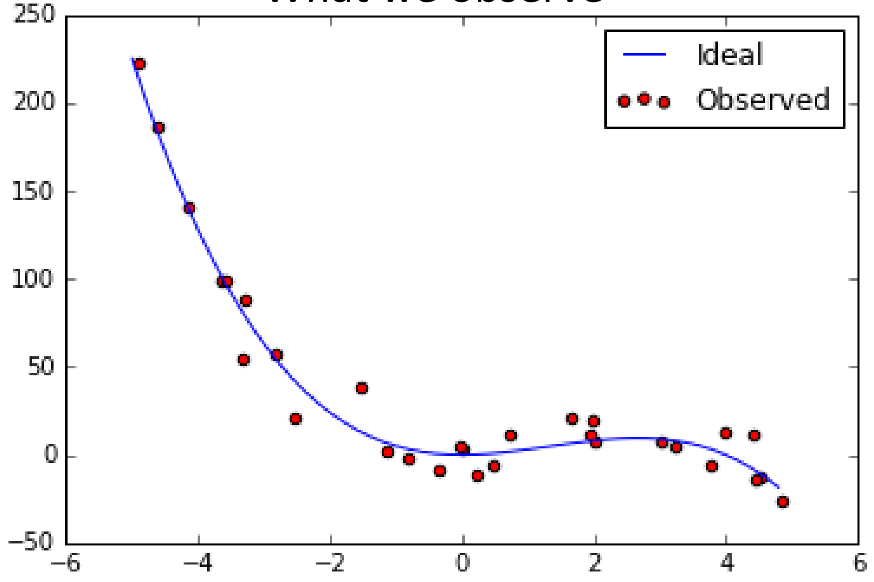
$$y^i = f\left(x^i\right) + N(0, \sigma)$$

$$f(x) = 4x^2 - x^3, \qquad \sigma = 5$$

Behind the scenes



What we observe



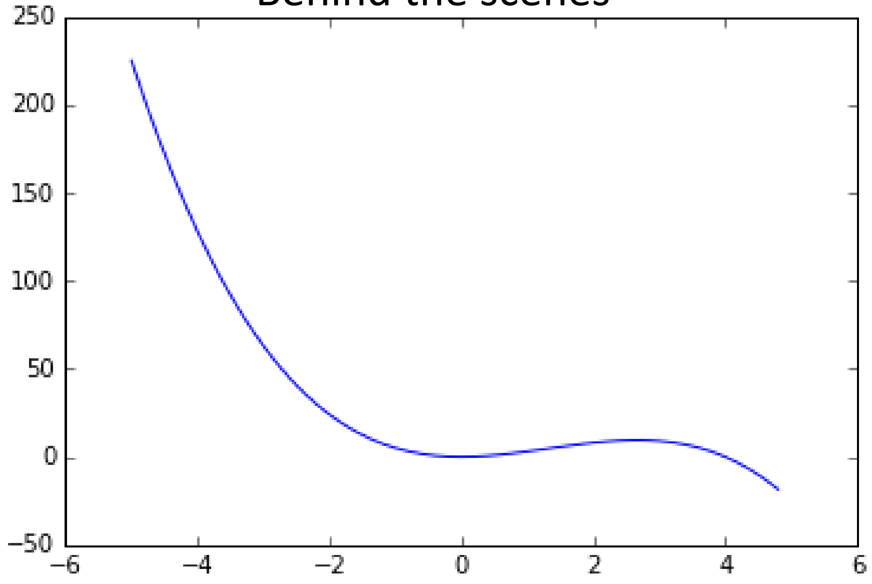
Running Example: a noisy cubic

Suppose our data model is:

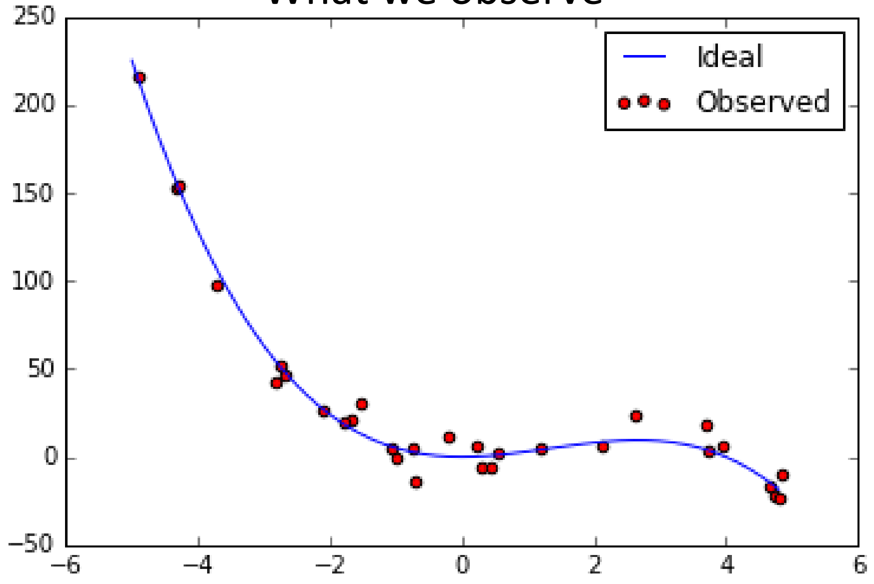
$$y^i = f\left(x^i\right) + N(0, \sigma)$$

$$f(x) = 4x^2 - x^3, \qquad \sigma = 5$$

Behind the scenes



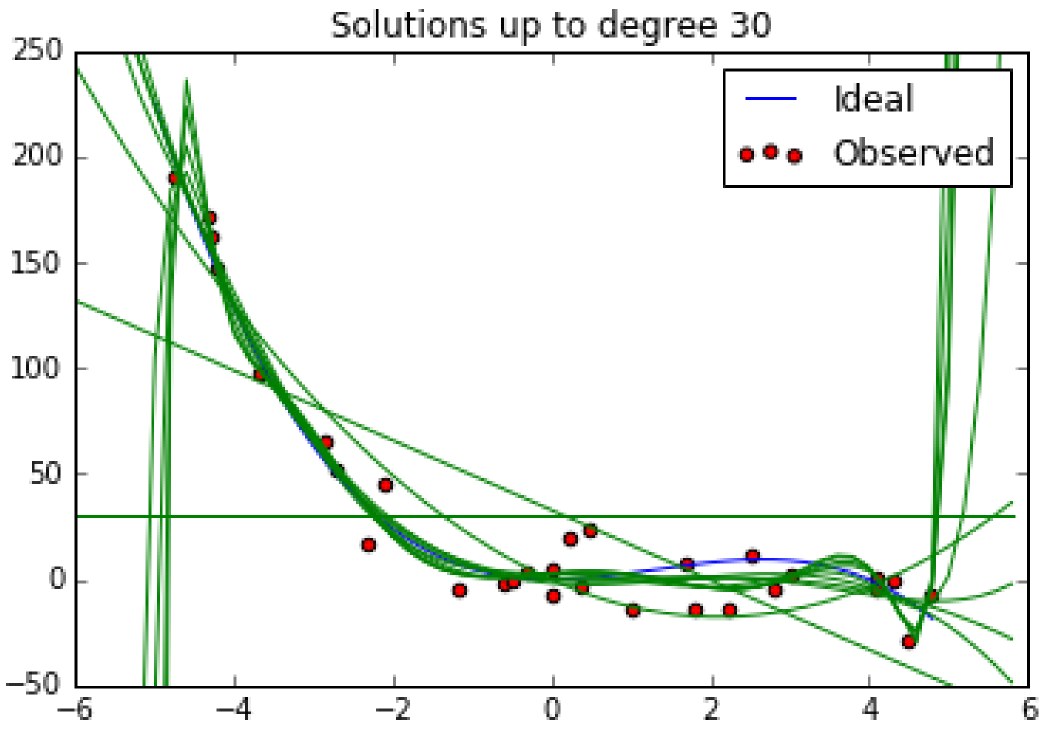
What we observe



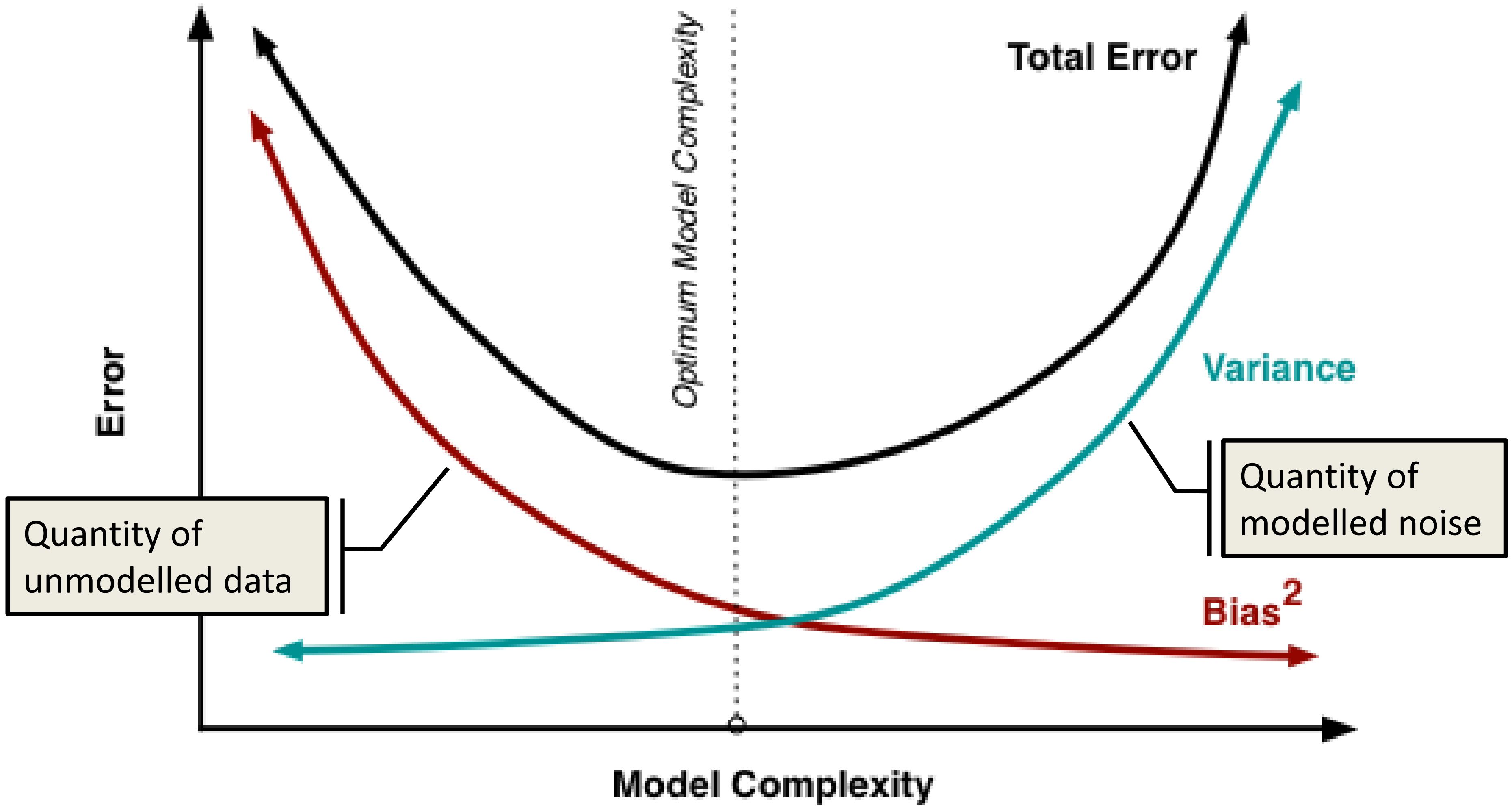
Increasing model complexity: when should we stop?

