Lecture 7

Searching the image

- Sliding windows
- Image scale pyramid

Dimensionality Reduction

- Curse of dimensionality
- Principal Components Analysis

Story so far

- Have introduced some methods to describe the appearance of an image/image patch via a feature vector. (SIFT HOG etc..)
- For patches of similar appearance their computed feature vectors should be similar while dissimilar if the patches differ in appearance.
- Feature vectors are designed to be invariant to common transformations that superficially change the pixel appearance of the patch.

Next problem

Have a training set of image patches each described by a feature vector $\mathbf{f}_r$ and a label ω such as *face* or *not face*.

$$\equiv \mathbf{f}_r, \ldots$$

face

Given a **novel image** identify the **patches** in this image that correspond to the **target class** (faces).

One part of the problem we have already partially **explored**.....

Know about classification

A patch from the novel image generates a feature vector $\mathbf{f}_n$, then can assign a label to $\mathbf{f}_n$ based on

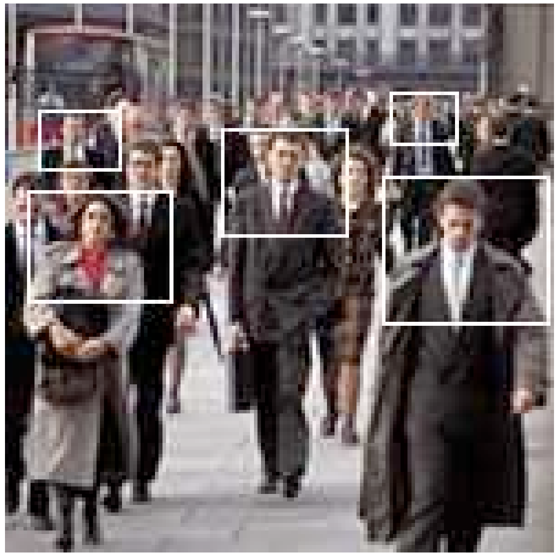
- a nearest neighbour classifier
- logistic regression
- a learnt separating hyper-plane etc..
- and some more methods you'll be learning about

However, which and how many different image patches do we extract from the novel image ??

Remember....

The sought after image patch can appear at:

- any spatial location in the image,
- any size (the size of an imaged object depends on its distance from the camera),
- multiple locations.



Sliding window technique

Must examine patches centered at different pixel **locations** and **sizes**.

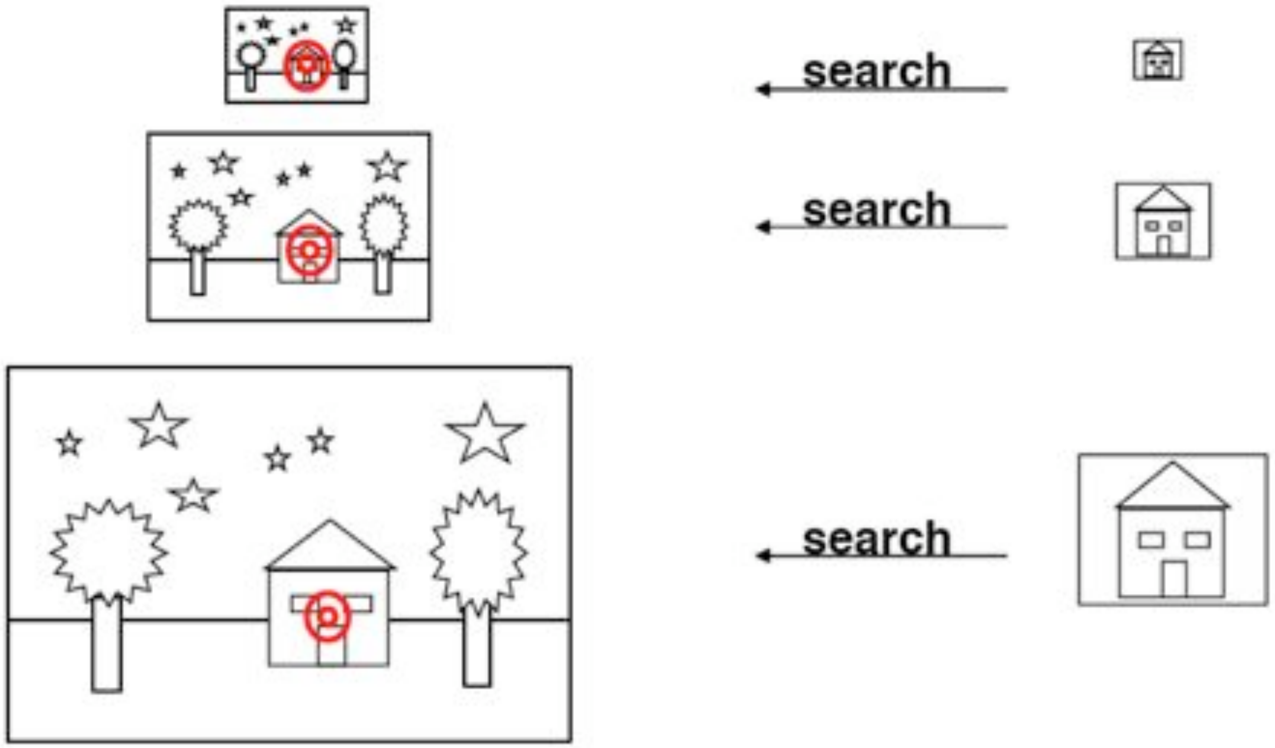
Naïve Option: Exhaustive search in the test image

```
for j = 1:n_s    %% vary size of patch
   n = n_min + j*n_step
   for x=0:x_max    %% vary x-position of patch
     for y=0:y_max    %% vary y-position of patch
       Extract image patch centred on pixel x, y of size n×n.
       Rescale it to the size of the reference patch
       Compute feature vector f.  Classify it.
```

This is computationally intensive, especially if it is expensive to compute $\mathbf{f}$, have $\texttt{n\_s} \times \texttt{x\_max} \times \texttt{y\_max}$ iterations.

Also if $\texttt{n}$ is large then it is probably very costly to compute $\mathbf{f}$.

What about a multi-scale search?



Irani & Basri

Scale pyramid option

Construct an image pyramid that represents an image at several resolutions. Then either

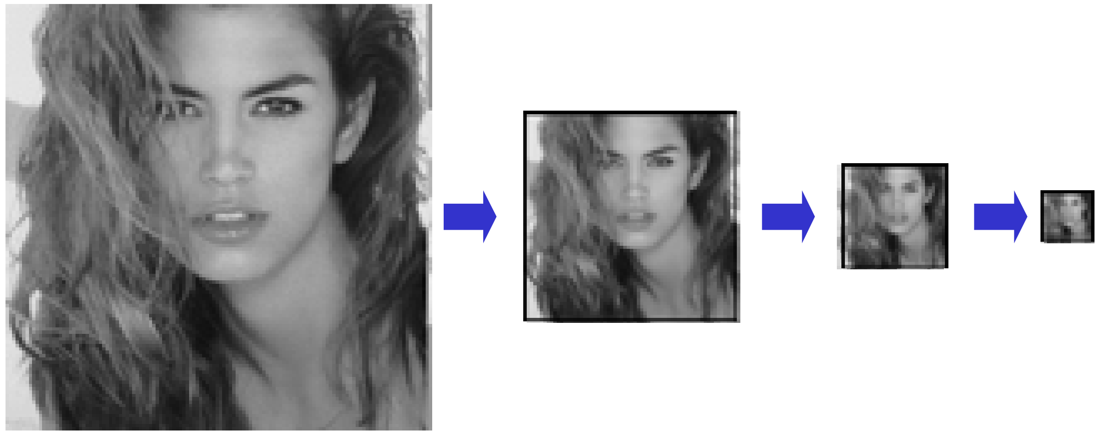
- Use the coarse scale to highlight promising image patches and then just explore these area in more detail at the finer resolutions. (Quick but may miss best image patches)

- Visit every pixel in the fine resolution image as a potential centre pixel, but simulate changing the window size by applying the same window size on the different images in the pyramid.

Now will review construction of the image pyramid..

