CS131 Review Section #2

Lectures 7-13

Core Topics (Lectures 7~10)

- Resizing
  - Seam carving
  - Energy vs forward-looking Energy
- Segmentation
  - Clustering is one way to do segmentation
  - Agglomerative clustering
  - k-Means
- Object recognition
  - k-Nearest Neighbors
  - Curse of dimensionality

(disclaimer: this list is not meant to be comprehensive)

Core Topics (Lectures 11~13)

- Dimensionality Reduction
  - SVD
  - PCA

- Face Recognition
  - The Eigenfaces Algorithm
  - LDA

(disclaimer: this list is not meant to be comprehensive)

Resizing

Seam Carving

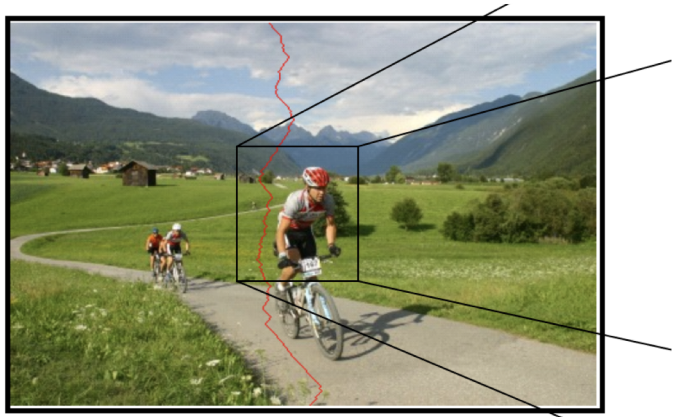
Energy vs forward-looking Energy

Seam Carving

A Seam

- A connected path of pixels from top to bottom (or left to right). Exactly one in each row

$$\mathbf{s}^{\mathbf{x}} = \{s_i^x\}_{i=1}^n = \{(x(i), i)\}_{i=1}^n, \text{ s.t. } \forall i, |x(i) - x(i-1)| \leq 1$$



Seam Carving

- Basic Idea: remove unimportant pixels from the image
  - Unimportant = pixels with less “energy”

$$E_1(\mathbf{I}) = |\frac{\partial}{\partial x}\mathbf{I}| + |\frac{\partial}{\partial y}\mathbf{I}|.$$

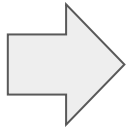
$$\Rightarrow s^* = \arg\min_s E(s)$$

Seam Carving in practice: **standard** energy cost

5	8	12	3
4	2	3	9
7	3	4	2
5	5	7	8

Energy - E(I,j)

Step 1



5	8	12	3
9	2		

Step 2

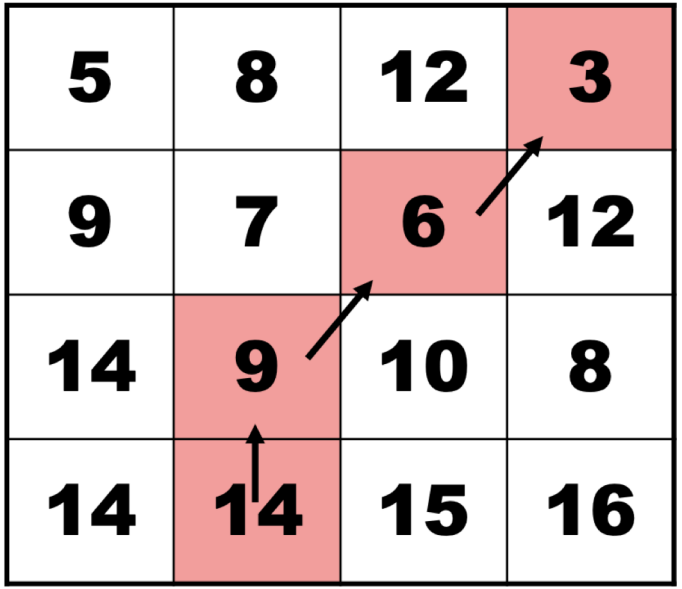
M(i,j) - minimal cost of a seam going through (i,j)

$$\mathbf{M}(i,j) = E(i,j) + \min\left(\mathbf{M}(i-1,j-1), \mathbf{M}(i-1,j), \mathbf{M}(i-1,j+1)\right)$$