On the same problem instance $(\mathbf{X}, \mathbf{y})$

High degree design matrix: $\phi(x) = [1, x, x^2, \ldots, x^N]$

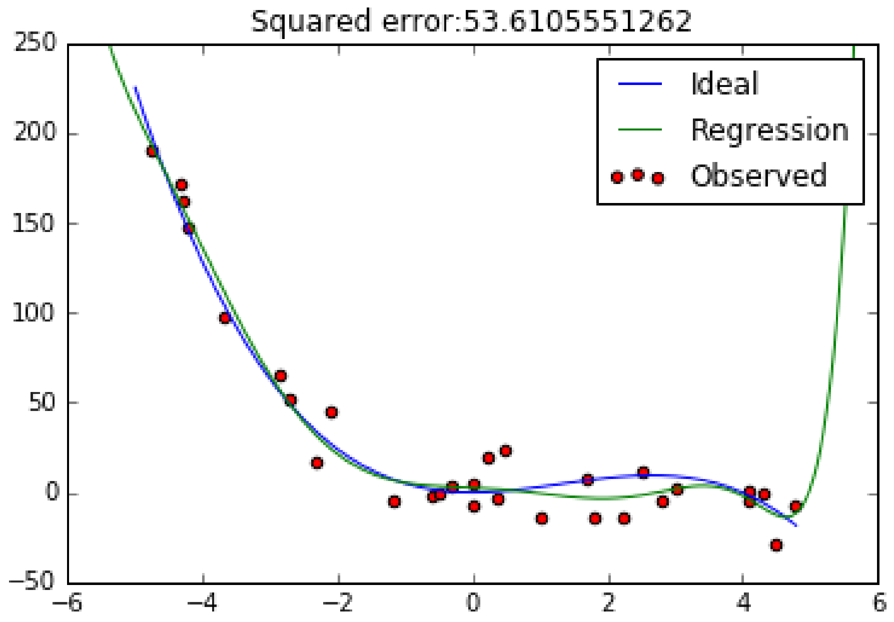


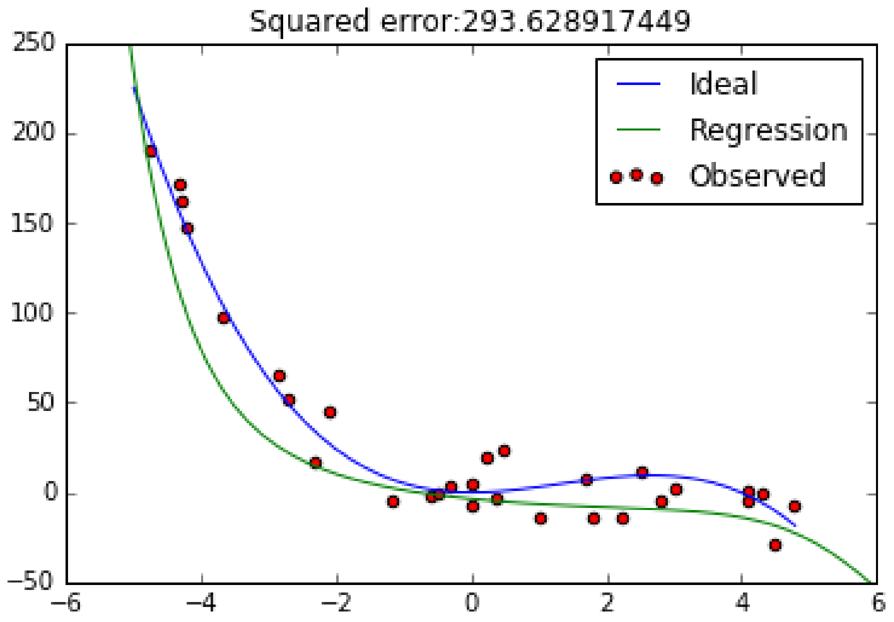
The Bias/Variance trade-off



Same model, different complexity

These two fits are both 10-degree polynomials



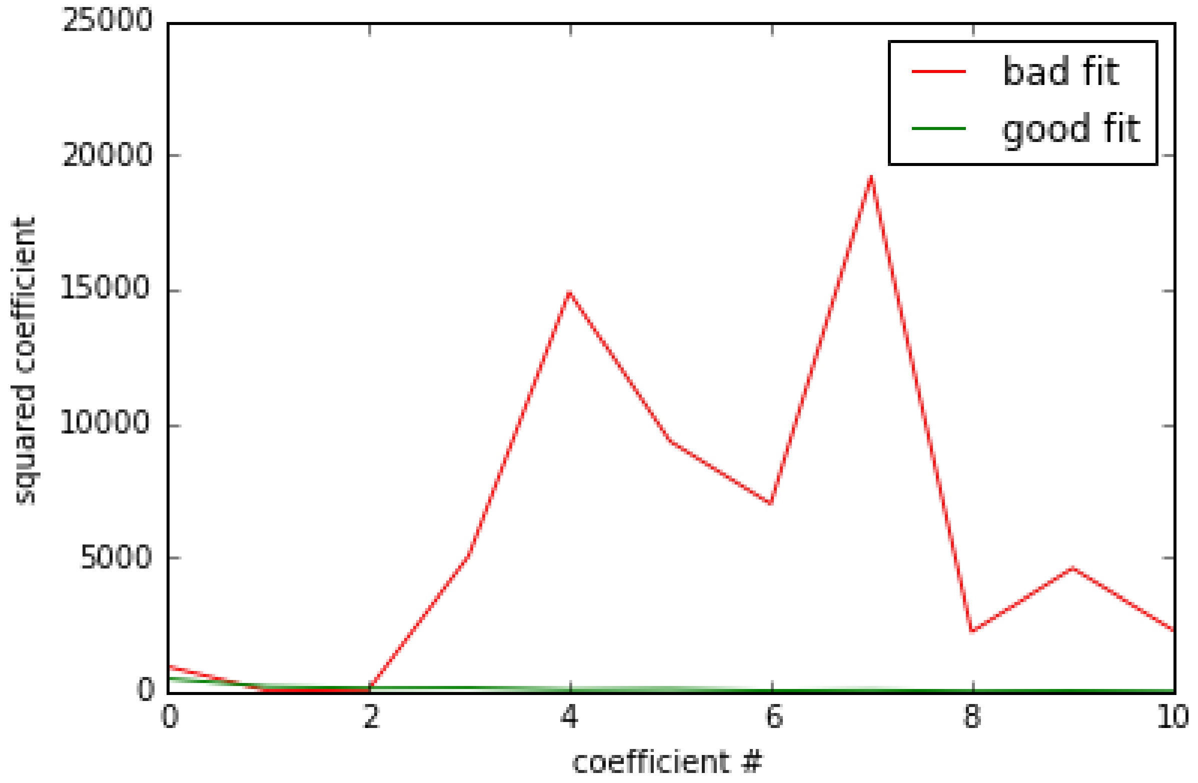


Which one is “better”?

What is the difference?

A look in the coefficients

A look into the squared coefficients (w_i^2) reveals:



Coefficient Magnitude

It turns out that coefficient magnitude is not a bad measure of fit complexity
What we do is *penalize* high coefficients by adding a *regularization term* to the loss:

$$\begin{aligned}\mathcal{L}(\mathbf{w}; \mathbf{X}, \mathbf{y}, \phi) &= \sum_{i=1}^{m} L\left(\phi\left(\mathbf{x}^{i}\right)^{T}\mathbf{w}, y^{i}\right) + \lambda(\mathbf{w}^{\boldsymbol{T}}\mathbf{w}) \\ &= \sum_{i=1}^{m} L\left(\phi\left(\mathbf{x}^{i}\right)^{T}\mathbf{w}, y^{i}\right) + \lambda \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \mathrm{w}_i^2\end{aligned}$$

Bias term in regularizer?

Is w_0 supposed to be included in the regularizer?

Right answer: No

Practical answer: it does not matter… if you center your data beforehand (remember normalization)

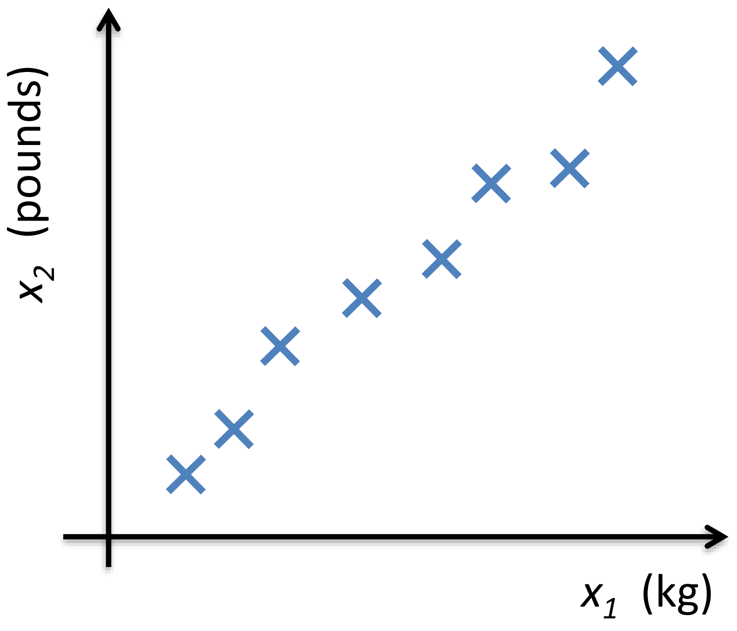
NOTE: Technically, many of the statements in this lecture about the "covariance" will be true only if the data has zero mean. In the rest of this lecture, we will take this assumption as implicit in our statements.

Some slides adapted from Andrew Bagdanov, Andrew Ng, Victor Lavrenko

SUBSPACE METHODS

True versus observed dimensionality

Imagine some instance of a problem, for which we are given the following measurements over two features {weight, pisua}



What is the dimensionality of this data?

"weight"= "pisua" in Basque…

True versus observed dimensionality

Imaging we have sensors in the city, or we receive information from some monitoring agency about:

- Number of patients with heat stroke
- Number of water bottles sold
- Number of parking spaces available in Castelldefels
- Noise levels in dB at the beach
- …

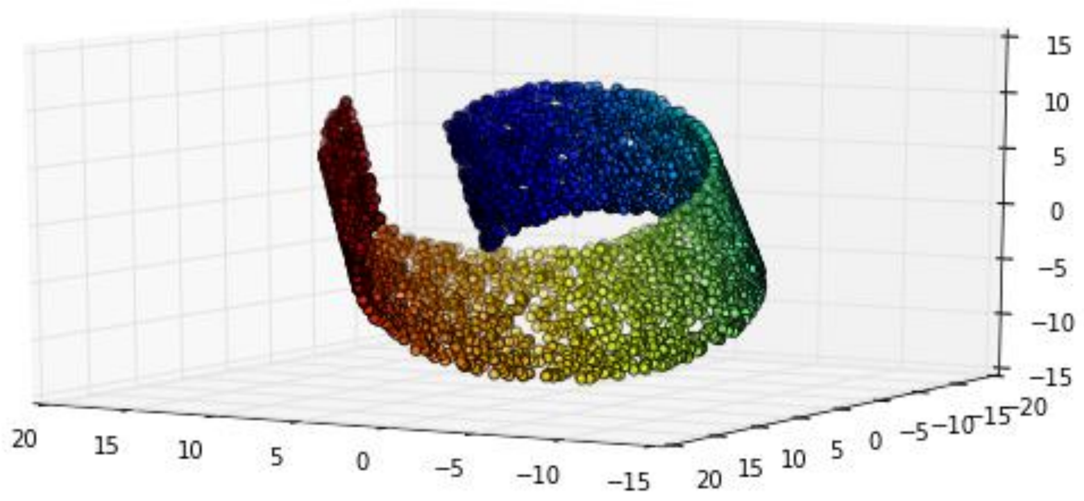
Maybe these can all be explained by a couple of **latent** ("hidden") variables: temperature, time of the day

Curse of dimensionality

Realistic data in machine learning are typically high dimensional:

- Images: a HD image comprises 10^5 pixels
- Text: English wikipedia counts 10^{10} words
- Gene expression experiments deal with 10^4 of genes
- …

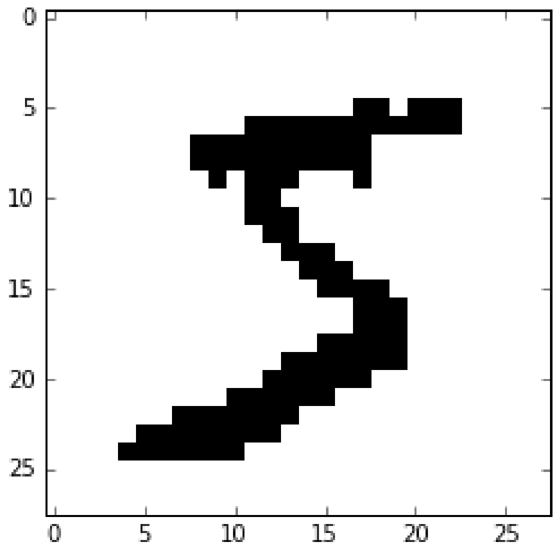
Is this the true dimensionality of the data? In most of the cases the true dimensionality is much lower, and data lie on a low dimension manifold in a high dimension space

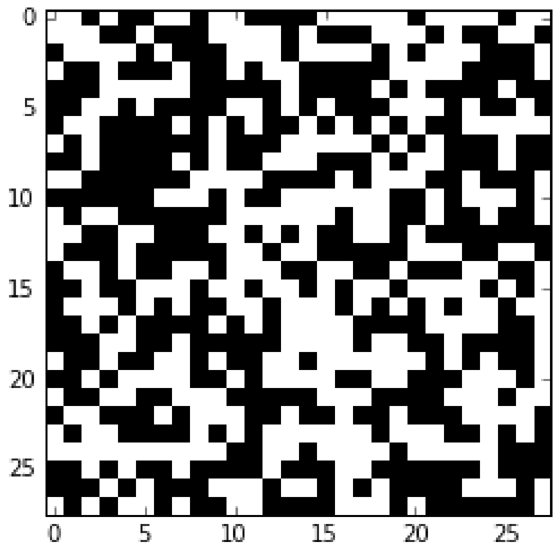


Curse of dimensionality

Example: Handwritten digits

- 20 x 20 bitmaps
- Each pixel can either be black or white: $\{0, 1\}^{400}$ possible events
- We will never see most of the events
- The actual digits are a tiny fraction of the possible events