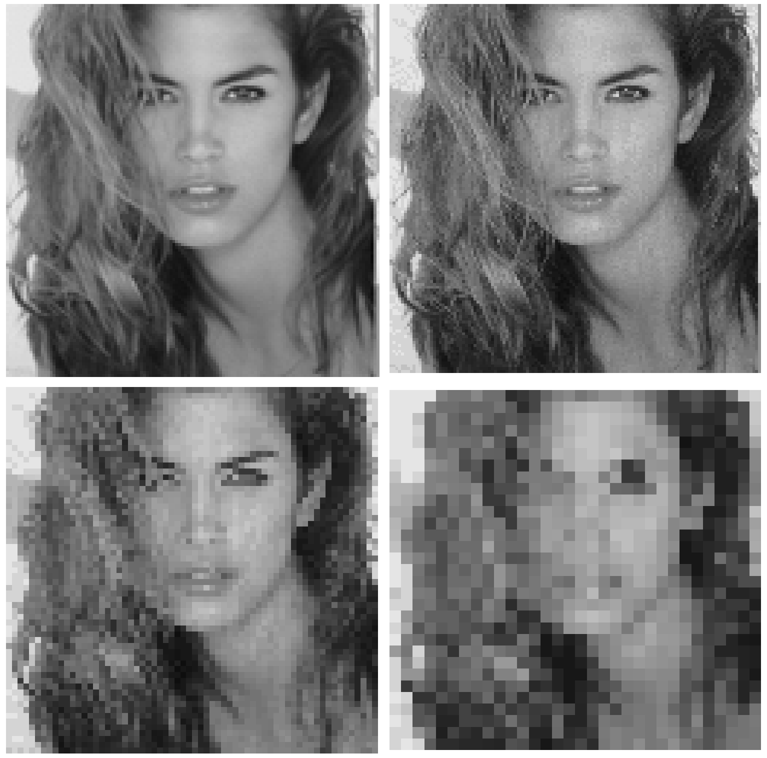
Image Scale Pyramids

Naive subsampling



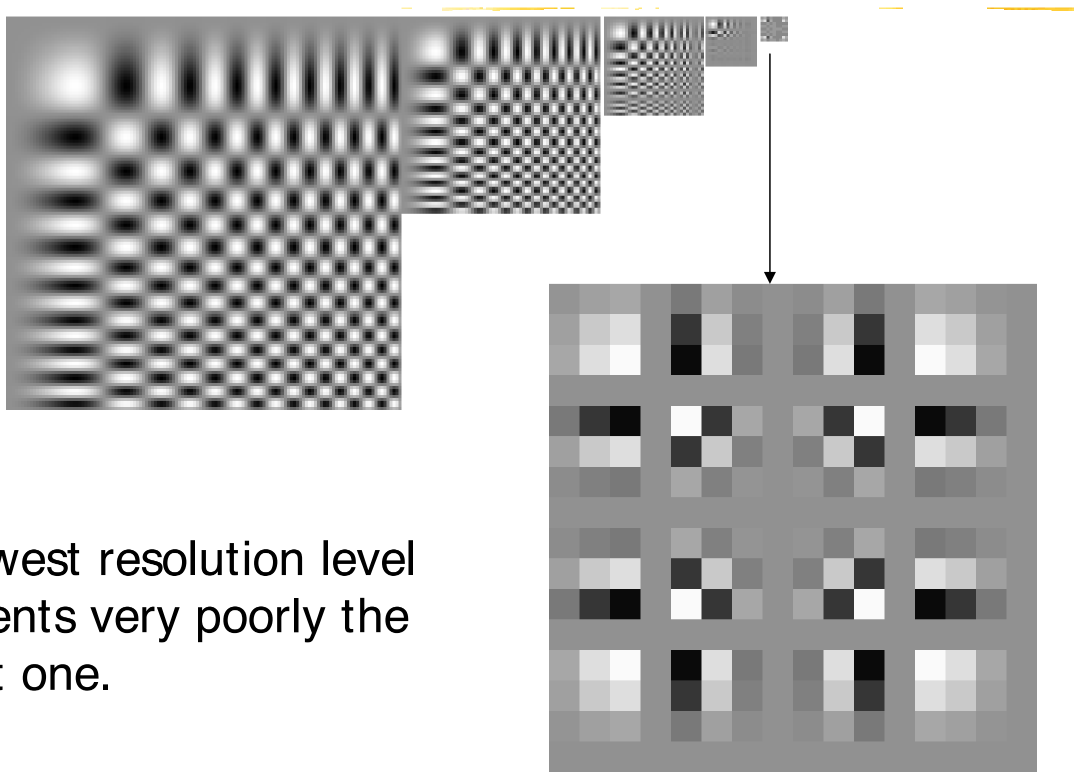
Pick every other pixel in both directions

Sub-sampling artifacts



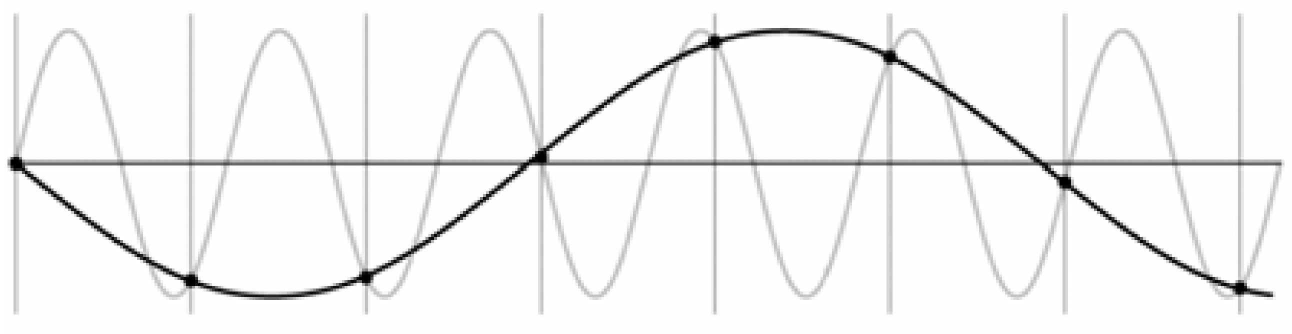
Particularly noticeable in high frequency areas, such as on the hair.

Synthetic example



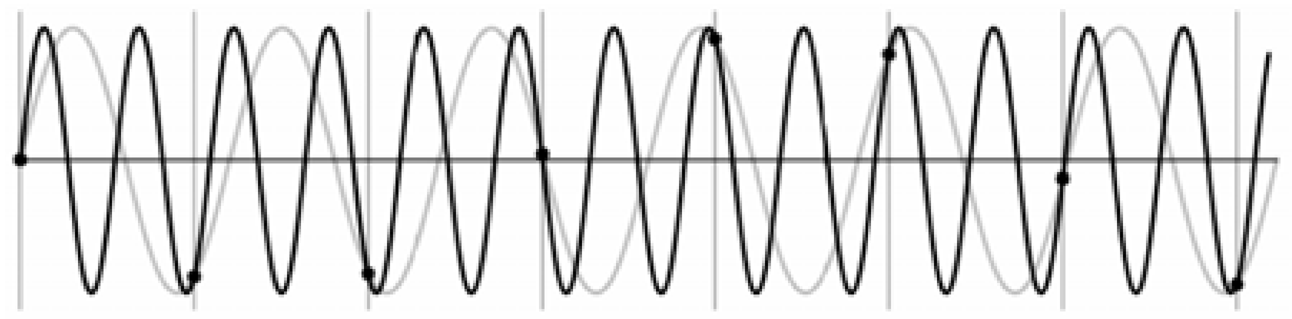
The lowest resolution level represents very poorly the highest one.

Under-sampling



Looks just like a lower frequency signal!

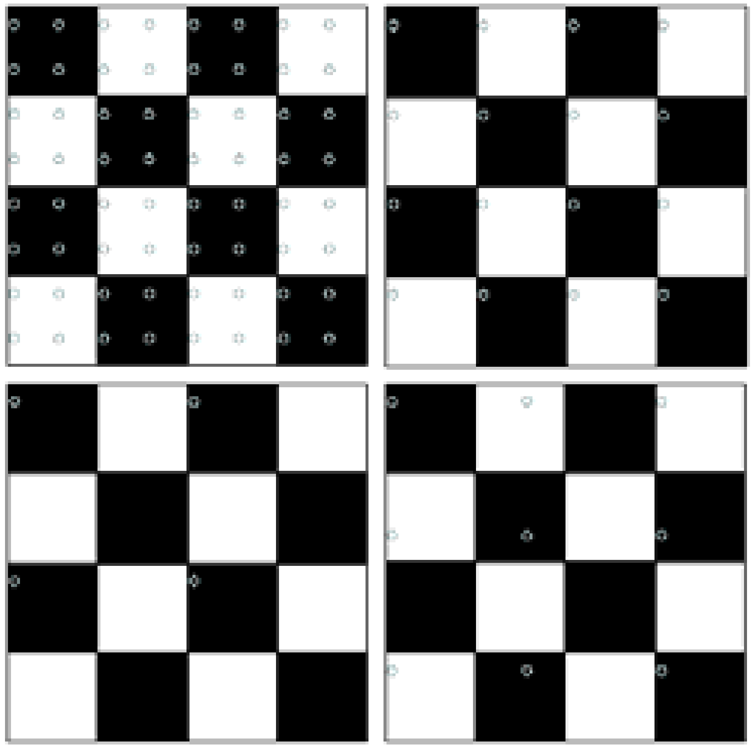
Under-sampling



Looks like higher frequency signal!

Aliasing: higher frequency information can appear as lower frequency information

2-D aliasing

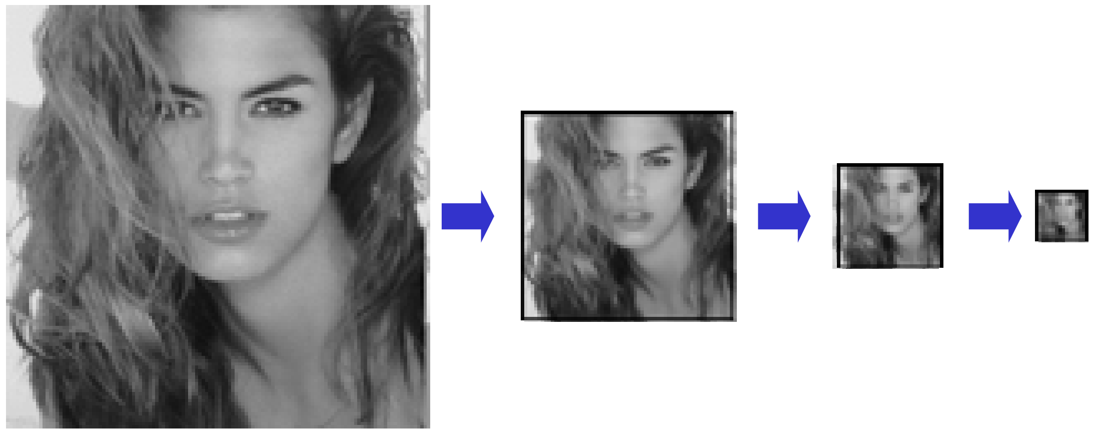


High frequency signal sampled lower than that of the signal yields a poor representation. **Therefore must remove high frequencies before sub-sampling.**

