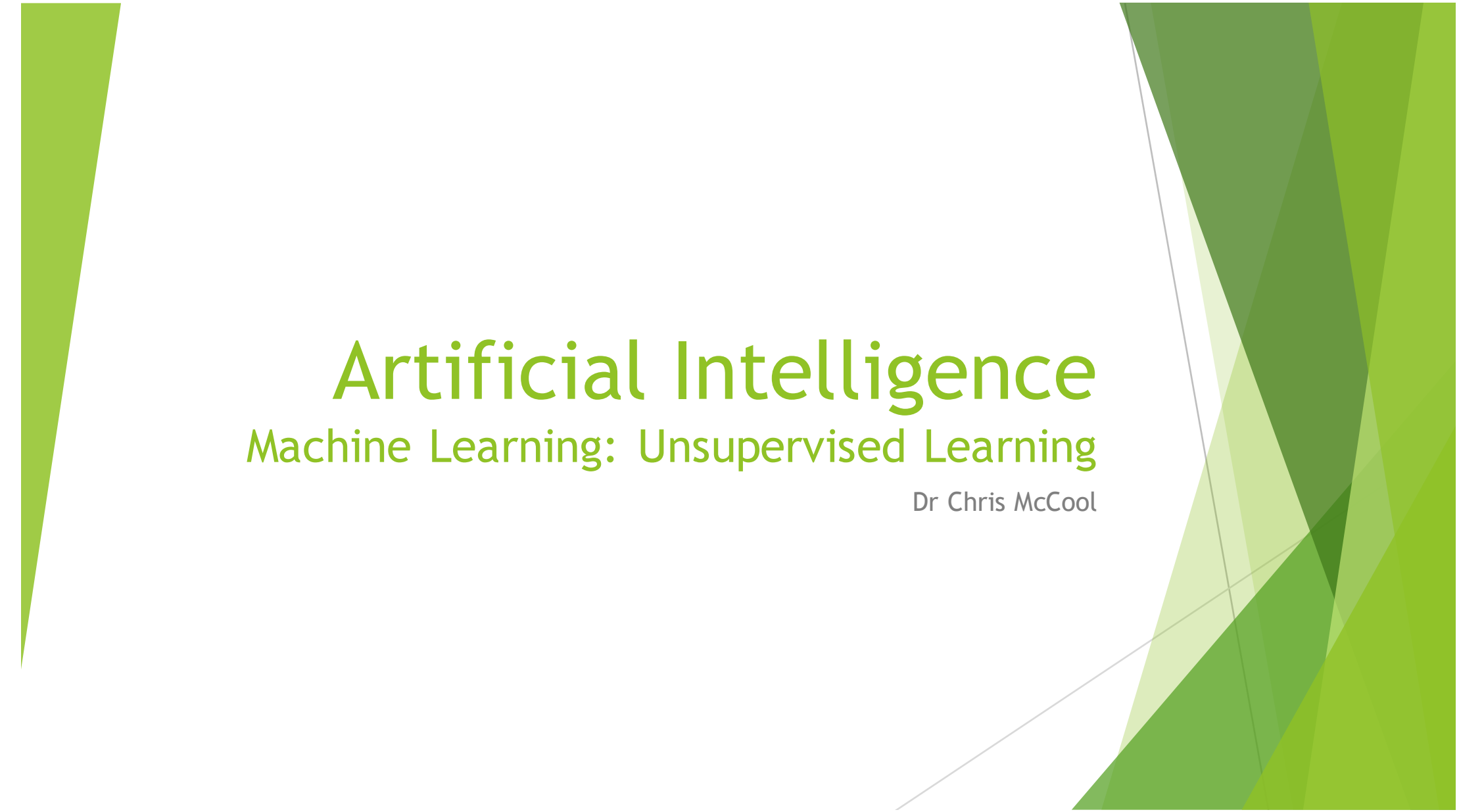
Artificial Intelligence

Machine Learning: Unsupervised Learning

Dr Chris McCool

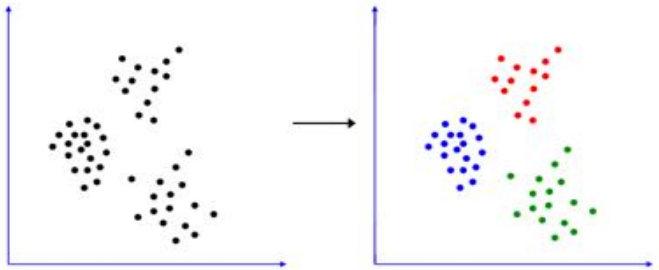
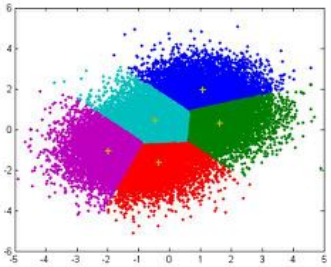


Overview

- Introduction
  - Unsupervised Approaches vs Supervised Approaches
- Clustering
  - K-means
  - Mixture of Gaussians
- Principal Component Analysis

Introduction

- **Unsupervised Learning**
  - Is the problem of trying to find hidden structure in unlabelled data. “See what you can find”


- **Supervised Learning**
  - Is the task of inferring a function (classifier) from labelled training data. “Learn from my examples”

Training data:



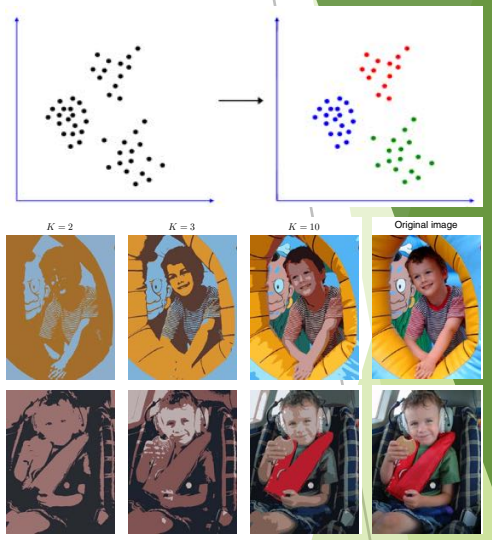
Testing:

Labels: Dog Cat Horse ?

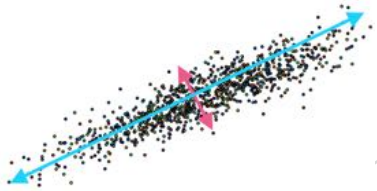
Introduction

Unsupervised Learning

- Clustering: Groups related data together
  - Image Segmentation
  - Finding groups of customers with similar behaviours
  - Social network analysis
  - Search results grouping