Seam Carving in practice: **standard** energy cost

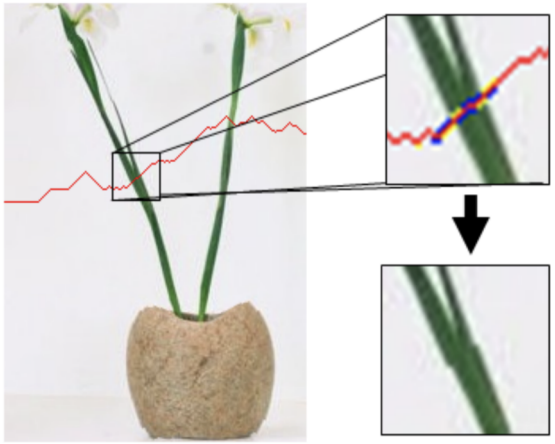
- Backtracking the seam

Step 3



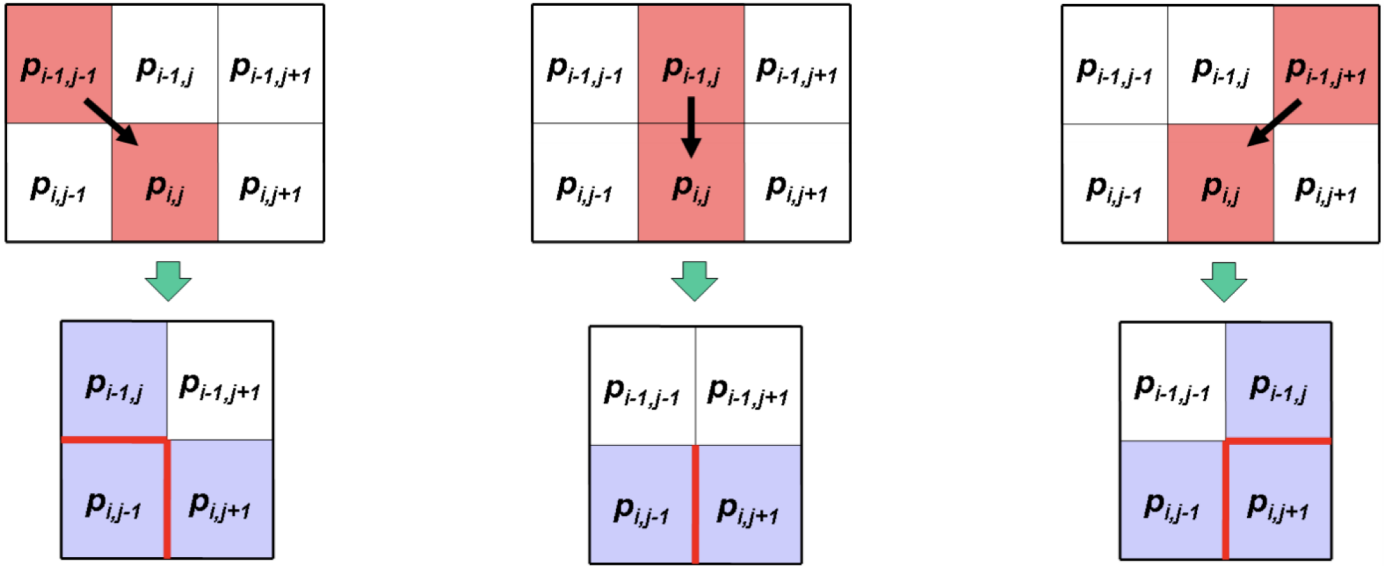
Seam Carving in practice: **forward** energy cost

- Instead of removing the seam of least energy, remove the seam that *inserts the least energy* to the image !



Why is it helpful?

Seam Carving in practice: **forward** energy cost



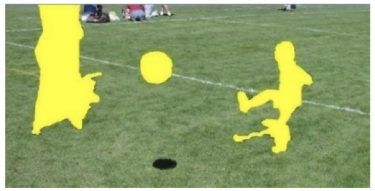
$$C_L(i,j) = |I(i,j+1) - I(i,j-1)| + |I(i-1,j) - I(i,j-1)| \quad C_V(i,j) = |I(i,j+1) - I(i,j-1)| \quad C_R(i,j) = |I(i,j+1) - I(i,j-1)| + |I(i-1,j) - I(i,j+1)|$$

$$M(i,j) = E(i,j) + \min \begin{cases} M(i-1,j-1) + C_L(i,j) \\ M(i-1,j) + C_U(i,j), \\ M(i-1,j+1) + C_R(i,j) \end{cases}$$

Seam Carving in practice: BYOEF: bring your own energy function!

1. Define an energy function **E(I)** (interest, importance, saliency)

2. Use some operator(s) to change the image **I**



Recompose

Setlur et al. [2005]

Seam Carving: what if we're removing **multiple seams**?

SEAM-CARVING(im, n') // size(im) = m x n

1. Do (n-n') times
   2.1. E ← Compute energy map on im
   2.2. s ← Find optimal seam in E
   2.3. im ← Remove s from im
2. Return im

Seam Carving: what if we're **adding** multiple seams (instead of **removing** multiple seams)

We now want to tackle the reverse problem of enlarging an image.
One naive way to approach the problem would be to duplicate the optimal seam iteratively until we reach the desired size.

Taken from HW4

The issue with `enlarge_naive` is that the same seam will be selected again and again, so this low energy seam will be the only to be duplicated.

Another way to get k different seams is to apply the process we used in function `reduce`, and keeping track of the seams we delete progressively. The function `find_seams(image, k)` will find the top k seams for removal iteratively.