Aliasing summary

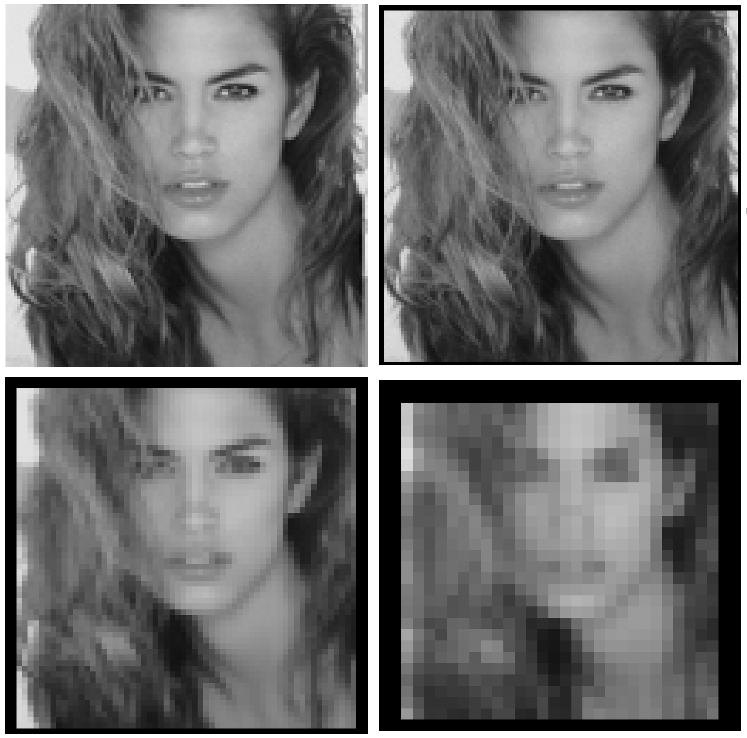
- Can't shrink an image by taking every second pixel due to sampling below the Nyquist rate
- If we do, characteristic errors appear such as
  - jaggedness in line features,
  - spurious highlights.
  - appearance of frequency patterns not present in the original image.

Gaussian pyramid



- Gaussian smooth image.
- Pick every other pixel in both directions.

Images in the pyramid



No aliasing but details are lost as high frequencies are progressively removed.

Scaled representation advantages

- Find template matches at all scales
  - Template size is constant, but image size changes
- Efficient search for correspondence
  - look at coarse scales, then refine with finer scales
  - much less cost, but may miss best match
- Examining of all levels of detail
  - Find edges with different amounts of blur
  - Find textures with different spatial frequencies

Back to Sliding Windows

Summary: Sliding windows

Pros

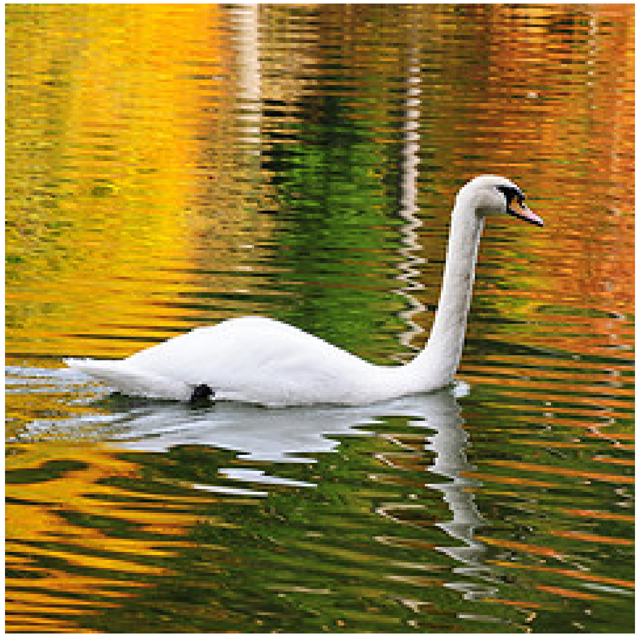
- Simple to implement.
- Good feature choices critical.
- Past successes for certain classes.
- Good detectors available.

Cons/Limitations

- High computational complexity
  - 250,000 locations x 30 orientations x 4 scales = 30,000,000 evaluations!
  - Puts constraints on the type of classifiers we can use.
  - If training binary detectors independently, this means cost increases linearly with number of classes.
- With so many windows, false positive rate better be low!!

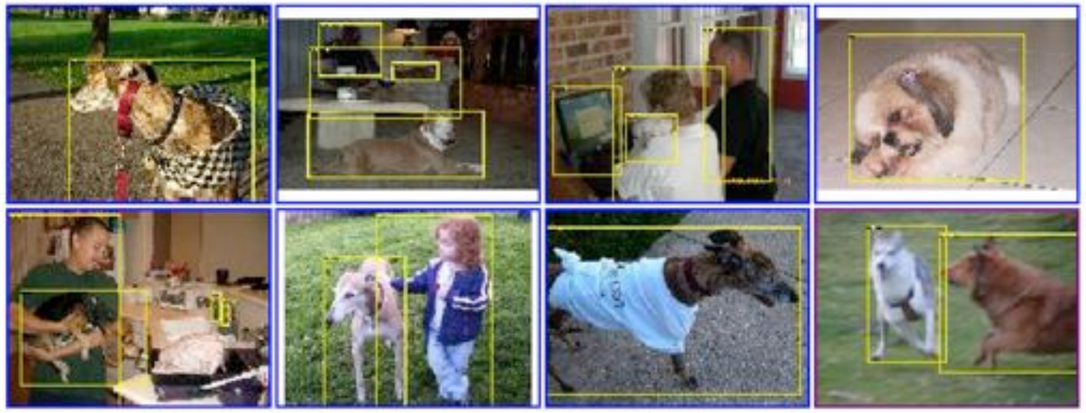
Limitations of sliding windows

- Not all object are **box** shaped.



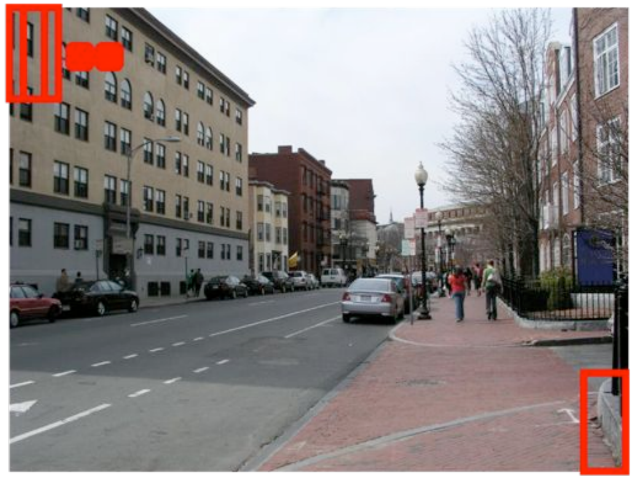
Limitations of sliding windows

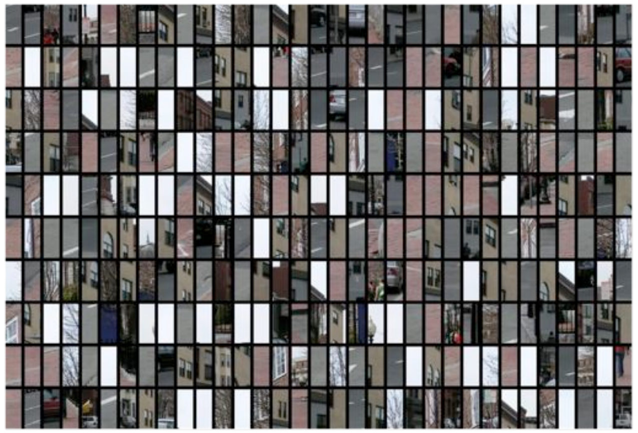
- Non-rigid, deformable objects not captured well with representations assuming a fixed 2D structure; or must assume fixed viewpoint
- Objects with less-regular textures not captured well with holistic appearance-based descriptions.



Limitations of sliding windows

- If considering windows in isolation, context is lost.


Sliding Window


Detector's View

Limitations of sliding windows

• In practice, often entails large, cropped training set.

• Using a global appearance description can lead to sensitivity to partial occlusions.

Need lots of training data

Partial occlusion a problem

The curse of dimensionality

The curse of dimensionality

The curse of dimensionality

- A term coined by Richard Bellman in 1961,
- Refers to the problems associated with estimation and classification of high-dimensional data.
- High-dimensions cause our intuition and many methods to break down.

There are many manifestations of the *curse of dimensionality*...

The curse of dimensionality

Consider a 3-class pattern recognition problem:

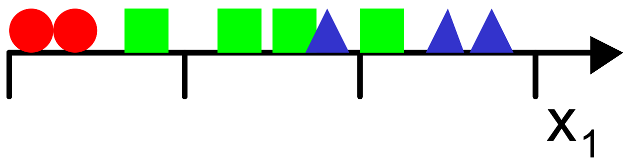
Have training which consisis of a set of feature vectors $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ and their associated class label $\{y_1, y_2, \ldots, y_n\}$. Given a query example $\mathbf{x}_u$ want to estimate its class label.