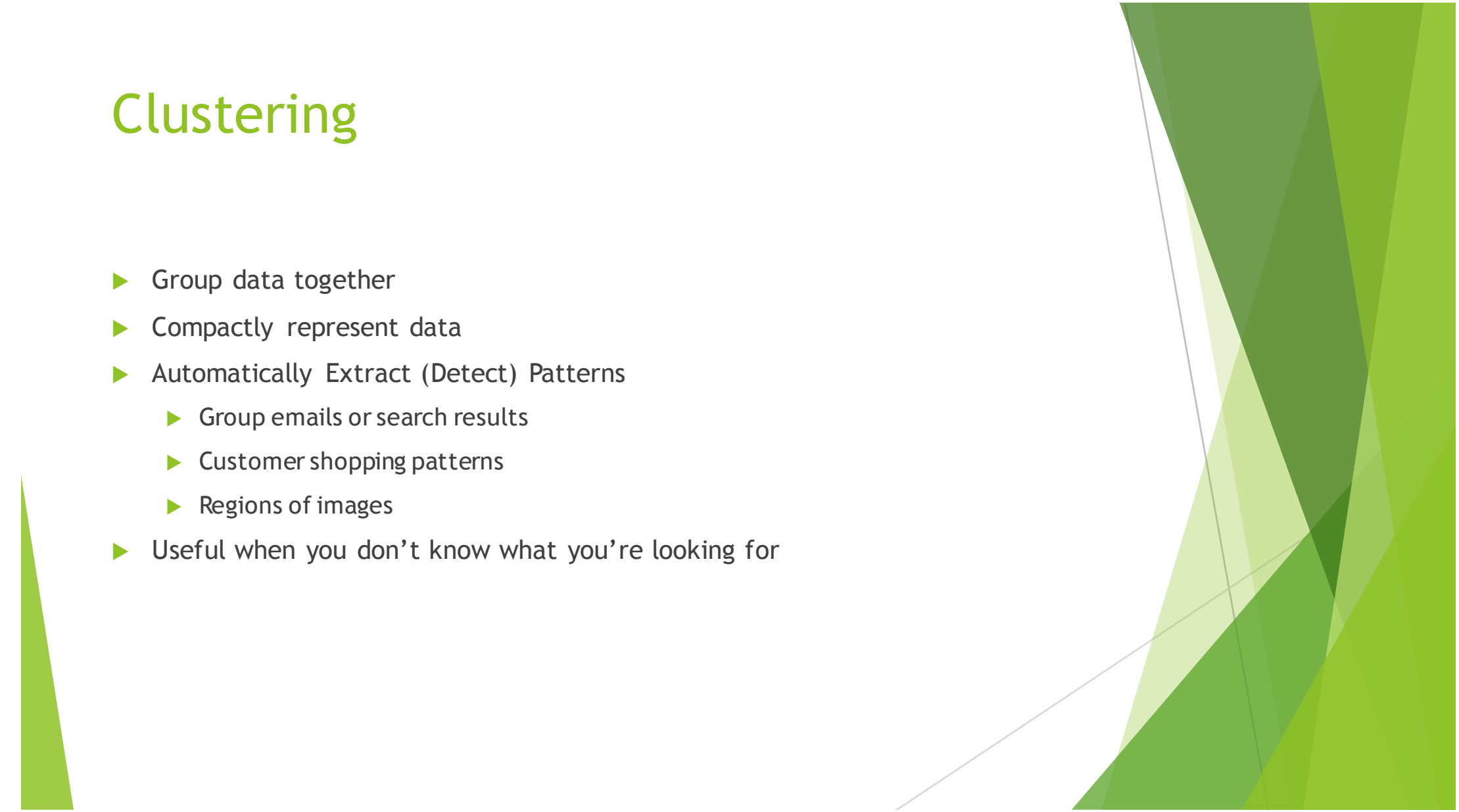


- Principal Component Analysis
  - Finding principal axes of variation
  - Dimensionality reduction



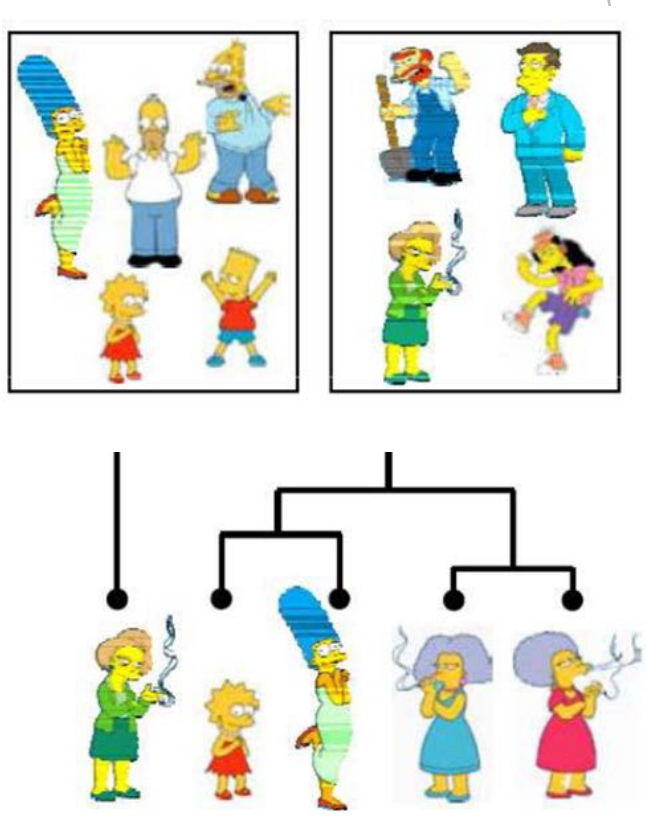
Clustering

- Group data together
- Compactly represent data
- Automatically Extract (Detect) Patterns
  - Group emails or search results
  - Customer shopping patterns
  - Regions of images
- Useful when you don’t know what you’re looking for



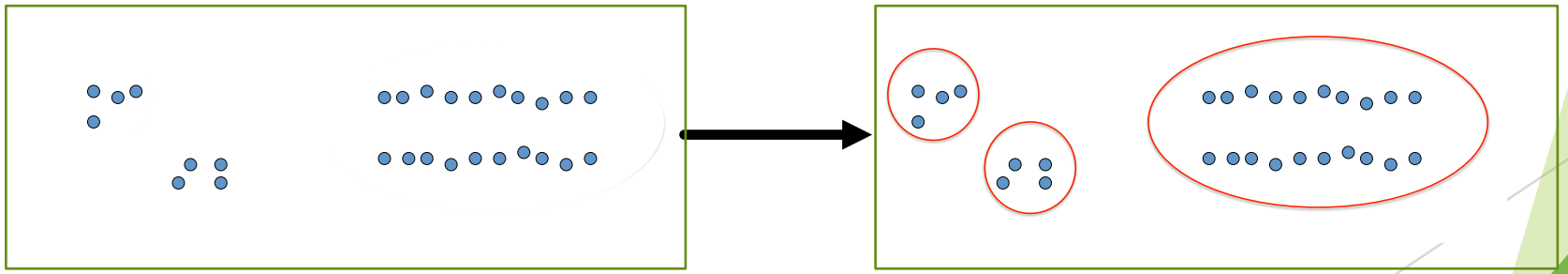
Clustering

- Partitioning (Flat)
  - K-means
  - Mixture of Gaussians
  - Spectral Clustering
- Hierarchical Algorithms
  - Bottom-up (Agglomerative)
  - Top-down (Divisive)



