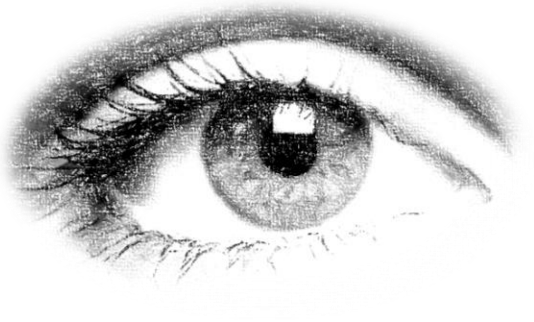


Machine Learning Basics (I)

簡韶逸 Shao-Yi Chien

Department of Electrical Engineering

National Taiwan University

References and Slide Credits

- Slides from *Deep Learning for Computer Vision*, Prof. Yu-Chiang Frank Wang, EE, National Taiwan University
- Slides from *CE 5554 / ECE 4554: Computer Vision*, Prof. J.-B. Huang, Virginia Tech
- Slides from *CSE 576 Computer Vision*, Prof. Steve Seitz and Prof. Rick Szeliski, U. Washington
- Slides from EECS 498-007/598-005 Deep Learning for Computer Vision, Prof. Justin Johnson
- Slides from CS291A Introduction to Pattern Recognition, Artificial Neural Networks, and Machine Learning, Prof. Professor Yuan-Fang Wang, UCSB
- Duda et al., *Pattern Classification*
- Bishop*, Pattern Recognition and Machine Learning*
- Reference papers

Outline

- Overview of recognition/classification pipeline
- Overview of machine learning
- From probability to Bayes decision rule
- Nonparametric techniques: Parzen window and nearest neighbor
- Unsupervised learning and supervised learning
- Unsupervised learning
    - Clustering: k-means
    - Dimension reduction: PCA and LDA
- Training, testing, & validation
- Supervised learning
    - Linear classification: support vector machine (SVM)
    - Combining models: decision tree, boosting
- Examples

Image Classification

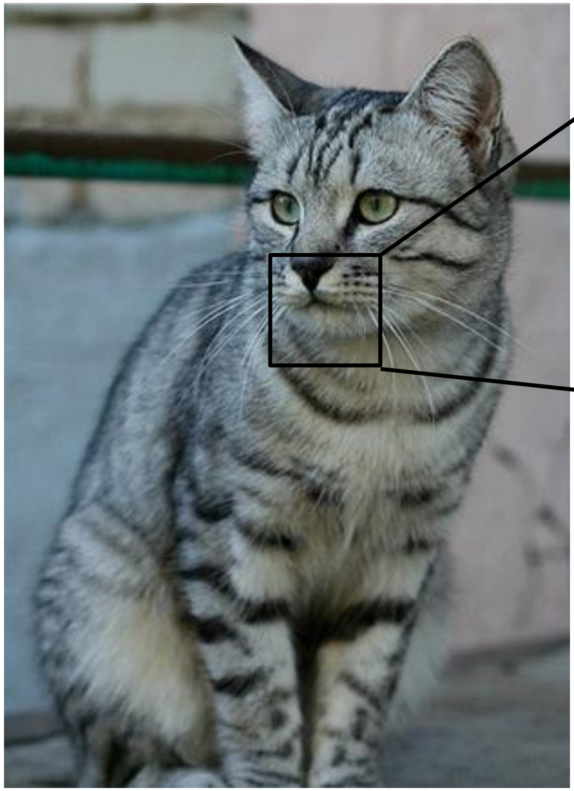
Input: image

This image by Nikita is licensed under CC-BY 2.0

Output: Assign image to one of a fixed set of categories

cat
bird
deer
dog
truck

Problem: Semantic Gap



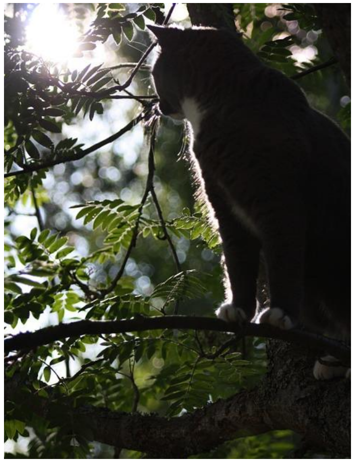
```
[[105 112 108 111 104  99 106  99  96 103 112 119 104  97  93  87]
 [ 91  98 102 106 104  79  98 103  99 105 123 136 110 105  94  85]
 [ 76  85  90 105 128 105  87  96  95  99 115 112 106 103  99  85]
 [ 99  81  81  93 120 131 127 100  95  98 102  99  96  93 101  94]
 [106  91  61  64  69  91  88  85 101 107 109  98  75  84  96  95]
 [114 108  85  55  55  69  64  54  64  87 112 129  98  74  84  91]
 [133 137 147 103  65  81  80  65  52  54  74  84 102  93  85  82]
 [128 137 144 140 109  95  86  70  62  65  63  63  60  73  86 101]
 [125 133 148 137 119 121 117  94  65  79  80  65  54  64  72  98]
 [127 125 131 147 133 127 126 131 111  96  89  75  61  64  72  84]
 [115 114 109 123 150 148 131 118 113 109 100  92  74  65  72  78]
 [ 89  93  90  97 108 147 131 118 113 114 113 109 106  95  77  80]
 [ 63  77  86  81  77  79 102 123 117 115 117 125 125 130 115  87]
 [ 62  65  82  89  78  71  80 101 124 126 119 101 107 114 131 119]
 [ 63  65  75  88  89  71  62  81 120 138 135 105  81  98 110 118]
 [ 87  65  71  87 106  95  69  45  76 130 126 107  92  94 105 112]
 [118  97  82  86 117 123 116  66  41  51  95  93  89  95 102 107]
 [164 146 112  80  82 120 124 104  76  48  45  66  88 101 102 109]
 [157 170 157 120  93  86 114 132 112  97  69  55  70  82  99  94]
 [130 128 134 161 139 100 109 118 121 134 114  87  65  53  69  86]
 [128 112  96 117 150 144 120 115 104 107 102  93  87  81  72  79]
 [123 107  96  86  83 112 153 149 122 109 104  75  80 107 112  99]
 [122 121 102  80  82  86  94 117 145 148 153 102  58  78  92 107]
 [122 164 148 103  71  56  78  83  93 103 119 139 102  61  69  84]]
```

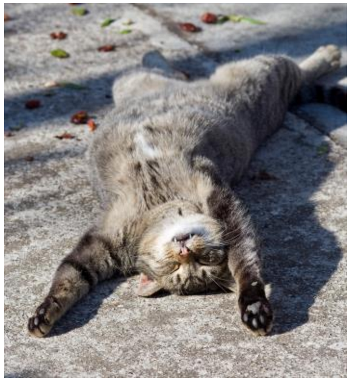
What the computer sees

An image is just a big grid of numbers between [0, 255]
e.g. 800 x 600 x 3
(3 channels RGB)

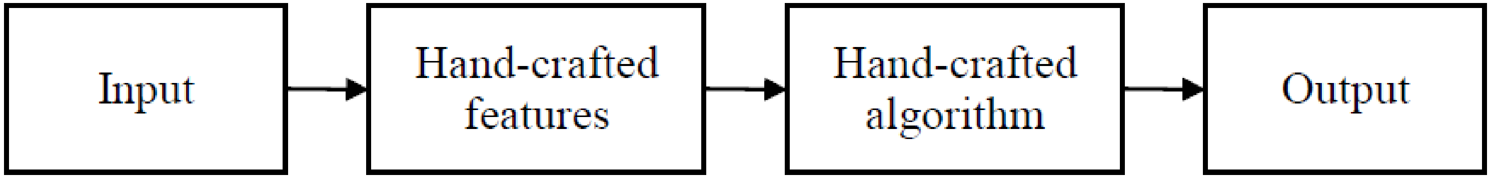
Challenges

- Viewpoint variation
- Intraclass variation
- Fine-grained categories
- Background clutter
- Illumination changes
- Deformation
- Occlusion
- …

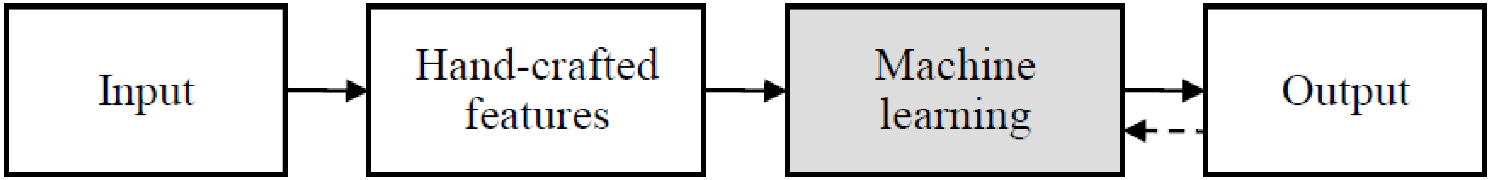




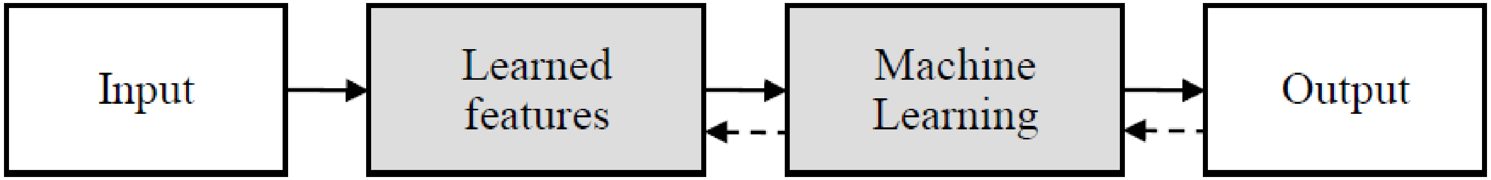
Overview of Recognition Pipeline



(a) Traditional vision pipeline

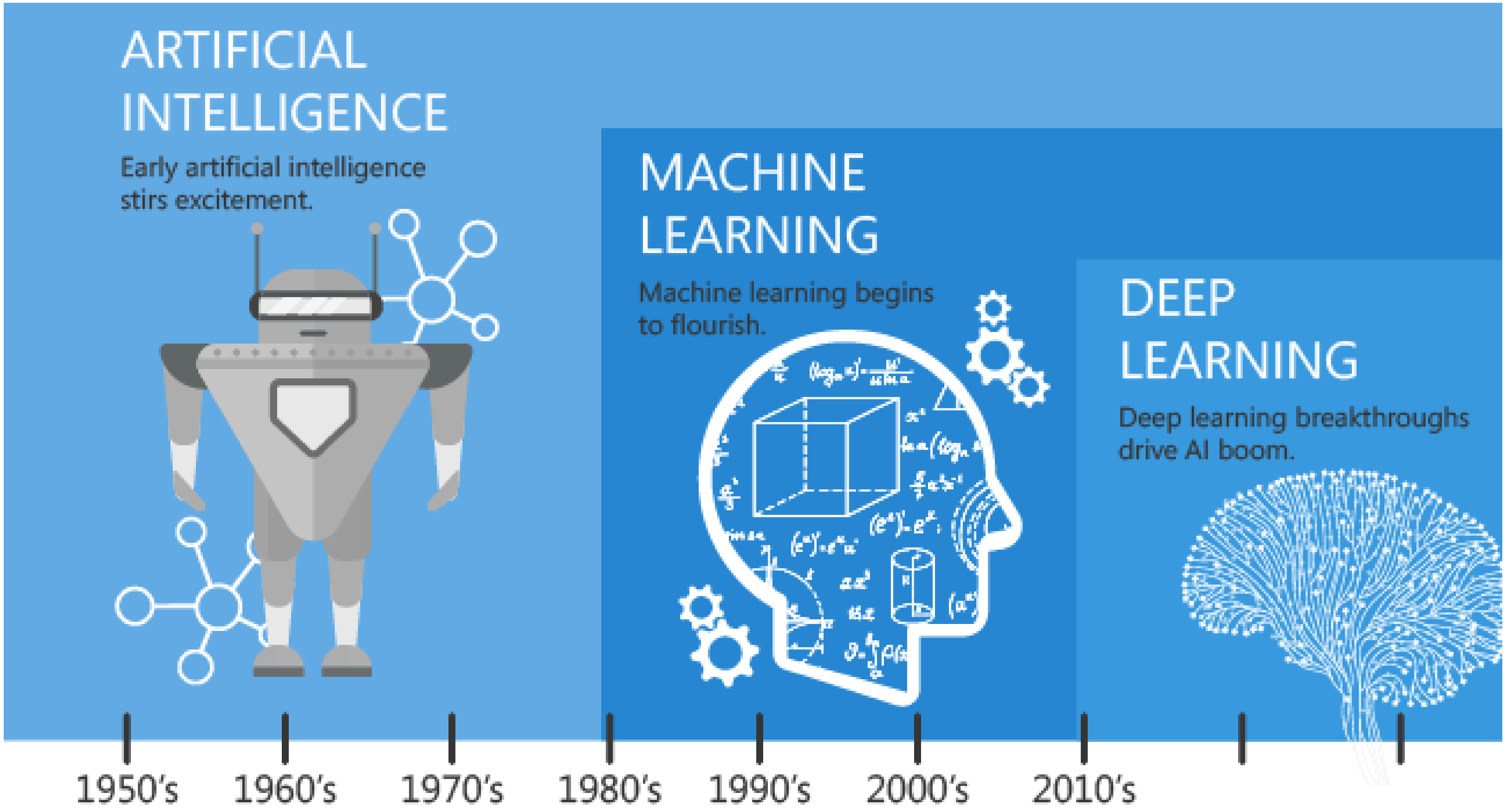


(b) Classic machine learning pipeline



(c) Deep learning pipeline

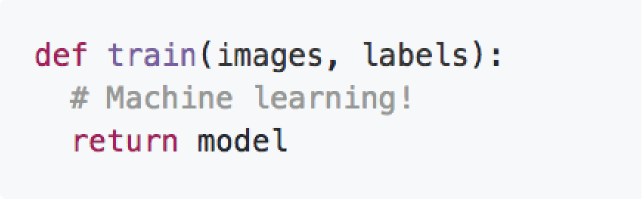
AI, Machine Learning, and Deep Learning


ARTIFICIAL INTELLIGENCE
Early artificial intelligence stirs excitement.
MACHINE LEARNING
Machine learning begins to flourish.
DEEP LEARNING
Deep learning breakthroughs drive AI boom.
1950's
1960's
1970's
1980's
1990's
2000's
2010's

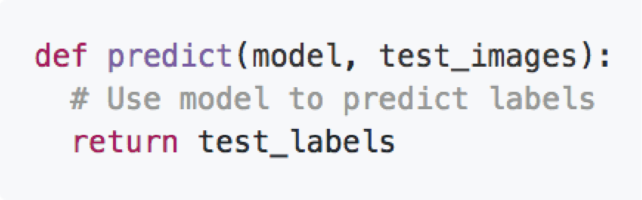
[Kaggle]

Machine Learning: Data-Driven Approach

1. Collect a dataset of images and labels
2. Use Machine Learning to train a classifier
3. Evaluate the classifier on new images

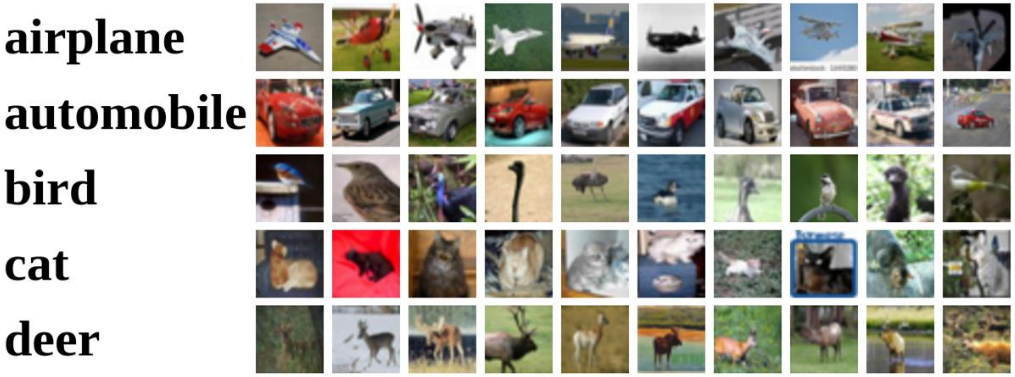


```
def train(images, labels):
  # Machine learning!
  return model
```