Seam Carving Practice #1

Look at HW4 input image:

```
test_img = np.array([[1.0, 2.0, 1.5],
                     [3.0, 1.0, 2.0],
                     [4.0, 0.5, 3.0]])
```

Make sure you can compute by hand:

- The energy matrix - E(i,j)
- The (minimal) cost matrix - M(i,j)
- The optimal seam to remove

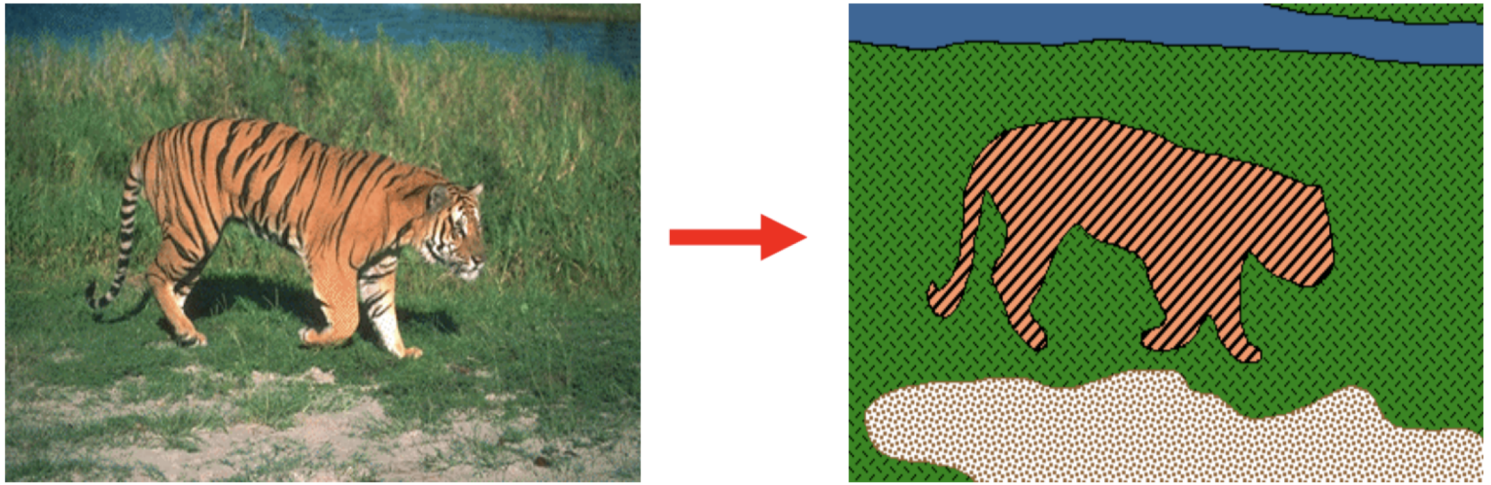
Segmentation: Clustering

Agglomerative clustering

k-Means

Mean-Shift clustering

Segmentation: identify groups of pixels that go together



- Clustering is one way to do segmentation
- There are many different clustering algorithms out there

Segmentation: identify groups of pixels that go together

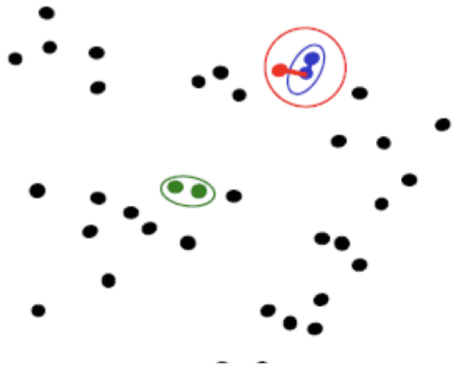
Clustering is an unsupervised learning method. Given items $x_1, \ldots, x_n \in \mathbb{R}^D$, the goal is to group them into clusters. We need a pairwise distance/similarity function between items, and sometimes the desired number of clusters.

When data (e.g. images, objects, documents) are represented by feature vectors, commonly used measures are:

- *Euclidean distance.* $dist(x, x') = \sqrt{\sum (x_i - x'_i)^2}$
- *cosine similarity.* $sim(x, x') = \frac{x^\top x'}{\sqrt{x^\top x}\sqrt{x'^\top x'}}.$

Agglomerative Clustering

– Start with each point as its own cluster and iteratively merge the closest clusters



1. Say "Every point is its own cluster"
2. Find "most similar" pair of clusters
3. Merge it into a parent cluster
4. Repeat

!/\! What does "nearest" clusters mean?

- Single Link -> Long, skinny clusters
- Complete Link -> Tight clusters
- Average Link -> Robust against noise

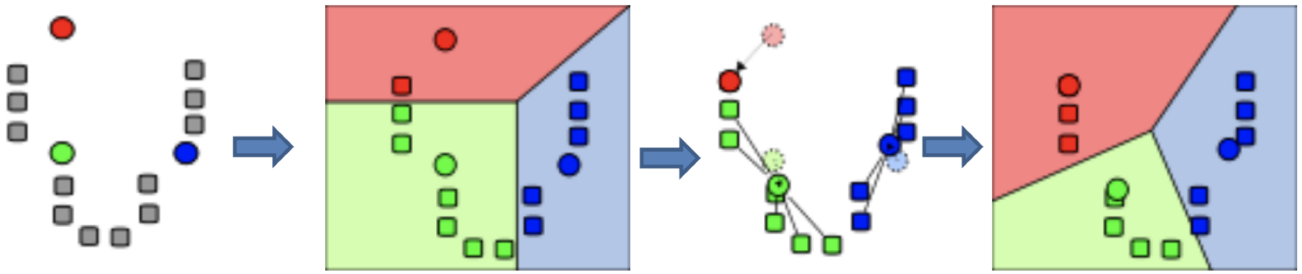
1. Initially each item $x_1, \ldots, x_n$ is in its own cluster $C_1, \ldots, C_n$.
2. Repeat until there is only one cluster left:
3. Merge the nearest clusters, say C_i and C_j.