A simple approach would be to:

- Divide the feature space into uniform bins
- Compute the ratio of examples for each class at each bin and,
- For a new example, find its bin and choose the predominant class in that bin

The curse: Toy example

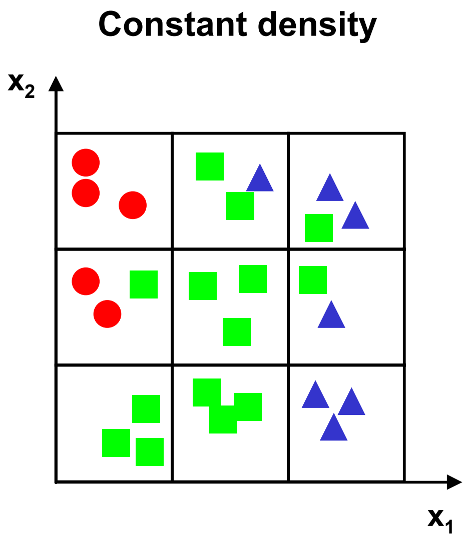
In our toy problem we decide to start with feature vectors of one dimension and divide the real line into 3 segments

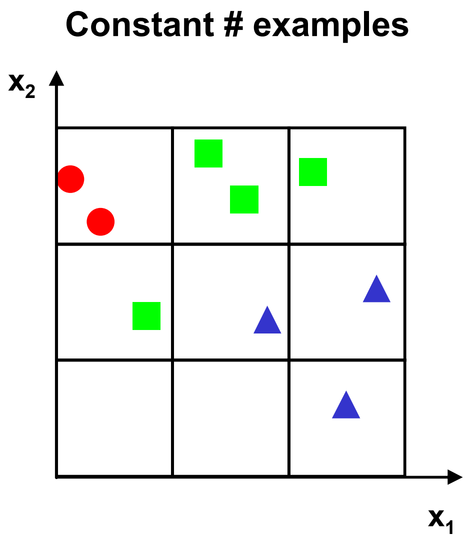


Notice, however, that there exists too much overlap among the classes, so we decide to incorporate a second feature to try and improve separability.

The curse: Toy example

Preserving the granularity of each axis raises the $\#$ of bins to $3^2 = 9$ (in 2D)





Then must decide to either

Maintain the density of examples per bin or **Keep the number of examples as for the 1D case**

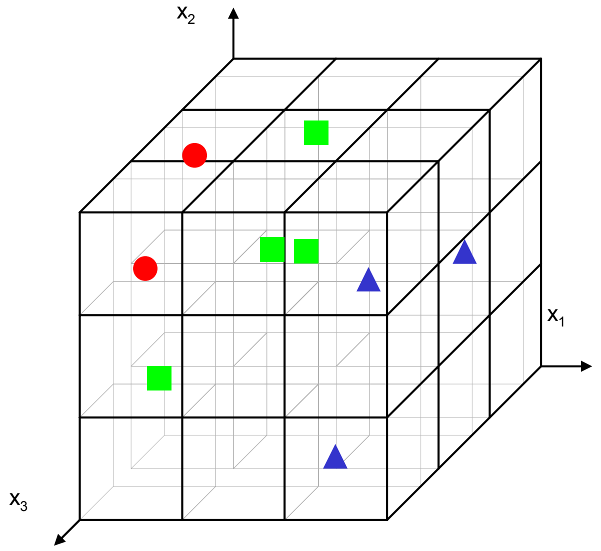
Consequences of this decision

- Maintaining the density increases the number of examples from 9 (in 1D) to 27 (in 2D)
- Maintaining the number of examples constant results in a very sparse 2D scatter plot

The curse: Toy example

Moving to three features makes the problem worse:

- The number of bins grows to $3^3 = 27$
- For the same density of examples the number of needed examples becomes 81
- For the same number of examples, well, the 3D scatter plot is almost empty

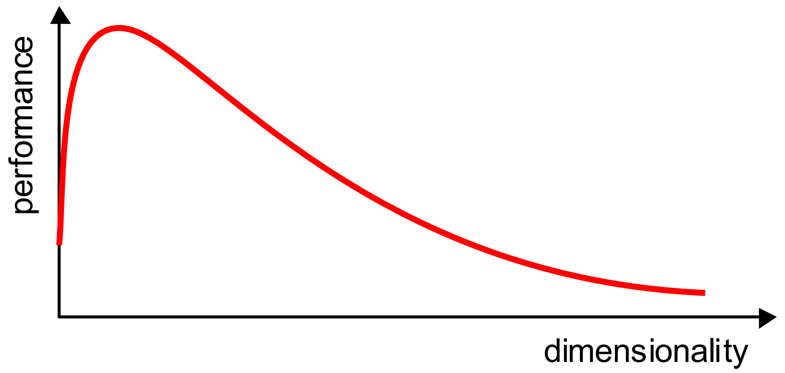


The curse of dimensionality

- This approach to divide the sample space into equally spaced bins was quite inefficient
  - There are other approaches that are much less susceptible to the curse of dimensionality, **but the problem still exists**.
- How do we beat the curse of dimensionality?
  - By incorporating prior knowledge
  - By providing increasing smoothness of the target function
  - By reducing the dimensionality

In practice, the curse of dimensionality means that, for a given sample size, there is a maximum number of features above which the performance of our classifier will degrade rather than improve

- In most cases, the additional information that is lost by discarding some features is (more than) compensated by a more accurate mapping in the lower dimensional space



The curse: Implications

- Exponential growth in the number of examples required to maintain a given sampling density
  - For a density of n examples/bin and d dimensions, the total number of examples is n^d
- Exponential growth in the complexity of the target function (a density estimate) with increasing dimensionality
  - "*A function defined in high-dimensional space is likely to be much more complex than a function defined in a lower-dimensional space, and those complications are harder to discern*" - Friedman

  This means that, in order to learn it well, a more complex target function requires denser sample points!

- What to do if it isn't Gaussian?
  - For one dimension a large number of density functions can be found in textbooks, but for high-dimensions only the multivariate Gaussian density is available. Moreover, for larger values of d the Gaussian density can only be handled in a simplified form!

- In high dimensions most data points are closer to the boundary of the sample space than to any other data point. The reason that this presents a problem is that prediction is much more difficult near the edges of the training sample. One must extrapolate from neighboring sample points rather than interpolate between them.

- Humans have an extraordinary capacity to discern patterns and clusters in 1, 2 and 3-dimensions, but these capabilities degrade drastically for 4 or higher dimensions.

Lecture 7

- The curse of dimensionality
- Dimensionality reduction & Feature selection vs. feature extraction

Dimensionality reduction

Feature extraction: new features from combinations of the existing features.

$$\underbrace{\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix} \longrightarrow \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_k \end{pmatrix} = f\left(\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix}\right)}_{\text{feature extraction}}$$

Feature selection: choose a subset of the features (the more informative ones)

$$\underbrace{\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix} \longrightarrow \begin{pmatrix} x_{i_1} \\ x_{i_2} \\ \vdots \\ x_{i_k} \end{pmatrix}}_{\text{feature selection}}$$

Feature extraction

The problem of feature extraction can be stated as

- Given a feature $\mathbf{x} \in \mathcal{R}^d$ find a mapping

  $$f : R^d \rightarrow \mathcal{R}^k \text{ with } k < d \text{ and } \mathbf{z} = f(\mathbf{x})$$

  such that the transformed feature vector $\mathbf{z} \in \mathcal{R}^k$ preserves (most of) the information or structure in $\mathcal{R}^d$.

- An optimal mapping with respect to a classification task $\mathbf{z} = f(\mathbf{x})$ will be one that results in no increase in the minimum probability of error.
  - That is, a Bayes decision rule applied to the initial space $\mathcal{R}^d$ and to the reduced space $\mathcal{R}^k$ yield the same classification rate.

Feature extraction

Generally, the optimal mapping $\mathbf{y} = f(\mathbf{x})$ is a non-linear function

- However, there is no systematic way to generate non-linear transforms
  - The selection of a particular subset of transforms is problem dependent
- For this reason, feature extraction is commonly limited to linear transforms: $\mathbf{z} = W\mathbf{x}$

$$\begin{pmatrix} x_1 \\ x_2 \\ \\ \vdots \\ \\ x_d \end{pmatrix} \longrightarrow \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_k \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1d} \\ w_{21} & w_{22} & \cdots & w_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k1} & w_{k2} & \cdots & w_{kd} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \\ \vdots \\ \\ x_d \end{pmatrix}$$

Lecture 7

- The curse of dimensionality
- Dimensionality reduction
- Feature selection vs. Feature extraction
- Signal representation vs. signal classification

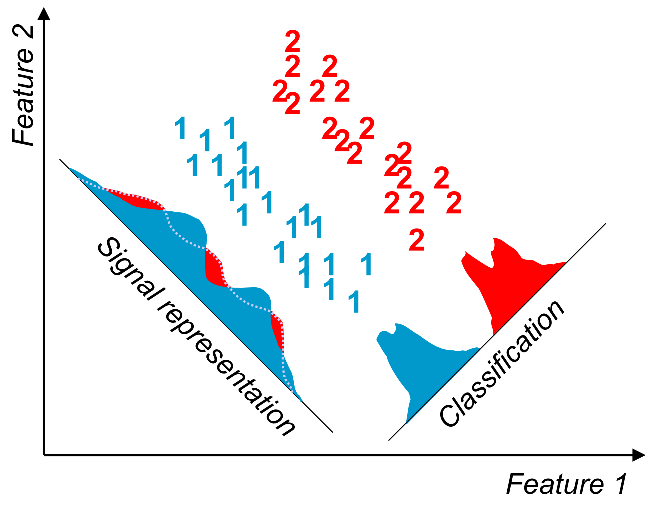
Signal representation Vs classification

- The selection of the feature extraction mapping $\mathbf{z} = f(\mathbf{x})$ is guided by an objective function that we seek to maximize (or minimize)

- Depending on the criteria used by the objective function, feature extraction techniques are grouped into two categories:

Signal representation The goal of the feature extraction mapping is to represent the samples accurately in a lower-dimensional space.

Classification The goal of the feature extraction mapping is to enhance the class-discriminatory information in the lower-dimensional space.