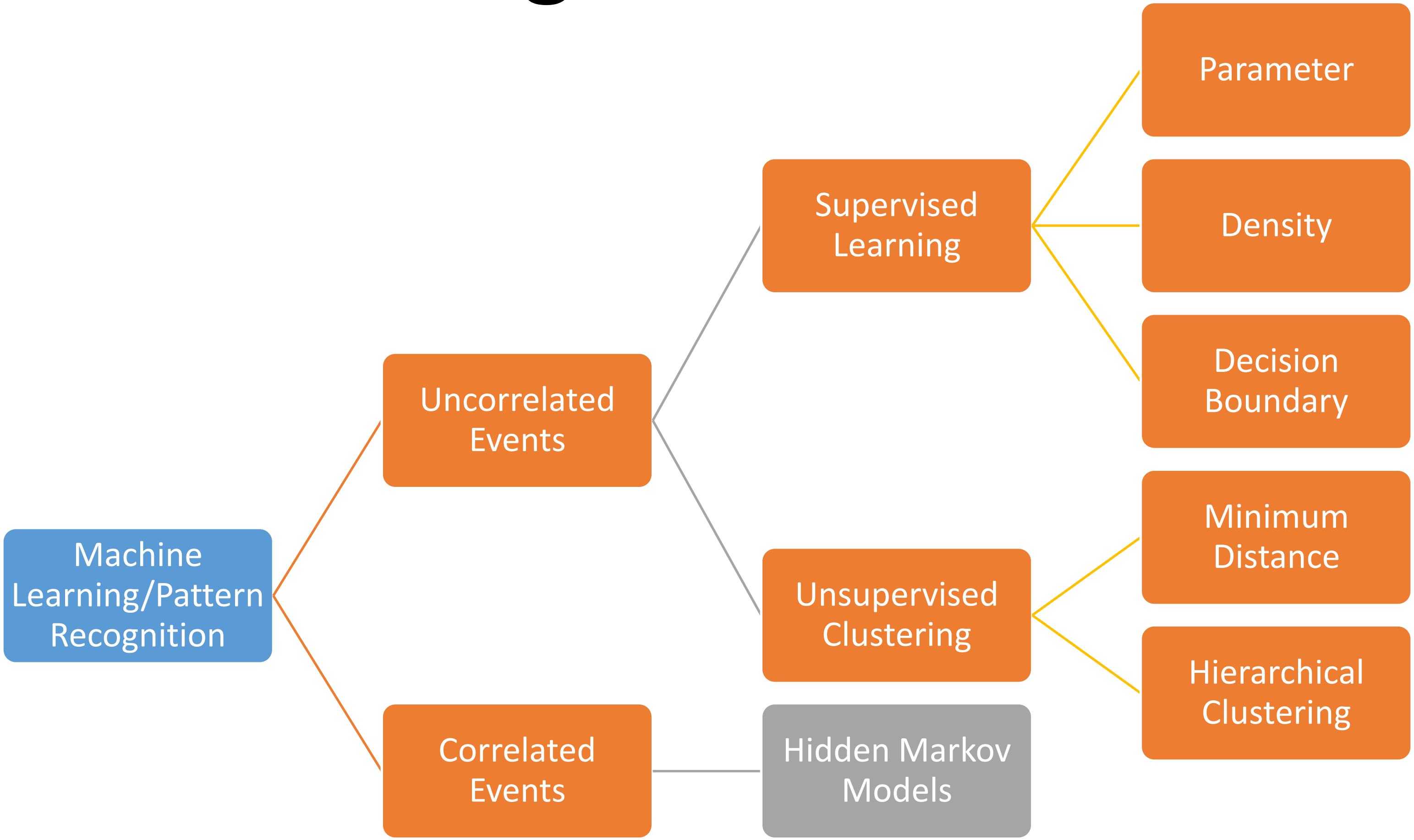


```
def predict(model, test_images):
  # Use model to predict labels
  return test_labels
```

Example training set



Machine Learning/ Pattern Recognition



Machine Learning/Pattern Recognition

Uncorrelated Events

Correlated Events

Supervised Learning

Unsupervised Clustering

Hidden Markov Models

Parameter

Density

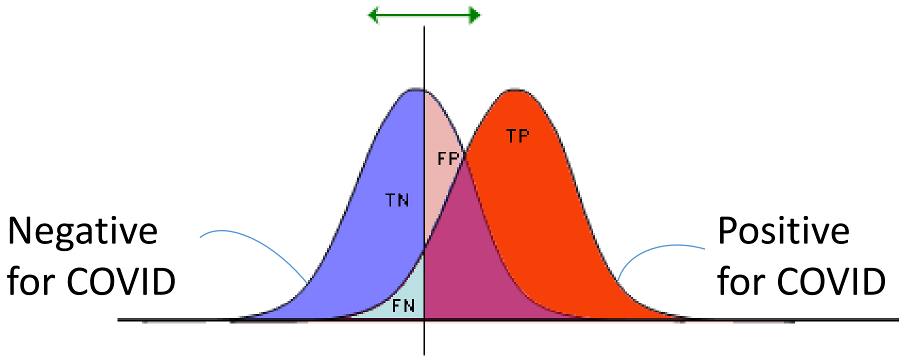
Decision Boundary

Minimum Distance

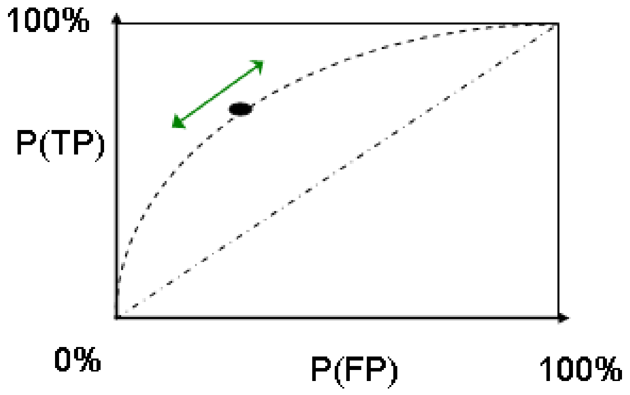
Hierarchical Clustering

From Probability to Bayes Decision Rule

- Example: Testing/Screening of COVID-19
- Distributions between positive/negative test results (e.g., PCR, antibody, etc.)
  - further away from each other
  - more accurate COVID diagnosis



		Ground Truth	
		Positive	Negative
Testing Result	+	TP	FP
	-	FN	TN



Bayesian Decision Theory

- Fundamental statistical approach to classification/detection tasks
- Take a 2-class classification/detection task as an example:
  - Let's see if a student would pass or fail the course of CV.
  - Define a probabilistic variable ω describe the case of pass or fail.
  - That is, $\omega = \omega_1$ for pass, and $\omega = \omega_2$ for fail.
- Prior Probability
  - The **a priori** or **prior** probability reflects the knowledge of how likely we expect a certain state of nature before observation.
  - $P(\omega = \omega_1)$ or simply $P(\omega_1)$ as the **prior** that the next student would pass CV.
  - The priors must exhibit *exclusivity* and *exhaustivity*, i.e.,

$$\sum_{j=1}^{C} p(w_j) = 1$$

  - Equal priors
    - If we have *equal* numbers of students pass/fail CV, then the priors are equal; in other words, the priors are uniform.

$$p(w_1) = p(w_2) = 0.5$$

Prior Probability (cont'd)

- Decision rule based on priors only
  - If the only available info is the prior,
    and the cost of any type of incorrect classification is equal,
    what would be a reasonable decision rule?
  - Decide ω_1 if

$$p(w_1) > p(w_2)$$

otherwise decide ω_2 .

  - What's the incorrect classification rate (or error rate) P_e?

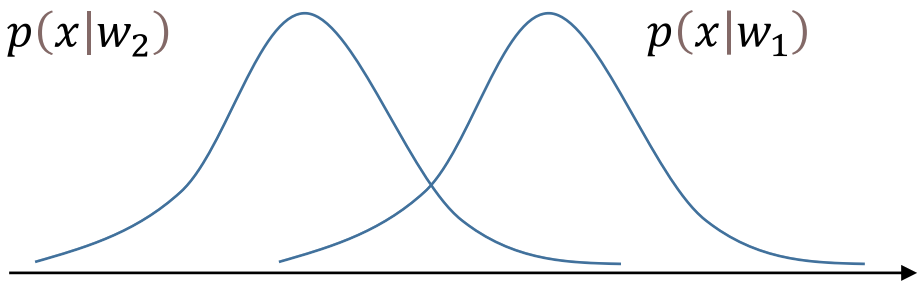
$$P_e = \min\{p(w_1), p(\omega_2)\}$$

Class-Conditional Probability Density (or Likelihood)

- The probability density function (PDF) for input/observation ***x*** given a state of nature ω is written as:

$$p(x|w_1)$$

- Here's (hopefully) the hypothetical class-conditional densities reflecting the time of the students spending on CV who eventually pass/fail this course.



Training data
(observe/collect in advance)

Maximum Likelihood (MLE)

Posterior Probability & Bayes Formula

- If we know the prior distribution and the class-conditional density, can we come up with a better decision rule?
  - Yes We Can!
  - By calculating the posterior probability.

- Posterior probability $P(\omega|\boldsymbol{x})$:
  - The probability of a certain state of nature ω given an observable ***x***.
- Bayes formula:

$$P(w_j,\boldsymbol{x}) = p(x|w_j)p(w_j) = p(w_j|x)p(x)$$

$$P(w_j|\boldsymbol{x}) = \frac{p(x|w_j)P(w_j)}{p(x)}$$

And, we have $\sum_{j=1}^{C} P(\omega_j|\boldsymbol{x}) = 1$.

Decision Rule & Probability of Error

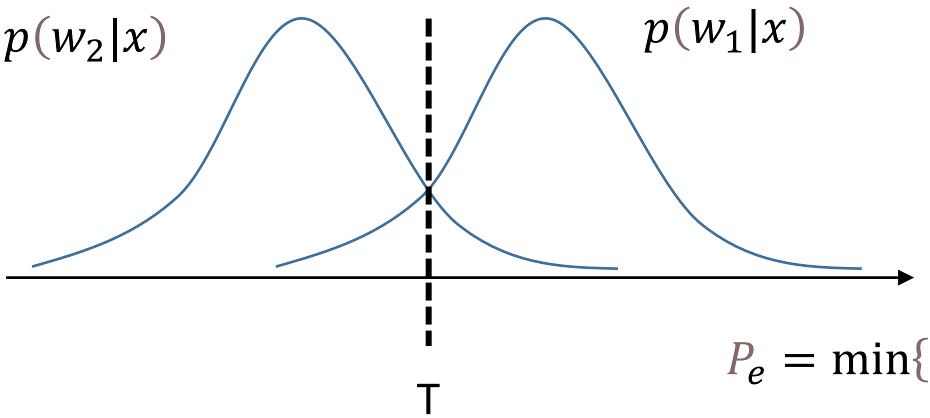
- For a given observable ***x*** (e.g., time you can spend for CV), the decision rule (to take CV or not) will be now based on:

$$\text{Decide } w_1 \text{ if } p(w_1|\boldsymbol{x}) > p(w_2|\boldsymbol{x})$$

$$w^* = \underset{i}{\operatorname{argmax}}\, p(w_i|x)$$

Maximum A Posterior (MAP)

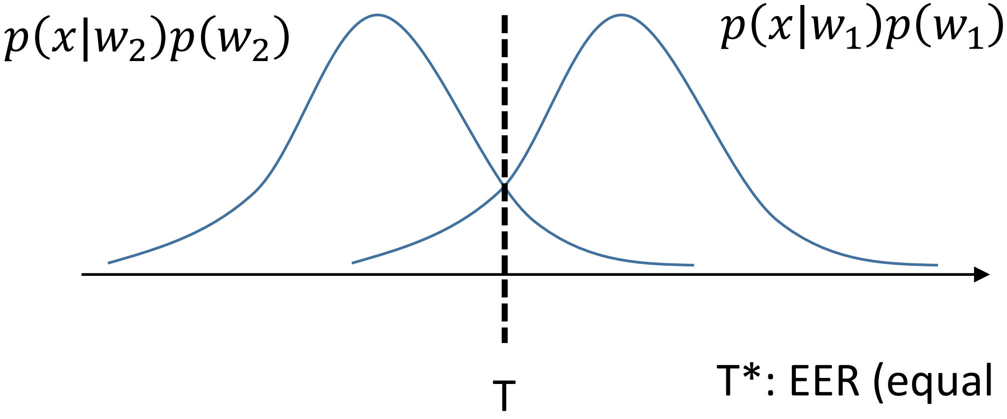
- What's the probability of error P(error) (or P_e)?



$$P_e = \min\{p(w_1|x), P(w_2|x)\} \text{ over all x}$$

From Bayes Decision Rule to Detection Theory

- Hit (detection, TP), false alarm (FA, FP), miss (false reject, FN), rejection (TN)

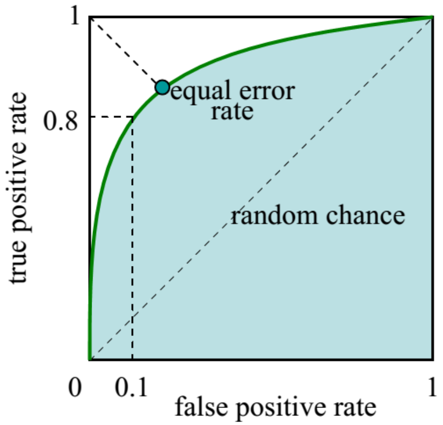


$$TP = \int_T^{\infty} p(x|w_1)p(w_1)\,dx$$

$$FP = \int_T^{\infty} p(x|w_2)p(w_2)\,dx$$

T*: EER (equal error rate): FP=FN

- Receiver Operating Characteristics (ROC)
  - To assess the effectiveness of the designed features/classifiers
  - False alarm (P_{FA} or FP) vs. detection (P_d or TP) rates



Nonparametric Techniques: Parzen Window

- Parzen-window approach to estimate densities: assume e.g. that the region R_n is a d-dimensional hypercube

$$V_n = h_n^d \quad (\mathrm{h_n : length\ of\ the\ edge\ of\ } \Re_n)$$

$$\mathit{Let}\ \varphi(\mathrm{u})\ \text{be the following hypercube window function :}$$

$$\varphi(\mathrm{u}) = \begin{cases} 1 & \left|\mathrm{u_j}\right| \le \dfrac{1}{2} \quad \mathrm{j} = 1,\ldots,\mathrm{d} \\ 0 & \text{otherwise} \end{cases}$$

- $\varphi((x-x_i)/h_n)$ is equal to unity if x_i falls within the hypercube of volume V_n centered at x, and equal to zero otherwise.

- The number of samples in this hypercube is:

$$k_n = \sum_{i=1}^{i=n} \varphi\left(\frac{x - x_i}{h_n}\right)$$

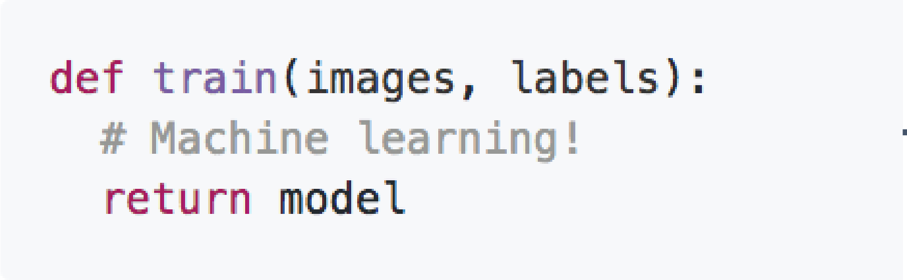
Substituting k_n in $p_n(x) = (k_n/n)/V_n$ we obtain:

$$p_n(x) = \frac{1}{n} \sum_{i=1}^{i=n} \frac{1}{V_n} \varphi\left(\frac{x - x_i}{h_n}\right)$$

$P_n(x)$ estimates $p(x)$ as an average of functions of x and the samples (x_i) $(i = 1, \dots ,n)$.

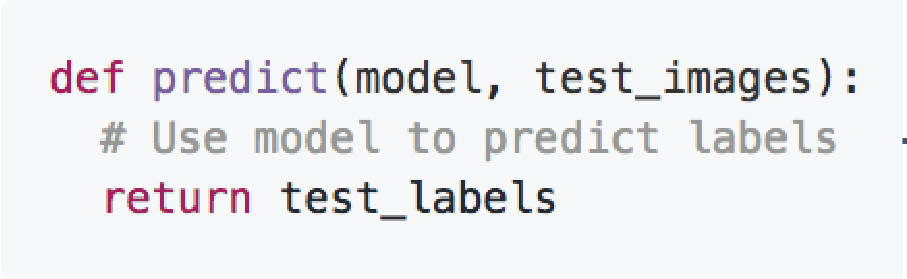
$$p_n(x) \underset{n \to \infty}{\longrightarrow} p(x)$$

Nonparametric Techniques: Nearest Neighborhood



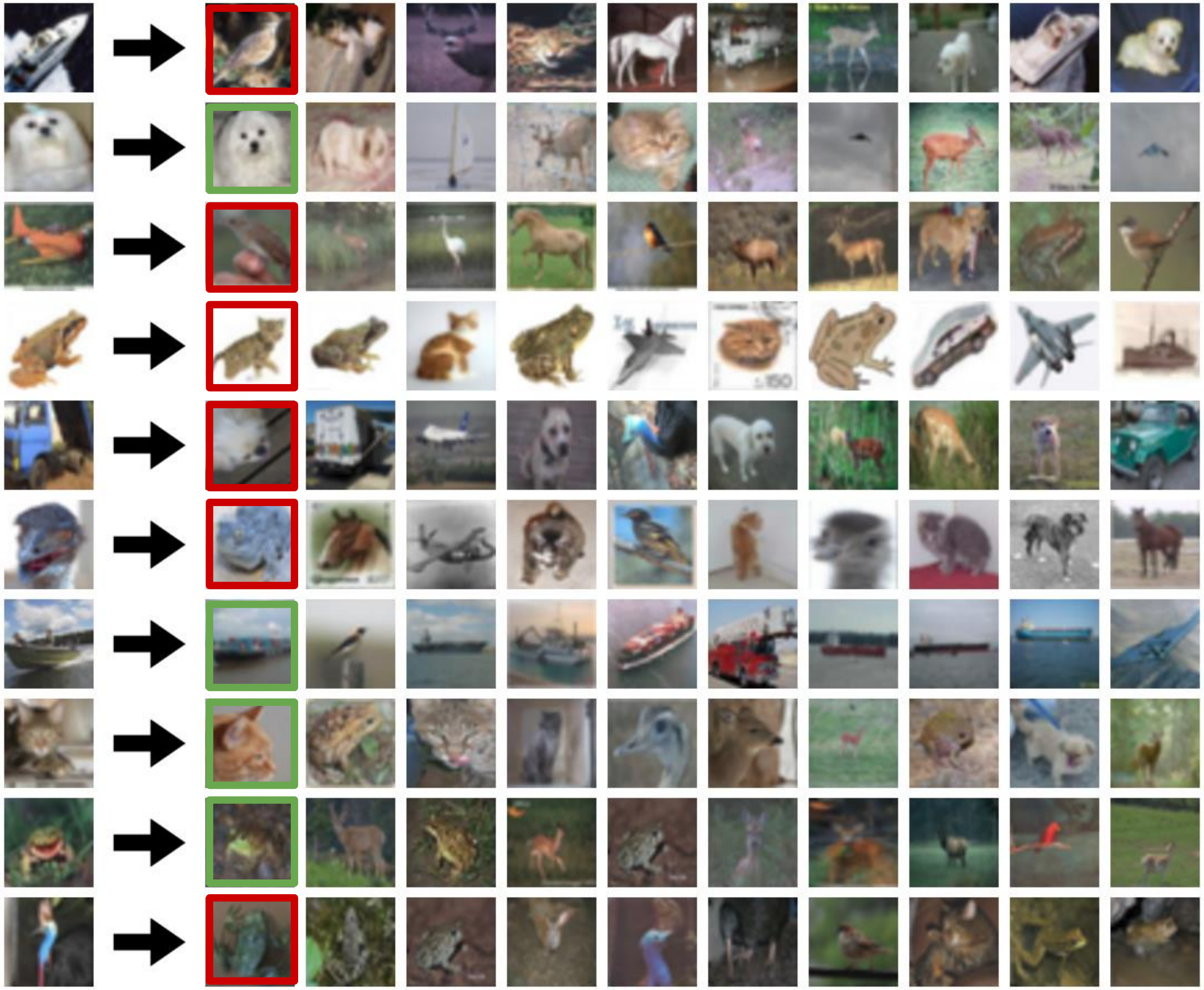
```
def train(images, labels):
  # Machine learning!
  return model
```