k-Means

– Iteratively re-assign points to the nearest cluster center

Goal: cluster to minimize variance in data given clusters

– Preserve information



k-Means

1. Initialize ($t = 0$): cluster centers $c_1, \ldots, c_K$
   - Commonly used: random initialization
   - Or greedily choose K to minimize residual
2. Compute δ^t: assign each point to the closest center
   - δ^t denotes the set of assignment for each x_j to cluster c_i at iteration t

$$\delta^t = \underset{\delta}{\operatorname{argmin}} \frac{1}{N} \sum_j^N \sum_i^K \delta_{ij}^{t-1} \left(c_i^{t-1} - x_j \right)^2$$

1. Computer c^t: update cluster centers as the mean of the points

$$c^t = \underset{c}{\operatorname{argmin}} \frac{1}{N} \sum_j^N \sum_i^K \delta_{ij}^{t} \left(c_i^{t-1} - x_j \right)^2$$

1. Update $t = t + 1$, Repeat Step 2-3 till stopped
   - c^t doesn't change anymore.

k-Means

How to choose the number of clusters?

Try different numbers of clusters in a validation set and look at performance.

- Generative
  - How well are points reconstructed from the clusters?

- Discriminative
  - How well do the clusters correspond to labels?
    - Can we correctly classify which pixels belong to the panda?
  - Note: unsupervised clustering does not aim to be discriminative as we don't have the labels.

Mean-Shift clustering

– Estimate modes of pdf

- Iterative Mode Search
  1. Initialize random seed, and window W
  2. Calculate center of gravity (the “mean”) of W: $\sum_{x \in W} xH(x)$
  3. Shift the search window to the mean
  4. Repeat Step 2 until convergence

Mean-Shift clustering

- Find features (color, gradients, texture, etc)
- Initialize windows at individual pixel locations
- Perform mean shift for each window until convergence
- Merge windows that end up near the same “peak” or mode

Clustering Practice #1

Pros and Cons of the various clustering algorithms?

agglomerative

k-means

Mean shift

Clustering Practice #1

Pros and Cons of the various clustering algorithms?

agglomerative

Good
- Simple to implement, widespread application.
- Clusters have adaptive shapes.
- Provides a hierarchy of clusters.
- No need to specify number of clusters in advance.

Bad
- May have imbalanced clusters.
- Still have to choose number of clusters or threshold.
- Does not scale well. Runtime of $O(n^3)$.
- Can get stuck at a local optima.

k-means

- Pros
  - Finds cluster centers that minimize conditional variance (good representation of data)
  - Simple and fast, Easy to implement
- Cons
  - Need to choose K
  - Sensitive to outliers
  - Prone to local minima
  - All clusters have the same parameters (e.g., distance measure is non-adaptive)
  - *Can be slow: each iteration is O(KNd) for N d-dimensional points
- Usage
  - Unsupervised clustering
  - Rarely used for pixel segmentation

Mean shift

- Pros
  - General, application-independent tool
  - Model-free, does not assume any prior shape (spherical, elliptical, etc.) on data clusters
  - Just a single parameter (window size h)
    - h has a physical meaning (unlike k-means)
  - Finds variable number of modes
  - Robust to outliers
- Cons
  - Output depends on window size
  - Window size (bandwidth) selection is not trivial
  - Computationally (relatively) expensive (~2s/image)
  - Does not scale well with dimension of feature space

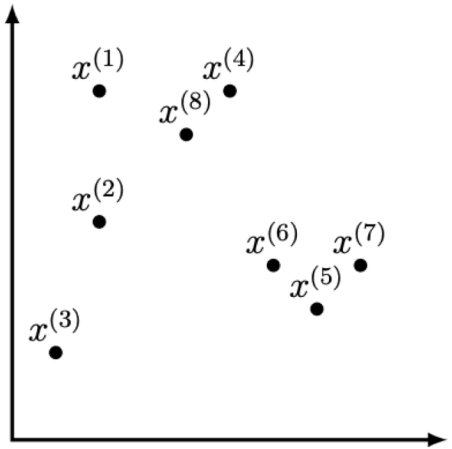
Clustering Practice #2

You have the following 2D data: what clusters does each clustering algorithm output?

$$x^{(1)} = (2, 8), \quad x^{(2)} = (2, 5), \quad x^{(3)} = (1, 2), \quad x^{(4)} = (5, 8),$$
$$x^{(5)} = (7, 3), \quad x^{(6)} = (6, 4), \quad x^{(7)} = (8, 4), \quad x^{(8)} = (4, 7).$$