When applying linear feature extraction, two techniques are commonly used

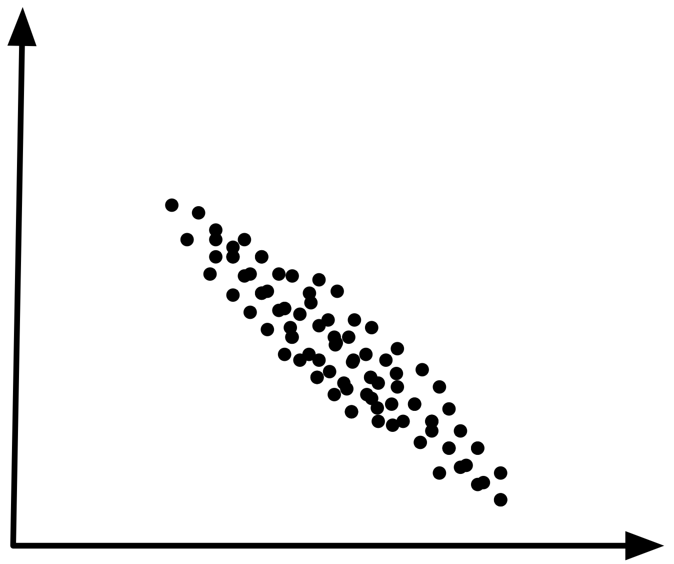
Principal Components Analysis (PCA) - uses a signal representation criterion

Linear Discriminant Analysis (LDA) - uses a signal classification criterion (remember this from lecture 6)

Lecture 7

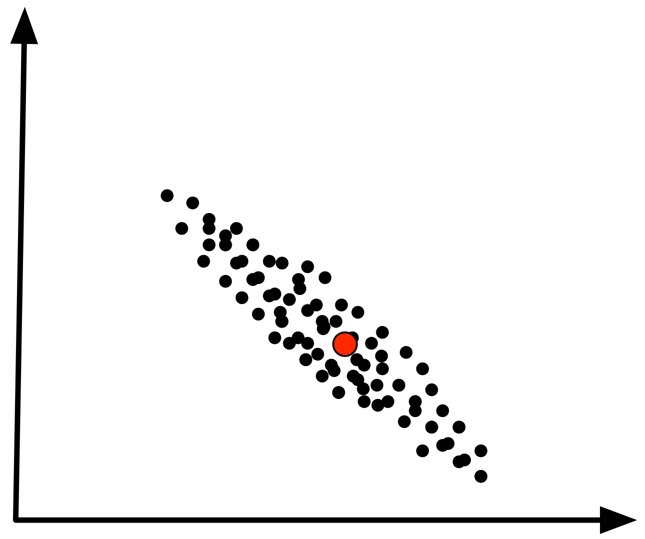
- The curse of dimensionality
- Dimensionality reduction
- Feature selection vs. feature extraction
- Signal representation vs. signal classification
- Principal Components Analysis

Intuitive motivation



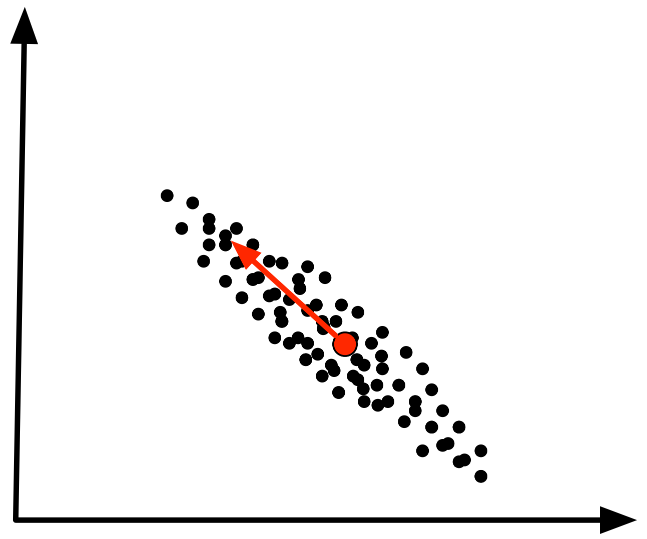
Want to **encode as accurately as possible** the position of the n points in this cluster. Can do so exactly with their x, y-coordinate locations - $2n$ numbers. However, say I only have bandwidth to send $n + 4$ numbers. **Intuitively, what should these numbers be ?**

Intuitive motivation



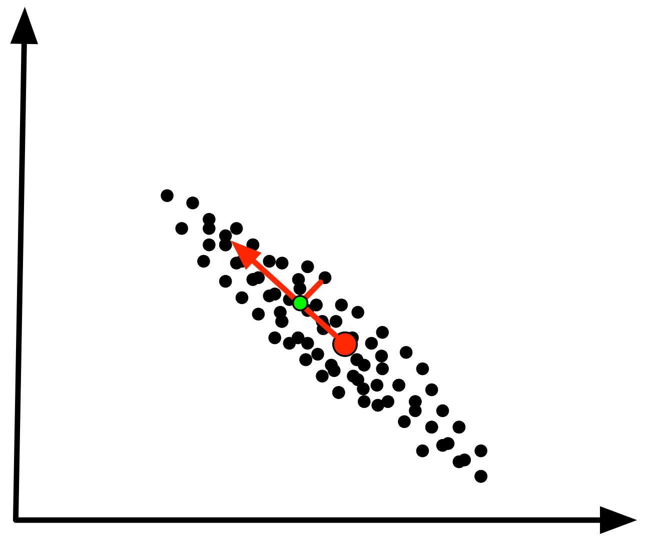
Devote 2 numbers to the center of mass of the points.

Intuitive motivation



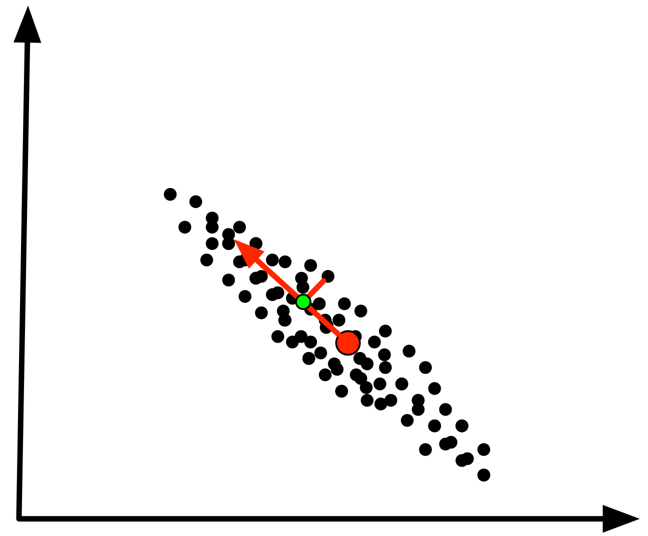
Use 2 numbers to define a direction which corresponds to the direction in which there is most variation.

Intuitive motivation



Let the other m numbers represent where each point is projected onto this line.

Intuitive motivation



Consider the variation in the direction $\mathbf{v}$ among the $\mathbf{x}_i$'s.

- Project each point $\mathbf{x}_i$ onto $\mathbf{v}$ via $z_i = \mathbf{v}^T\mathbf{x}_i$.
- The mean of the z_i's is:

$$\frac{1}{n}\sum z_i = \frac{1}{n}\sum \mathbf{v}^T\mathbf{x}_i = \mathbf{v}^T\left(\frac{1}{n}\sum \mathbf{x}_i\right) = \mathbf{v}^T\boldsymbol{\mu}$$

- The variance, σ_z^2, of the z_i's is

$$\propto \sum_i \left(\mathbf{v}^T(\mathbf{x}_i - \boldsymbol{\mu})\right)^2$$

Which unit vector $\mathbf{v}$ maximizes σ_z^2 ?

Which unit vector $\mathbf{v}$ minimizes σ_z^2 ?

Intuitive motivation

Want to find the unit $\mathbf{v}$ that maximizes/minimises:

$$\sum_i \left((\mathbf{x}_i - \boldsymbol{\mu})^T \mathbf{v}\right)^2 = \sum_i \mathbf{v}^T (\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^T \mathbf{v} = \mathbf{v}^T \left[\sum_i (\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^T\right] \mathbf{v} = \mathbf{v}^T A \mathbf{v}$$

where $A = \sum_i (\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^T$

The two optimization problems are to find

$$\mathbf{v}^* = \arg\max_{\mathbf{v}} \mathbf{v}^T A \mathbf{v} \quad \text{subject to } \mathbf{v}^T \mathbf{v} = 1$$

$$\mathbf{v}^* = \arg\min_{\mathbf{v}} \mathbf{v}^T A \mathbf{v} \quad \text{subject to } \mathbf{v}^T \mathbf{v} = 1$$

Solution:

Constrained optimization problem $\implies$ use Lagrange Multipliers.

Therefore construct the Lagrangian:

$$\mathcal{L} = \mathbf{v}^T A \mathbf{v} + \lambda \, (1 - \mathbf{v}^T \mathbf{v})$$

Take its derivative wrt $\mathbf{v}$, remembering A is symmetric, and set to 0

$$\frac{\partial \, \mathcal{L}}{\partial \mathbf{v}} = 2A\mathbf{v} - 2 \, \lambda \, \mathbf{v} = 0 \implies A \, \mathbf{v} = \lambda \, \mathbf{v}$$

Therefore the optimum $\mathbf{v}$ is an eigenvector of A and the value of the cost function at this optimum is $\mathbf{v}^T A \mathbf{v} = \lambda \mathbf{v}^T \mathbf{v} = \lambda$.

Maximum occurs when $\mathbf{v}$ is eigenvector of A with **largest** eigenvalue.

Minimum occurs when $\mathbf{v}$ is eigenvector of A with **smallest** eigenvalue.

PCA derivation

A more formal derivation of the PCA basis for d dimensional data.