Memorize all data and labels

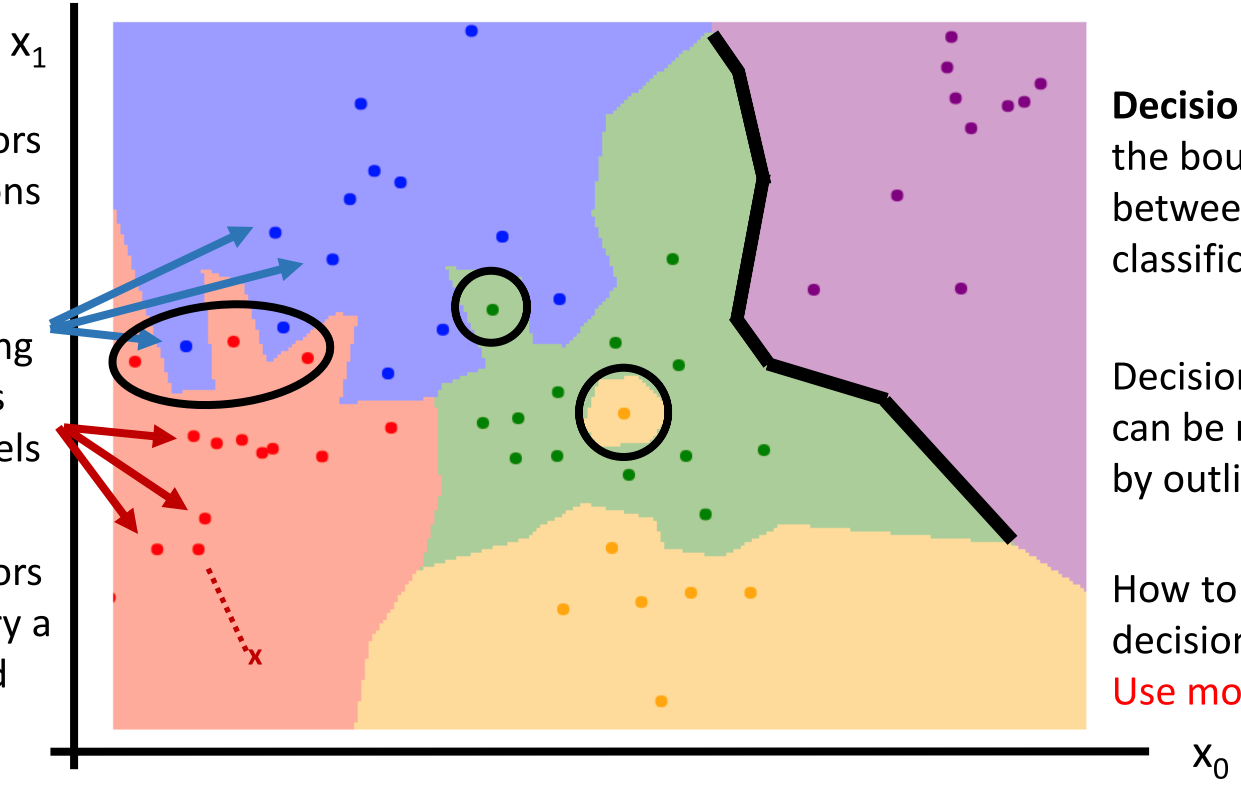


```
def predict(model, test_images):
  # Use model to predict labels
  return test_labels
```

Predict the label of the most similar training image



Nearest Neighbor Decision Boundaries



Nearest neighbors in two dimensions

Points are training examples; colors give training labels

Background colors give the category a test point would be assigned

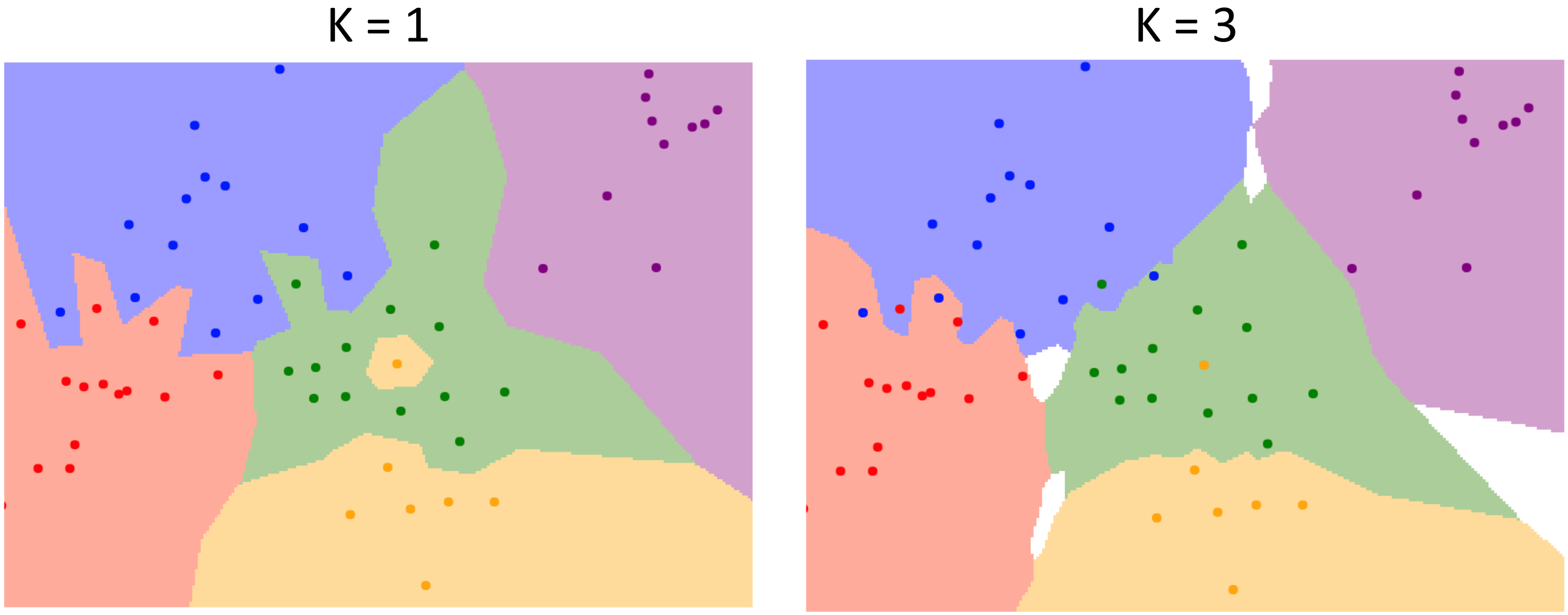
Decision boundary is the boundary between two classification regions

Decision boundaries can be noisy; affected by outliers

How to smooth out decision boundaries? Use more neighbors!

K-Nearest Neighbors (kNN)

- Instead of copying label from nearest neighbor, take majority vote from K closest points



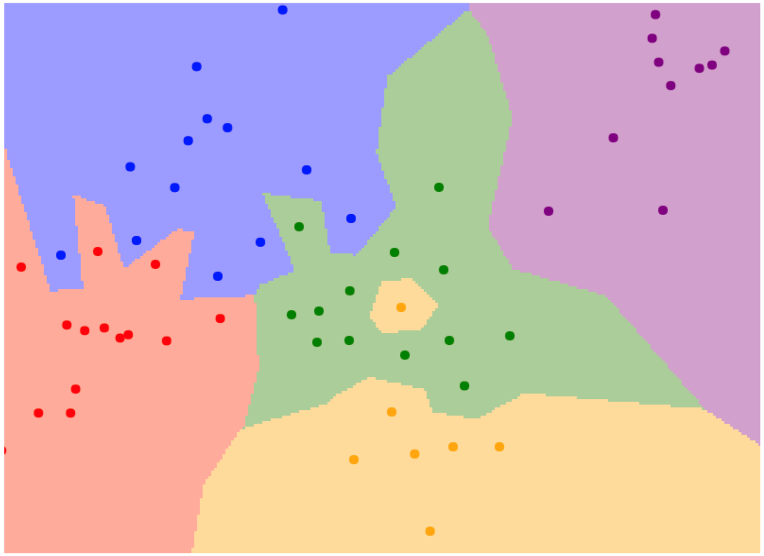
K = 1

K = 3

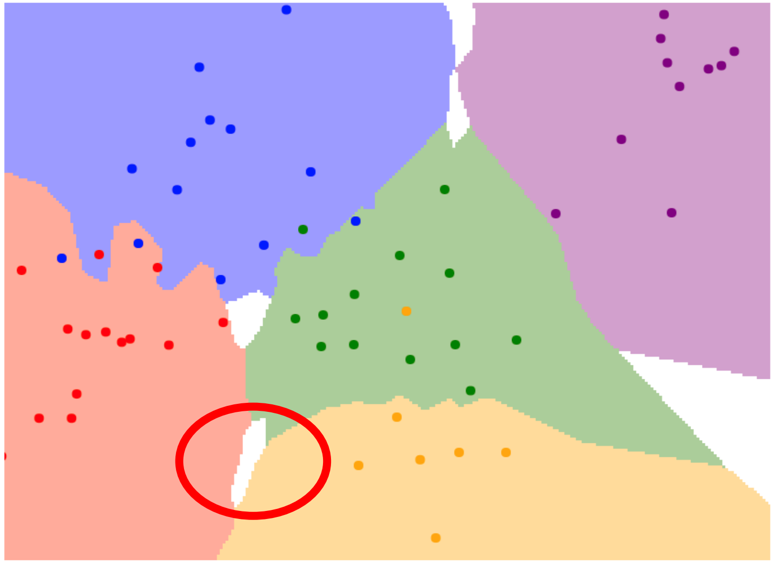
K-Nearest Neighbors (kNN)

- Make the decision boundary more smooth
- Reduce the effect of outliers

K = 1



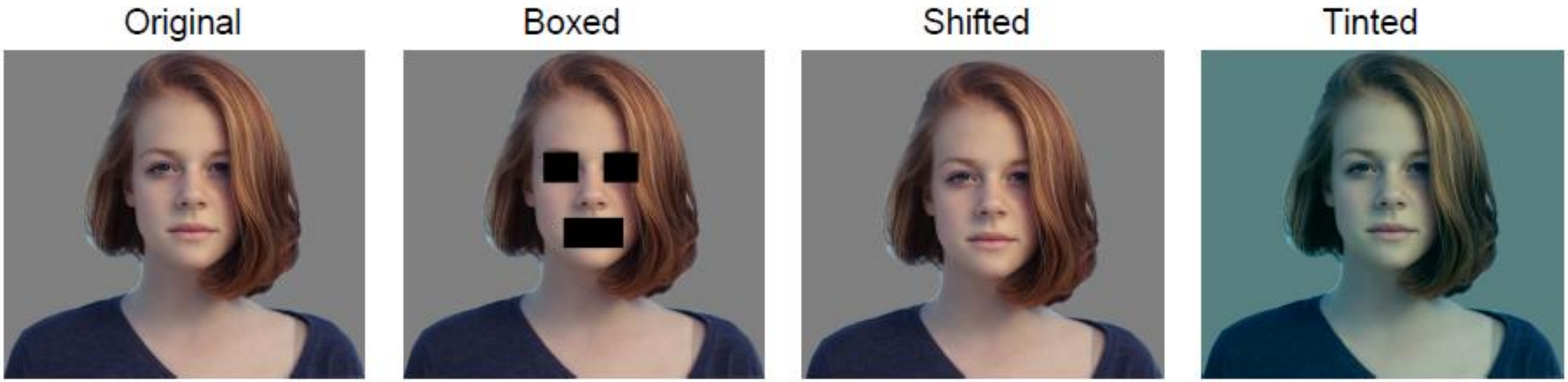
K = 3



http://vision.stanford.edu/teaching/cs231n-demos/knn/

Minor Remarks on NN-based Methods

- k-NN is easy to implement but not of much interest in practice. Why?
  - Choice of distance metrics might be an issue (see example below)
  - Measuring distances in high-dimensional spaces might not be a good idea.
  - Moreover, NN-based methods require lots of space and computation time! (NN-based methods are viewed as *data-driven* approaches.)



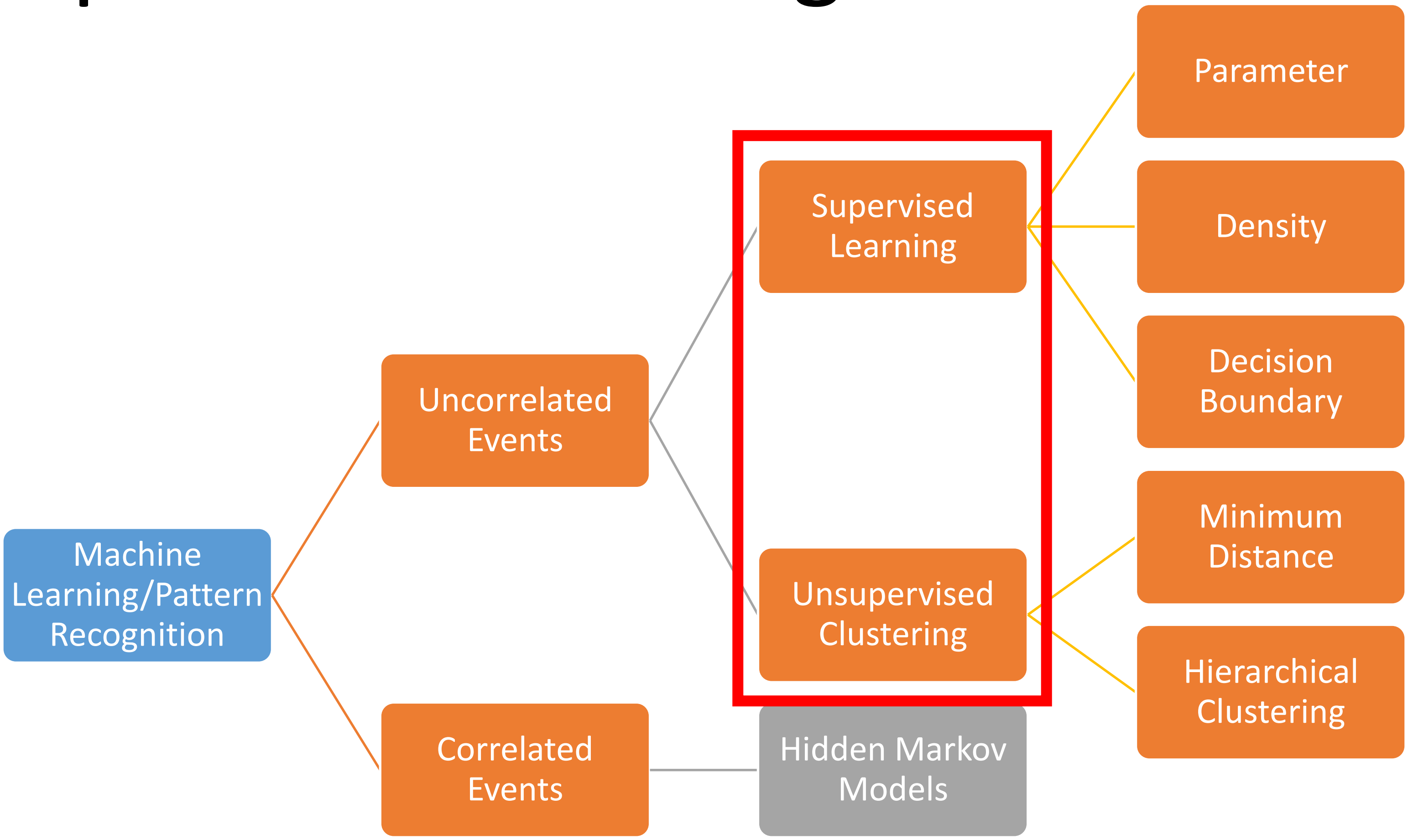
All three images have the same Euclidean distance to the original one.