Initialization:

$$\mu^{(1)} = x^{(3)}$$
$$\mu^{(2)} = x^{(4)}$$
$$\mu^{(3)} = x^{(6)}$$



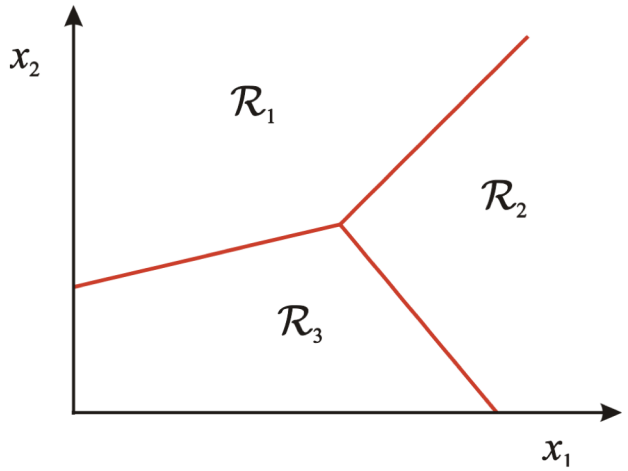
Object Recognition
k-Nearest Neighbors
Curse of dimensionality

- Design algorithms that have the capability to:
  - –Classify images or videos
  - –Detect and localize objects
  - –Estimate semantic and geometrical attributes
  - – Classify human activities and events

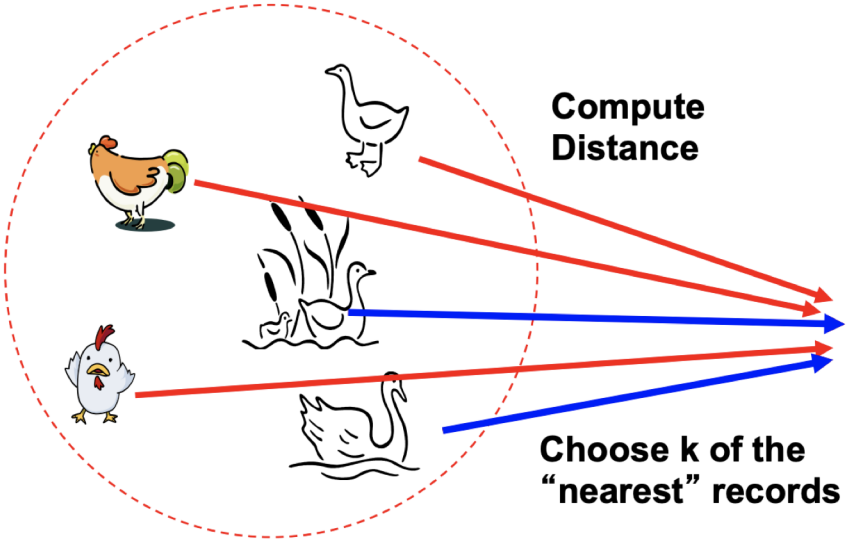
k-Nearest Neighbor lets us do classification

Classification

- Assign input vector to one of two or more classes
- Any decision rule divides input space into *decision regions* separated by *decision boundaries*



k-Nearest Neighbor is one way to do classification

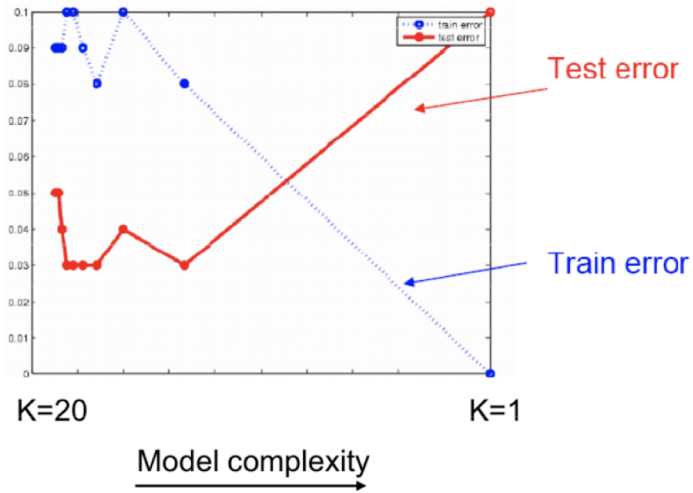


- Simple, a good one to try first
- Very flexible decision boundaries

- Choosing the value of k:
  - If too small, sensitive to noise points (overfitting)
  - If too large, neighborhood may include points from other classes (underfitting)

k-Nearest Neighbor

– Solution: cross validate!



- Choosing the value of k:
  - If too small, sensitive to noise points
  - If too large, neighborhood may include points from other classes

k-Nearest Neighbor

- Choosing the value of k:
  - If too small, sensitive to noise points
  - If too large, neighborhood may include points from other classes
  - Solution: cross validate!
- Can produce counter-intuitive results (using Euclidean measure)
  - Solution: normalize the vectors to unit length
- Curse of Dimensionality
  - Solution: no good one

Underfitting vs Overfitting, Bias vs Variance

- **Underfitting:** model is too “simple” to represent all the relevant class characteristics
  - High bias and low variance
  - High training error and high test error
- **Overfitting:** model is too “complex” and fits irrelevant characteristics (noise) in the data
  - Low bias and high variance
  - Low training error and high test error

No free lunch theorem

- No classifier is inherently better than any other: you need to make assumptions to generalize

- Three kinds of error
  - Inherent: unavoidable
  - Bias: due to over-simplifications
  - Variance: due to inability to perfectly estimate parameters from limited data