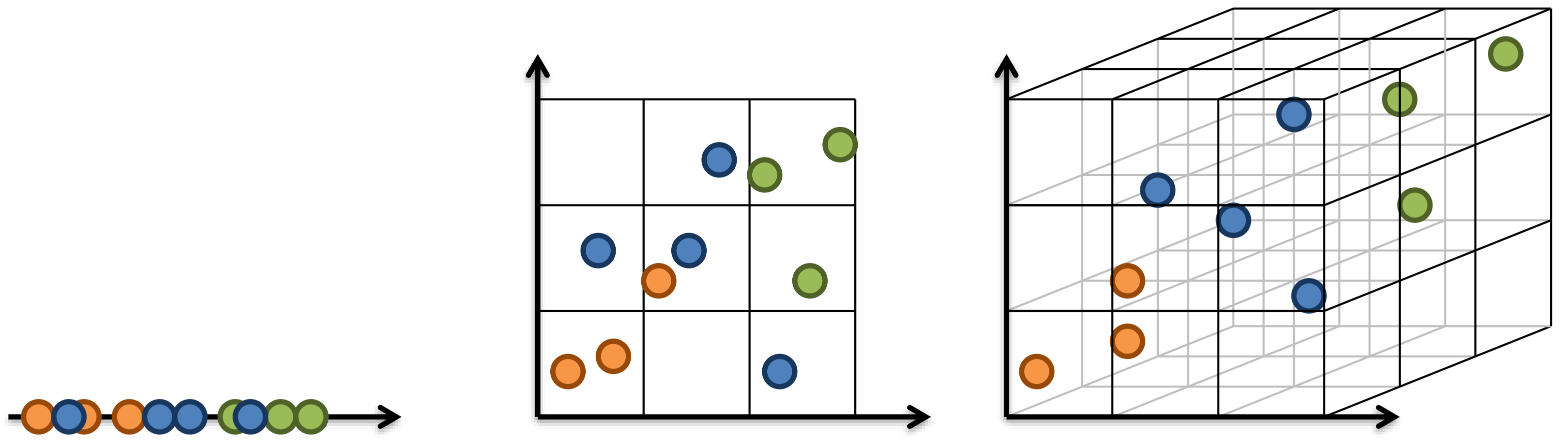


Curse of dimensionality

High dimensionality is a problem because most of the methods we have are statistical

- Count observations (frequency) in various regions of some feature space
- Use counts to construct the predictor h(x)

As dimensionality grows we have fewer observations per region



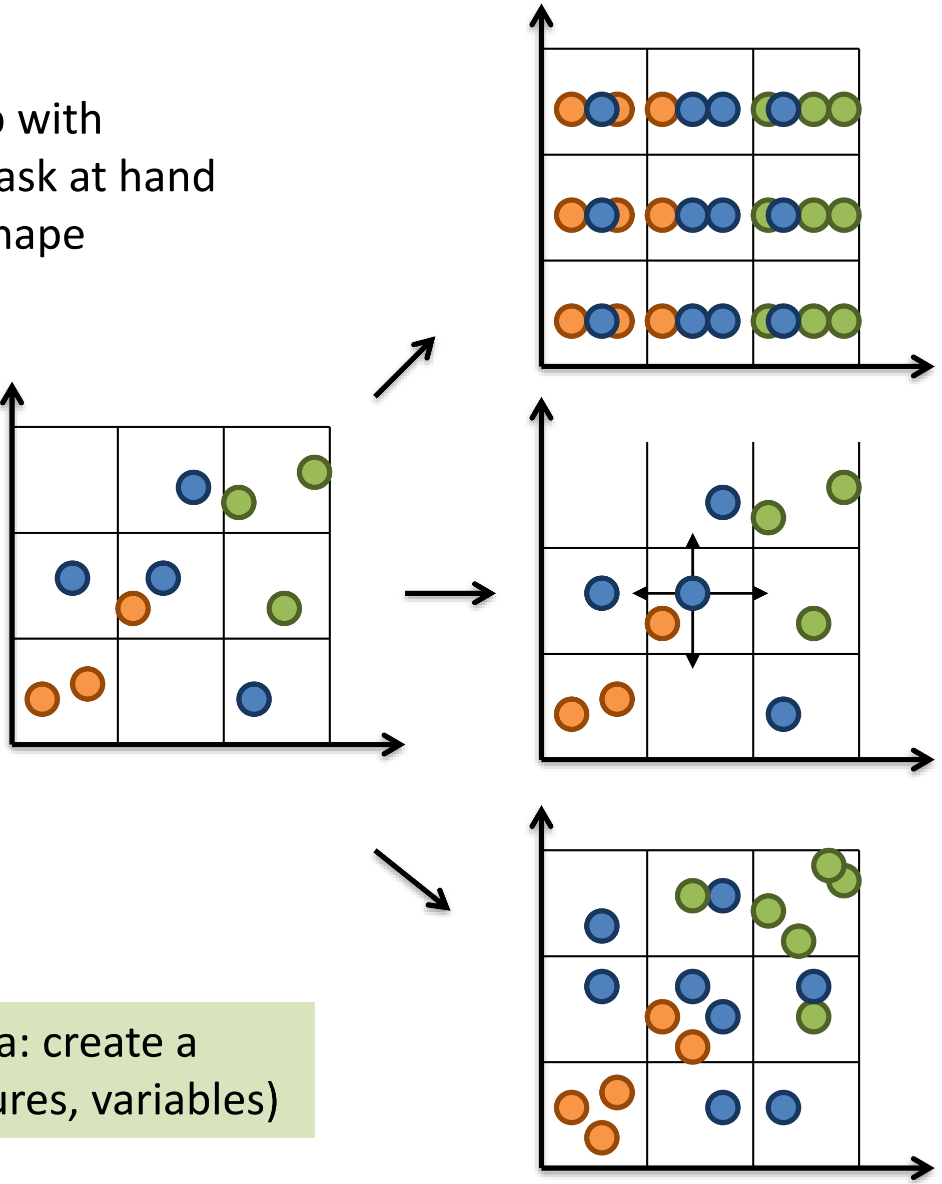
Dealing with high dimensionality

Use domain knowledge, e.g. come up with handcrafted features specific to the task at hand (e.g. in computer vision: SIFT, HOG, shape descriptors)

Make assumptions about your data

- Independence: count along each dimension separately
- Smoothness: propagate class counts to neighbouring regions
- Symmetry: e.g. invariance to order of features

Reduce the dimensionality of the data: create a new, smaller, set of dimensions (features, variables)



Dimensionality reduction

Whatever is useful for the problem at hand

Goal: represent samples with fewer features
- Try to preserve as much **structure** in the data as possible

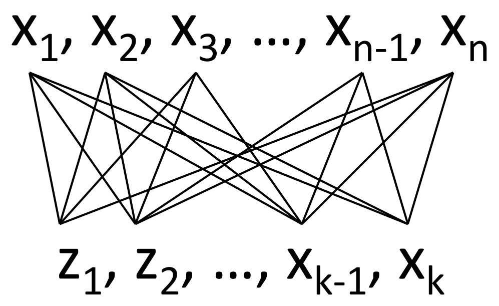
Feature selection
- Pick a subset of the original dimensions
- E.g. using information gain to decide which features to pick
- You are throwing out some of the feature

$$x_1, x_2, x_3, \dots, x_{n-1}, x_n$$

$$\Downarrow$$

$$x_1, \mathbf{x_2}, x_3, \dots, x_{n-1}, \mathbf{x_n}$$

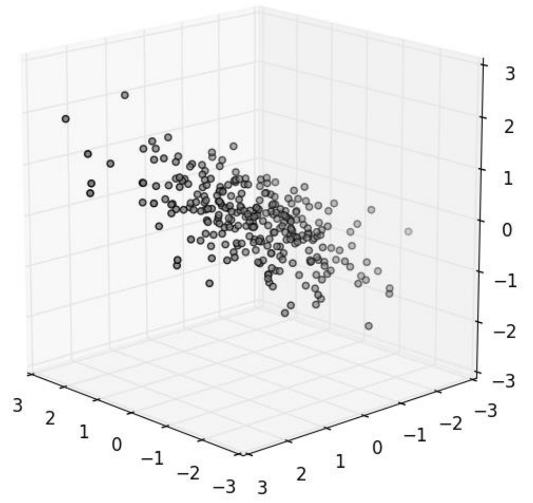
Feature extraction
- Construct a new set of *k* features (with *k* < *n*) combining existing ones
- The *i*^th feature given by: $z_i = f(x_1, x_2, \dots, x_n)$
- The easiest way is by linearly combining the original features

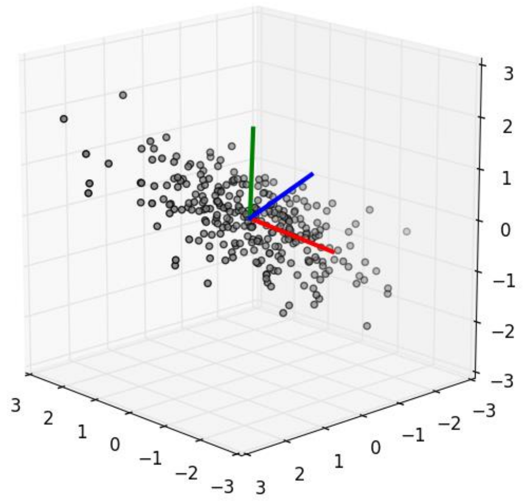

$x_1, x_2, x_3, \dots, x_{n-1}, x_n$
$z_1, z_2, \dots, x_{k-1}, x_k$

Principal Components Analysis

PCA defines a set principal components (a new set of dimensions, a new set of features)

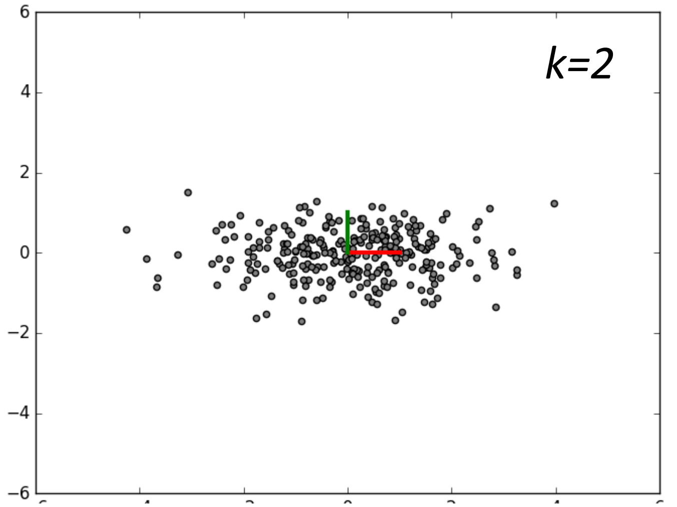
- 1st dimension: direction of the greatest variability in the data
- 2nd dimension: perpendicular to the 1st, greatest variability of what's left to explain
- 3rd dimension: perpendicular to all the previous ones, greatest variability of what's left to explain
- … and so on until n (the original dimensionality)

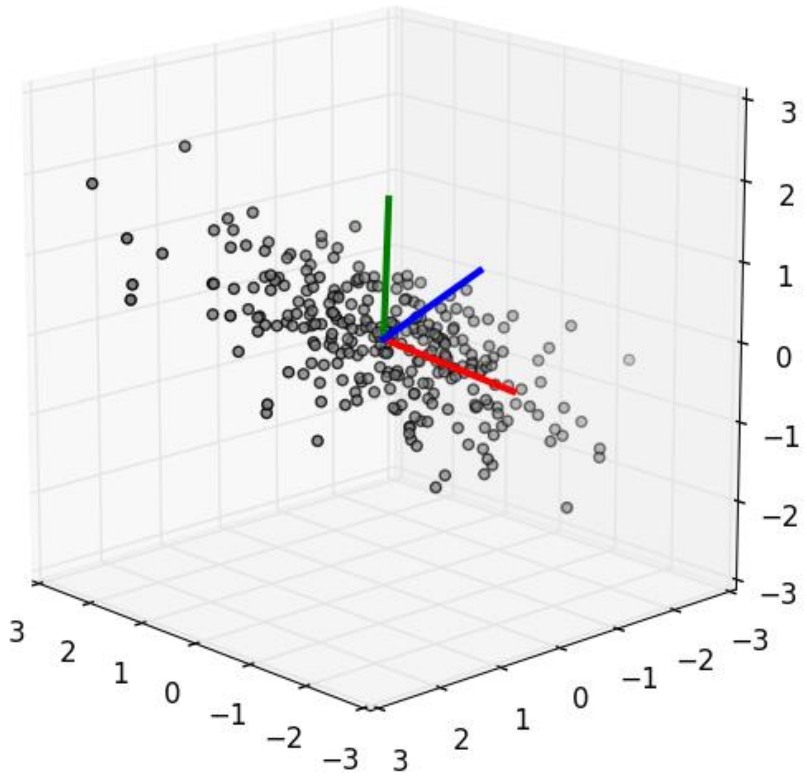


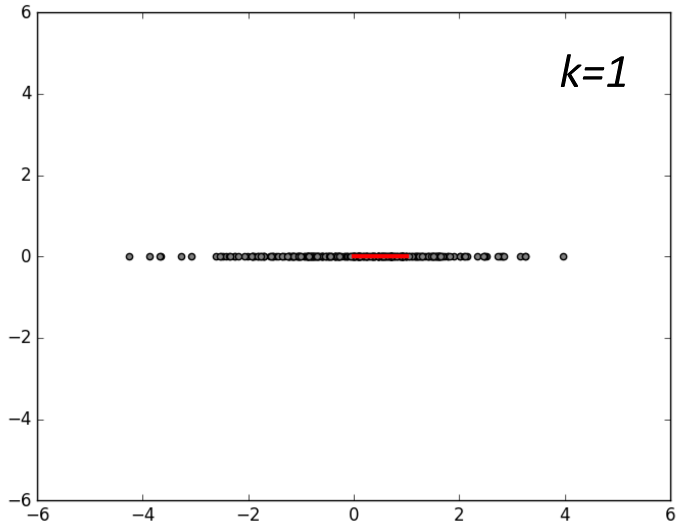


Principal Components Analysis

Once all principal components have been calculated, we select *k* << *n* components, which become the new dimensions