Image credit: Stanford CS231n

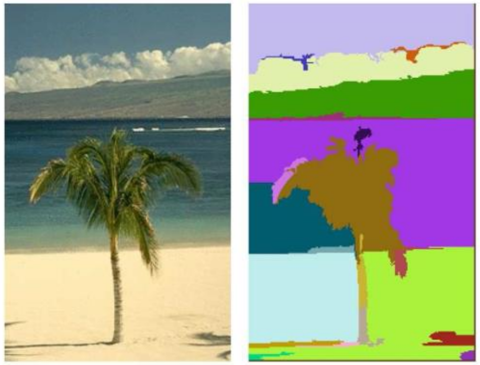
Nonparametric Techniques

- kNN is also a nonparametric technique:
    - Specify k_n as a function of n, such as $k_n = \sqrt{n}$; the volume V_n is grown until it encloses k_n neighbors of x

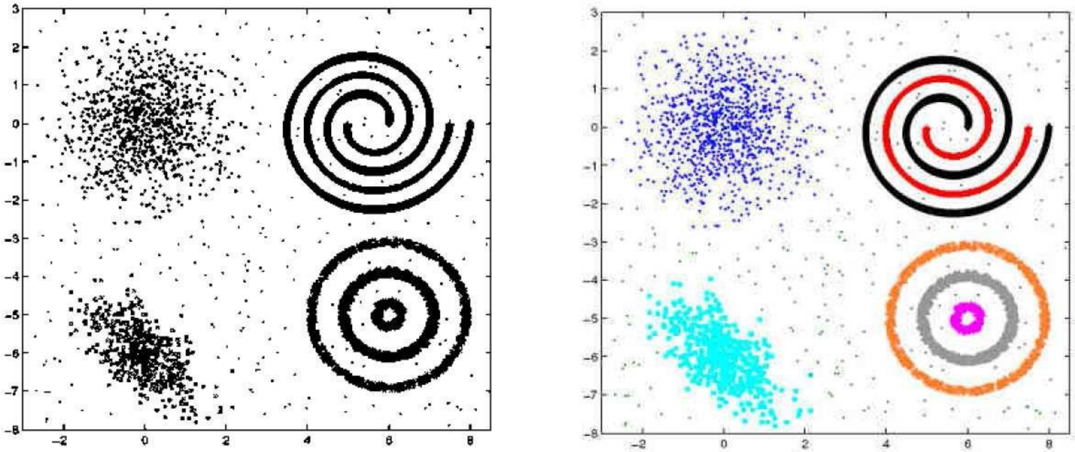
Unsupervised Learning and Supervised Learning



Clustering



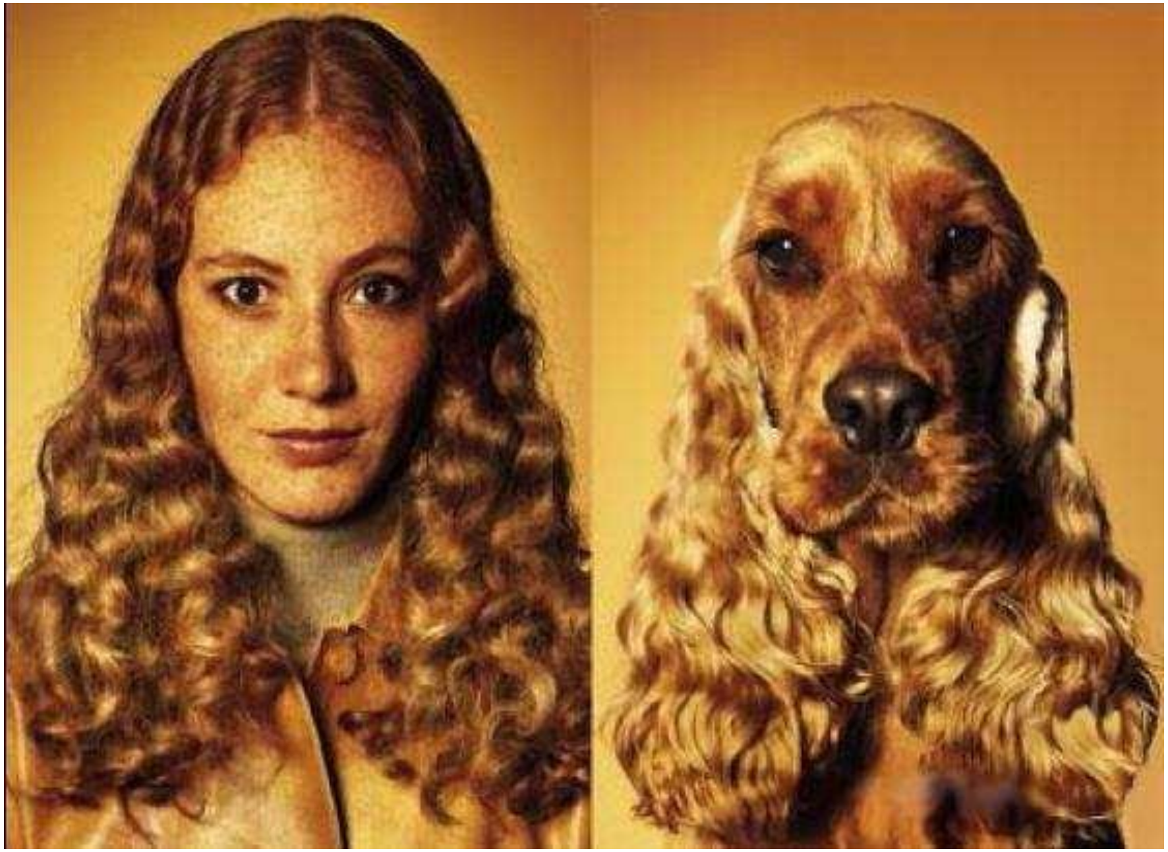
- Clustering is an unsupervised algorithm.
  - Given:
    a set of N unlabeled instances {$\boldsymbol{x}_1$, …, $\boldsymbol{x}_N$}; # of clusters K
  - Goal: group the samples into K partitions
- Remarks:
  - High within-cluster (intra-cluster) similarity
  - Low between-cluster (inter-cluster) similarity
  - But…how to determine a proper similarity measure?



(a) Input data

(b) Desired clustering

Similarity is NOT Always Objective…

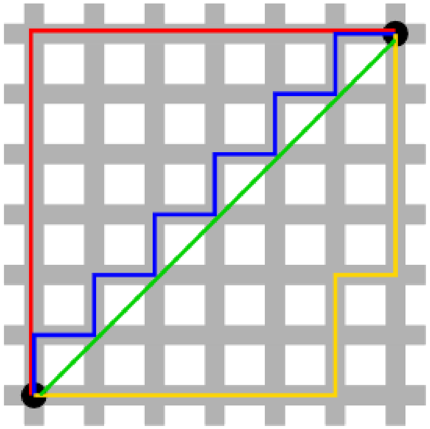


Clustering (cont'd)

- Similarity:
  - A key component/measure to perform data clustering
  - Inversely proportional to distance
  - Example distance metrics:
    - Euclidean distance (L2 norm): $d(x, z) = \|x - z\|_2 = \sqrt{\sum_{i=1}^{D} (x_i - z_i)^2}$
    - Manhattan distance (L1 norm): $d(x, z) = \|x - z\|_1 = \sum_{i=1}^{D} |x_i - z_i|$
    - Note that *p*-norm of *x* is denoted as:

$$L_P(\boldsymbol{x}, \boldsymbol{z}) = \left\{ \sum_{i=1}^{D} (x_i - z_i)^p \right\}^{1/p}$$

$$L_0(\boldsymbol{x}, \boldsymbol{z}) = \lim_{p \to 0} \left\{ \sum_{i=1}^{D} (x_i - z_i)^p \right\}^{1/p}$$



Clustering (cont'd)

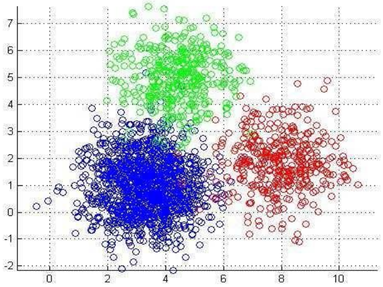
- Similarity:
  - A key component/measure to perform data clustering
  - Inversely proportional to distance
  - Example distance metrics:
    - Kernelized (non-linear) distance:

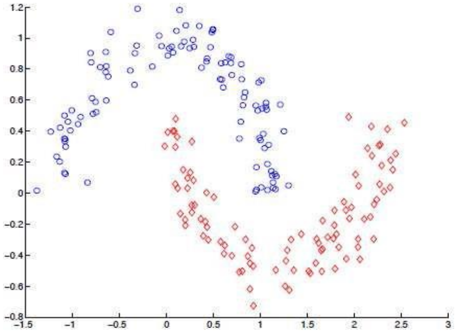
$$d(x,z) = \|\Phi(x) - \Phi(z)\|_2^2 = \|\Phi(x)\|_2^2 + \|\Phi(z)\|_2^2 - 2\Phi(x)^T\Phi(z)$$

    - Taking *Gaussian kernel* for example: $K(x,z) = \Phi(x)^T\Phi(z) = exp\left(-\frac{\|x-z\|_2^2}{2\sigma^2}\right)$,

      we have $\|\Phi(x)\|_2^2 = \Phi(x)^T\Phi(x) = 1$

      distance is more sensitive smaller σ.
    - For example, L2 or kernelized distance metrics for the following two cases?



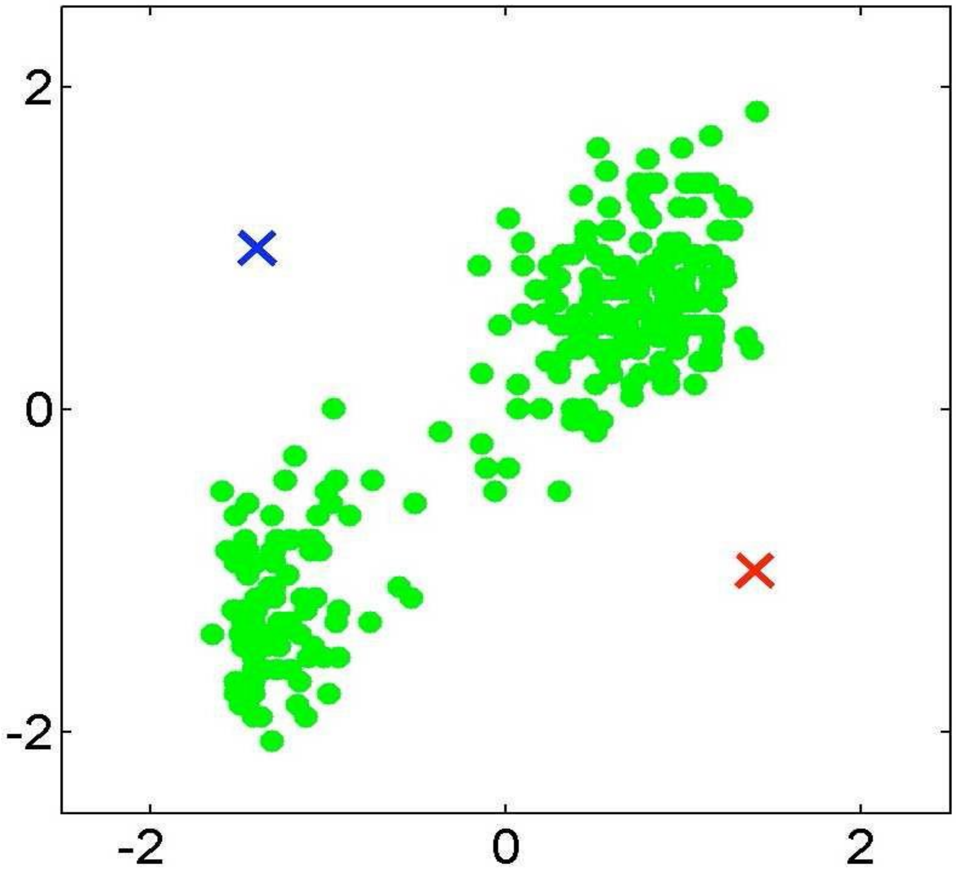


K-Means Clustering

- Input: N examples $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$ ($\boldsymbol{x}_n \in \mathrm{R}^D$); number of partitions K
- Initialize: K cluster centers $\mu_1, \ldots, \mu_K$. Several initialization options:
  - Randomly initialize $\mu_1, \ldots, \mu_K$ anywhere in R^D
  - Or, simply choose any K examples as the cluster centers
- Iterate:
  - Assign each of example $\boldsymbol{x}_\mathrm{n}$ to its closest cluster center
  - Recompute the new cluster centers μ_k (mean/centroid of the set C_k)
  - Repeat while not converge
- Possible convergence criteria:
  - Cluster centers do not change anymore
  - Max. number of iterations reached
- Output:
  - K clusters (with centers/means of each cluster)