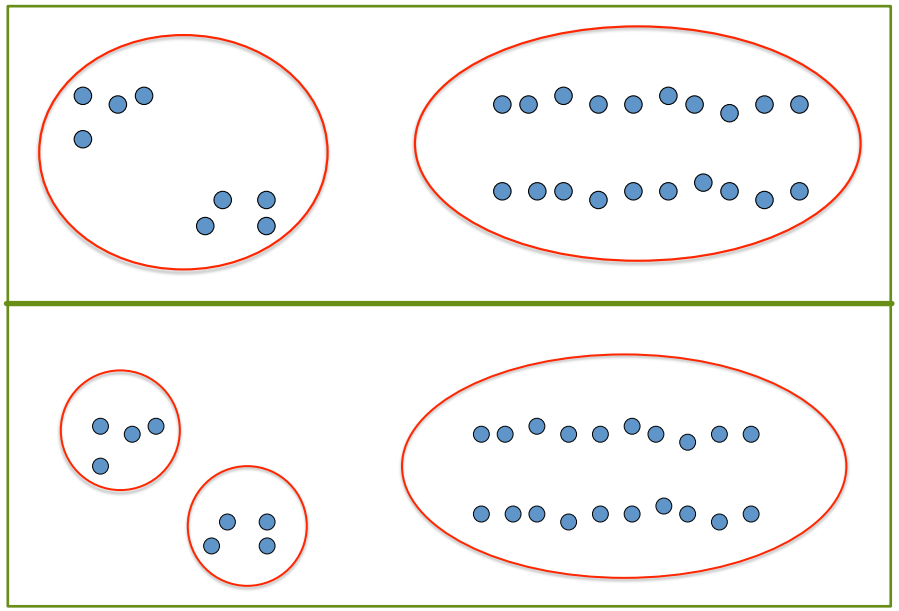
Clustering

- Group similar instances together
- Example: using 2D points



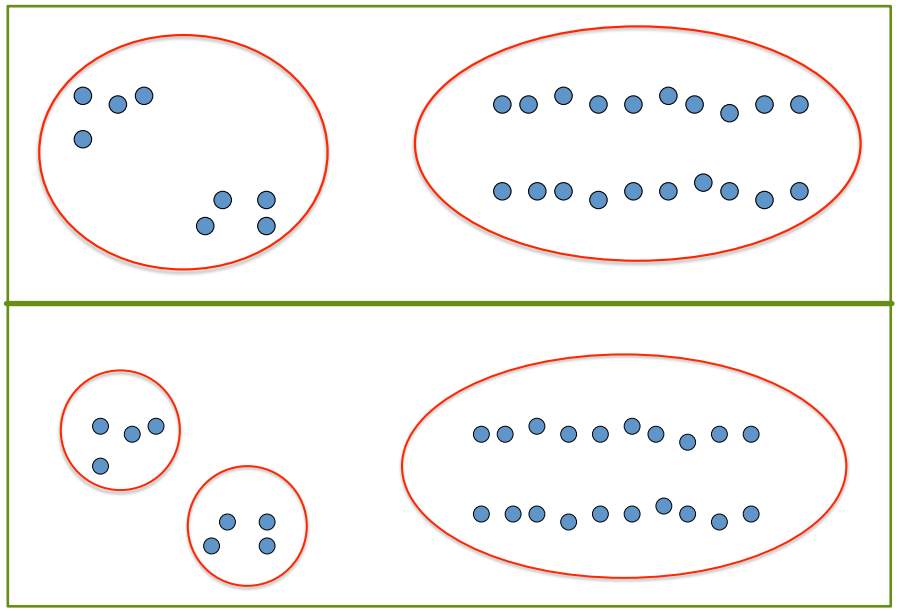
Clustering

- Group similar instances together
- Example: using 2D points
- The result is highly dependent on how we measure the *similarity* of the points



Clustering

- What is a good result for clustering?
  - Internal (within the cluster) distances should be small
  - External (intra-cluster) distances should be large
- Clustering is a way to discover new categories

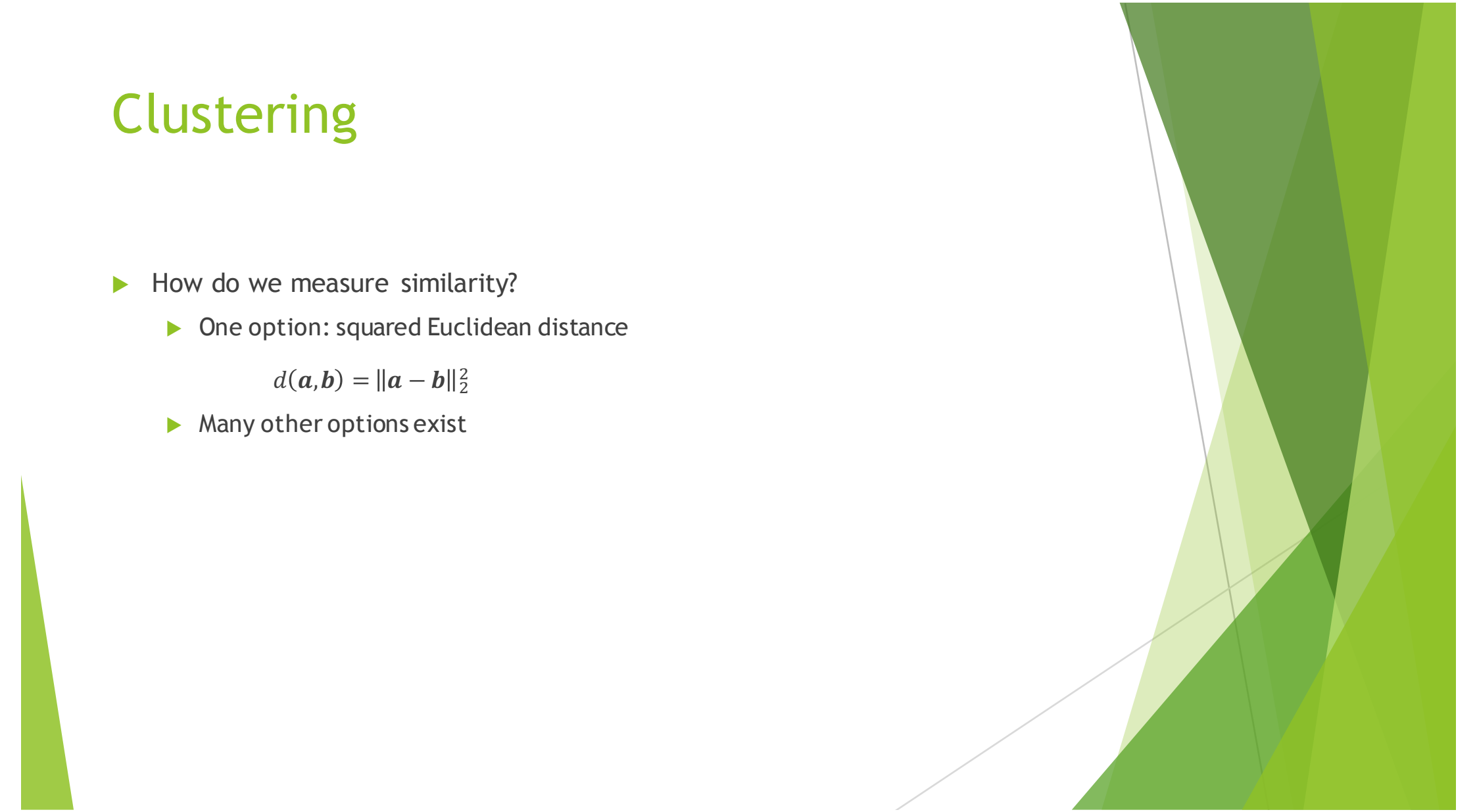


Clustering

- How do we measure similarity?
  - One option: squared Euclidean distance

$$d(\boldsymbol{a}, \boldsymbol{b}) = \|\boldsymbol{a} - \boldsymbol{b}\|_2^2$$

  - Many other options exist



Clustering

- Euclidean distance

  $$d(\boldsymbol{a},\boldsymbol{b}) = \sqrt{\sum_{d=1}^{D}(\boldsymbol{a}_d - \boldsymbol{b}_d)^2}$$

  - Translation invariant

- Manhattan (city block) distance

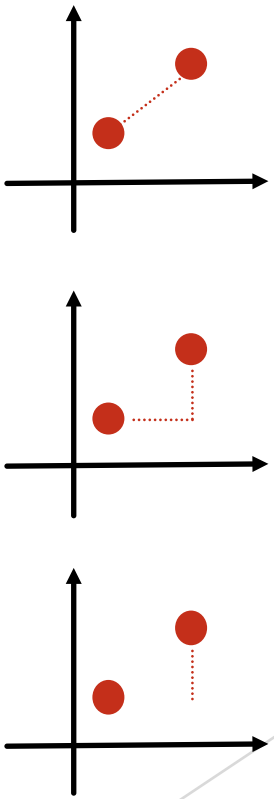
  $$d(\boldsymbol{a},\boldsymbol{b}) = \sum_{d=1}^{D}|\boldsymbol{a}_d - \boldsymbol{b}_d|$$

  - Approximation to Euclidean distance, cheaper to compute

- Chebyshev distance

  $$d(\boldsymbol{a},\boldsymbol{b}) = \max_{1\le d\le D}|\boldsymbol{a}_d - \boldsymbol{b}_d|$$

  - Approximation to Euclidean distance, cheapest to compute

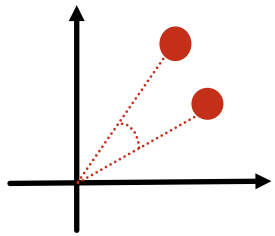


Clustering

- Cosine similarity

$$d(\boldsymbol{a},\boldsymbol{b}) = \frac{\boldsymbol{a}^T\boldsymbol{b}}{\|\boldsymbol{a}\|\,\|\boldsymbol{b}\|}$$

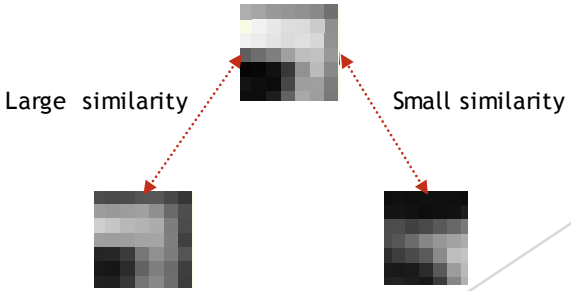
  - smaller the angle, larger the similarity
  - scale invariant measure
  - popular in text retrieval