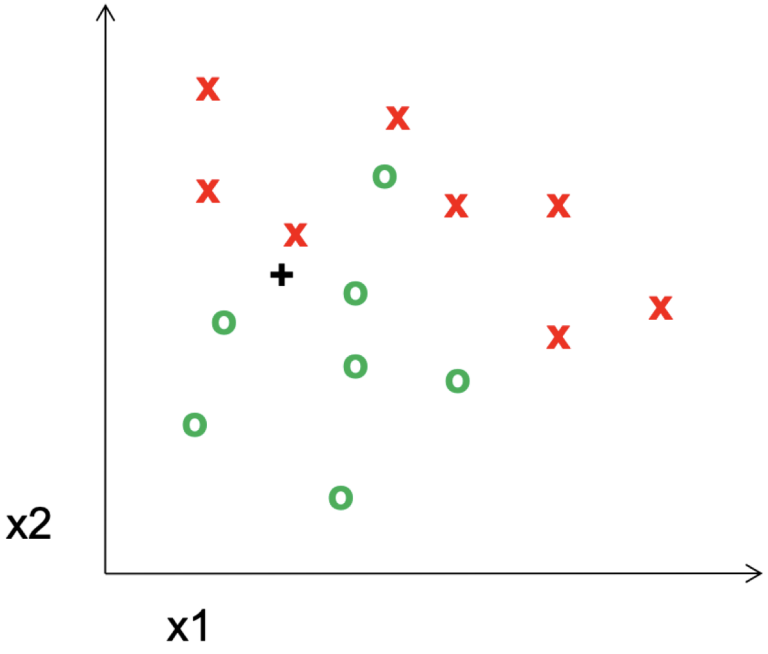
Curse of Dimensionality

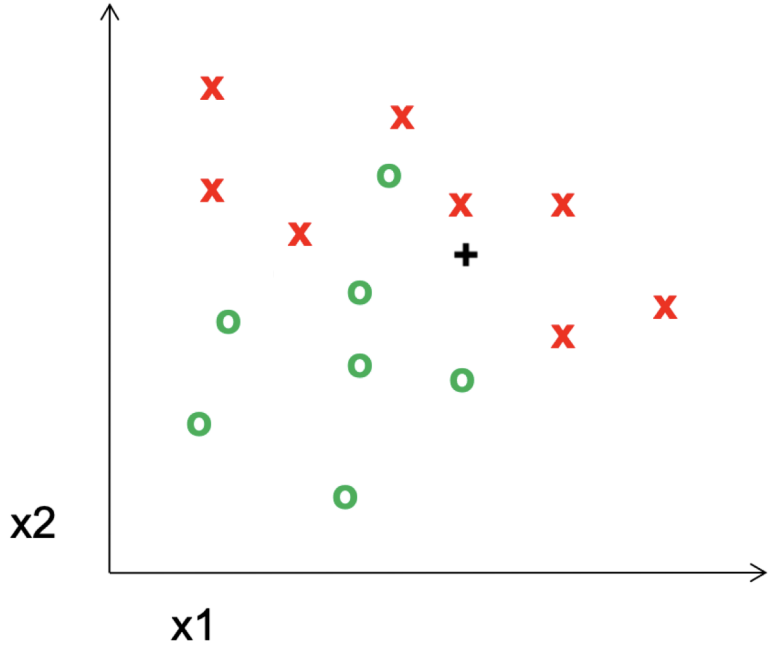
- In a high-dimensional space, most points [taken from a random finite set of N data points inside a finite volume] are far away from each other.

- As the number of features or dimensions grows, the amount of data we need to generalize accurately grows exponentially

Object Recognition Practice

What is the label given to this test point using 1-Nearest Neighbor?

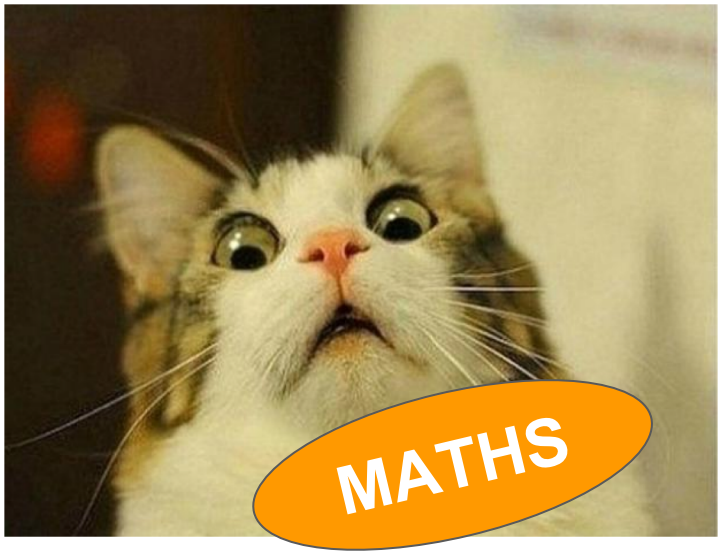




Dimensionality Reduction

SVD

PCA



SVD

$$\underset{U}{\begin{bmatrix} -.39 & -.92 \\ -.92 & .39 \end{bmatrix}} \times \underset{\Sigma}{\begin{bmatrix} 9.51 & 0 & 0 \\ 0 & .77 & 0 \end{bmatrix}} \times \underset{V^T}{\begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix}} = \underset{A}{\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}}$$