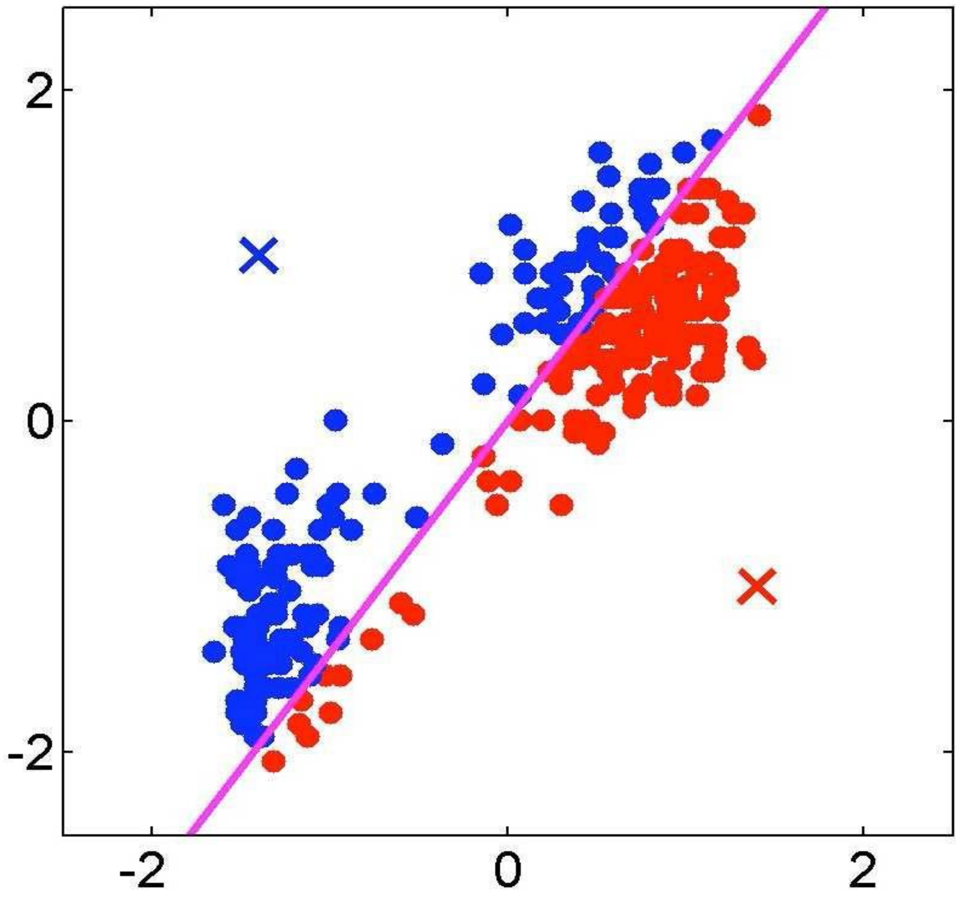
K-Means Clustering

- Example (K = 2): Initialization, iteration #1: pick cluster centers



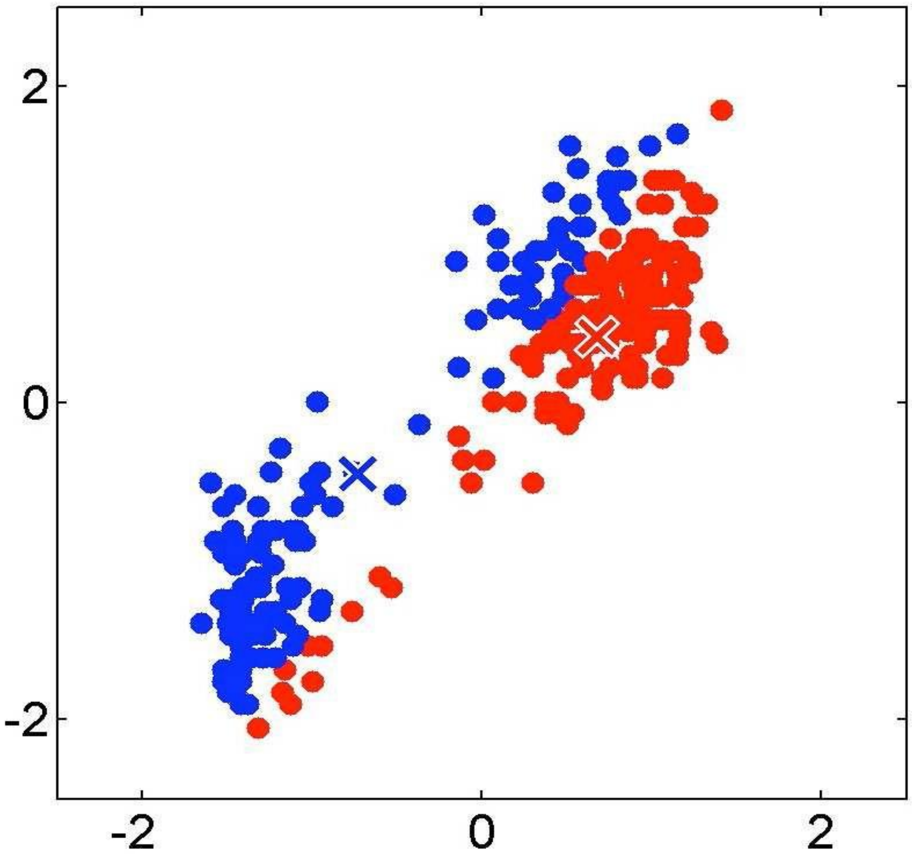
K-Means Clustering

- Example (K = 2): iteration #1-2, assign data to each cluster



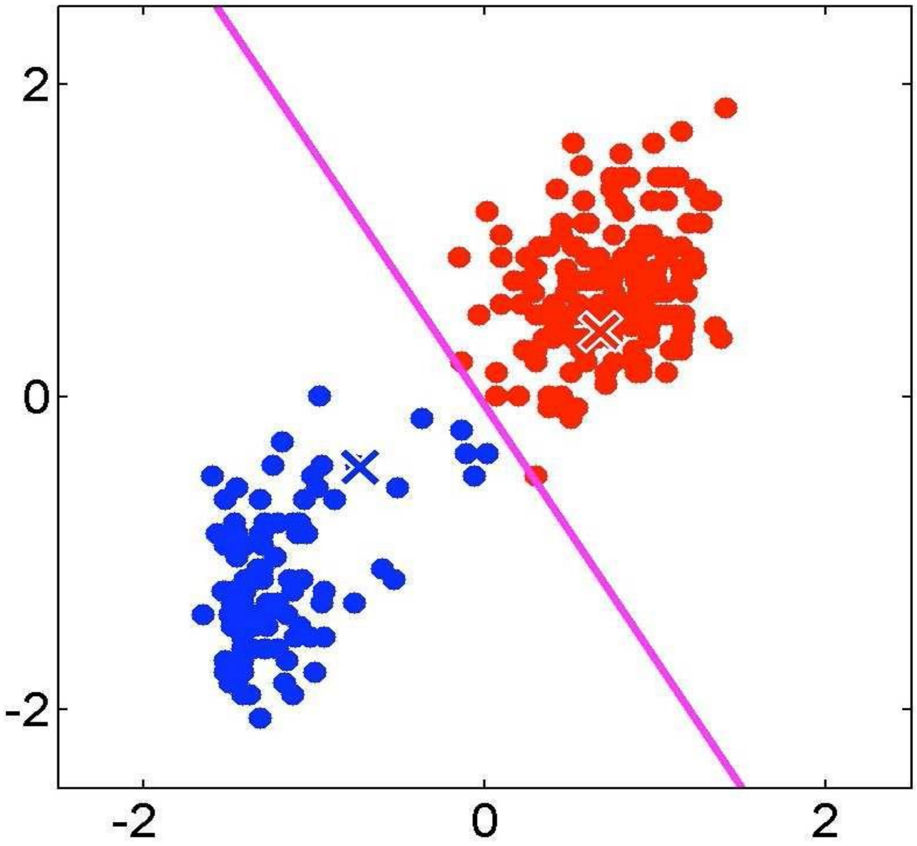
K-Means Clustering

- Example (K = 2): iteration #2-1, update cluster centers



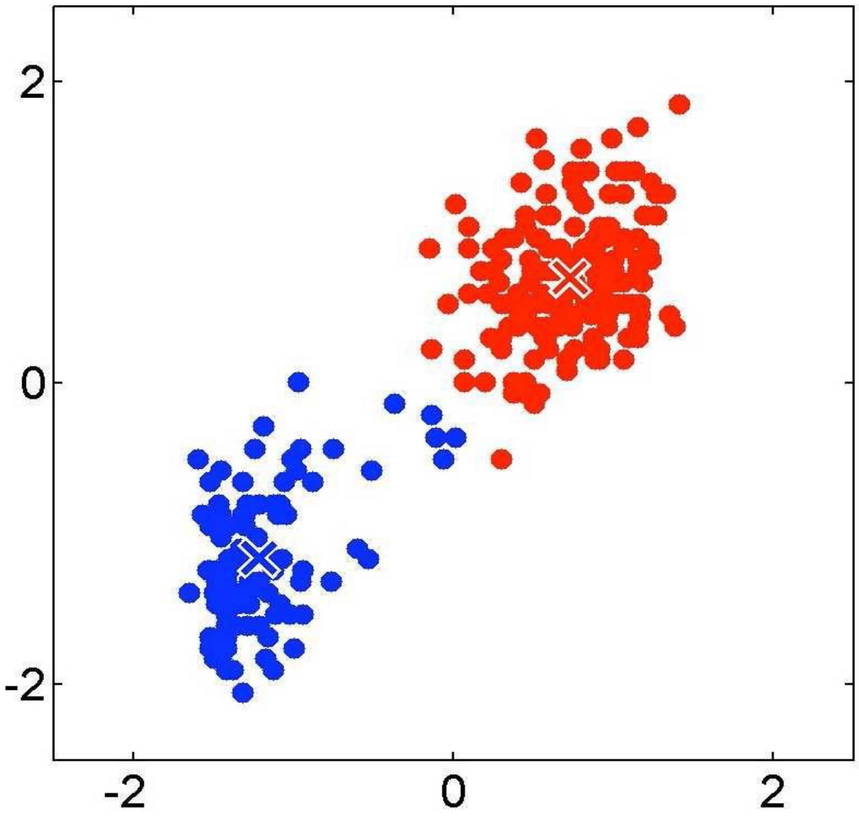
K-Means Clustering

- Example (K = 2): iteration #2, assign data to each cluster



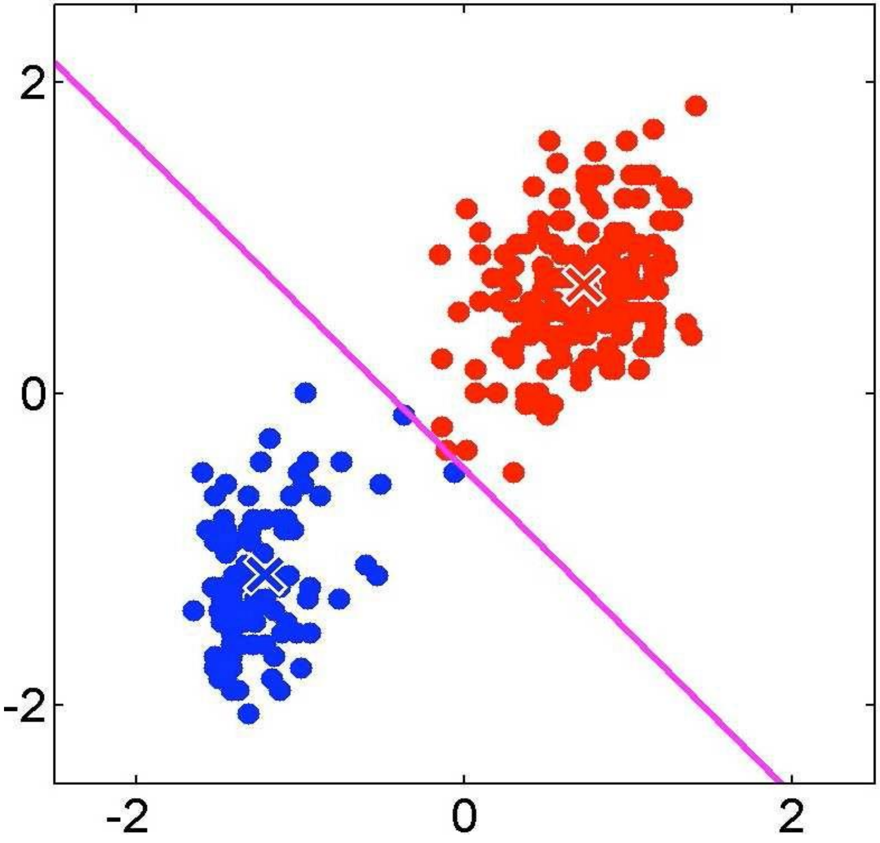
K-Means Clustering

- Example (K = 2): iteration #3-1



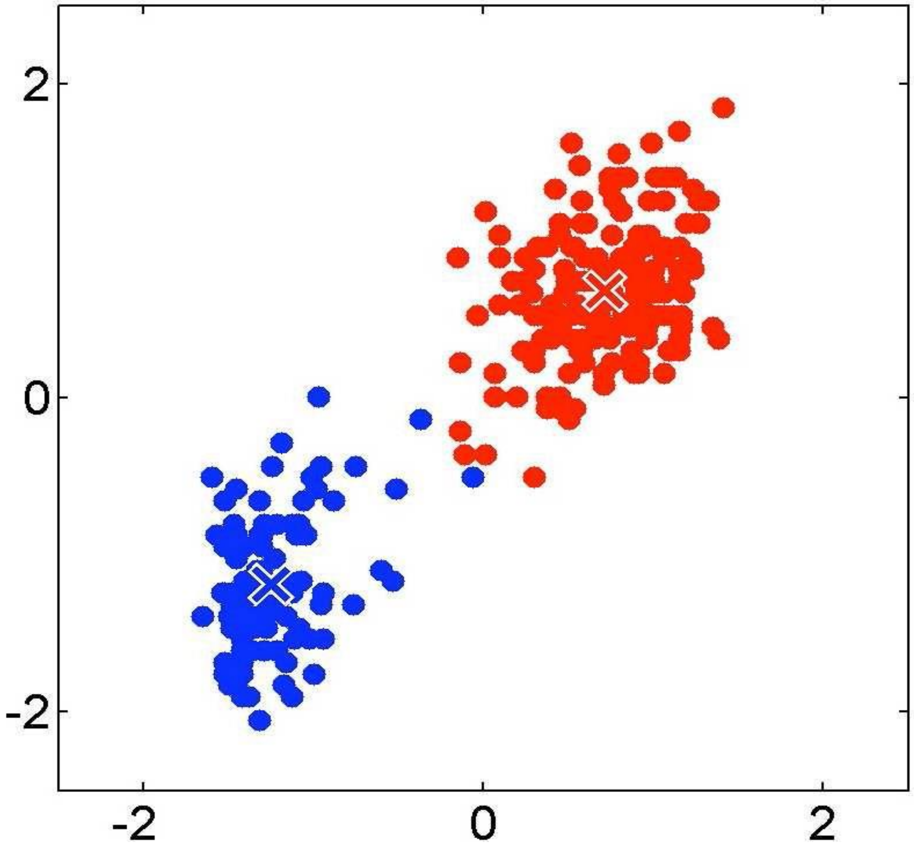
K-Means Clustering

- Example (K = 2): iteration #3-2



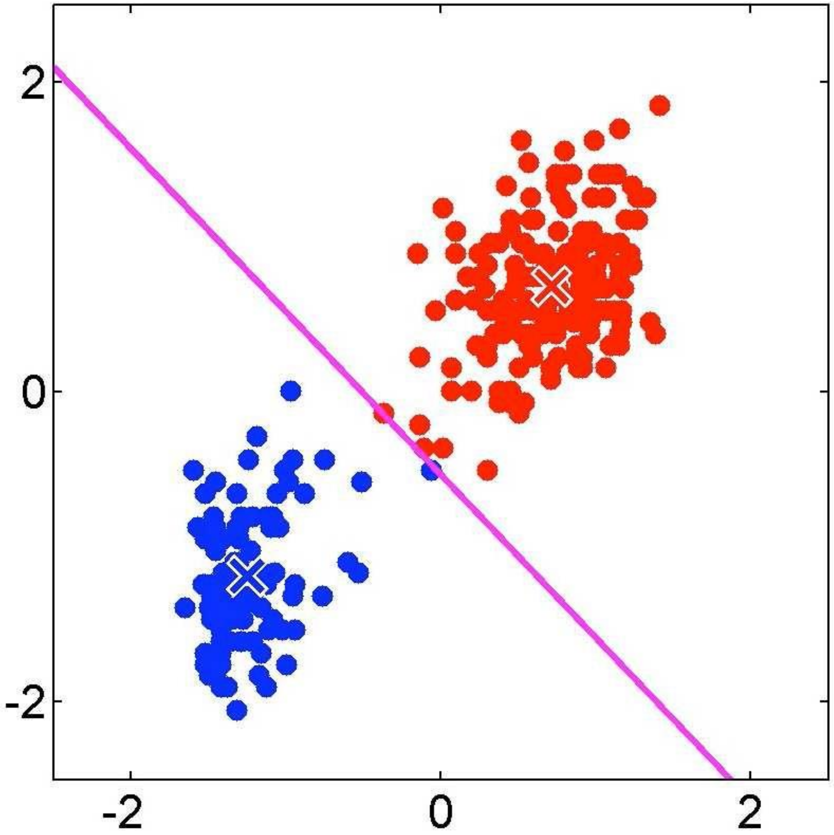
K-Means Clustering

- Example (K = 2): iteration #4-1



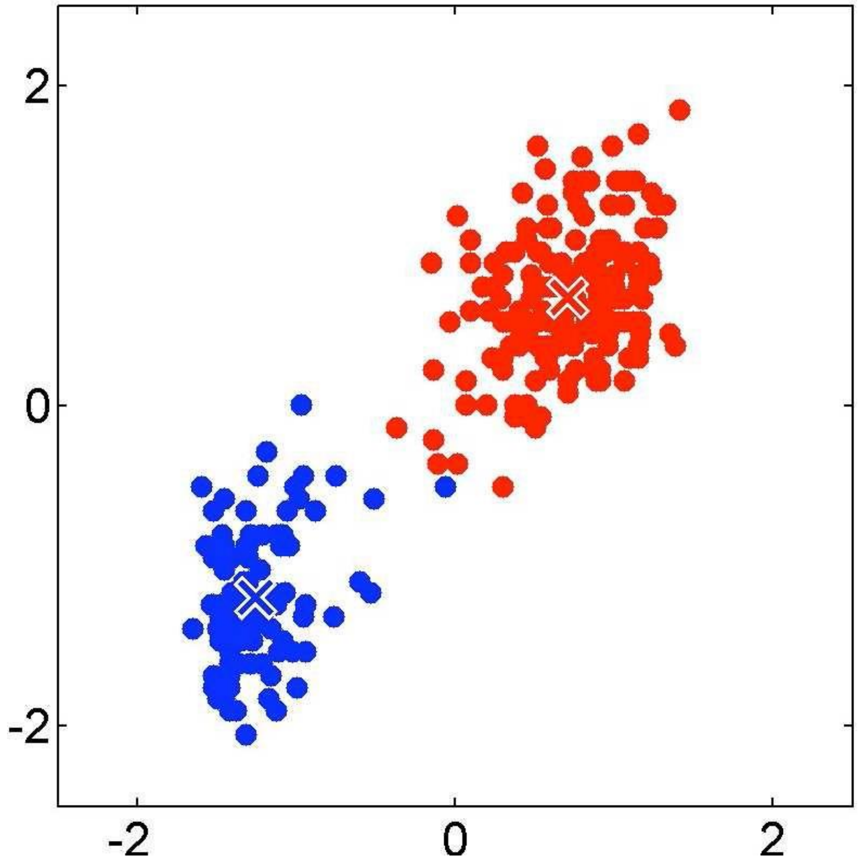
K-Means Clustering

- Example (K = 2): iteration #4-2



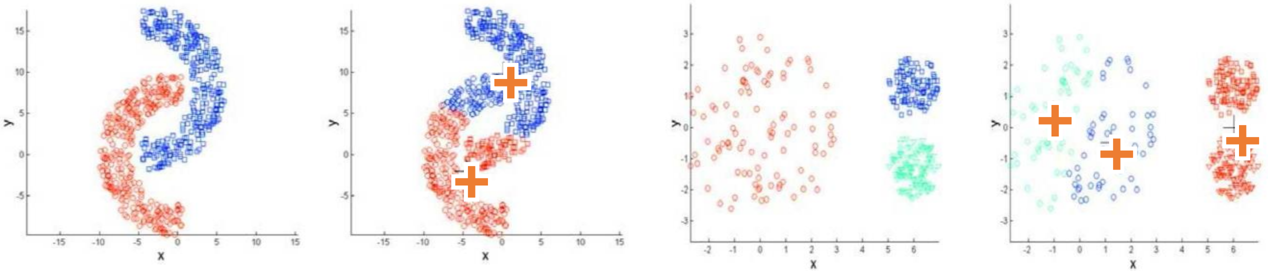
K-Means Clustering

- Example (K = 2): iteration #5, cluster means are not changed.



K-Means Clustering (cont'd)

- Limitation
    - Preferable for round shaped clusters with similar sizes

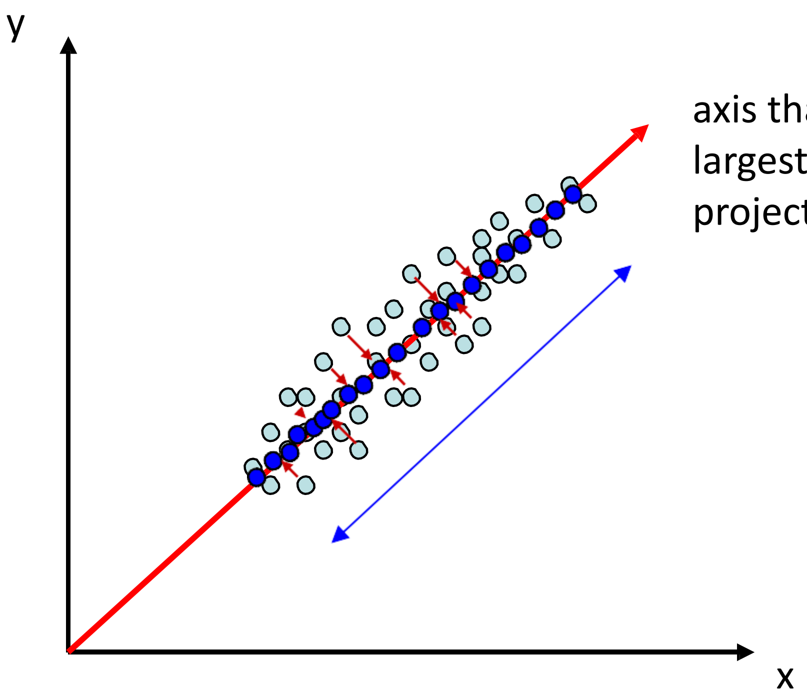


    - Sensitive to initialization; how to alleviate this problem?
    - Sensitive to outliers; possible change from K-means to…
    - Hard assignment only.