- Correlation coefficient

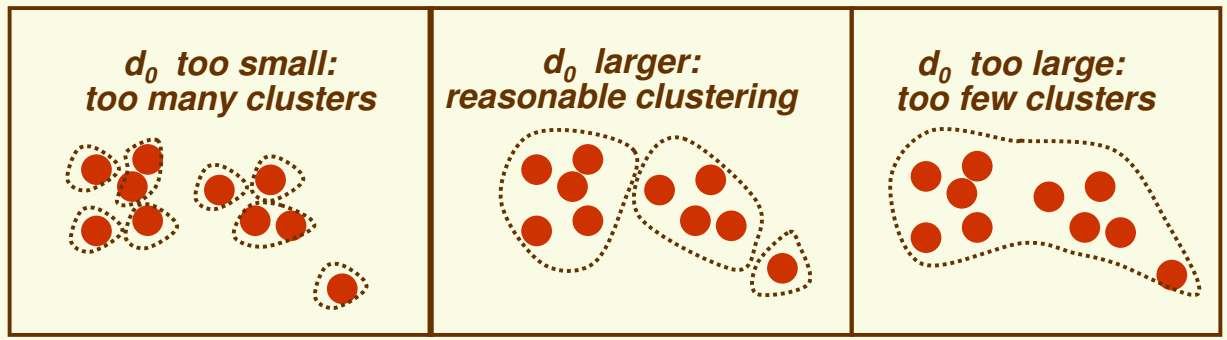
$$d(\boldsymbol{a},\boldsymbol{b}) = \frac{\sum_{d=1}^{D}(\boldsymbol{a}_d-\boldsymbol{\mu}_d)(\boldsymbol{b}_d-\boldsymbol{\mu}_d)}{\left[\sum_{d=1}^{D}(\boldsymbol{a}_d-\boldsymbol{\mu}_d)^2\sum_{d=1}^{D}(\boldsymbol{b}_d-\boldsymbol{\mu}_d)^2\right]^{1/2}}$$

  - popular in image processing



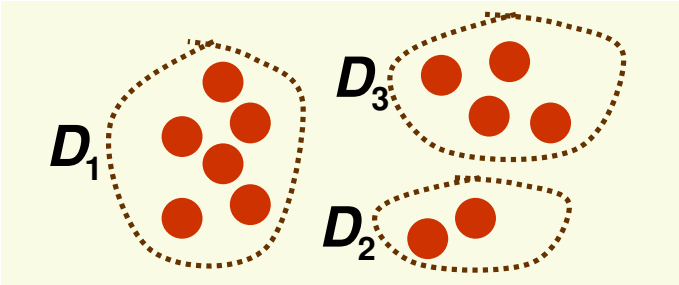
Clustering

- Simplest Algorithm
  - Using the desired distance/similarity measure
  - Go through all sample pairs
    - Put them in the same cluster if the distance between them is lower than a threshold τ
  - Pros: simple to understand and implement
  - Cons: very dependent on the threshold τ



Clustering

- Another approach
- Suppose we have N samples $(\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N)$
- Set the number of subsets K and partition the samples into K subsets $(\boldsymbol{D}_1, \boldsymbol{D}_2, \dots, \boldsymbol{D}_K)$



- Can we define a function to measure the quality of a partitioning $J(\boldsymbol{D}_1, \boldsymbol{D}_2, \dots, \boldsymbol{D}_K)$?
  - If so, the problem is well defined: the optimal clustering is the one that optimises the measure
- However, there are approximately ${}^{K^N}/_{K!}$ distinct partitions!

Clustering

- Let N_i be the number of samples in D_k the mean of the samples in D_k is

  $\boldsymbol{\mu}_k = \frac{1}{N_i} \sum_{n=1}^{N} r_{nk} \boldsymbol{x}_n$

- $\boldsymbol{x}_n$ is the n-th sample
- $\boldsymbol{\mu}_k$ is the k-th mean
- r_{nk} an indicator variable (1 or 0), does the n-th sample belong to the k-th mean?

- Then the sum-of-squared errors criterion function (to minimise) is

  $J_{SSE} = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \| \boldsymbol{x}_n - \boldsymbol{\mu}_k \|^2$

- Where the number of clusters K is fixed

Clustering

- Other criterion functions can be obtained by replacing $\|\boldsymbol{x}_n - \boldsymbol{\mu}_k\|^2$ by other distance functions
  - Median
  - Maximum
  - and more…
- But, given there are approximately ${}^{K^N}/_{K!}$ distinct partitions how do we find the optimal clustering?
- We can find a local optimum by using an iterative approach
  - Find a reasonable partition
  - Move samples from one group to another such that (s.t.) the object function J is improved

Clustering

- Example: K-means
- One of the most famous algorithms and still commonly used today
- Uses the SSE

$$J_{SSE} = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \| \boldsymbol{x}_n - \boldsymbol{\mu}_k \|^2$$

- Fix K and take an initial guess at what the means $(\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_K)$ should be

1. Assign the n-th sample $\boldsymbol{x}_n$ to the mean closest to it. Do this for all N samples.
2. Compute the sample mean for each of the K means using

$$\boldsymbol{\mu}_k = \frac{1}{N_i} \sum_{n=1}^{N} r_{nk} \boldsymbol{x}_n$$

3. If not converged, repeat steps 2 and 3

Clustering

K-means

- This is an example of a well known Expectation Maximisation (EM) Algorithm)
- Iteratively minimises J

$$J_{SSE} = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \| \boldsymbol{x}_n - \boldsymbol{\mu}_k \|^2$$

1. Keeping $\boldsymbol{\mu}_k$ constant and solving for r_{nk} [Expectation or E-Step]
2. Keeping r_{nk} constant and solving for $\boldsymbol{\mu}_k$ [Maximisation or M-Step]

Clustering

E-Step: solving for r_{nk}

- Objective function

  $J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \|\boldsymbol{x}_n - \boldsymbol{\mu}_k\|^2$

- r_{nk} can be optimised by choosing the k (the mean) which minimises $\|\boldsymbol{x}_n - \boldsymbol{\mu}_k\|^2$

- Setting r_{nk} to 1 for this k and the remainder to 0

$$r_{nk} = \begin{cases} 1 & if\ k = argmin_j \left\|\boldsymbol{x}_n - \boldsymbol{\mu}_j\right\|^2 \\ 0 & otherwise \end{cases}$$

Clustering

M-Step: solving for $\boldsymbol{\mu}_k$

- Objective function

  $J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \|\boldsymbol{x}_n - \boldsymbol{\mu}_k\|^2$

- Differentiate with respect to $\boldsymbol{\mu}_k$

  $\frac{\partial J}{\partial \boldsymbol{\mu}_k} = 2 \sum_{n=1}^{N} r_{nk} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)$