- Let $\mathbf{x}$ be an d-dimensional random vector, represented as a linear combination of orthonormal basis vectors $(\boldsymbol{\phi}_1, \boldsymbol{\phi}_2, \ldots, \boldsymbol{\phi}_d)$ as

$$\mathbf{x} = \sum_{i=1}^{d} z_i \, \boldsymbol{\phi}_i, \text{ where } \boldsymbol{\phi}_i^T \boldsymbol{\phi}_j = \delta_{ij}$$

- Suppose we choose to represent $\mathbf{x}$ with only k $(k < d)$ of the basis vectors. Do this by replacing the components $(z_{k+1}, \ldots, z_d)^T$ with some pre-selected constants b_i

$$\hat{\mathbf{x}}(k) = \sum_{i=1}^{k} z_i \, \boldsymbol{\phi}_i + \sum_{i=k+1}^{d} b_i \, \boldsymbol{\phi}_i$$

- The representation error is then

$$\Delta\,\mathbf{x}(k) = \mathbf{x} - \hat{\mathbf{x}}(k) = \sum_{i=1}^{d} z_i\,\boldsymbol{\phi}_i - \left(\sum_{i=1}^{k} z_i\,\boldsymbol{\phi}_i + \sum_{i=k+1}^{d} b_i\,\boldsymbol{\phi}_i \right) = \sum_{i=k+1}^{d} (z_i - b_i)\,\boldsymbol{\phi}_i$$

- We can measure this representation error by the mean-squared magnitude of $\Delta\mathbf{x}$.

- Our goal is to find the basis vectors $\boldsymbol{\phi}_i$ and constants b_i that minimize this mean-square error

$$\begin{aligned} \bar{\epsilon}^2(k) &= \mathrm{E}\left[|\Delta\mathbf{x}(k)|^2 \right] \\ &= \mathrm{E}\left[\sum_{i=k+1}^{d} \sum_{j=k+1}^{d} (z_i - b_i)(z_j - b_j)\,\boldsymbol{\phi}_i^T \boldsymbol{\phi}_j \right] \\ &= \sum_{i=k+1}^{d} \mathrm{E}\left[(z_i - b_i)^2 \right], \quad \text{as the } \boldsymbol{\phi}_i\text{'s are orthonormal} \end{aligned}$$

PCA derivation ctd

- The optimal values of b_i can be found by computing the partial derivative of the objective function and setting it to zero

$$\frac{\partial}{\partial b_i} \mathrm{E}\left[(z_i - b_i)^2\right] = -2\left(\mathrm{E}\left[z_i\right] - b_i\right) = 0 \implies b_i = \mathrm{E}\left[z_i\right]$$

  Therefore, replace the discarded dimensions z_i's by their expected value

- The mean-square error can then be written as

$$\begin{aligned}
\bar{\epsilon}^2(k) &= \sum_{i=k+1}^{d} \mathrm{E}\left[(z_i - \mathrm{E}\left[z_i\right])^2\right] \\
&= \sum_{i=k+1}^{d} \mathrm{E}\left[\left(\boldsymbol{\phi}_i^T \mathbf{x} - \mathrm{E}\left[\boldsymbol{\phi}_i^T \mathbf{x}\right]\right)\left(\boldsymbol{\phi}_i^T \mathbf{x} - \mathrm{E}\left[\boldsymbol{\phi}_i^T \mathbf{x}\right]\right)^T\right] \\
&= \sum_{i=k+1}^{d} \boldsymbol{\phi}_i^T \mathrm{E}\left[(\mathbf{x} - \mathrm{E}\left[\mathbf{x}\right])(\mathbf{x} - \mathrm{E}\left[\mathbf{x}\right])^T\right] \boldsymbol{\phi}_i = \sum_{i=k+1}^{d} \boldsymbol{\phi}_i^T \Sigma_{\mathbf{x}} \boldsymbol{\phi}_i
\end{aligned}$$

- We want the solution that minimizes this expression subject to the ortho-normality constraints. Incorporate these into the expression using a set of Lagrange multipliers λ_i

$$\bar{\epsilon}^2(k) = \sum_{i=k+1}^{d} \boldsymbol{\phi}_i^T \Sigma_{\mathbf{x}} \boldsymbol{\phi}_i + \sum_{i=k+1}^{d} \lambda_i (1 - \boldsymbol{\phi}_i^T \boldsymbol{\phi}_i)$$

- Compute the partial derivative with respect to the basis vectors, for $j = k + 1, \ldots, d$

$$\frac{\partial}{\partial \boldsymbol{\phi}_j} \bar{\epsilon}^2(k) = \frac{\partial}{\partial \boldsymbol{\phi}_j} \left[\sum_{i=k+1}^{d} \boldsymbol{\phi}_i^T \Sigma_{\mathbf{x}} \boldsymbol{\phi}_i + \sum_{i=k+1}^{d} \lambda_i (1 - \boldsymbol{\phi}_i^T \boldsymbol{\phi}_i) \right] = 2 \left(\Sigma_{\mathbf{x}} \boldsymbol{\phi}_j - \lambda_j \boldsymbol{\phi}_j \right)$$

  Setting this equal to zero implies

$$\Sigma_{\mathbf{x}} \boldsymbol{\phi}_j = \lambda_j \boldsymbol{\phi}_j$$

$\boldsymbol{\phi}_j$ and λ_j are the eigenvectors and eigenvalues of the covariance matrix $\Sigma_{\mathbf{x}}$.

The mean-square error can now be written as

$$\bar{\epsilon}^2(k) = \sum_{i=k+1}^{d} \boldsymbol{\phi}_i^T \Sigma_{\mathbf{x}} \boldsymbol{\phi}_i = \sum_{i=k+1}^{d} \boldsymbol{\phi}_i^T \lambda_i \boldsymbol{\phi}_i = \sum_{i=k+1}^{d} \lambda_i$$

To minimize this measure, choose λ_i's to be the smallest eigenvalues.

Therefore, to represent the $\mathbf{x}_i$'s with minimum square-square error, choose $\boldsymbol{\phi}_i$ (for $i = 1, \ldots, k$) to be the eigenvectors of $\Sigma_{\mathbf{x}}$ with the k largest eigenvalues.

PCA summary

- The eigenvectors of $\Sigma_{\mathbf{x}}$ define a new coordinate system.
  - eigenvectors with largest eigenvalues capture the most variation among training vectors $\mathbf{x}_i$,
  - eigenvectors with smallest eigenvalues have the least variation.
- Can compress the data by only using the top few eigenvectors
  - corresponds to choosing a *linear subspace*
  - these eigenvectors are known as the **principal components**.

Properties of PCA

Have shown that the mean square error between $\mathbf{x}$ and its reconstruction using only k principle eigenvectors is given by the expression:

$$\sum_{j=k+1}^{d} \lambda_j$$

Interpretation

- PCA minimizes reconstruction error.
- PCA maximizes the variance of projection.
- Finds a more *natural* coordinate system for the sample data.

PCA and Images

- An image is a point in a high dimensional space.
  - An $W \times H$ image can be viewed as a point in $\mathcal{R}^{WH}$
- The set of faces is a **subspace** of the set of all images
  - Suppose it is k dimensional.
  - Can find the best subspace using PCA
  - This is like fitting a **hyper-plane** to the set of faces
- This **hyper-plane** is spanned by the eigenvectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$.
  - Any face is then approximated by:

$$\mathbf{x} \approx \boldsymbol{\mu} + (\mathbf{v}_1^T \mathbf{x})\, \mathbf{v}_1 + (\mathbf{v}_2^T \mathbf{x})\, \mathbf{v}_2 + \cdots + (\mathbf{v}_k^T \mathbf{x})\, \mathbf{v}_k$$

PCA and Images

If the matrix $W \in \mathcal{R}^{k \times d}$ is formed such that each of its rows corresponds to one of the eigenvectors $\mathbf{v}_i$ then

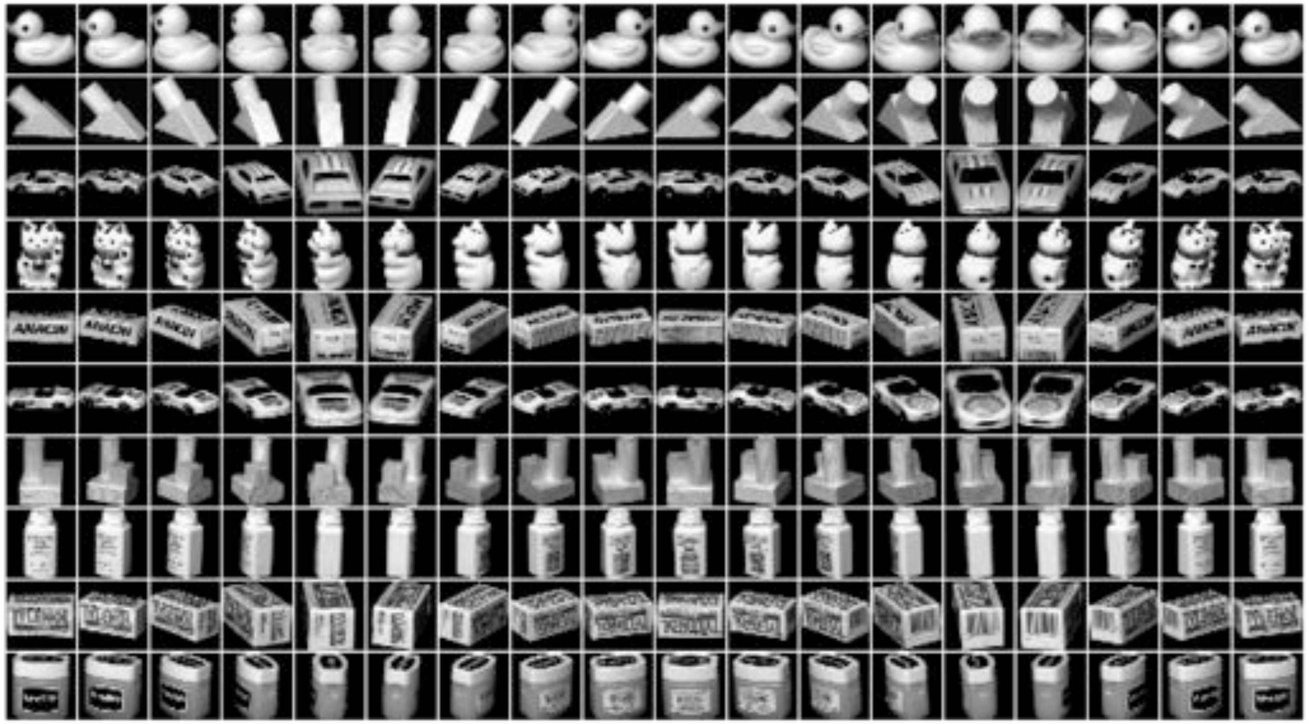
$$\hat{\mathbf{x}} = W^T W \, \mathbf{x}$$

is the point on the **hyper-plane** that is closest to $\mathbf{x}$.