- Remarks
    - Expectation-maximization (EM) algorithm
    - Speed-up possible by hierarchical clustering (e.g., 100 = 10^2 clusters)

Dimension Reduction

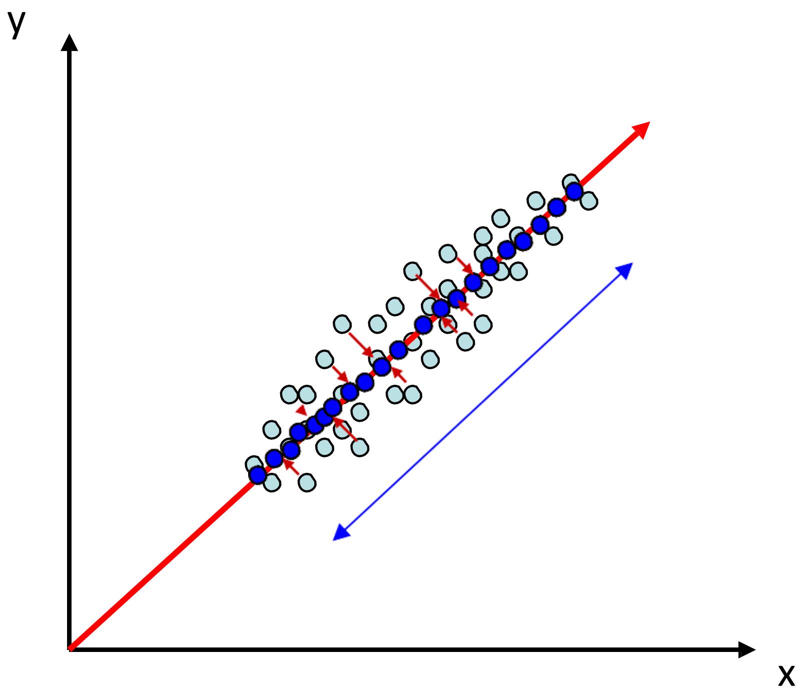
- Principal Component Analysis (PCA)
  - Unsupervised & linear dimension reduction
  - Related to Eigenfaces, etc. feature extraction and classification techniques
  - Still very popular despite of its simplicity and effectiveness.
  - Goal:
    - Determine the projection, so that the variation of projected data is maximized.



axis that describes the largest variation for data projected onto it.

Formulation & Derivation for PCA

- Input: a set of instances ***x*** without label info
- Output: a projection vector $\boldsymbol{u_1}$ maximizing the variance of the projected data



the variance of the projected data is given by

$$\frac{1}{N}\sum_{n=1}^{N}\left\{\mathbf{u}_1^{\mathrm{T}}\mathbf{x}_n - \mathbf{u}_1^{\mathrm{T}}\overline{\mathbf{x}}\right\}^2 = \mathbf{u}_1^{\mathrm{T}}\mathbf{S}\mathbf{u}_1$$

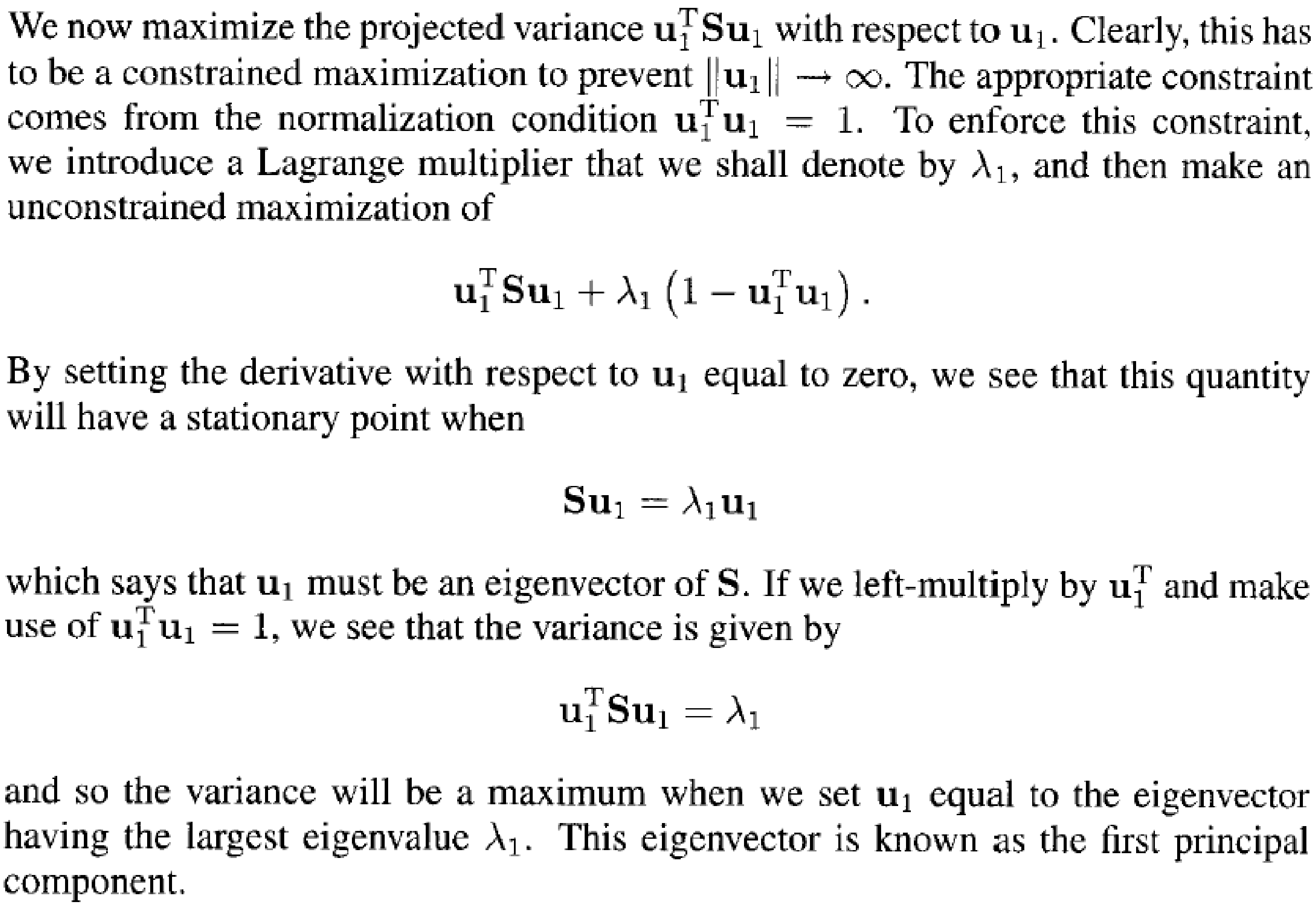
where $\mathbf{S}$ is the data covariance matrix defined by

$$\mathbf{S} = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_n - \overline{\mathbf{x}})(\mathbf{x}_n - \overline{\mathbf{x}})^{\mathrm{T}}.$$

$$S = TT^T$$

T be the matrix of preprocessed training examples, where each column contains one mean-subtracted image.

Formulation & Derivation for PCA



We now maximize the projected variance $\mathbf{u}_1^{\mathrm{T}}\mathbf{S}\mathbf{u}_1$ with respect to $\mathbf{u}_1$. Clearly, this has to be a constrained maximization to prevent $\|\mathbf{u}_1\| \to \infty$. The appropriate constraint comes from the normalization condition $\mathbf{u}_1^{\mathrm{T}}\mathbf{u}_1 = 1$. To enforce this constraint, we introduce a Lagrange multiplier that we shall denote by λ_1, and then make an unconstrained maximization of

$$\mathbf{u}_1^{\mathrm{T}}\mathbf{S}\mathbf{u}_1 + \lambda_1 \left(1 - \mathbf{u}_1^{\mathrm{T}}\mathbf{u}_1\right).$$

By setting the derivative with respect to $\mathbf{u}_1$ equal to zero, we see that this quantity will have a stationary point when

$$\mathbf{S}\mathbf{u}_1 = \lambda_1\mathbf{u}_1$$

which says that $\mathbf{u}_1$ must be an eigenvector of $\mathbf{S}$. If we left-multiply by $\mathbf{u}_1^{\mathrm{T}}$ and make use of $\mathbf{u}_1^{\mathrm{T}}\mathbf{u}_1 = 1$, we see that the variance is given by

$$\mathbf{u}_1^{\mathrm{T}}\mathbf{S}\mathbf{u}_1 = \lambda_1$$

and so the variance will be a maximum when we set $\mathbf{u}_1$ equal to the eigenvector having the largest eigenvalue λ_1. This eigenvector is known as the first principal component.

Formulation & Derivation for PCA

However $\mathbf{T}\mathbf{T}^T$ is a large matrix, and if instead we take the eigenvalue decomposition of

$$\mathbf{T}^T\mathbf{T}\mathbf{u}_i = \lambda_i \mathbf{u}_i$$

then we notice that by pre-multiplying both sides of the equation with **T**, we obtain

$$\mathbf{T}\mathbf{T}^T\mathbf{T}\mathbf{u}_i = \lambda_i \mathbf{T}\mathbf{u}_i$$

Meaning that, if $\mathbf{u}_i$ is an eigenvector of $\mathbf{T}^T\mathbf{T}$, then $\mathbf{v}_i = \mathbf{T}\mathbf{u}_i$ is an eigenvector of **S**. If we have a training set of 300 images of 100 × 100 pixels, the matrix $\mathbf{T}^T\mathbf{T}$ is a 300 × 300 matrix, which is much more manageable than the 10,000 × 10,000 covariance matrix.

Eigenanalysis

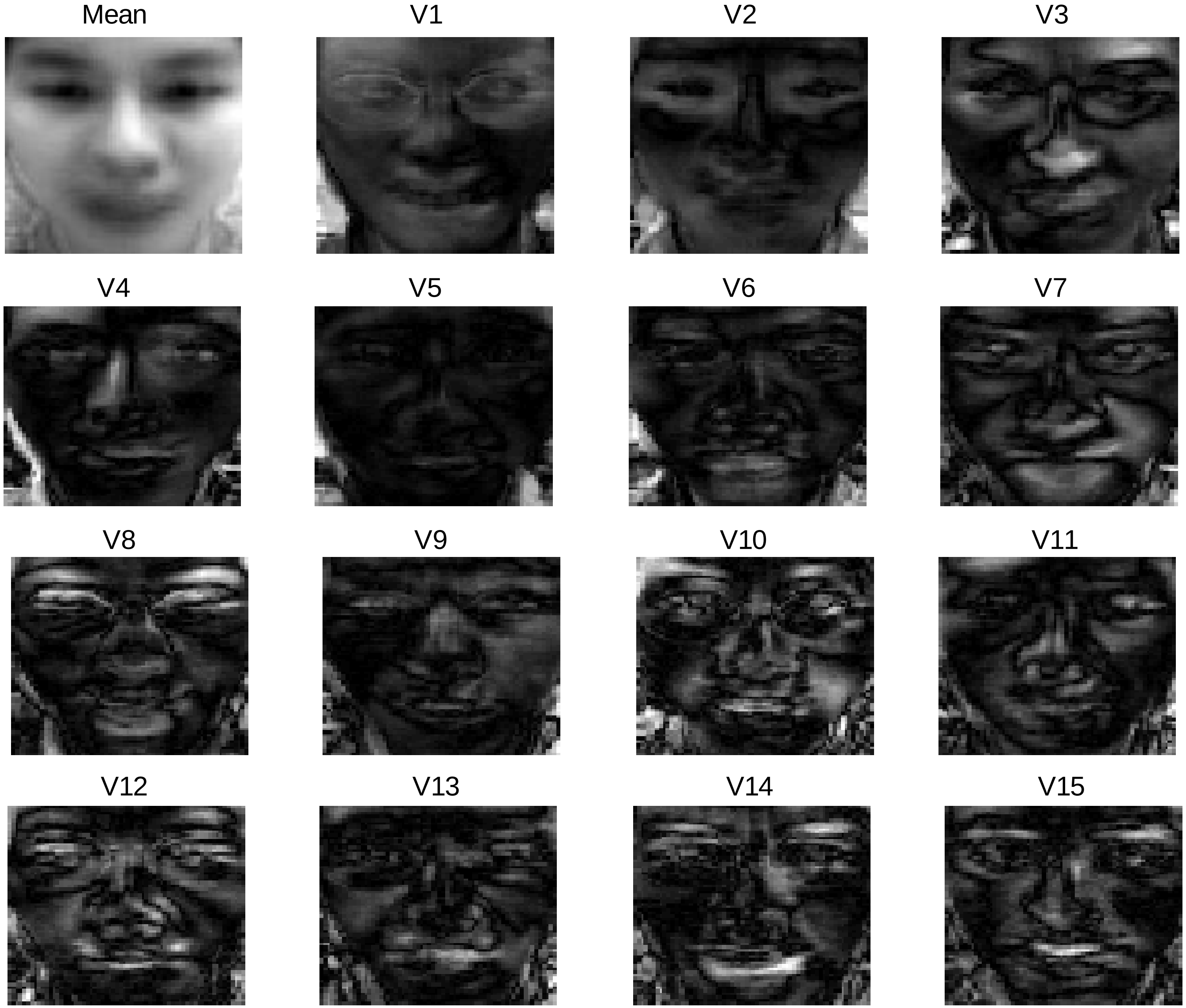
- A *d* x *d* covariance matrix contains a maximum of *d* eigenvector/eigenvalue pairs.
  - How dimension reduction is realized? how to reconstruct the input data?

- Expanding a signal via eigenvectors as bases
  - With symmetric matrices (e.g., covariance matrix), eigenvectors are orthogonal.
  - They can be regarded as unit basis vectors to span any instance in the d-dim space.

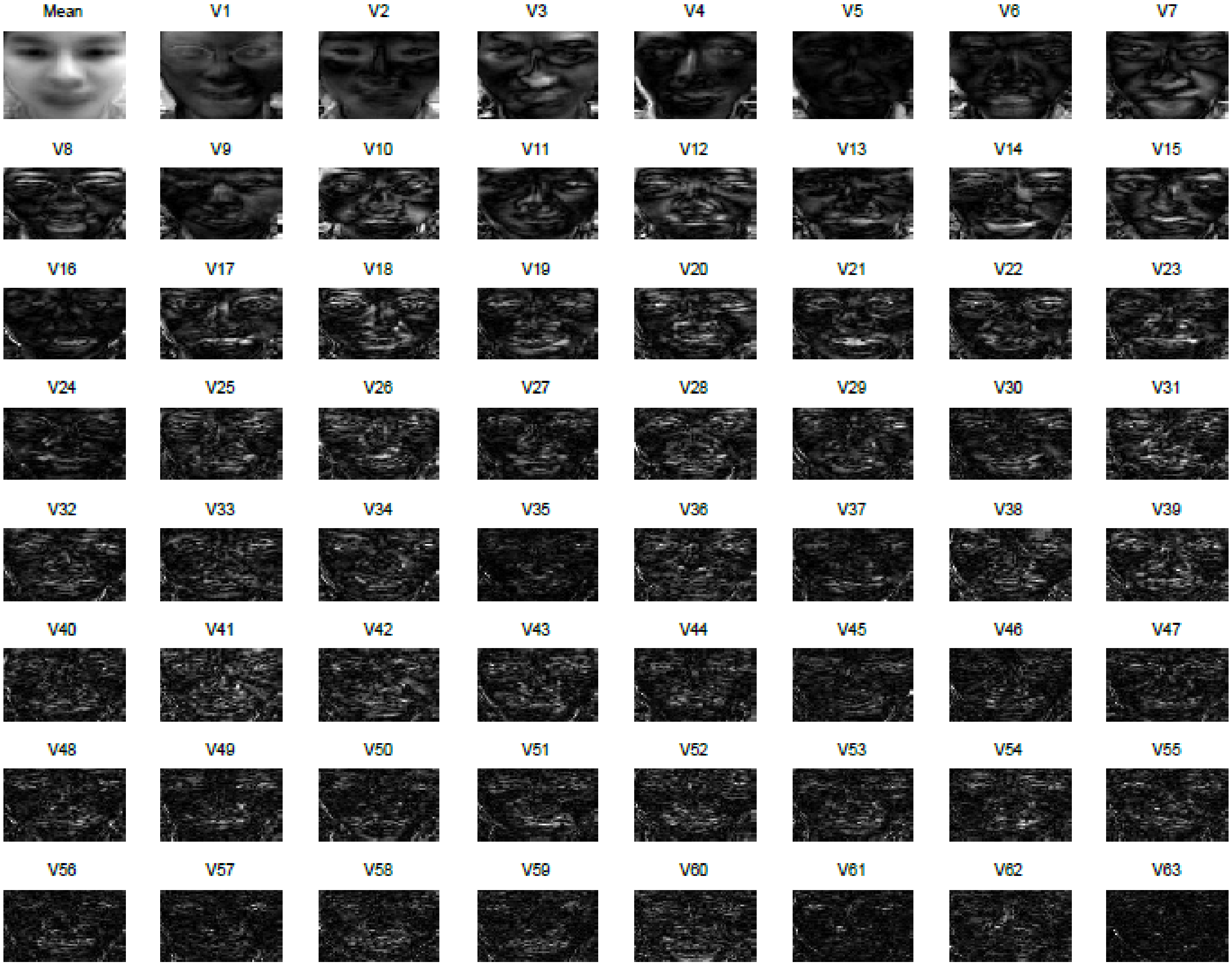
Let’s See an Example (CMU AMP Face Database)

- Let’s take 5 face images x 13 people = 65 images, each is of size 64 x 64 = 4096 pixels.
- # of eigenvectors are expected to use for perfectly reconstructing the input = 64.
- Let’s check it out!