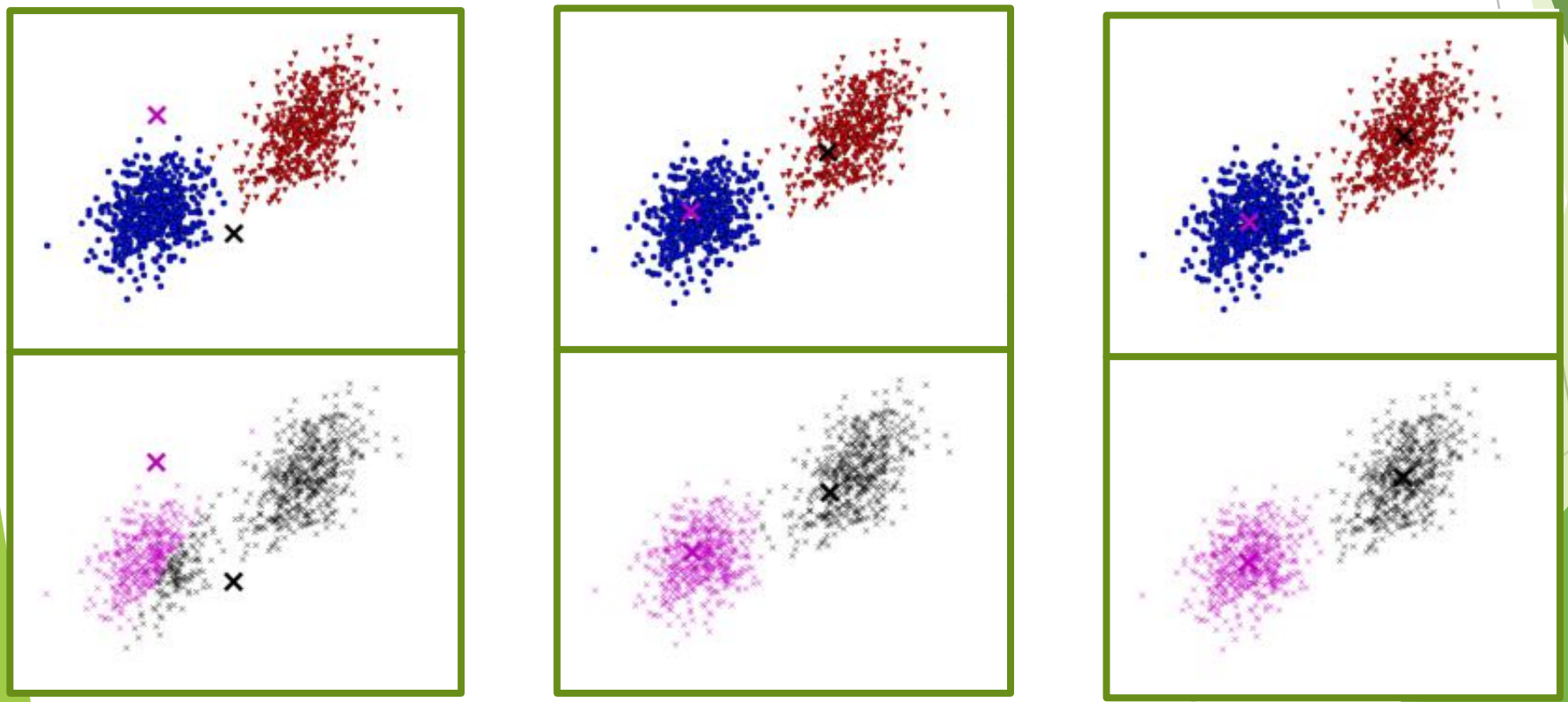
- Minimise by setting to 0

  $0 = 2 \sum_{n=1}^{N} r_{nk} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)$

- Re-arrange to solve for $\boldsymbol{\mu}_k$

  $\boldsymbol{\mu}_k = \frac{\sum_n r_{nk} \boldsymbol{x}_n}{\sum_n r_{nk}}$

Clustering



Clustering

- Practical examples of what K-means can be used for
- Image compression: vector quantisation
  - Take a colour image, each pixel (RGB) is a sample $\boldsymbol{x}_n$ find the means that will encode this image compactly
  - Using K values instead of 256^3 possible values
  - In this example the means are called $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_K$ *code-book vectors*
    - This trick is also used to extract features for image classification (bag-of-visual-words)

$K = 4$

$K = 8$

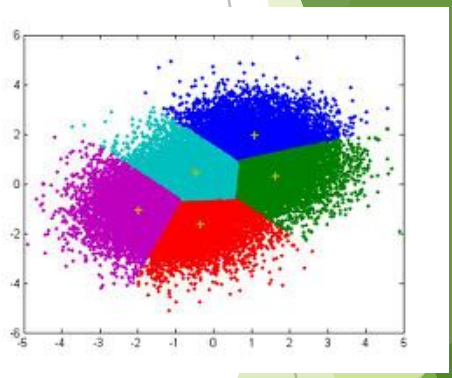
$K = 16$

$K = 32$

Clustering

K-means

- Is a common approach to clustering
- Finds a local optimum (not a global one)
  - Usually good enough, but has several failure cases
  - Initialisation is difficult many approaches (K-means++)
- K is hard to choose and very important
- Hard assignment of samples to a “centroid”
  - Probabilistic extension: mixture of Gaussians
- This and other approaches are used in audio/image classification
  - bag of visual words, Gaussian mixture models (GMMs), fishervectors



Unsupervised Learning

Curse of Dimensionality

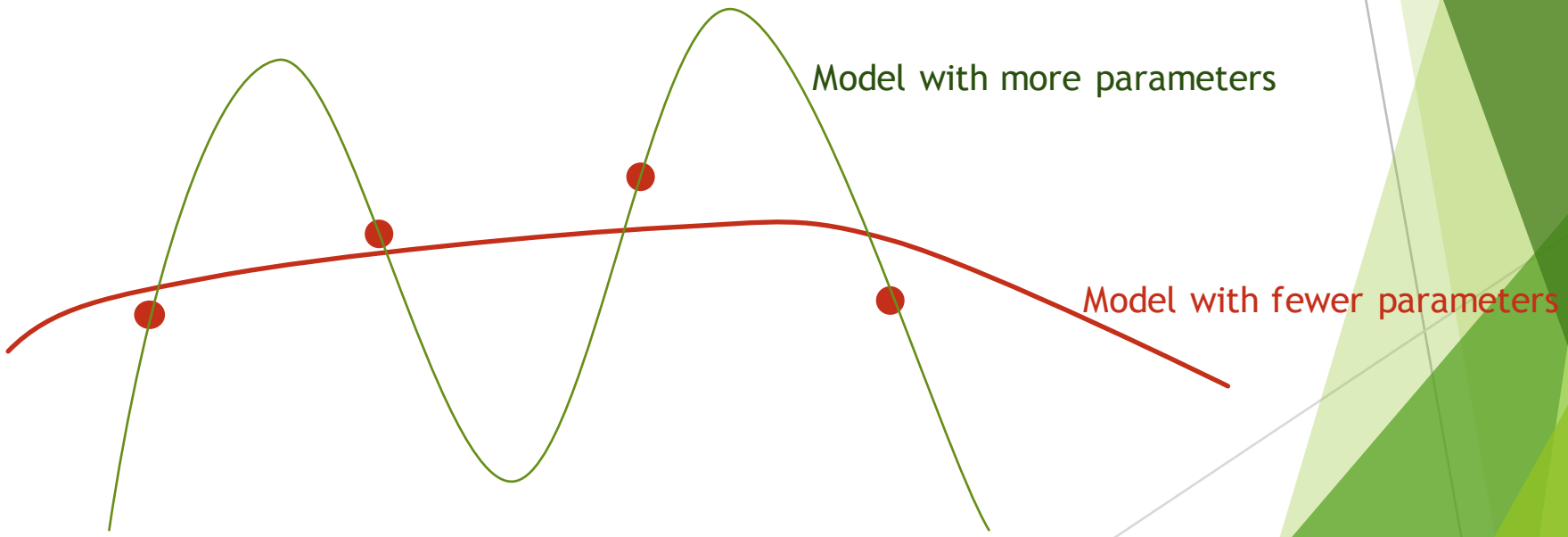
- Complexity increases as the number of dimensions D increases
- Yet, we often have high dimensional data (images, audio, etc.)
- Many methods have complexity of $O(ND^2)$
  - N is the number of samples
- For example, a covariance matrix has D^2 parameters

$$\begin{bmatrix} \sigma_{11}^2 & \cdots & \sigma_{1D}^2 \\ \vdots & \ddots & \vdots \\ \sigma_{D1}^2 & \cdots & \sigma_{DD}^2 \end{bmatrix}$$

- To accurately estimate this N should be much bigger than D^2
- Otherwise model is too complicated for the data (*overfitting*)

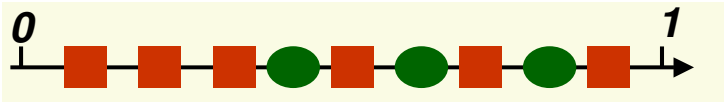
Curse of Dimensionality

- Overfitting



Curse of Dimensionality

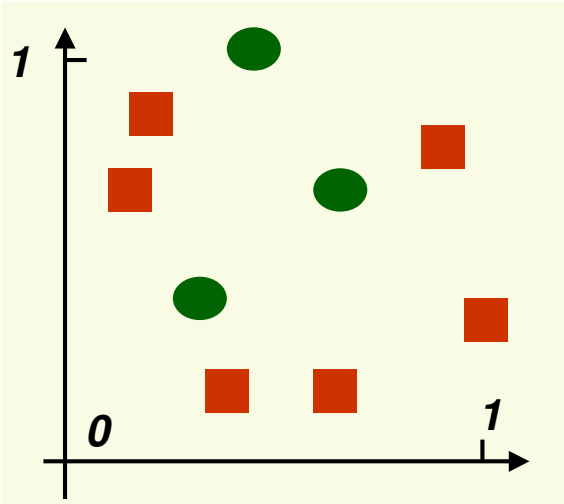
- Suppose we want to use the nearest neighbour (NN) approach with $D = 1$



- This will not be discriminative enough (classes aren't well separated)
- So… we decide to use $D = 2$
  - Problem for NN to work well we need a lot of samples (dense)
- To maintain the same density as $D = 1$ (9 samples per unit length) how many samples do we need?