- A has shape m x n
- Σ is a diagonal matrix of shape m x n
- U and V are orthogonal matrices:
  - UT . U = I
  - VT . V = I

SVD can be used to compress images

$$U\Sigma \quad V^T \quad A_{partial}$$

$$\begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix} \times \begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix} \quad \begin{bmatrix} 1.6 & 2.1 & 2.6 \\ 3.8 & 5.0 & 6.2 \end{bmatrix}$$

$$A$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

$$U\Sigma \quad V^T \quad A_{partial}$$

$$\begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix} \times \begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix} \quad \begin{bmatrix} -.6 & -.1 & .4 \\ .2 & 0 & -.2 \end{bmatrix}$$

SVD Practice #1

I want to use SVD to reduce the dimensionality of an image.

I apply SVD to an image of size m x n.

I decide to keep only the top 20 eigenvalues.

What is the size of the output image?

How many floats do I need to store the image?

PCA

Training

- Given sample $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, $x_i \in \mathcal{R}^d$
  - compute sample mean: $\hat{\mu} = \frac{1}{n}\sum_i(\mathbf{x}_i)$
  - compute sample covariance: $\hat{\Sigma} = \frac{1}{n}\sum_i(\mathbf{x}_i - \hat{\mu})(\mathbf{x}_i - \hat{\mu})^T$
  - compute eigenvalues and eigenvectors of $\hat{\Sigma}$

$$\hat{\Sigma} = \Phi\Lambda\Phi^T, \ \Lambda = diag(\sigma_1^2, \ldots, \sigma_n^2) \ \ \Phi^T\Phi = I$$

  - order eigenvalues $\sigma_1^2 > \ldots > \sigma_n^2$
  - if, for a certain k, $\sigma_k << \sigma_1$ eliminate the eigenvalues and eigenvectors above k.

Testing

- Given principal compoenents $\phi_i, i \in 1, \ldots, k$ and a test sample $\mathcal{T} = \{\mathbf{t}_1, \ldots, \mathbf{t}_n\}$, $t_i \in \mathcal{R}^d$
  - subtract mean to each point $\mathbf{t}'_i = \mathbf{t}_i - \hat{\mu}$
  - project onto eigenvector space $\mathbf{y}_i = \mathbf{A}\mathbf{t}'_i$ where

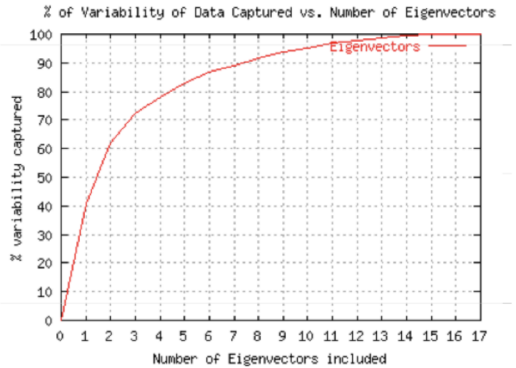
$$\mathbf{A} = \begin{bmatrix} \phi_1^T \\ \vdots \\ \phi_k^T \end{bmatrix}$$

  - use $\mathcal{T}' = \{\mathbf{y}_1, \ldots \mathbf{y}_n\}$ to estimate class conditional densities and do all further processing on $\mathbf{y}$.

PCA

Rule of thumb for finding the number of PCA components

- A natural measure is to pick the eigenvectors that explain p% of the data variability
  - Can be done by plotting the ratio r_k as a function of k

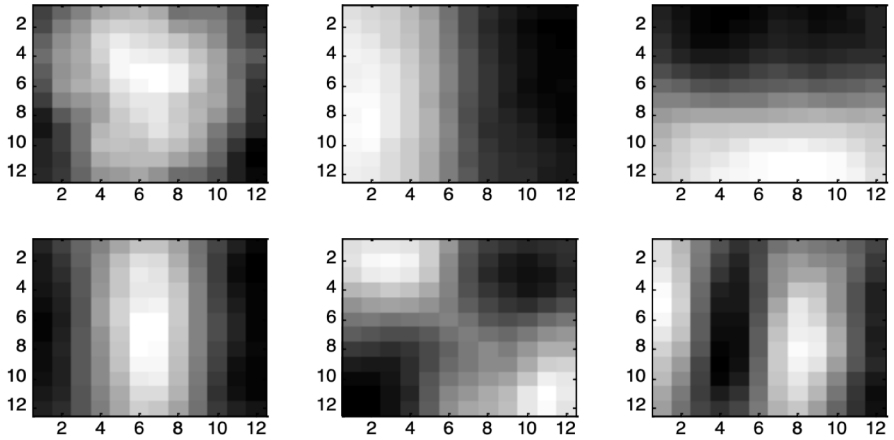


$$r_k = \frac{\sum_{i=1}^{k} \lambda_i^2}{\sum_{i=1}^{n} \lambda_i^2}$$

  - E.g. we need 3 eigenvectors to cover 70% of the variability of this dataset

PCA on 1 image: the “vectors” whose dimensionality we are reducing are the patches