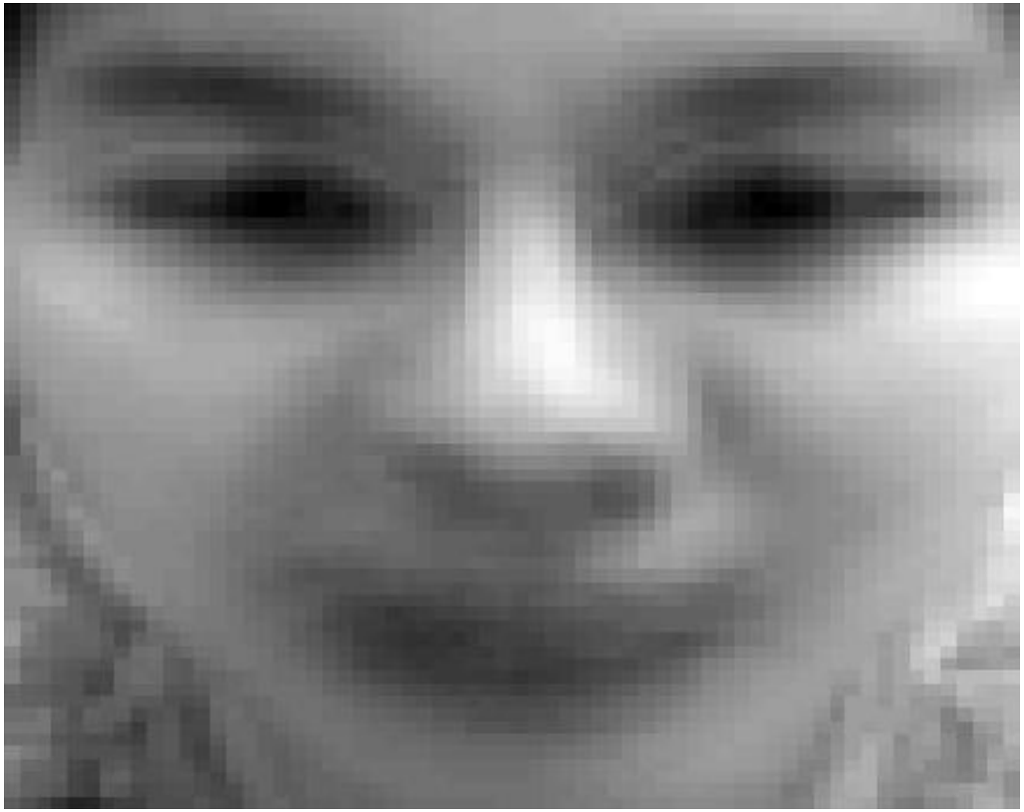
What Do the Eigenvectors/Eigenfaces Look Like?



All 64 Eigenvectors, do we need them all?