Projecting to a line

Note that the direction that maximises variability also minimises the distances between original points and their projections

What we are trying to do can be expressed as:

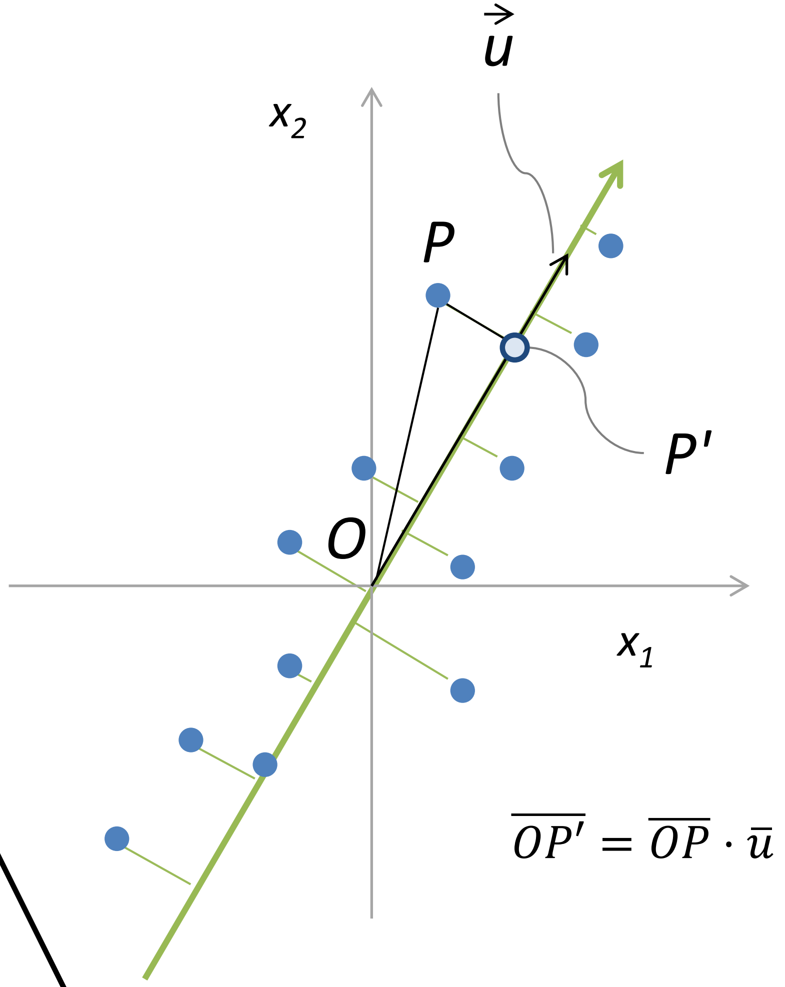
Maximise variability:

$$\max\left\{\sum_i(\overline{OP'}-\mu)^2\right\} = \max\left\{\sum_i\left(\left(x^{(i)}\right)'-\mu\right)^2\right\}$$

$$\Updownarrow$$

Minimize L2 error:

$$\min\left\{\sum_i(\overline{PP'})^2\right\} = \min\left\{\sum_i\left(x^{(i)}-\left(x^{(i)}\right)'\right)^2\right\}$$



$$\overline{OP'} = \overline{OP}\cdot\overline{u}$$

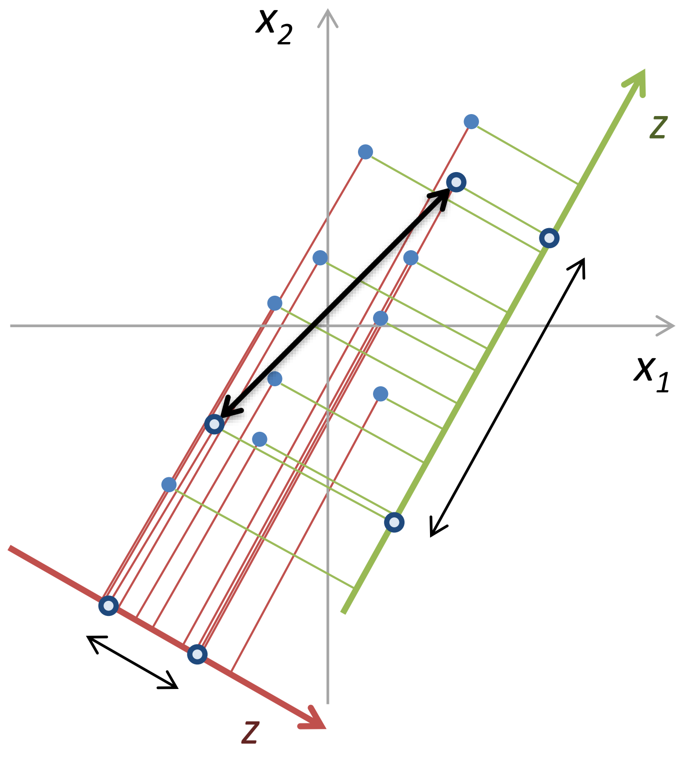
Remember we have centred the data, μ=0

Why maximum variability

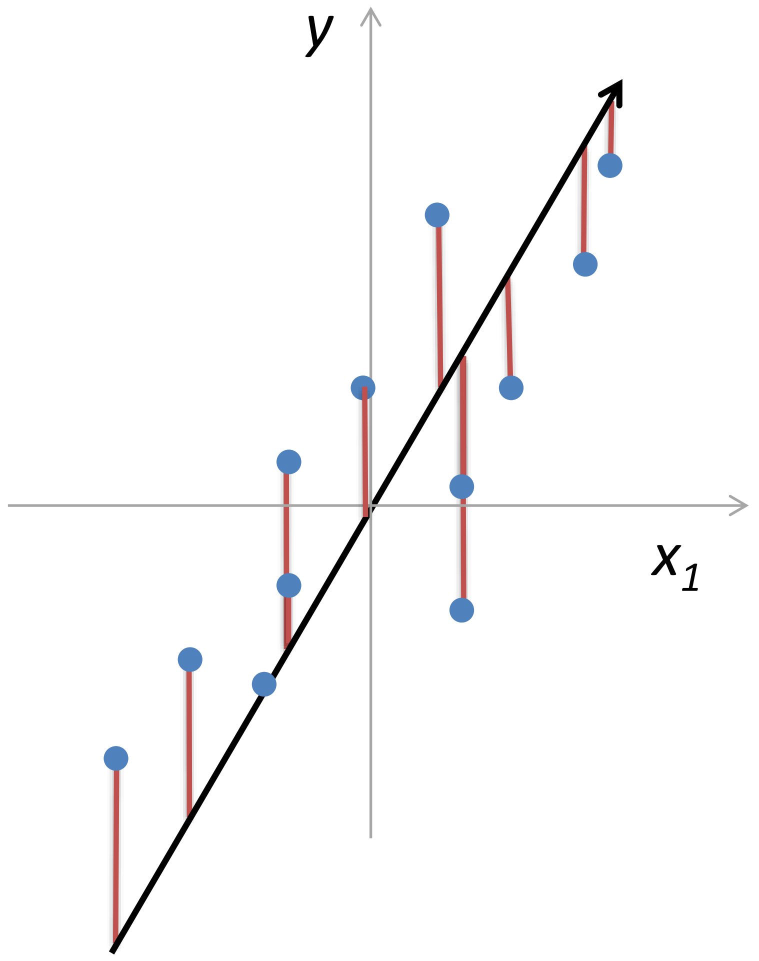
An example, reducing from $\mathbb{R}2$ to $\mathbb{R}1$

If we pick a direction *z* that maximises variability (green in the plot), it will maintain to a certain degree the relative distance between points in the original space:

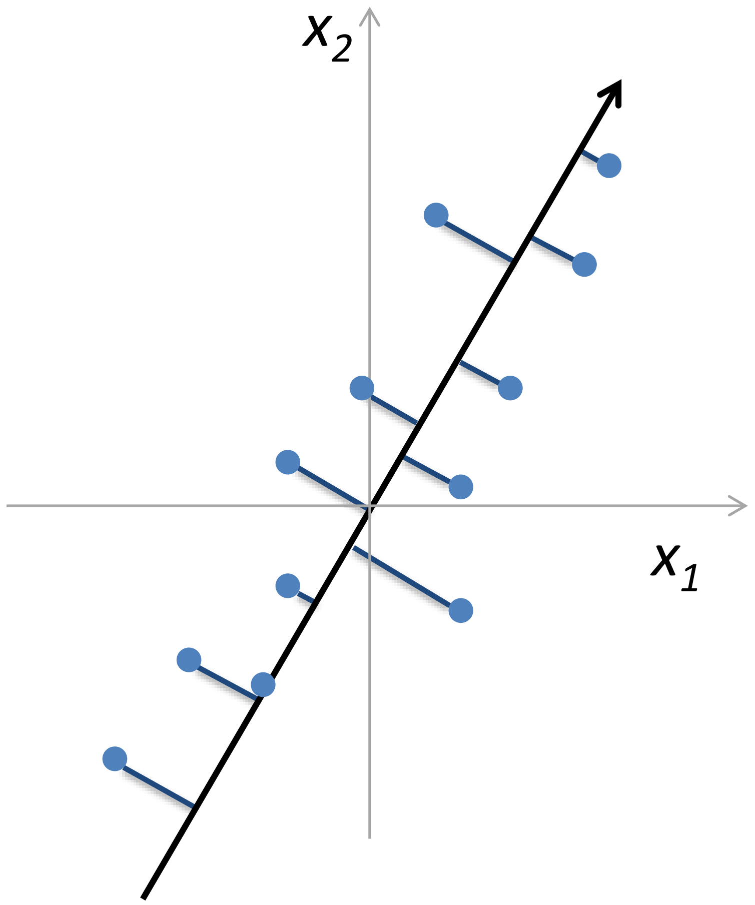
If two points are far in the original (x_1, x_2)-space, they will most probably be far in the new (z)-space



PCA is not linear regression



Linear regression



PCA

Data pre-processing

Training set: $x^{(1)}, x^{(2)}, \dots, x^{(m)}$

Centre the data at zero:

$$\mu_j = \frac{1}{m}\sum_{i=0}^{m} x_j^{(i)} \qquad x_j^{(i)} := x_j^{(i)} - \mu_j^{(i)}$$

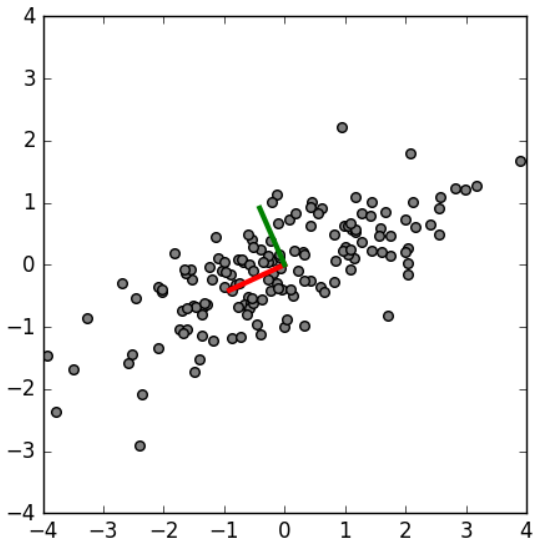
If different features are on different scales, scale features:

$$x_j^{(i)} = \frac{x_j^{(i)} - \mu_j}{s_j}$$

The covariance matrix Σ

Compute the covariance matrix Σ

Covariance of dimensions x_1 and x_2 tell you whether x_1 and x_2 tend to increase together, or does x_2 decrease as x_1 increases



$$\underbrace{\Sigma}_{n \times n} = \frac{1}{m}\sum_{i=1}^{m} \underbrace{\left(x^{(i)}\right)}_{n \times 1}\underbrace{\left(x^{(i)}\right)^T}_{1 \times n} = \underbrace{X^T}_{n \times m}\underbrace{X}_{m \times n}$$

Notice we do not subtract the mean, cause it is zero

$$\begin{array}{c|cc} & x_1 & x_2 \\ \hline x_1 & \boxed{2.0} & \boxed{0.8} \\ x_2 & \boxed{0.8} & \boxed{0.6} \end{array}$$

$$cov(x_1, x_2) = \frac{1}{m}\sum_{i=1}^{m} x_1^{(i)} x_2^{(i)}$$

$$var(x_1) = \frac{1}{m}\sum_{i=1}^{m} \left(x_1^{(i)}\right)^2$$

Principal Components

$$\Sigma = \begin{bmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{bmatrix}$$

Slope

$$\begin{bmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 1.2 \\ 0.2 \end{bmatrix}$$

0.1666

$$\begin{bmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{bmatrix} \begin{bmatrix} 1.2 \\ 0.2 \end{bmatrix} = \begin{bmatrix} 2.56 \\ 1.08 \end{bmatrix}$$