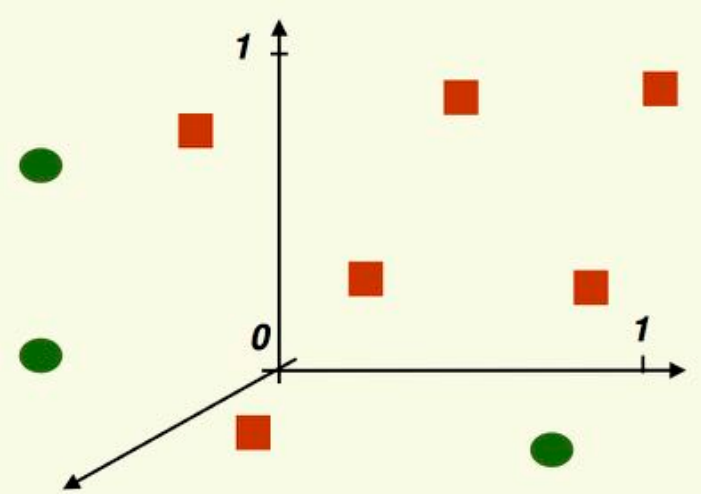
Curse of Dimensionality

- We could decide to continue, even if we don't have any more samples
- Problem: this will be too sparse for NN to work well



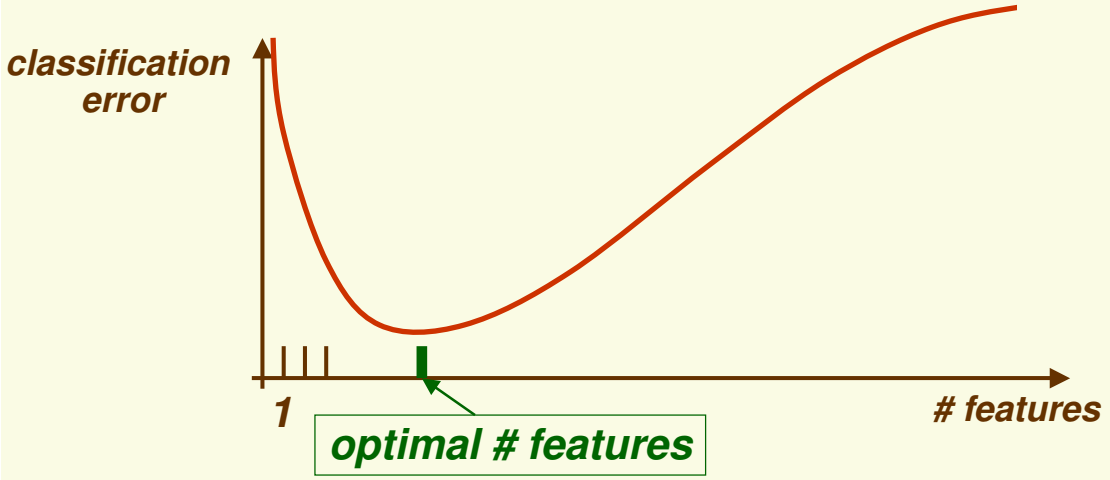
Curse of Dimensionality

- Things become even more problematic as we go from $D = 2$ to $D = 3$
- If **9** samples was for $D = 1$ then for $D = 3$ we need **729** samples!



Curse of Dimensionality

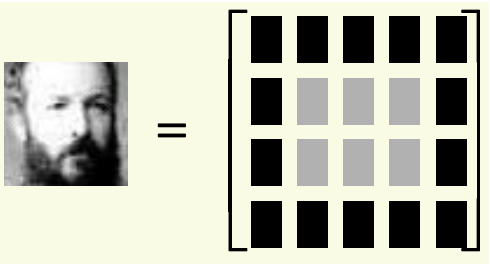
- For a fixed number of samples was we add features, the graph of classification error



- For each fixed sample size N there is an optimal number of features to use

Curse of Dimensionality

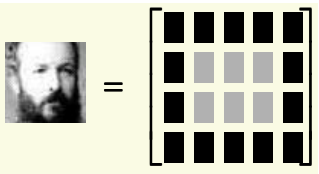
- Generally: we should avoid trying to create a lot of features
- Often, this is not possible because we start with a lot of features



- This image has $H \times W$ pixels ($D = H \times W$)
- Even for a small image, this is a lot of pixels
  - For a $H = W = 20$ (20×20) image this is $D = 400$
- If **10** samples is dense enough for $D = 1$ then we would need $\mathbf{10^{400}}$ samples!

Curse of Dimensionality

- Even though we have set this up as a problem with $D = 400$ dimesions



- This does not mean it really is…
- Space of all $H \times W$ images has $D = 400$ dimensions
  - But this is for all images, not just faces
- Space of all $H \times W$ faces must be much smaller
- Most likely we are not setting the problem up with the right features
- If we had better features than we wouldn't need $D = 400$

Dimensionality Reduction

- High dimensional data is challenging and often redundant
- There are often repeating patterns
- It is natural to try to reduce dimensionality

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_D \end{bmatrix} \rightarrow f\left(\begin{bmatrix} x_1 \\ \vdots \\ x_D \end{bmatrix}\right) = \begin{bmatrix} y_1 \\ \vdots \\ y_K \end{bmatrix} = \boldsymbol{y}$$

- Where $K < D$
- Example:

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \rightarrow \begin{bmatrix} x_1 + x_2 \\ x_3 + x_4 \end{bmatrix} = \boldsymbol{y}$$

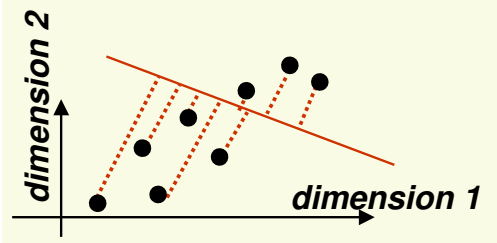
Dimensionality Reduction

- Ideally the new (reduced dimension) vector $\boldsymbol{y}$ should contain all the important information for classification
- Most likely this will be a non-linear function
  - But, linear functions are easier to find
- If we assume that $f(\boldsymbol{x})$ is a linear mapping we can represent this as a matrix $\boldsymbol{W}$

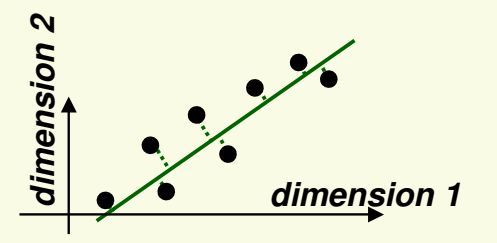
$$\begin{bmatrix} x_1 \\ \vdots \\ x_D \end{bmatrix} \rightarrow \boldsymbol{W} \begin{bmatrix} x_1 \\ \vdots \\ x_D \end{bmatrix} = \begin{bmatrix} w_{11} & \cdots & w_{1D} \\ \vdots & \ddots & \vdots \\ w_{K1} & \cdots & w_{KD} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_D \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_K \end{bmatrix}$$

Dimensionality Reduction

- Main idea: seek an accurate data representation in a lower dimensional space
- Example for 2D



Large projection errors,
bad line to project to

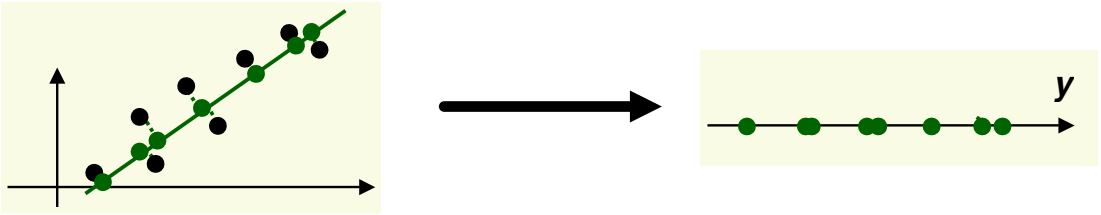


Small projection errors,
good line to project to

- Note: the good line to use for projection lies in the direction of largest variance