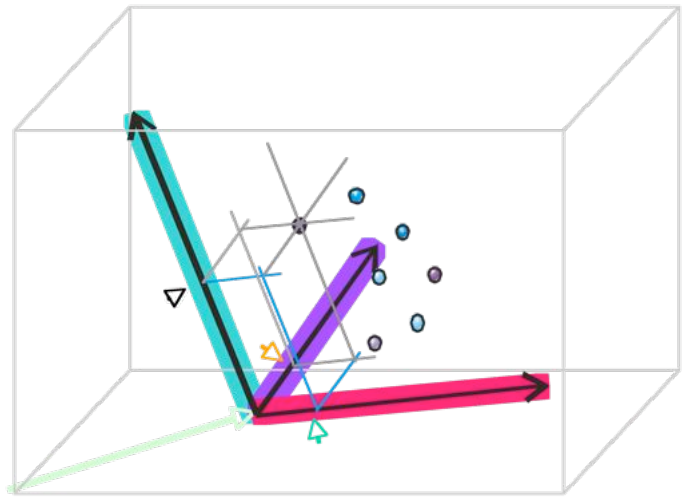
The error of the reconstruction is then

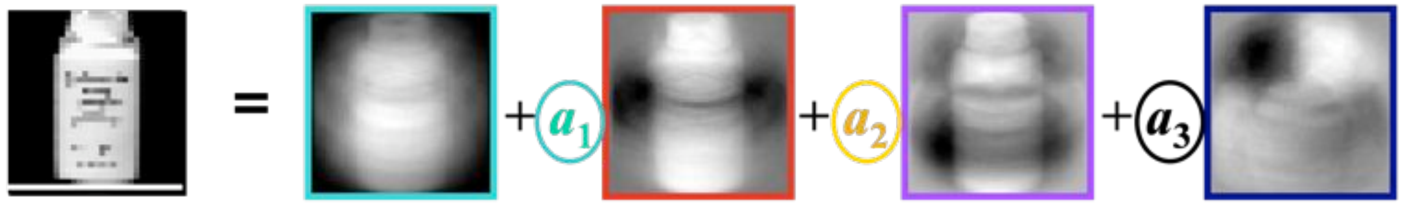
$$\|\mathbf{x} - W^T W \mathbf{x}\|$$

Object representation



PCA representation





Object detection: distance to eigenspace

Slide a window over the image and classify the window as object or non-object as follows:

1. Project window, $\mathbf{x}$, to the eigen-subspace and reconstruct.

$$\hat{\mathbf{x}} = W^T W \mathbf{x}$$

2. Compute the reprojection error $\epsilon = \|\mathbf{x} - \hat{\mathbf{x}}\|$.

3. Local minima of the reprojection error over all image locations $\Longrightarrow$ object location.

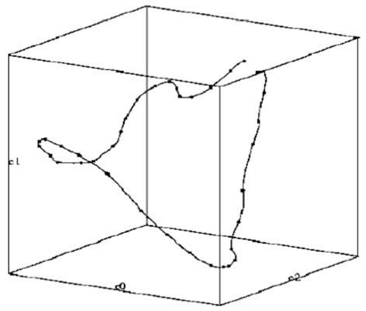
4. Repeat at different scales.

Object identification: distance in eigenspace

Object represented by its coordinates in an k-dimensional eigenspace.

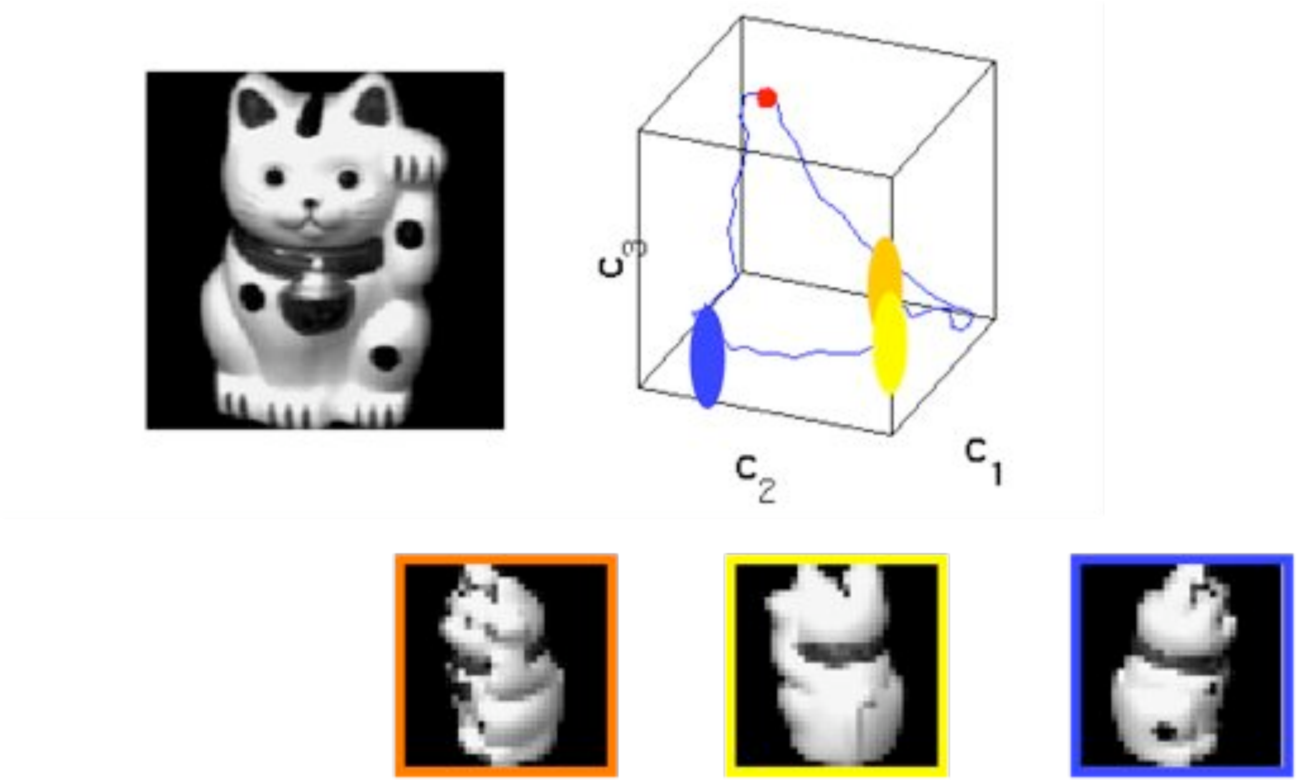
Example:

A parametric eigenspace is a set of points representing individual objects which differ according to some parameter, such as orientation, pose, or illumination.



Estimate novel object properties by finding the nearest neighbour in the eigenspace.

Parametric eigenspace



Object identifcation / pose estimation

Eigenfaces: Key ideas

- Assume that most faces lie on a low-dimensional subspace determined by the $k < d$ directions of the maximum variance.
- Use PCA to determine the vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$ that span this subspace.

$$\mathbf{x} \approx \boldsymbol{\mu} + a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \cdots + a_k\mathbf{v}_k$$

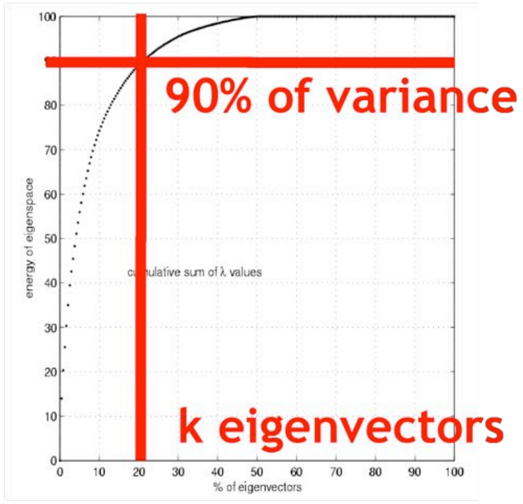
- Represent each face using its **face space** coordinates $\mathbf{z} = (a_1, a_2, \ldots, a_k)$.
- Can then perform face recognition using nearest-neighbour recognition in **face space**.

Choosing the dimension k

How many eigenfaces to use ?

Look at the decay of the eigenvalues

- the eigenvalues tell you the amount of variance *in the direction* of that eigenface
- ignore eigenfaces with low variance.