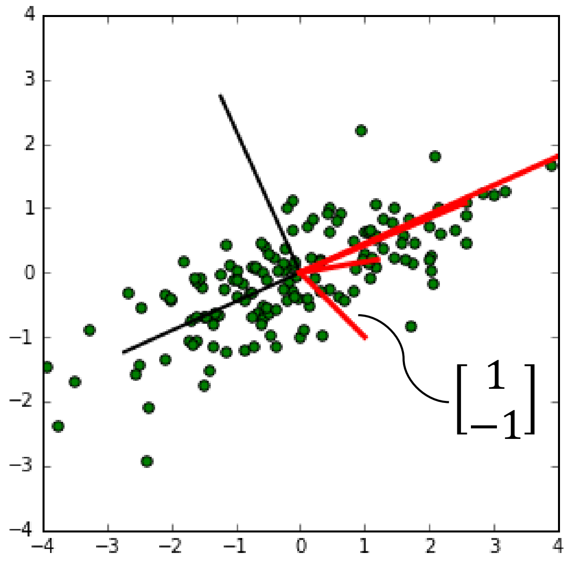
0.4218

$$\begin{bmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{bmatrix} \begin{bmatrix} 2.56 \\ 1.08 \end{bmatrix} = \begin{bmatrix} 5.984 \\ 2.696 \end{bmatrix}$$

0.4505

$$\begin{bmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{bmatrix} \begin{bmatrix} 5.984 \\ 2.696 \end{bmatrix} = \begin{bmatrix} 14.1248 \\ 6.4048 \end{bmatrix}$$

0.4534



Multiplying with the covariance matrix turns a vector towards the direction of the maximum variance… this offers a nice tool to calculate these directions

Find the vectors z which are NOT turned when multiplied with the covariance matrix: $\Sigma\mathbf{z} = \lambda\mathbf{z}$

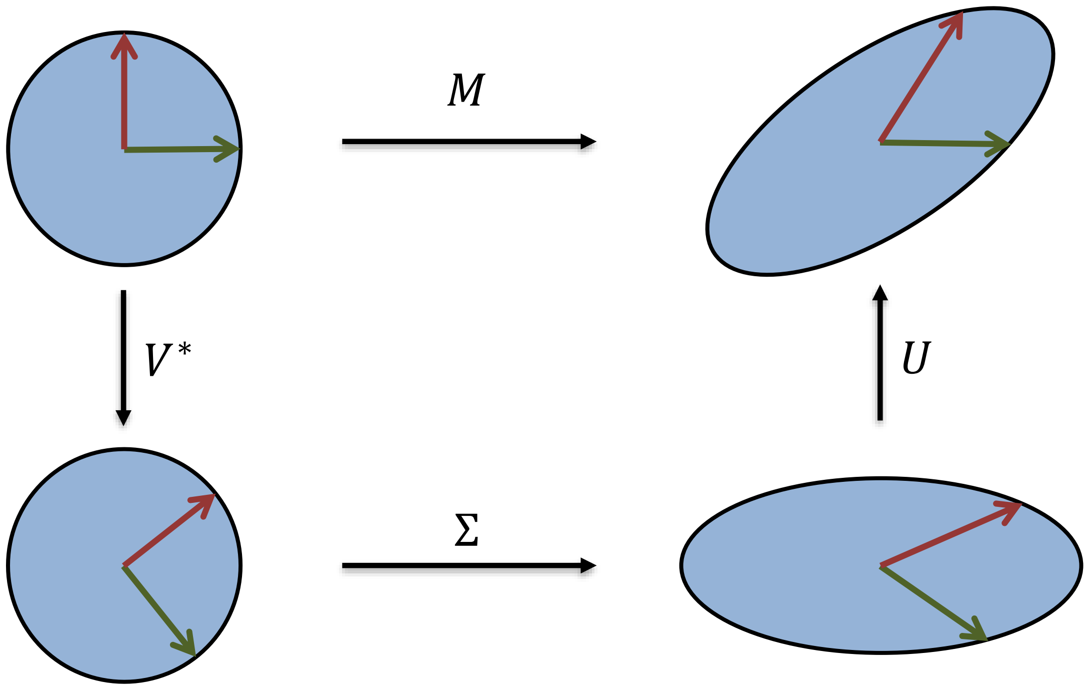
- **z** are the eigenvectors of Σ, λ are the corresponding eigenvalues
- The principal components are the eigenvalues of the covariance matrix

The SVD transform

Instead of calculating the eigenvectors we can use the Singular Value Decomposition (SVD) of the covariance matrix

For a matrix M, the SVD calculates the decomposition:

$$M = U \Sigma V^*$$



The SVD transform

For a matrix M, the SVD calculates the decomposition:

$$M = U\,\Sigma\,V^*$$

$$U = \left[\begin{matrix} \underbrace{\begin{matrix} | & | & \\ u^{(1)} & u^{(2)} & \dots \\ | & | & \end{matrix}}_{k} & \begin{matrix} | \\ u^{(n)} \\ | \end{matrix}\end{matrix}\right] \in \mathbb{R}^{n\times n}$$

$\mathrm{z} \in \mathbb{R}^k$ $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ $x \in \mathbb{R}^n$

$$z^{(i)} = \underbrace{\begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \dots & u^{(k)} \\ | & | & & | \end{bmatrix}^T}_{n\times k} \underbrace{\mathrm{x}^{(\mathrm{i})}}_{n\times 1} = \underbrace{\begin{bmatrix} - & \left(u^{(i)}\right)^T & - \\ & \vdots & \\ - & \left(u^{(k)}\right)^T & - \end{bmatrix}}_{k\times n} \underbrace{\mathrm{x}^{(\mathrm{i})}}_{n\times 1}$$

PCA Algorithm

```
U, S, V = np.linalg.svd(np.cov(samples))
U_reduced = U[:,0:2]
z = np.dot(U_reduced.T, samples)
```

Before applying PCA to a new data point, remember to first centre your data, and optionally to scale them

… the same way as you did with your training set

Choosing the k

Average squared projection error: $$\frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)} - x_{projected}^{(i)}\right\|^2$$

Total variation in the data: $$\frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)}\right\|^2$$

Typically choose k to be the smallest value so that:

$$\frac{\frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)} - x_{projected}^{(i)}\right\|^2}{\frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)}\right\|^2} \leq 0.01$$

0.99 variance retained

Choosing the k

Instead of checking for each k if:

$$\frac{\frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)}-x_{projected}^{(i)}\right\|^2}{\frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)}\right\|^2} \leq 0.01$$

we can use the SVD factorization:

```
U, S, V = np.linalg.svd(np.cov(samples))
```

$$\boxed{\begin{bmatrix} S_{11} & 0 & 0 & \dots & 0 \\ 0 & S_{22} & 0 & \dots & 0 \\ 0 & 0 & S_{33} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & S_{nn} \end{bmatrix}}$$

The eigenvalues are the variances along the corresponding eigenvector direction

$$1-\frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \leq 0.01 \rightarrow \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \geq 0.99$$

Choosing the k

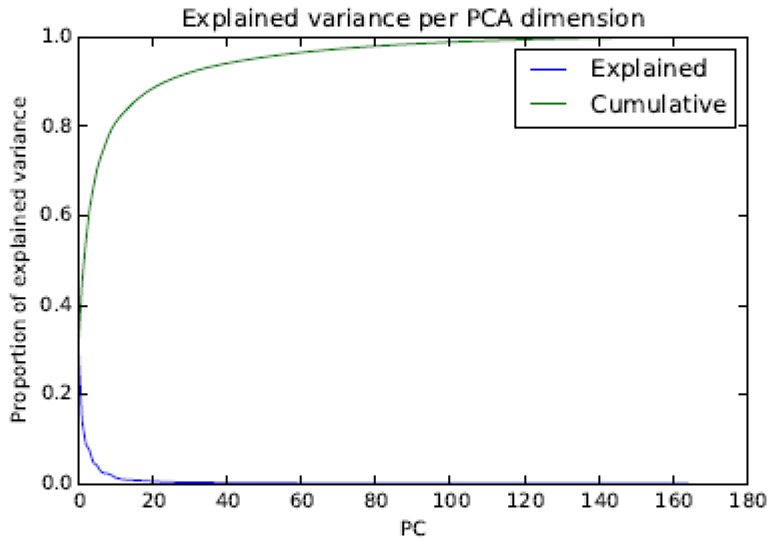
Suppose we start with very high dimensional data (n = 100k + dimensions)

How many principal directions can we have?

- In principle, we can have n eigenvectors for an $n \times n$ covariance matrix
- In practice, we cannot have more than m eigenvectors

How many principal directions do we actually need?

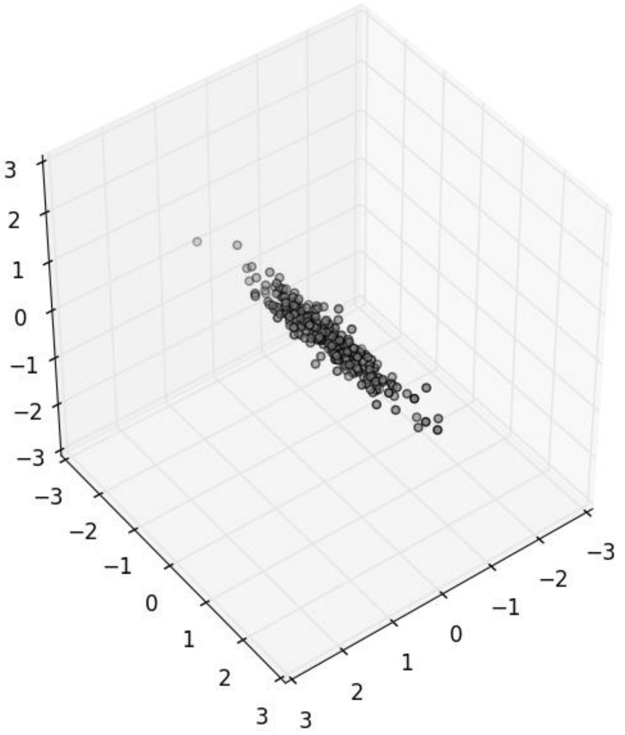
- In reality, most variance is explained by a few principal components

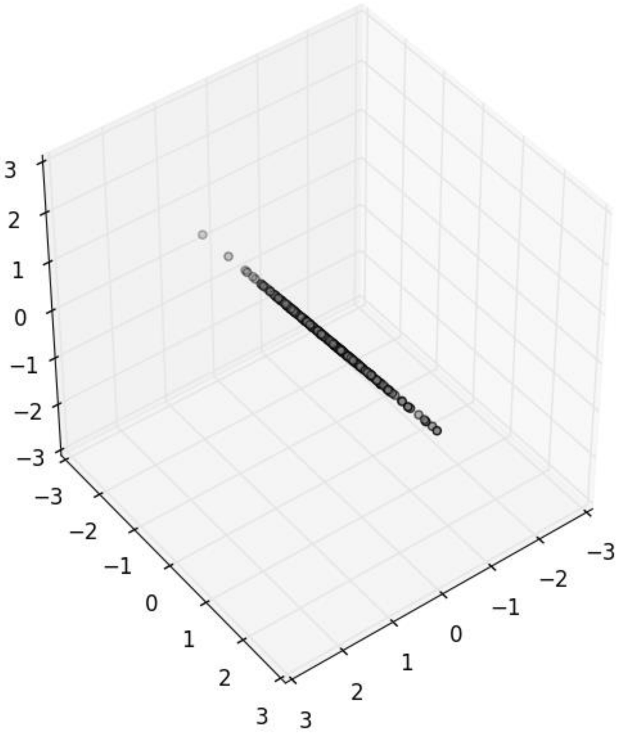


PCA algorithm summary

1. We start with correlated, high-dimensional data, $x \in \mathbb{R}^n$
2. Centre the points (optionally scale the features)
3. Computer the covariance matrix
4. Find the eigenvectors and the eigenvalues of the covariance matrix (e.g. using SVD)
5. Pick the $k \ll n$ eigenvectors with the highest eigenvalues
6. Project data points to the selected eigenvectors
7. Obtain uncorrelated low-dimensional data, $z \in \mathbb{R}^k$

Projecting Back





$$z = U_{reduced}^T \, x$$

$z \in \mathbb{R}^k$ — $k \times n$ — $x \in \mathbb{R}^n$

$$x_{approx} = U_{reduced} \, z$$

$x_{approx} \in \mathbb{R}^n$ — $n \times k$ — $z \in \mathbb{R}^k$

DEALING WITH IMAGES

Technical note: Images

We usually want for each of the features to have a similar range of values to the others (and to have a mean close to zero).