Dimensionality Reduction

- After the data is projected onto the best line we can transform the coordinate system to a $D = 1$ representation for $\boldsymbol{y}$



- The new data $\boldsymbol{y}$ has the same variance as the old data $\boldsymbol{x}$ in the direction of the green line
- Intuition: we should exploit the variance in the dataset in some way

Dimensionality Reduction

- Let's consider what we are searching for
  - We have an input space $\boldsymbol{X}$
  - We want to find a feature map $\boldsymbol{W}$
  - So that the weights decorrelate: minimise the relationship (redundancy) between dimensions

  $$(\boldsymbol{WX})(\boldsymbol{WX})^{\boldsymbol{T}} = N\boldsymbol{I}$$
- How can we solve this?

  $$\boldsymbol{WXX}^{T}\boldsymbol{W}^{T} = N\boldsymbol{I}$$

  $$\boldsymbol{WCov}(\boldsymbol{X})\boldsymbol{W}^{T} = \boldsymbol{I}$$

Dimensionality Reduction

- How can we solve this?

  $$(\boldsymbol{WX})(\boldsymbol{WX})^{\boldsymbol{T}} = N\boldsymbol{I}$$

  $$\boldsymbol{WXX}^{T}\boldsymbol{W}^{T} = N\boldsymbol{I}$$

  $$\boldsymbol{WCov}(\boldsymbol{X})\boldsymbol{W}^{T} = \boldsymbol{I}$$

- Covariance matrices are symmetric positive definite and have orthogonal eigenvectors and eigenvalues
- Also, can be factorised by

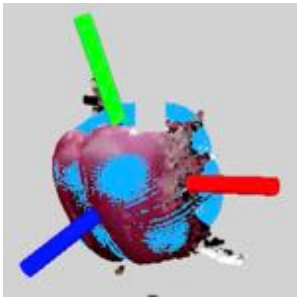
  $$\boldsymbol{UAU}^{T} = \boldsymbol{\Lambda}$$

- Where $\boldsymbol{U}$ has eigenvectors of $\boldsymbol{A}$ in its columns and $\boldsymbol{\Lambda}$ has the eigenvalues on the diagonal

Dimensionality Reduction

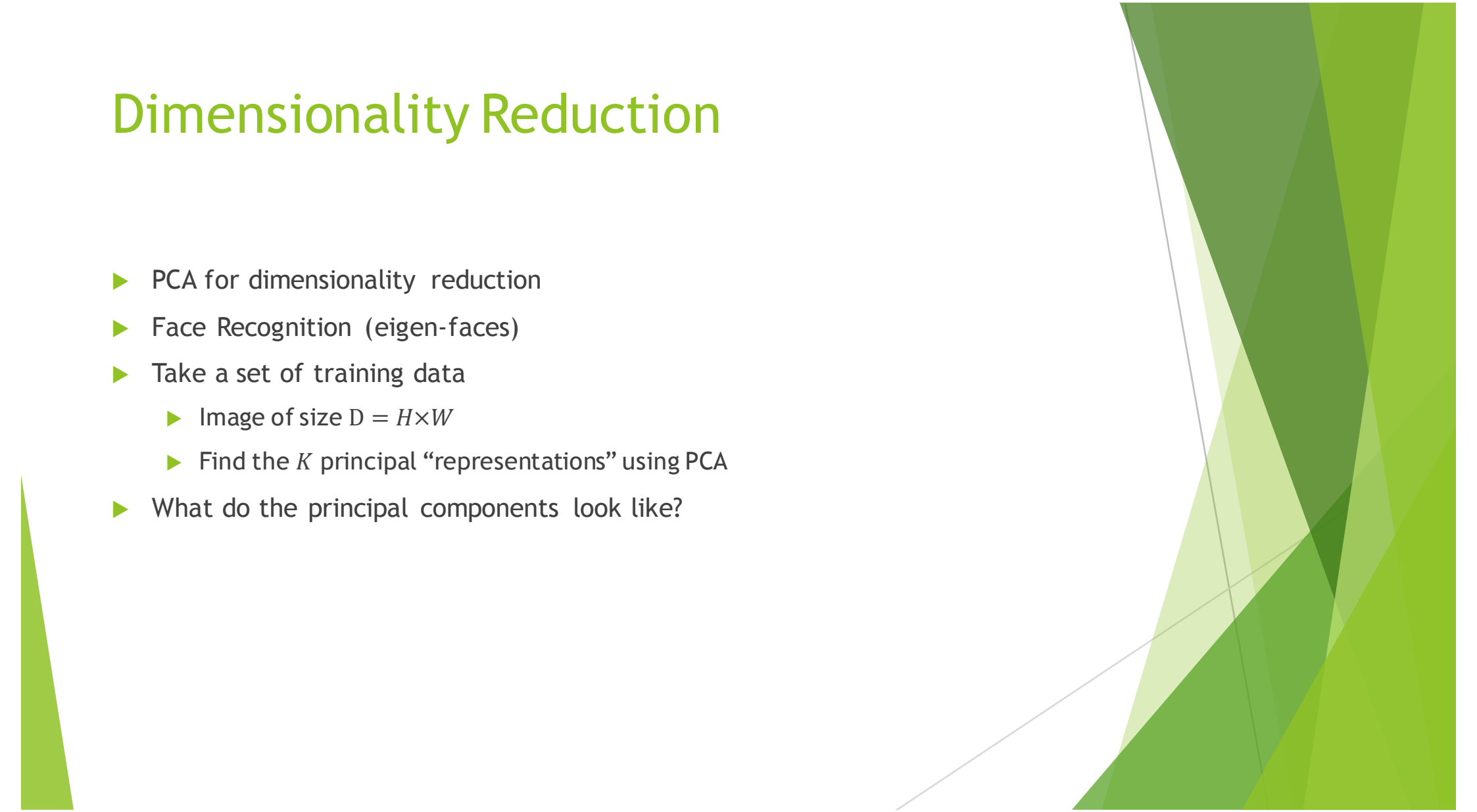
- What does this mean?
  - The eigenvectors are the “unit” vectors that decorrelate the original input space
  - The eigenvalues are the variance represented by the eigenvector
- Practical examples
  - We have 2D data and want to find the ellipsoid (potentially rotated) that represents this

  - We have a 3D object and want to find the orientation of the object



Dimensionality Reduction

- PCA for dimensionality reduction
- Face Recognition (eigen-faces)
- Take a set of training data
  - Image of size $\mathrm{D} = H \times W$
  - Find the K principal "representations" using PCA
- What do the principal components look like?