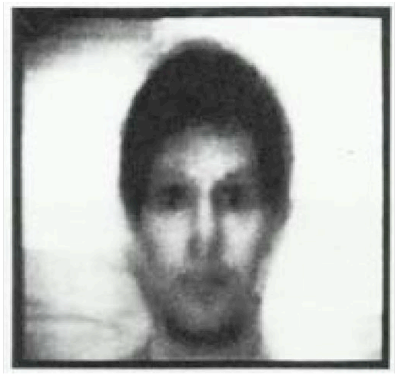


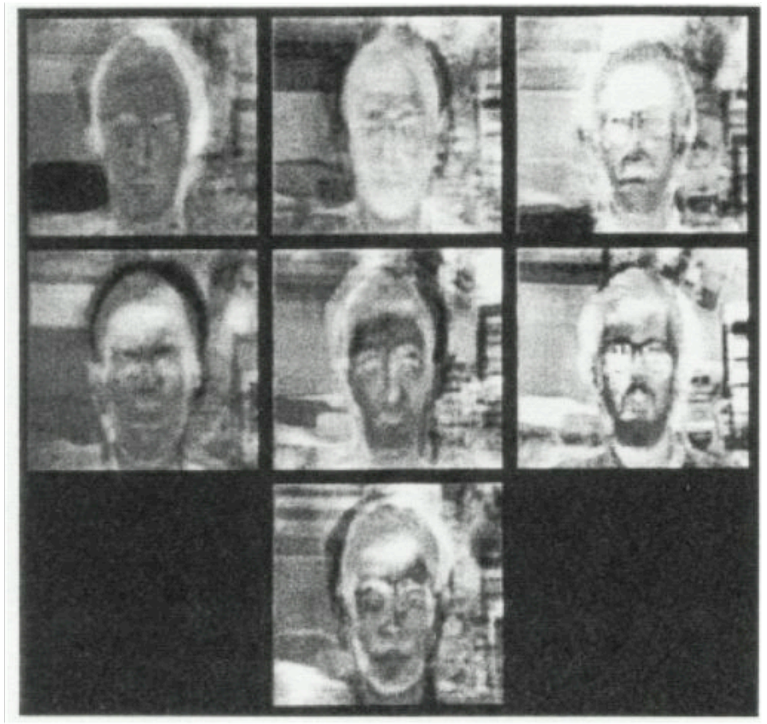
Cumulative influence of eigenvectors

Eigenfaces example

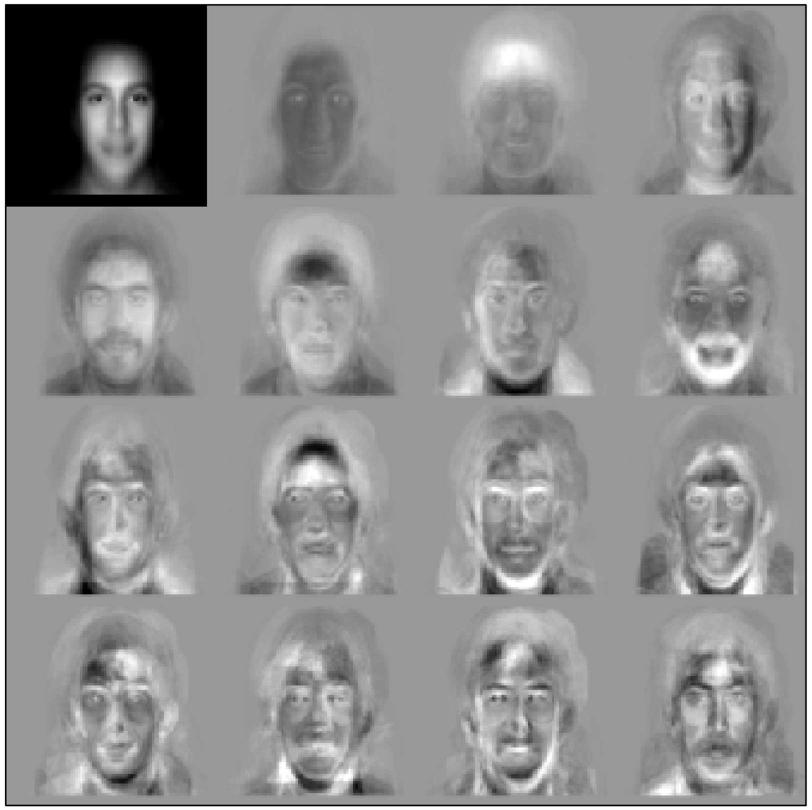
Let $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ be a collection of feature vectors. Each feature vector $\mathbf{x}_i \in [0, 255]^{WH}$ corresponds to the pixel values of a visual image $(W \times H)$ of a face.

Eigenfaces example


Mean face: μ


Top eigenvectors: $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$

Example 2, better alignment


mean and **eigenfaces**

Eigenfaces example

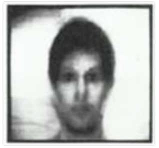
Face $\mathbf{x}$ in *face space* coordinates

novel face

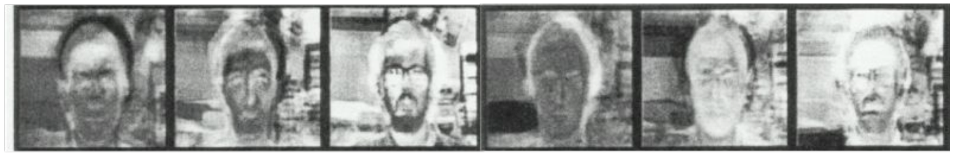
$$\mathbf{x} \rightarrow (\mathbf{v}_1^T(\mathbf{x}-\boldsymbol{\mu}), \ldots, \mathbf{v}_k^T(\mathbf{x}-\boldsymbol{\mu})) = (a_1, \ldots, a_k)$$

Reconstruction

=



+



$$\mathbf{x} = \boldsymbol{\mu} + \sum_{i=1}^{k} a_i \, \mathbf{v}_i$$

Summary: Recognition with eigenfaces

Process labelled training images:

- Find mean $\boldsymbol{\mu}$ and covariance matrix $\Sigma_{\mathbf{x}}$
- Find k principal components (eigenvectors of Σ) $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$
- Project each training image $\mathbf{x}_i$ onto subspace spanned by principal components:

$$(a_{i1}, \ldots, a_{ik}) = (\mathbf{v}_1^T(\mathbf{x}_i - \boldsymbol{\mu}), \ldots, \mathbf{v}_k^T(\mathbf{x}_k - \boldsymbol{\mu}))$$

Summary: Recognition with eigenfaces

Given novel image $\mathbf{x}$:

- Project onto subspace

$$(a_1, \ldots, a_k) = (\mathbf{v}_1^T(\mathbf{x} - \boldsymbol{\mu}), \ldots, \mathbf{v}_k^T(\mathbf{x} - \boldsymbol{\mu}))$$

- **Optional**: check reconstruction error $\|\mathbf{x} - \hat{\mathbf{x}}\|$ to determine whether image is really a face.

- Classify as closest training face in k-dimensional subspace.

Important footnote

Don't really implement PCA this way! Why?

1. How big is Σ?
   - $d \times d$, where d is the number of pixels in an image. Can be a big number.
   - However, we only have n training examples.
     Typically $n \ll d^2 \implies \Sigma$ will a rank of most n!

2. You only need the first k eigenvectors.

Remember SVD

Any arbitrary $k \times n$ matrix X can be converted to the product of an orthogonal matrix, a diagonal matrix and another orthogonal matrix via singular value decomposition:

$$X = USV^T$$

$S \sim$ a diagonal matrix $(k \times n)$ with non-negative entries $(\sigma_1, \sigma_2, \ldots, \sigma_s)$

where $s = \min(k, n)$ are called the singular values.

Singular values are the square roots of the eigenvalues of XX^T

$U \sim$ a square $(k \times k)$ orthogonal matrix, **columns of U are the eigenvectors of XX^T**

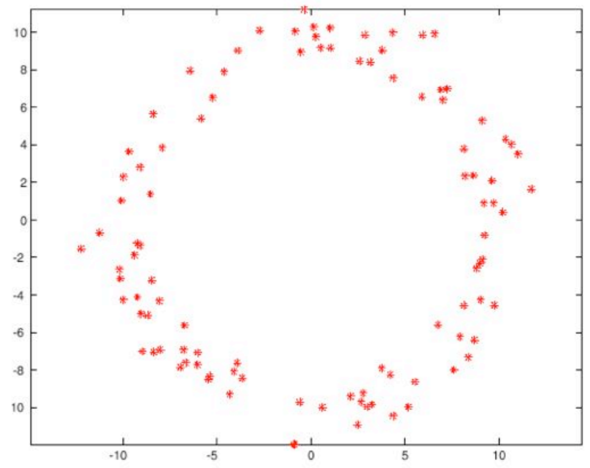
$V \sim$ a square $(n \times n)$ orthogonal matrix, **columns of V are the eigenvectors of X^TX**

SVD PCA - Implementation guide

- Given n data points $\mathbf{x}_i$ each of dimension d.
- Compute the mean $\boldsymbol{\mu} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i$ and subtract it from each data point.
$$\mathbf{x}_i^c = \mathbf{x}_i - \boldsymbol{\mu}$$
- Compute the data matrix X where each column is a data point $\mathbf{x}_i^c$.
- Let $Y = \frac{1}{\sqrt{n}}X^T$ and **perform SVD** such that $Y = WSV^T$.
- The **principal components** are the k singular vectors with highest singular values (rows of V^T).

Limitations

- Global appearance method: it is not robust to misalignment or background variation.
- PCA implicitly assumes the data has a Gaussian distribution



These points are not well described by its principal components

Pen & Paper Assignment

- Details available on the course website.
- You will implement compute the eigenfaces of the Bush dataset and for another face dataset.
- Mail me about any errors you spot in the Exercise notes.
- I will notify the class about errors spotted and corrections via the course website and mailing list.