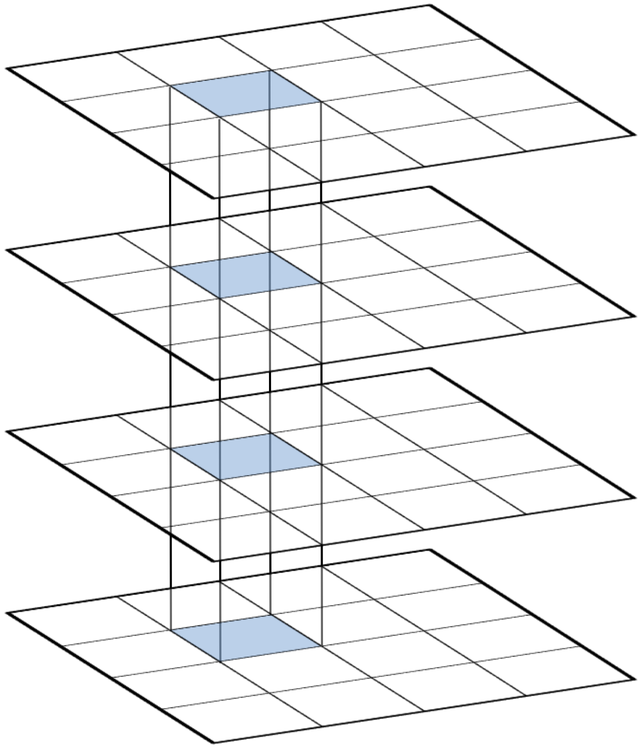
We have seen how to scale features, one by one, processing each feature separately estimating its mean and variance over the dataset

For natural images, where our features are the values of the image pixels, it makes little sense to estimate a separate mean and variance for each pixel (for each feature) because the statistics in one part of the image should (theoretically) be the same as any other. This property of images is called **stationarity**.

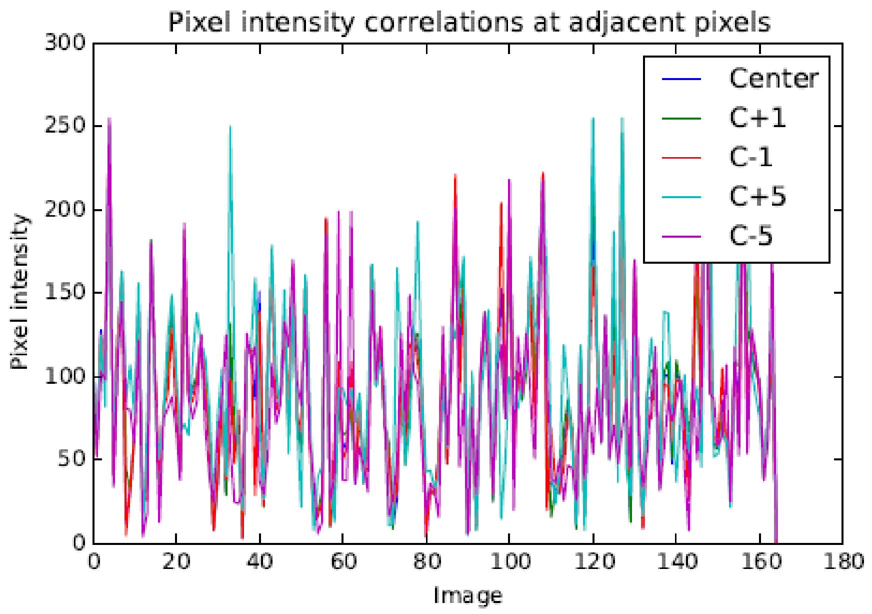
As all features share the same statistics, they are already in the same range, so no variance normalization is needed. We only need to perform mean normalization (zero centre the features)

Technical note: Images

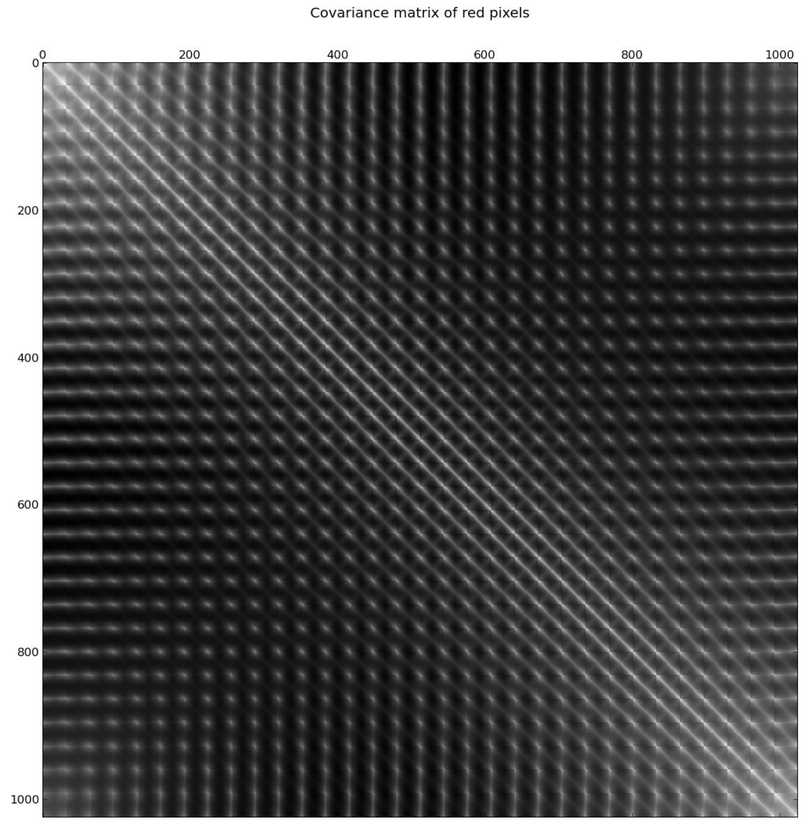
Whether the image is overall more bright or more dark is not usually relevant – we are not interested in the mean intensity of the image. We can subtract this value as a form of mean normalization. This is a per-image normalization.



Correlations between pixels



The values of the centre pixel and nearby ones across a dataset of 165 images



Covariance matrix over a dataset of 80 million tiny (32x32) images

Whitening

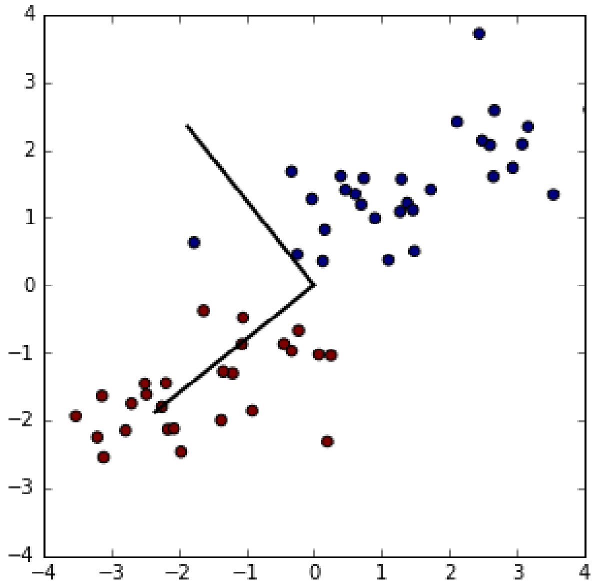
When working with images, nearby pixels are highly correlated. Whitening makes the data less redundant.

What you intuitively want is for your model to discover interesting regularities (higher-order correlations), and not get distracted by the fact that nearby pixels are correlated (merely learn that nearby pixels are similar)

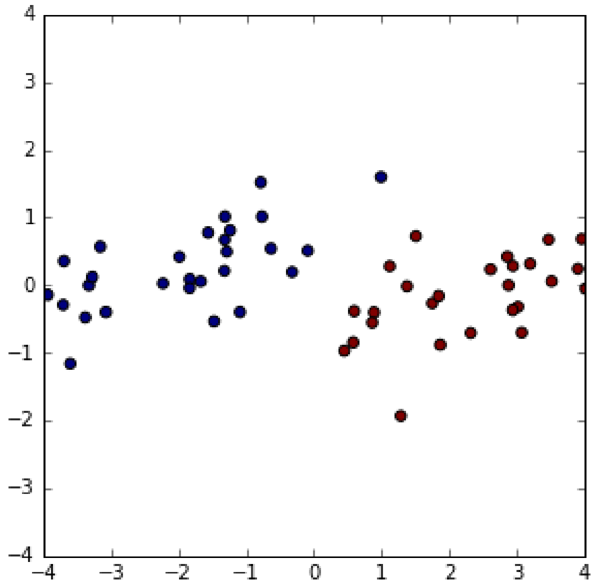
What we want from our learning algorithm at the end of the day is:

- That the features are less correlated with each other
- That the features all have the same variance

Whitening

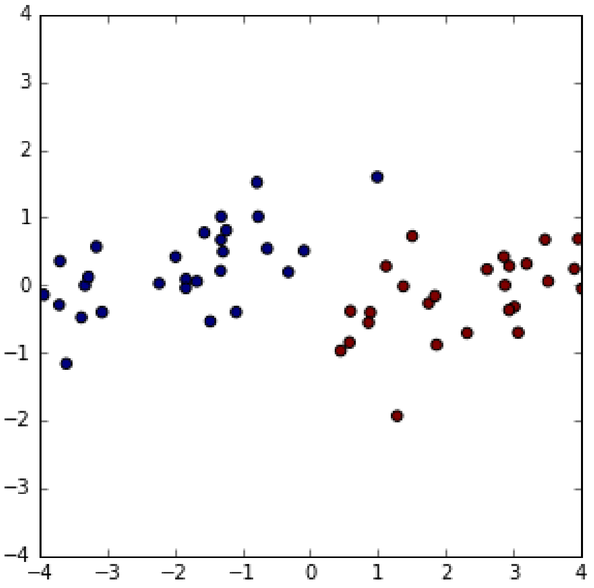


$$\Sigma = \begin{bmatrix} 3.6 & 2.2 \\ 2.2 & 2.0 \end{bmatrix}$$



$$\Sigma = \begin{bmatrix} 5.2 & 0.0 \\ 0.0 & 0.4 \end{bmatrix}$$

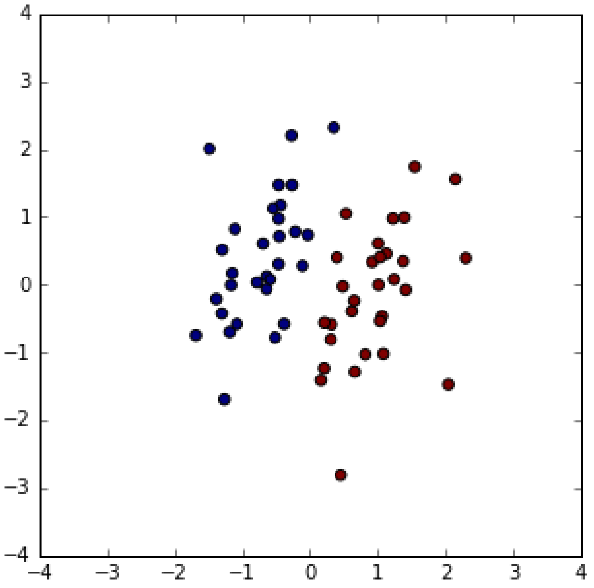
Whitening



$$\Sigma = \begin{bmatrix} 5.2 & 0.0 \\ 0.0 & 0.4 \end{bmatrix}$$

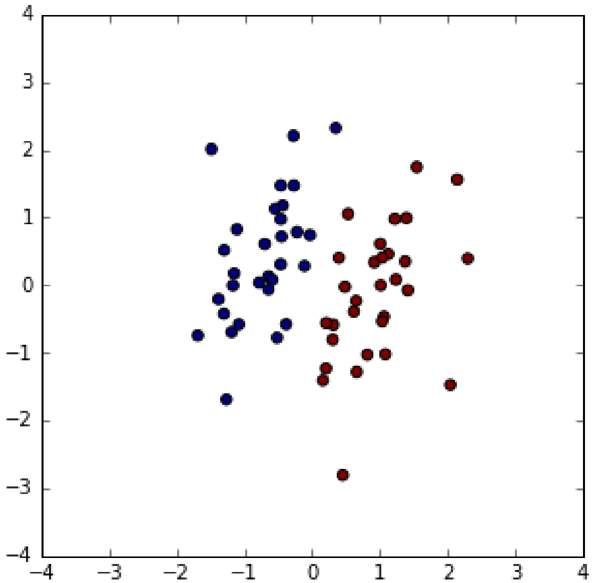
To make each of our input features have unit variance, we can simply rescale each feature z_i by $1/\lambda_i$

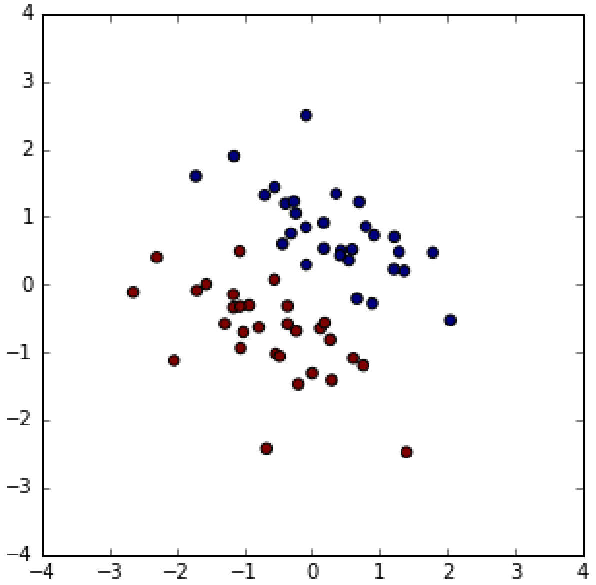
$$x_{PCAwhite,i} = \frac{x_{PCA,i}}{\sqrt{\lambda_i}}$$



$$\Sigma = \begin{bmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \end{bmatrix}$$

ZCA Whitening





$$\Sigma = \begin{bmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \end{bmatrix}$$

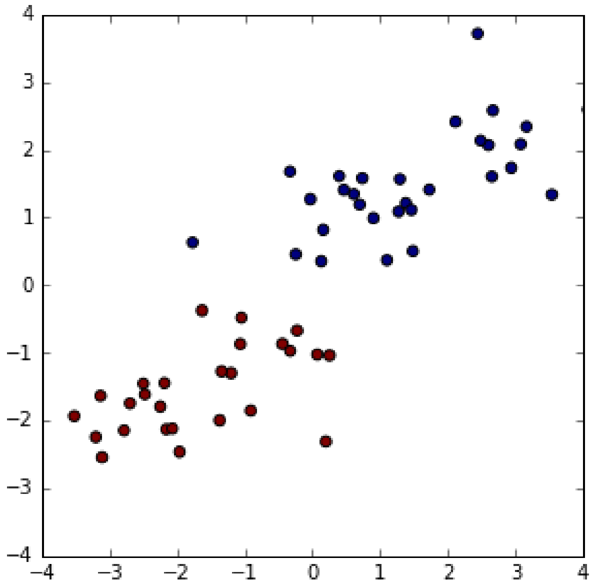
Problem: there is no unique way to get to this result. Any rotation would also have the same identity covariance

Formally: if R is an orthogonal matrix that satisfies $RR^T = R^T R = I$ (a rotation/reflection matrix), then $Rx_{PCAwhite}$ will also have identity covariance.