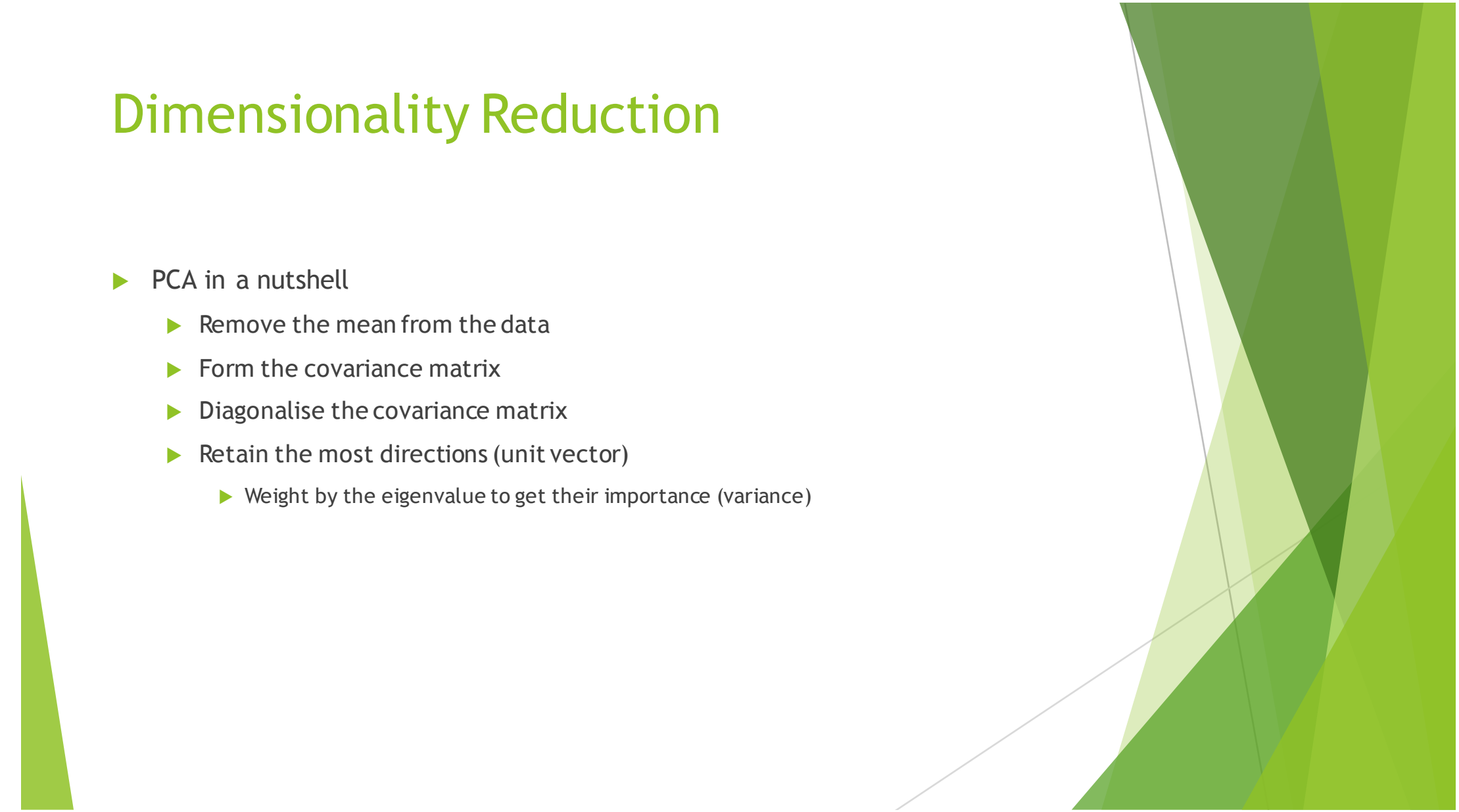


PCA

- Each face image can now be represented by a linear combination of the representations

Dimensionality Reduction

- PCA in a nutshell
  - Remove the mean from the data
  - Form the covariance matrix
  - Diagonalise the covariance matrix
  - Retain the most directions (unit vector)
    - Weight by the eigenvalue to get their importance (variance)



PCA

- Each face image can now be represented by a linear combination of the representations


Original


D=10

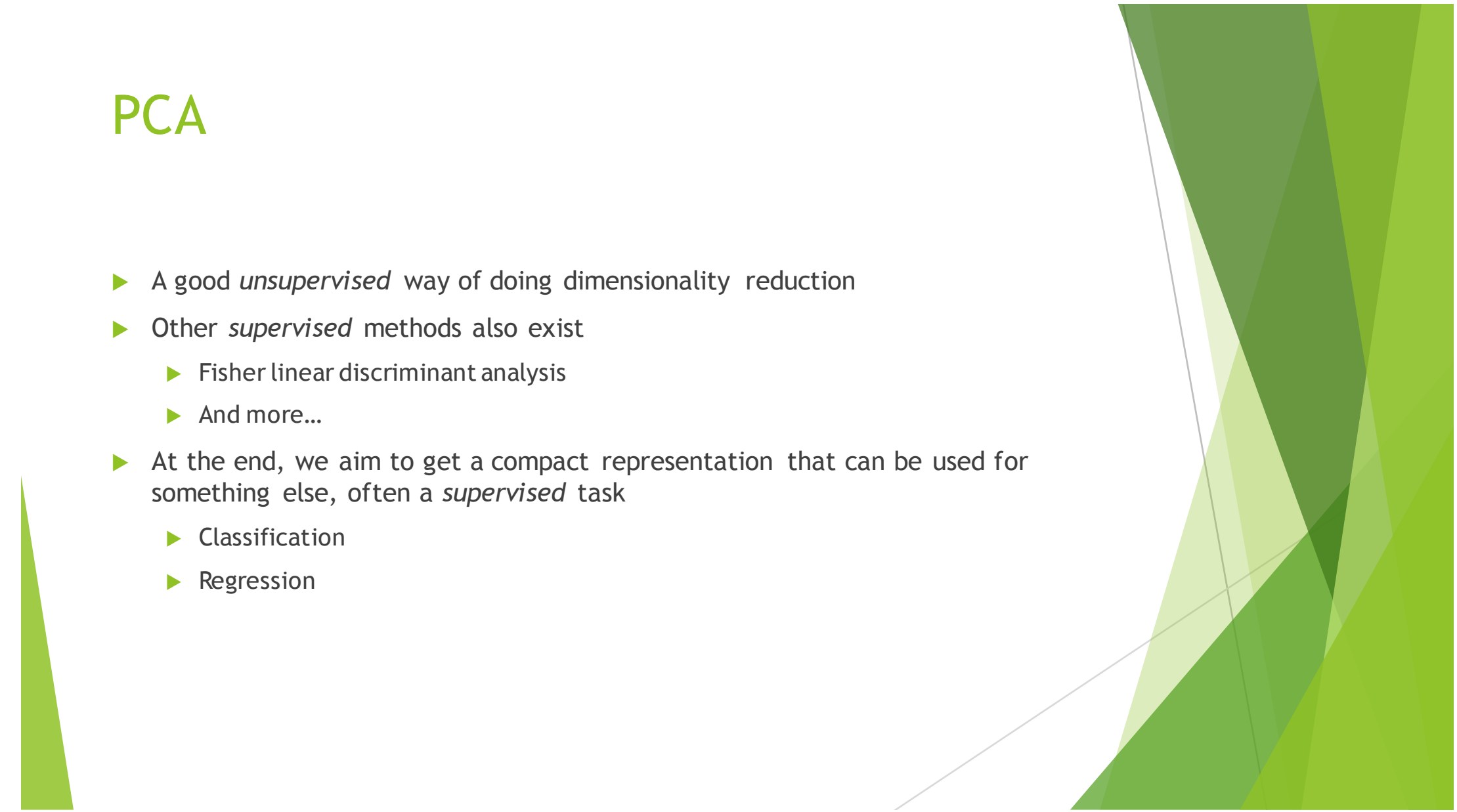

D=100


D=500


D=1000

PCA

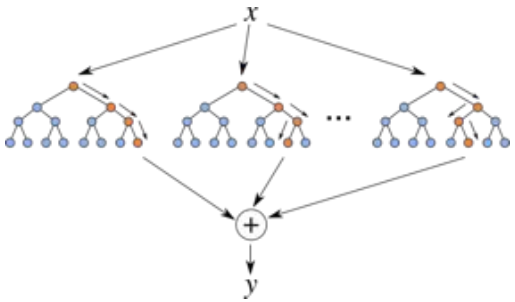
- A good *unsupervised* way of doing dimensionality reduction
- Other *supervised* methods also exist
  - Fisher linear discriminant analysis
  - And more…
- At the end, we aim to get a compact representation that can be used for something else, often a *supervised* task
  - Classification
  - Regression

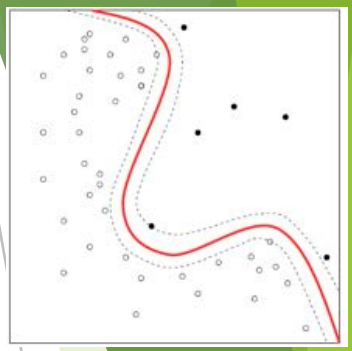


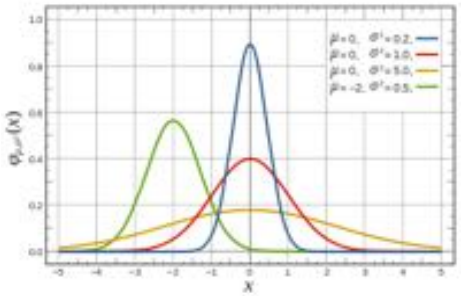
Supervised Learning

Classifiers

- **Support Vector Machines**
- Naïve Bayes Classifiers
- Boosted Classifiers
- **Neural Networks** (and deep learning)
- Random Forests







Classifiers

- Supervised Learning
  - Classification Tasks
- Is the topic for next week