One image is made of many patches of size 12x12

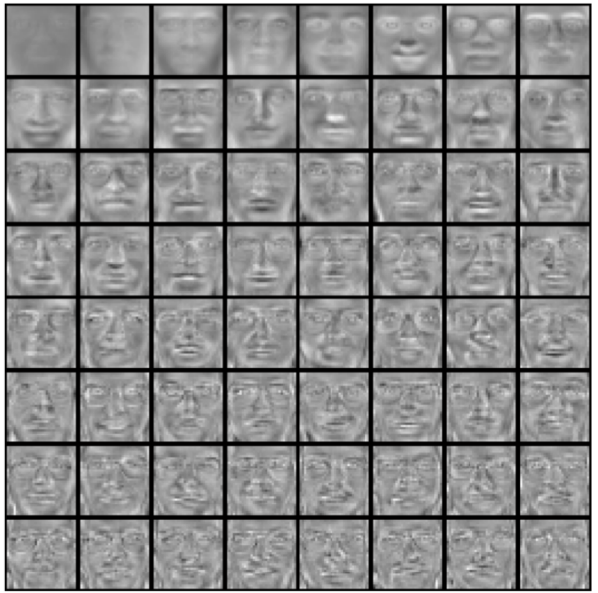


Top Eigenvectors

Each image patch will be projected on the top-k eigenvectors

PCA on a dataset: the “vectors” whose dimensionality we are reducing are the images

Training Images



Top Eigenvectors

Each image will be projected on the top-k eigenvectors

PCA Practice #1

I want to use PCA to reduce the dimensionality of 1 image of size 100x200.

I break down an image into patches of size 20x20: the image has 50 patches.

I decide to project each patch on a subspace of dimension 10 (instead of 20x20).

What is the size of the output image?

How many floats do I need to store the image?

Face Recognition

The Eigenfaces Algorithm

LDA

Detection finds the faces in images $\neq$ Recognition recognizes WHO the person is

The Eigenfaces Algorithm (PCA)

- Assume that most face images lie on a low-dimensional subspace determined by the first k (k<<d) directions of maximum variance
- Use PCA to determine the vectors or “eigenfaces” that span that subspace
- Represent all face images in the dataset as linear combinations of eigenfaces

At test time,

- project compute (centered) projection of query image into eigenface space
- compare projection w with all N training projections: which face is it?

Eigenface Algorithm Practice

Pros and Cons of Eigenface Algorithm?

Pros

- Non-iterative, globally optimal solution

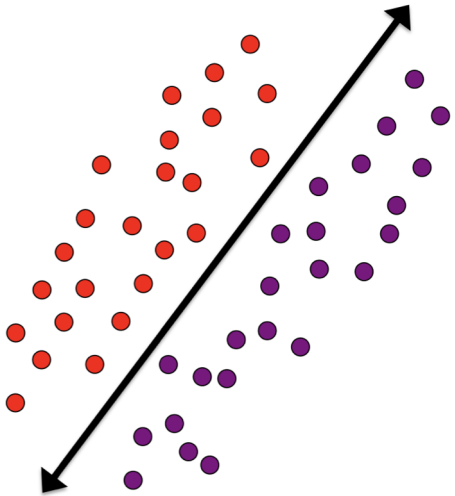
Limitations

- PCA projection is **optimal for reconstruction** from a low dimensional basis, but **may NOT be optimal for discrimination…** Is there a better dimensionality reduction?

Fisherfaces Algorithm (LDA)

- PCA preserves maximum variance

- LDA preserves discrimination
  - Find projection that maximizes scatter between classes and minimizes scatter within classe



$$J(w) = \max \frac{between\ \ class\ \ scatter}{within\ \ class\ \ scatter}$$

Fisherfaces Algorithm (LDA)

- N Sample images: $\{x_1, \cdots, x_N\}$
- C classes: $\{Y_1, Y_2, \ldots, Y_c\}$
- Average of each class: $\mu_i = \frac{1}{N_i} \sum_{x_k \in Y_i} x_k$
- Average of all data: $\mu = \frac{1}{N} \sum_{k=1}^{N} x_k$
- Scatter of class i: $S_i = \sum_{x_k \in Y_i} (x_k - \mu_i)(x_k - \mu_i)^T$
- Within class scatter: $S_W = \sum_{i=1}^{c} S_i$
- Between class scatter: $S_B = \sum_{i=1}^{c} \sum_{j \neq i} (\mu_i - \mu_j)(\mu_i - \mu_j)^T$

Fisherfaces Algorithm (LDA)

Recall that we want to learn a projection W such that the projection converts all the points from x to a new space z:

$$z = w^T x \qquad z \in \mathbf{R}^m \quad x \in \mathbf{R}^n$$

$$W_{opt} = \arg\max_{\mathbf{W}} \frac{\left| W^T S_B W \right|}{\left| W^T S_W W \right|}$$

- Solve generalized eigenvector problem:

$$S_B w_i = \lambda_i S_W w_i \qquad i = 1, \ldots, m$$

Linear Algebra

You should know your transformation matrices

You should know what homogeneous coordinates are, and how to go from cartesian to homogeneous coordinates