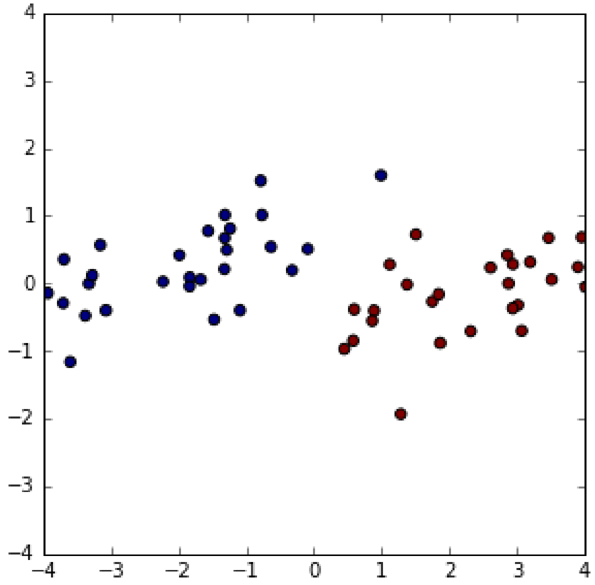
What is the "*right*" / "*best*" way to rotate?

For ZCA whitening we choose $R = U$ (remember SVD?), so it is $x_{ZCAwhite} = Ux_{PCAwhite}$

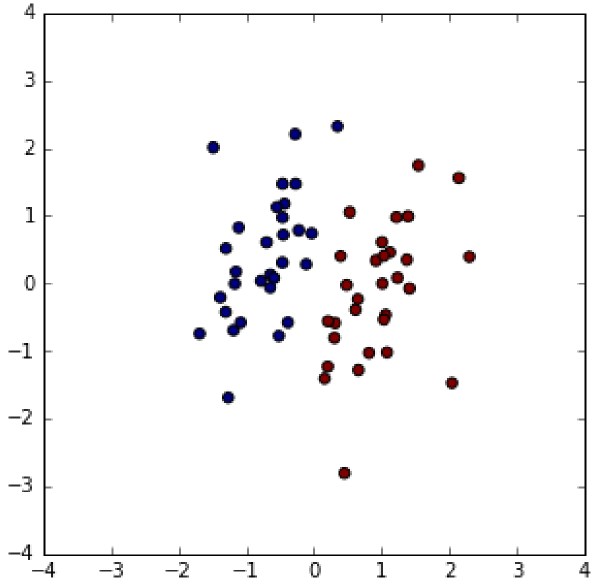
ZCA Whitening



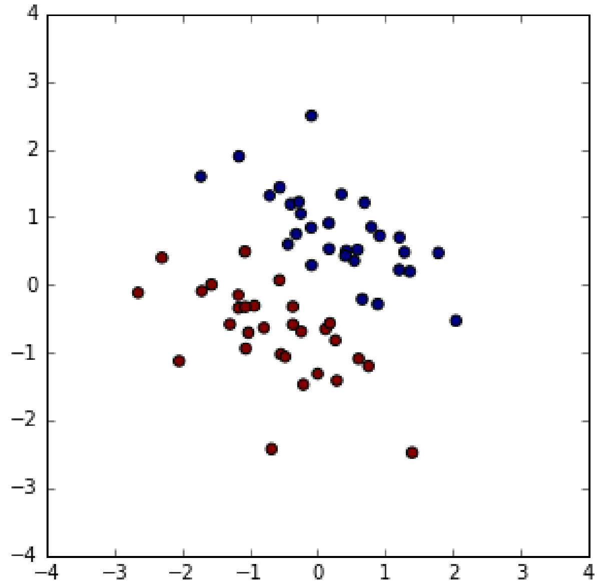
Original data



PCA



PCA Whitened

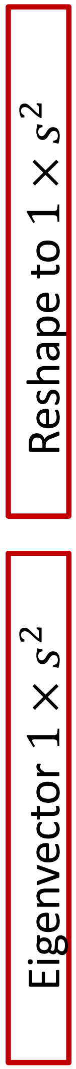


ZCA whitened

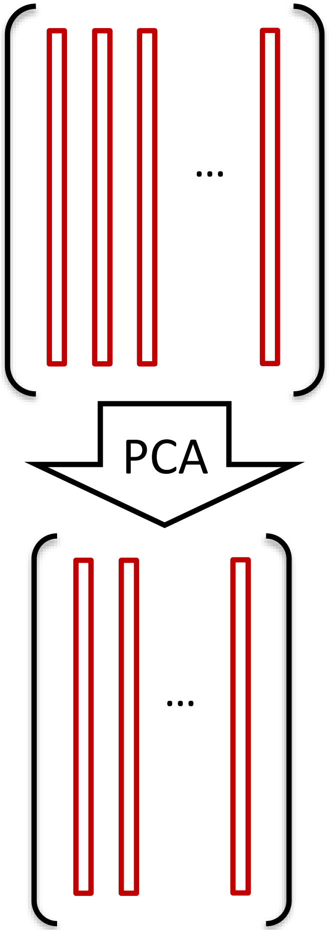
An example: eigenfaces

Input: dataset of m face images

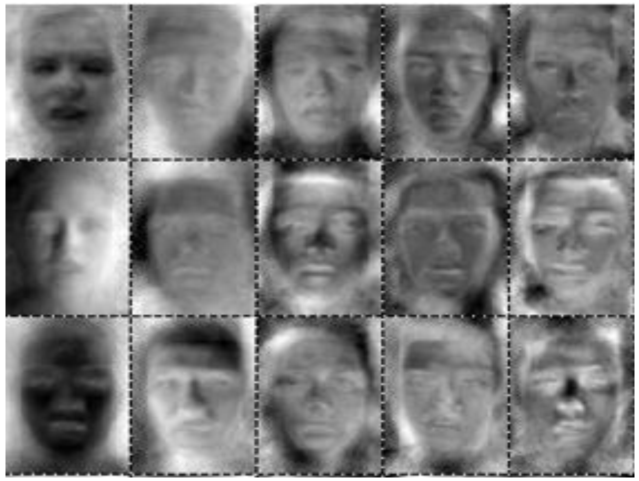
Face: $s \times s$ bitmap



Arrange in a matrix



Set of k eigenvectors



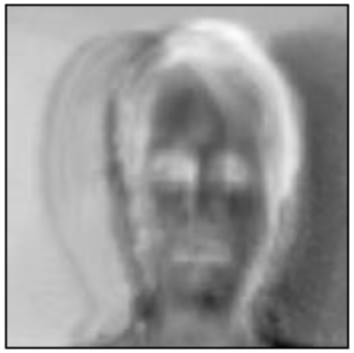
Eigenvectors can be visualised

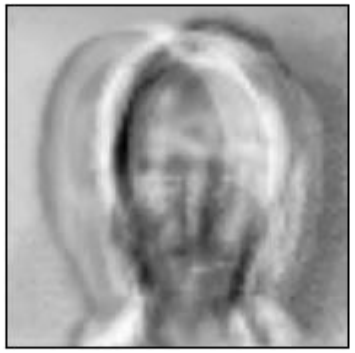
Reshape into a $s \times s$ bitmap

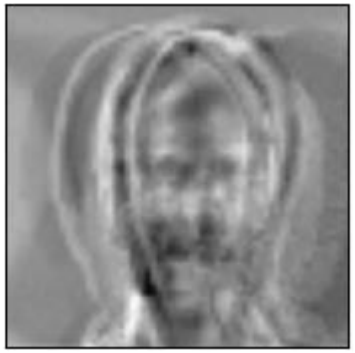
Zero-phase Component Analysis

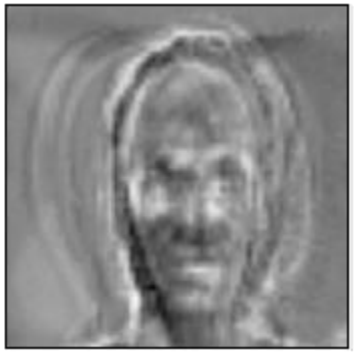
In a sense, ZCA components are “faces” more than PCA components

They tend to preserve edges and smooth constant areas


ZC=10

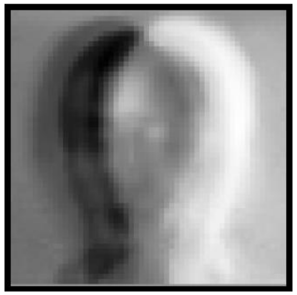

ZC=20


ZC=30

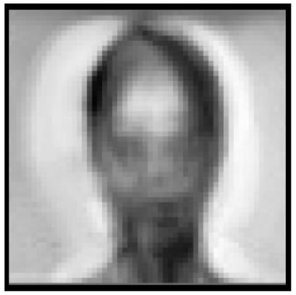

ZC=50

An example: eigenfaces

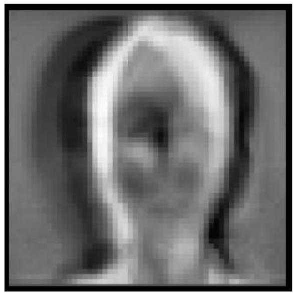
=mean+0.7*



-0.3*



+0.5*

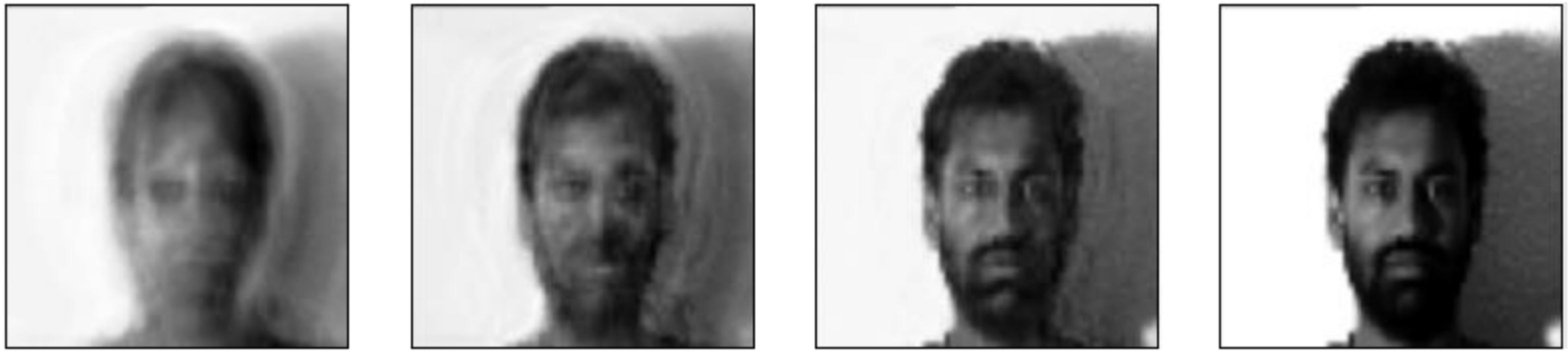


+...

Project new face to space of eigen-faces

Represent vector as a linear combination of principal components

How many do we need?



10 PCs 50 PCs 100 PCs 165 PCs

WHEN NOT TO USE PCA

Practical issues

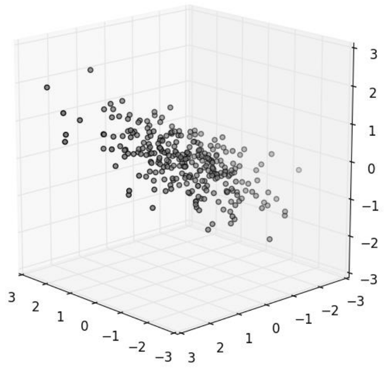
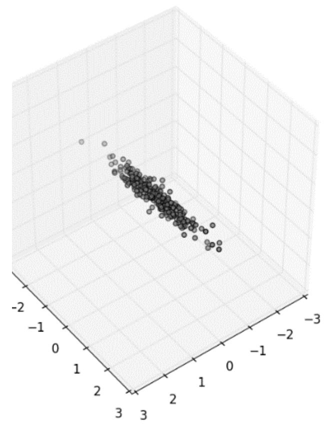
Covariance is extremely sensitive to large values

- One way to mess up with the PCA is to multiply a dimension by 1000
- This dimension will then dominate covariance
- It will become a principal component
- Good practice: centre and scale your data

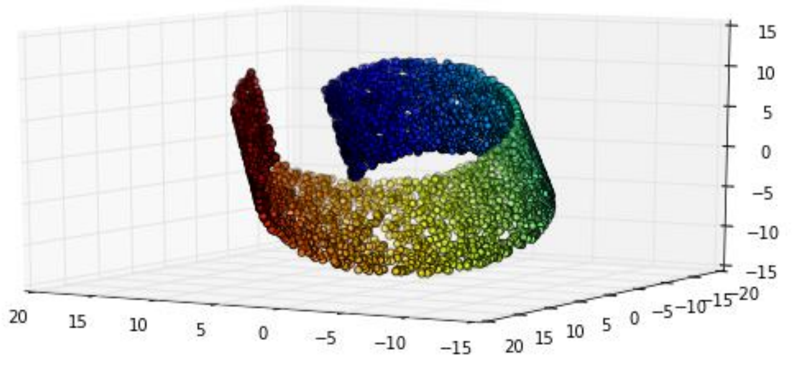
$$x_j^{(i)} = \frac{x_j^{(i)} - \mu_j}{s_j}$$

PCA assumes underlying subspace is linear

- Good practice: transform to handle non-linear spaces (manifolds)

vs



PCA should not be used to prevent overfitting

Common thought: Use $z^{(i)}$ instead of $x^{(i)}$ to reduce the number of features from n to $k \ll n$. This gets us fewer features, thus it is less likely to overfit…

This might work, but is not a way to address overfitting. The reason is that PCA does NOT take into account the labels of the dataset, just the feature values. Use regularization instead:

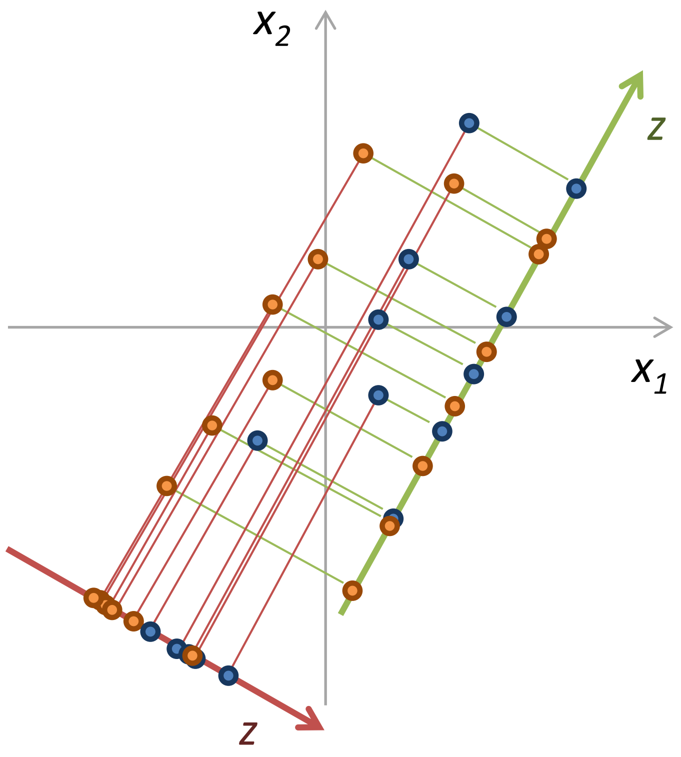
$$\mathcal{L}(\mathbf{w}; \mathbf{X}, \mathbf{y}, \phi) = \sum_{i=1}^{m} L\left(\phi\left(\mathbf{x}^{i}\right)^{T}\mathbf{w}, y^{i}\right) + \lambda(\mathbf{w}^{\boldsymbol{T}}\mathbf{w})$$

PCA and classification

PCA can sometimes hurt instead of help, as it does not take into account the class labels

PCA is unsupervised
- Maximises overall variance along a small set of directions
- Does not know anything about class labels

Discriminative approach
- Look for a dimension that makes it easy to separate classes



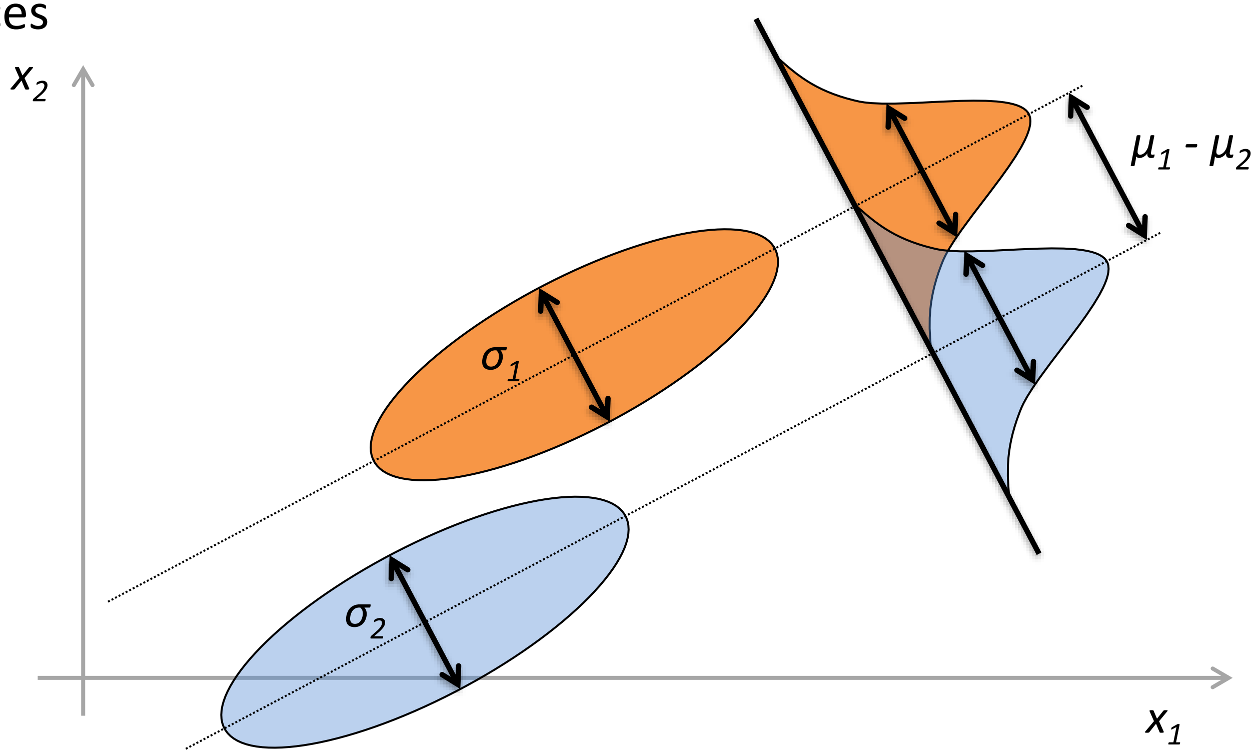
Linear Discriminant Analysis

LDA picks a new dimension that gives:

- Maximum separation between means of projected classes
- Minimum variance within each projected class

Solution: eigenvectors based on between-class and within-class covariance matrices

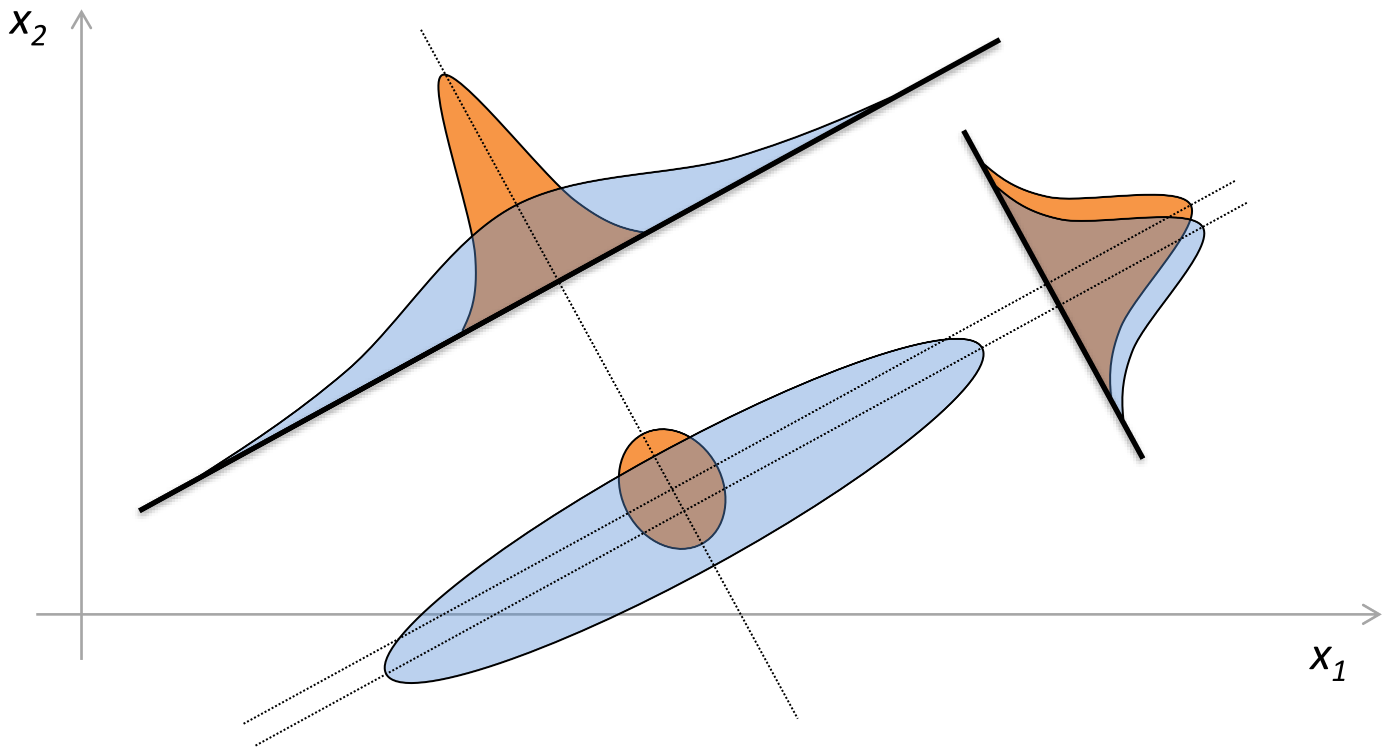
$$\max \frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2}$$



Linear Discriminant Analysis

LDA is not guaranteed to be better for classification

- Assumes that distributions are unimodal Gaussians
- Assumes that they are separable
- Fails when the discriminatory information is not in the mean but in the variance of the data



Summary

Data in high dimensions is typically highly correlated

This means that the data lives in a lower-dimensional sub-space, possibly much lower dimensional than the original one

Principal Component Analysis (PCA) is a subspace method that identifies the principal axes of variation in a sample

It can be used to decorrelate and to reduce dimensionality of the data

Whitening is the process of equalizing variance in all directions

Zero-phase Component Analysis (ZCA) is often used on image data since its components can be interpreted as images

ZCA-whitening rotates data back into the original directions so that each component is as similar as possible to the original data

Linear Discriminant Analysis takes into account the class labels, and finds directions that maximise separation between means of projected classes, while minimising variance within each projected class

LDA is generally better for classification (when unimodal Gaussian distributions can be assumed and the discriminatory information is in the means)

What’s Next

		Mondays	Tuesdays				
		16:00 - 18:00	15:00 - 17:00				
Practical Sessions		**M**	**T**	**W**	**T**	**F**	**Lectures**
	Feb	8	9	10	11	12	Introduction and Linear Regression
P0. Introduction to Python, Linear Regression		15	16	17	18	19	Logistic Regression, Normalization
P1. Text non-text classification (Logistic Regression)		22	23	24	25	26	Regularization, Bias-variance decomposition
	Mar	29	1	2	3	4	Normalization and subspace methods (dimensionality reduction)
		7	8	9	10	11	Probabilities, Bayesian inference
Discussion of intermediate deliverables / project presentations		14	15	16	17	18	Parameter Estimation, Bayesian Classification
		21	22	23	24	25	Easter Week
	Apr	28	29	30	31	1	Clustering, Gausian Mixture Models, Expectation Maximisation
P2. Feature learning (k-means clustering, NN, bag of words)		4	5	6	7	8	Nearest Neighbour Classification
		11	12	13	14	15	
		18	19	20	21	22	Kernel methods
Discussion of intermediate deliverables / project presentations		25	26	27	28	29	Support Vector Machines, Support Vector Regression
P3. Text recognition (multi-class classification using SVMs)	**May**	2	3	4	5	6	Neural Networks
		9	10	11	12	13	Advanced Topics: Metric Learning, Preference Learning
		16	17	18	19	20	Advanced Topics: Deep Nets
Final Project Presentations		23	24	25	26	27	Advanced Topics: Structural Pattern Recognition
	Jun	30	31	1	2	3	Revision

LEGEND	
🟦	Project Follow Up
▨	Project presentations
🟩	Lectures
🟪	Project Deliverable due date
🟥	Vacation / No Class