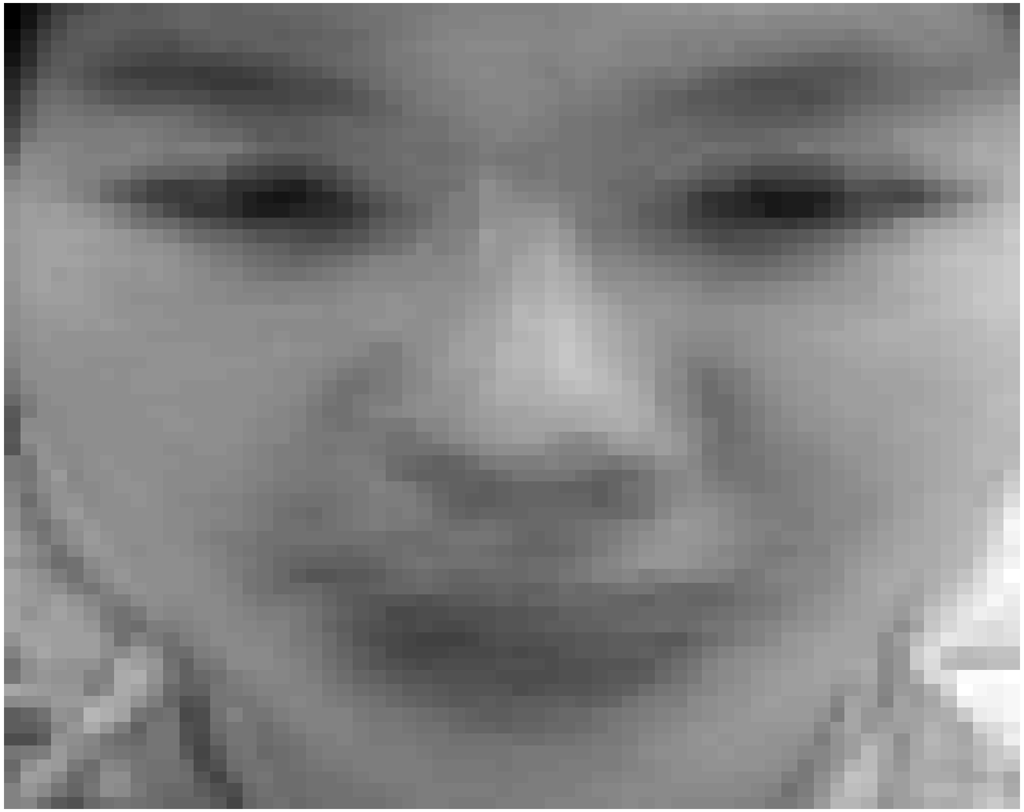
Use only 1 eigenvector, MSE = 1233



MSE=1233.16

Use 2 eigenvectors, MSE = 1027

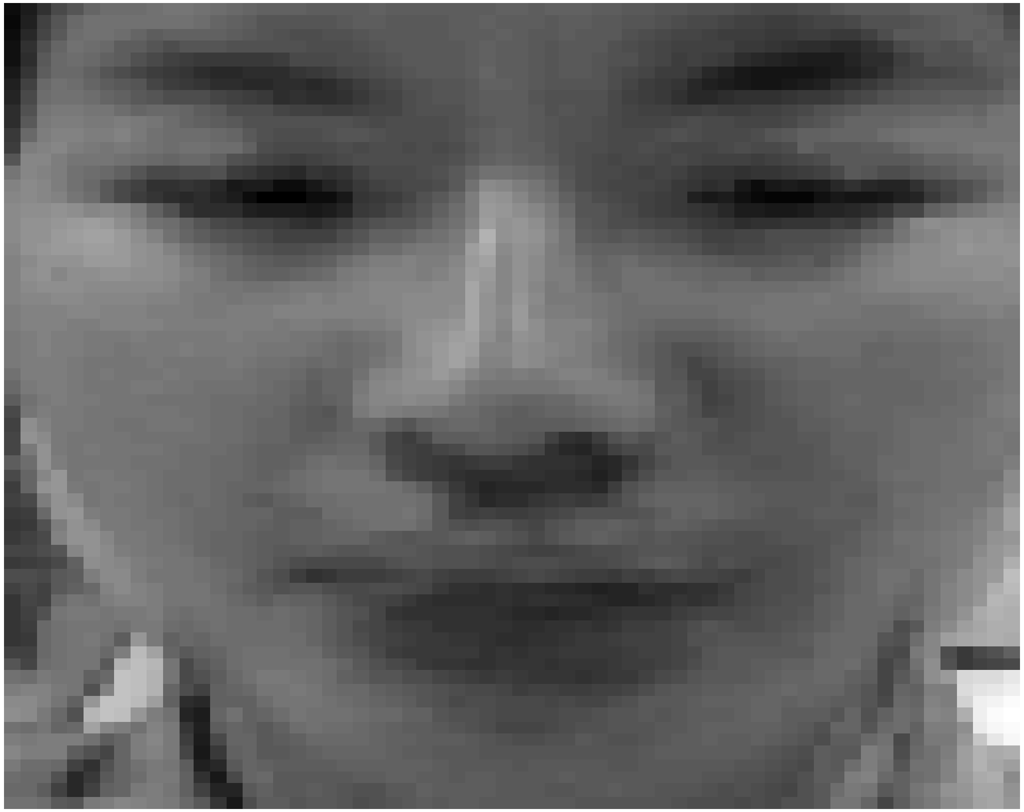
MSE=1027.63



Use 3 eigenvectors, MSE = 758

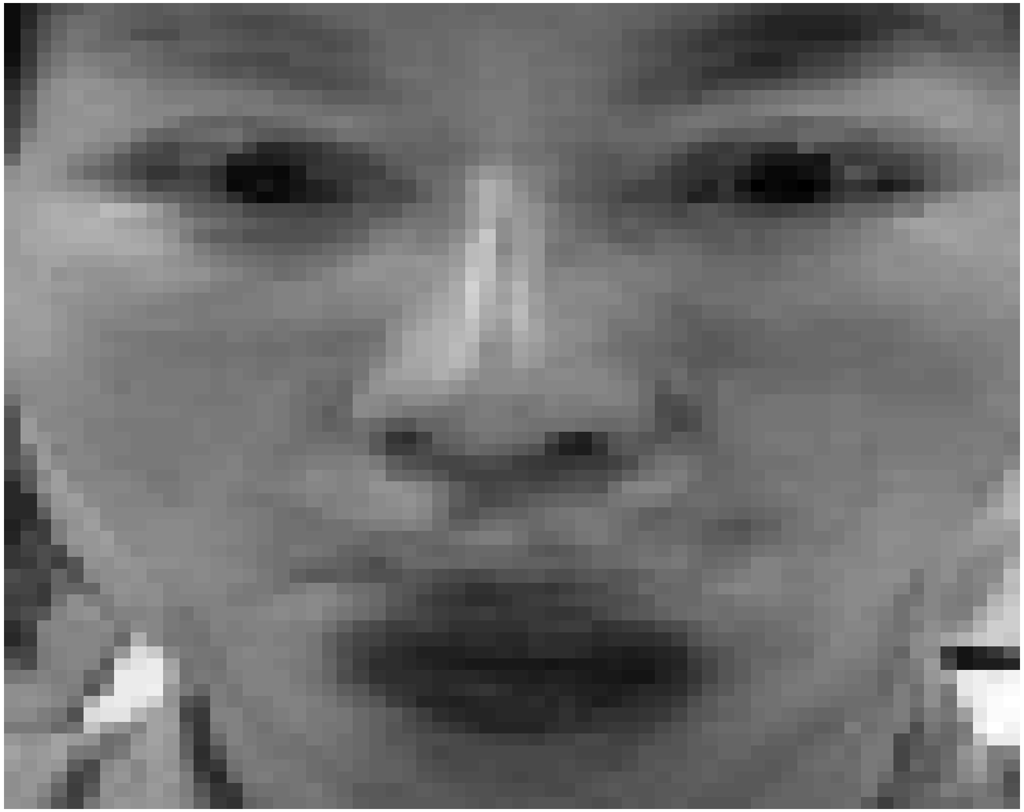
Use 4 eigenvectors, MSE = 634

MSE=634.54



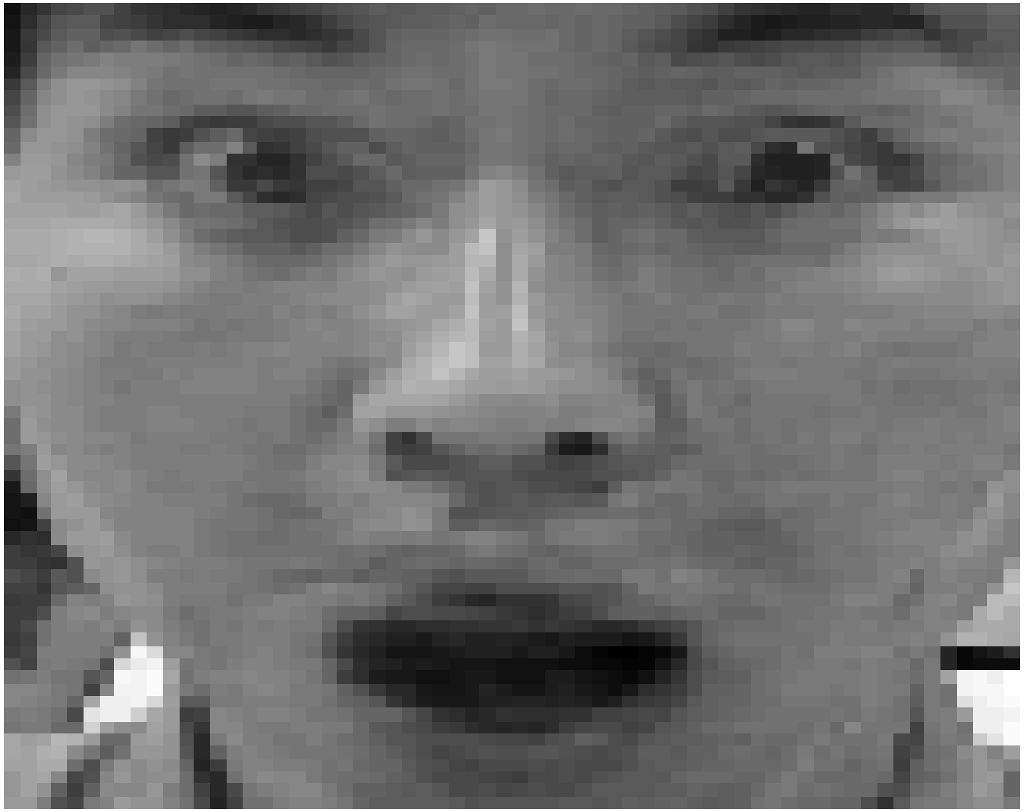
Use 8 eigenvectors, MSE = 285

MSE=285.08

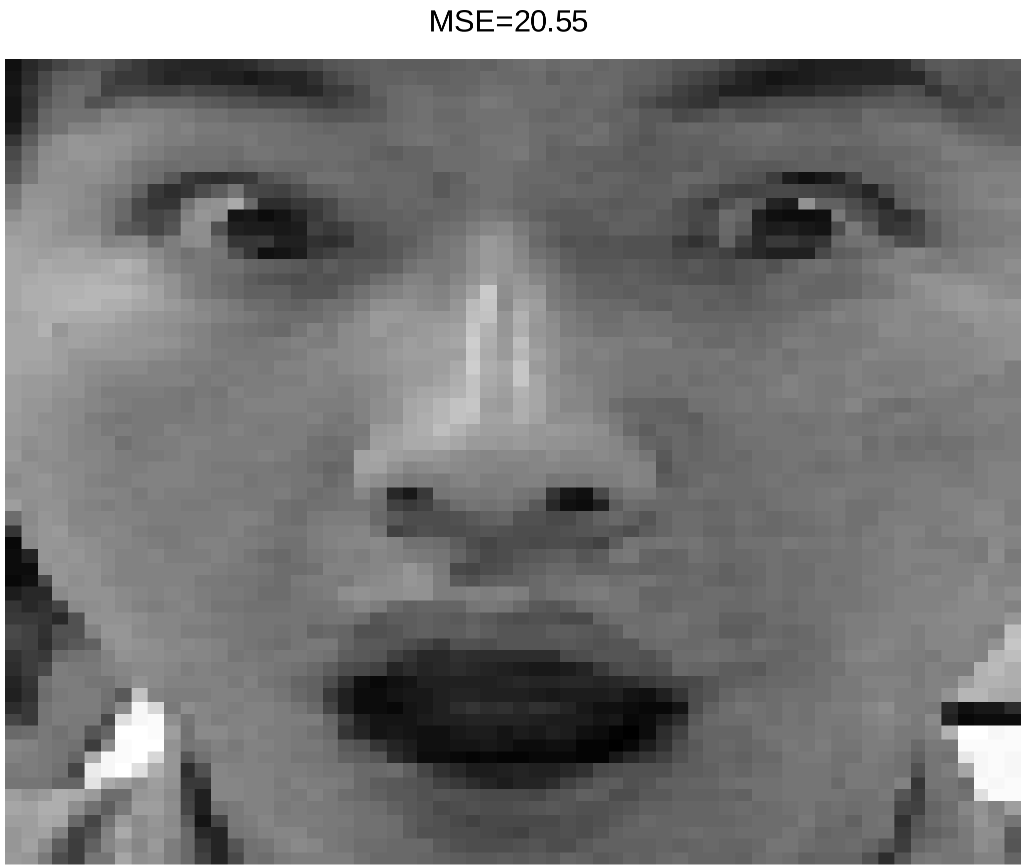


With 20 eigenvectors, MSE = 87

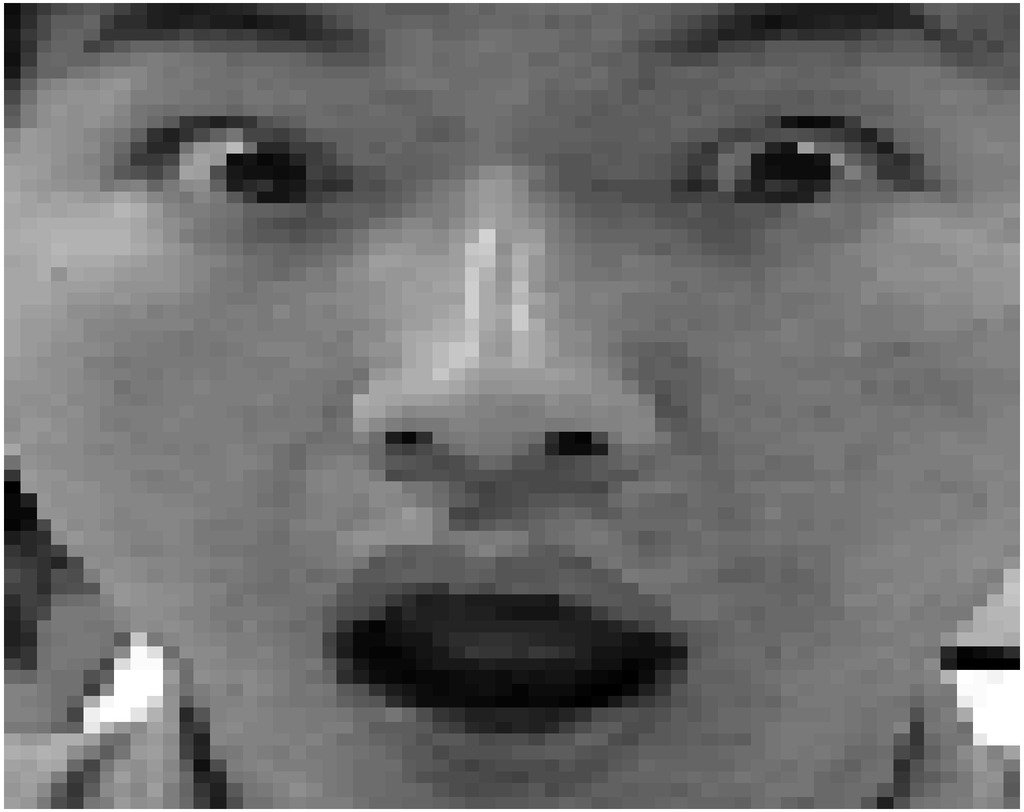
MSE=87.93



With 30 eigenvectors, MSE = 20

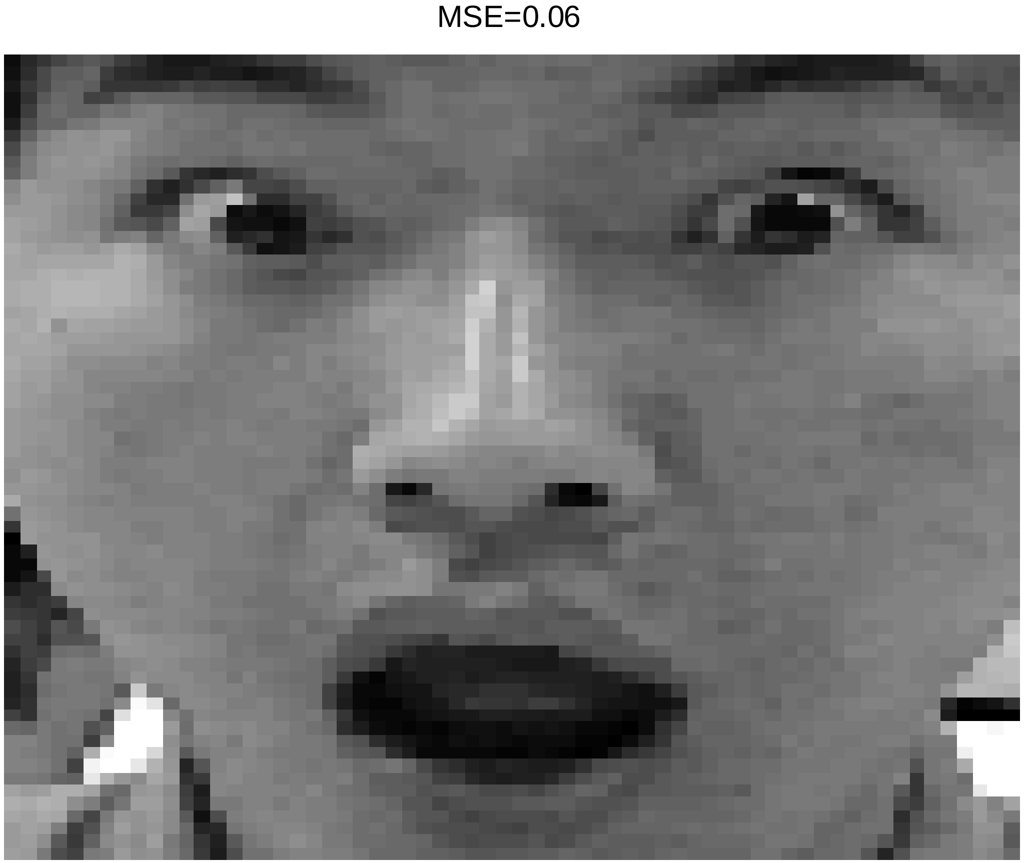


With 50 eigenvectors, MSE = 2.14



MSE=2.14

With 60 eigenvectors, MSE = 0.06



All 64 eigenvectors, MSE = 0

MSE=0.00

Linear Discriminant Analysis(LDA)

- Linear Discriminant Analysis(LDA)
  - Classify objects into one of two or more groups
  - Base on a set of features
- The transform tries to maximize the ratio of between variance to within class variance
- Between class variance
  - $$S_b = \frac{1}{m} \cdot \sum_{i=1}^{k} \sum_{j=1}^{m_i} (x_{ij} - \bar{x}) \cdot (x_{ij} - \bar{x})^T$$
- Within class variance
  - $$S_w = \frac{1}{m} \cdot \sum_{i=1}^{k} \sum_{j=1}^{m_i} (x_{ij} - \bar{x}_i) \cdot (x_{ij} - \bar{x}_i)^T = \sum_{i=1}^{k} p_i \times (cov_i)$$

Different

Mathematical Operations

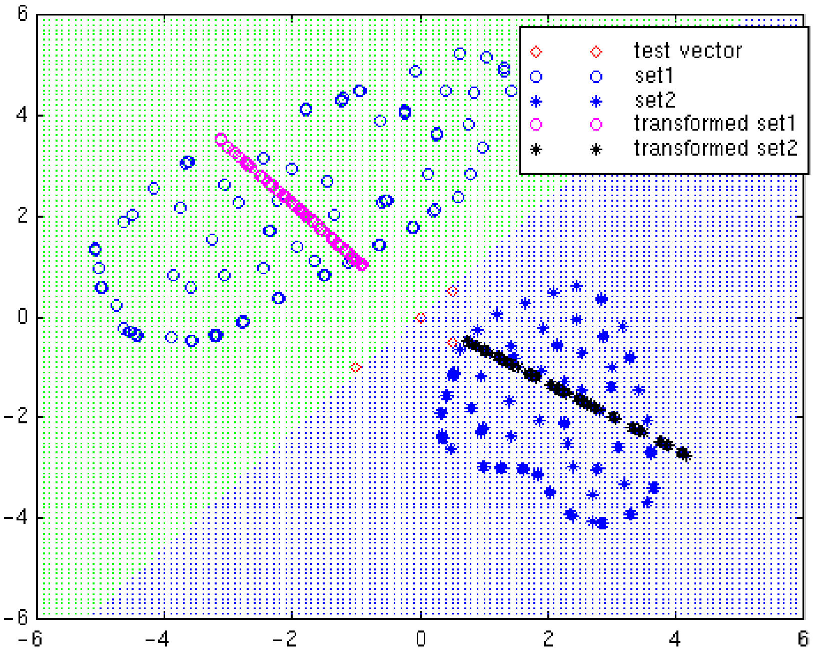
- Maximize $J = \frac{|S_b|}{|S_w|}$
- If y is the transform of x
  - $\mathbf{y} = \mathbf{W}^T\mathbf{x}$
- Compute J after the transform
  - $S'_w = W^T S_w W$
  - $S'_b = W^T S_b W$
  - $J' = \frac{|S'_b|}{|S'_w|} = \frac{|W^T S_b W|}{|W^T S_w W|}$
    - Find W to maximize J'

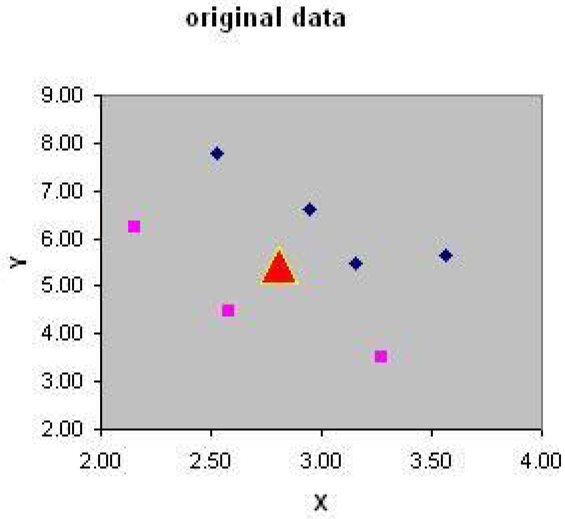
Find W

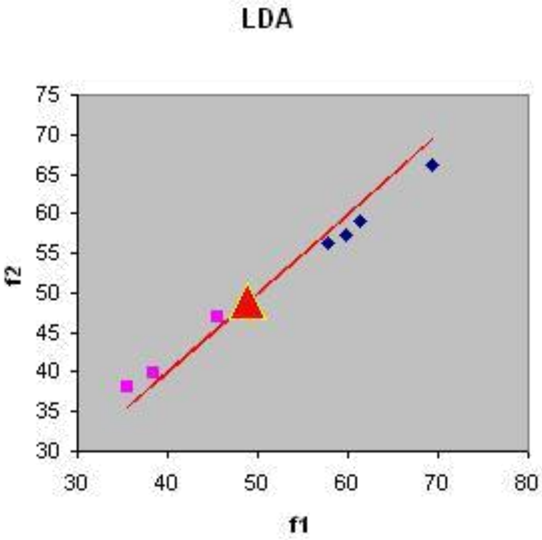
- If we are lucky, $\mathrm{S_w}$ is a non-singular matrix
  - We can find $\mathrm{S_w^{-1}}$
  - $\mathrm{S_b}\mathbf{w} = \lambda \mathrm{S_w}\mathbf{w}$
    - Calculate the eigenvector of $\mathrm{S_w^{-1} S_b}$

- If not, well……….It's a tough work to do.
  - Everyone tries to avoid this
    - Using PCA

Small Example

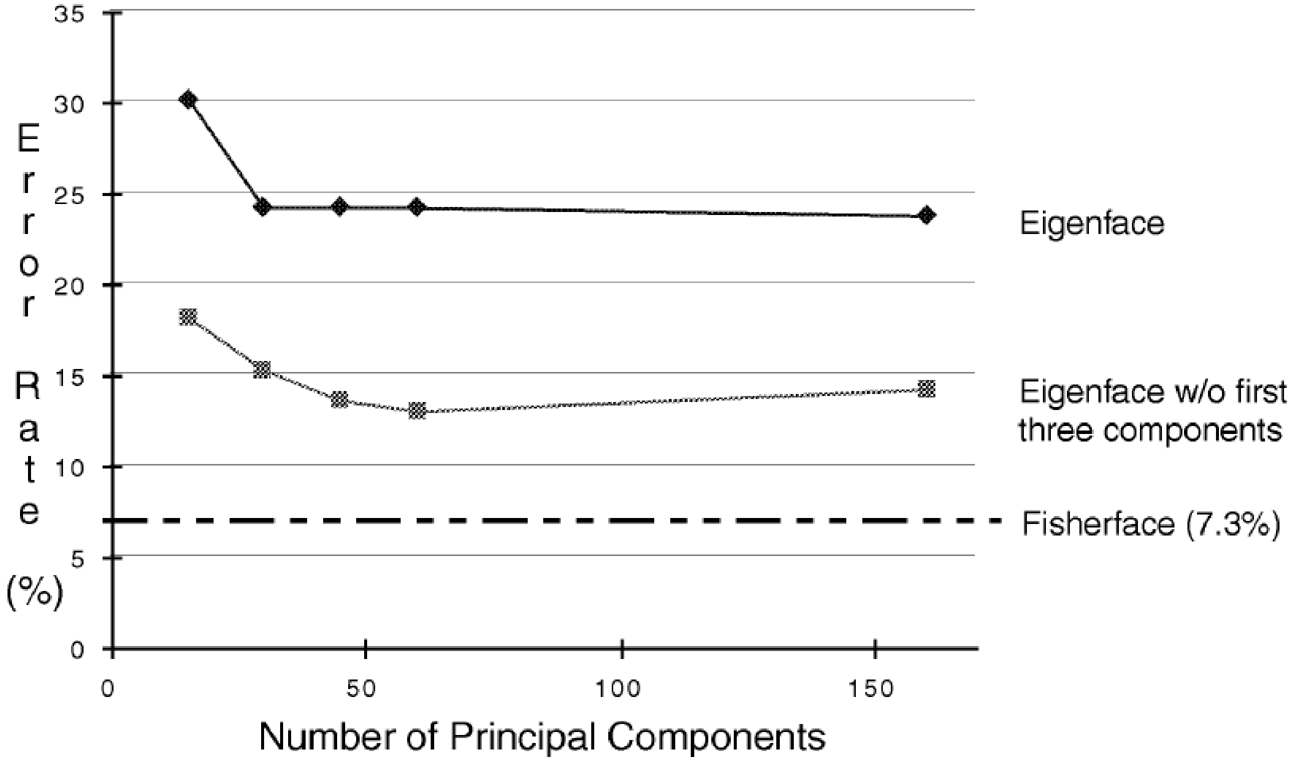






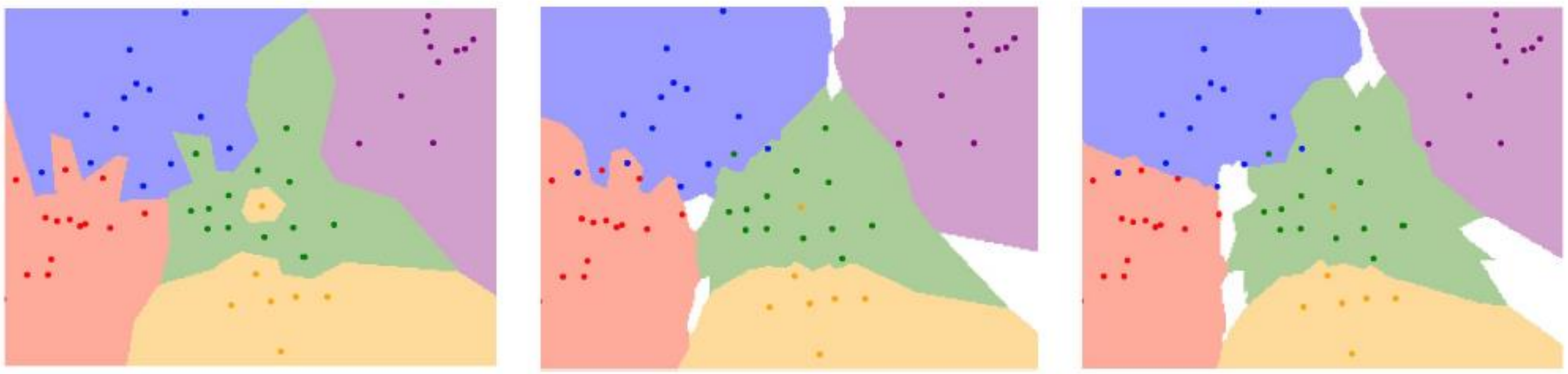
Experiment

Matthew Turk and Alex Pentland, “Eigenfaces for Recognition,” Journal of Cognitive Neuroscience, Match 1991.
Peter N. Belhumeur, Joao P. Hespanha, and David J. Kriegman, “Eigenfaces vs. Fisherfaces: Recognition Using Class Specific Linear Projection,” *IEEE Transactions on Pattern Analysis And Machine Intelligence*, 1997.



Hyperparameters in ML

- Recall that for k-NN, we need to determine the k value in advance.
  - What is the best k value?
  - Or, take PCA for example, what is the best reduced dimension number?
- **Hyperparameters**: parameter choices for the learning model/algorithm
  - We need to determine such hyperparameters instead of guessing.
  - Let’s see what we can and cannot do…



k = 1

k = 3

k = 5

Image credit: Stanford CS231n

How to Determine Hyperparameters?

- Idea #1
  - Let’s say you are working on face recognition.
  - You come up with your very own feature extraction/learning algorithm.
  - You take a dataset to train your model, and select your hyperparameters (e.g., k of k-NN) based on the resulting performance.
  -

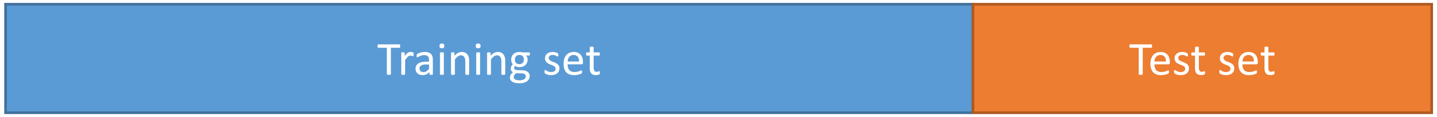
  - Might not generalize well.

Dataset

How to Determine Hyperparameters? (cont’d)

- Idea #2
  - Let’s say you are working on face recognition.
  - You come up with your very own feature extraction/learning algorithm.
  - For a dataset of interest, you split it into training and test sets.
  - You train your model with possible hyperparameter choices (e.g., k in k-NN), and select those work best on test set data.
  -

  - That’s called cheating…



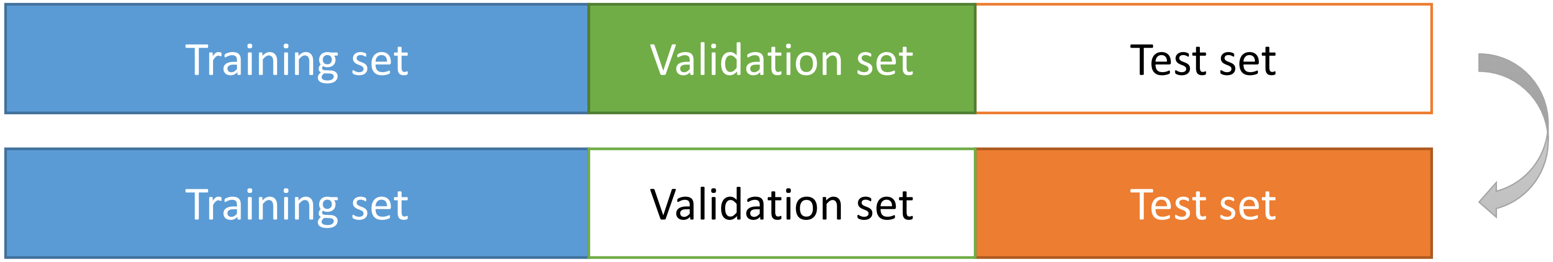
Training set

Test set

How to Determine Hyperparameters? (cont’d)

- Idea #3
  - Let’s say you are working on face recognition.
  - You come up with your very own feature extraction/learning algorithm.
  - For the dataset of interest, it is split it into training, validation, and test sets.
  - You train your model with possible hyperparameter choices (k in k-NN), and select those work best on the validation set.
  -

  - OK, but…

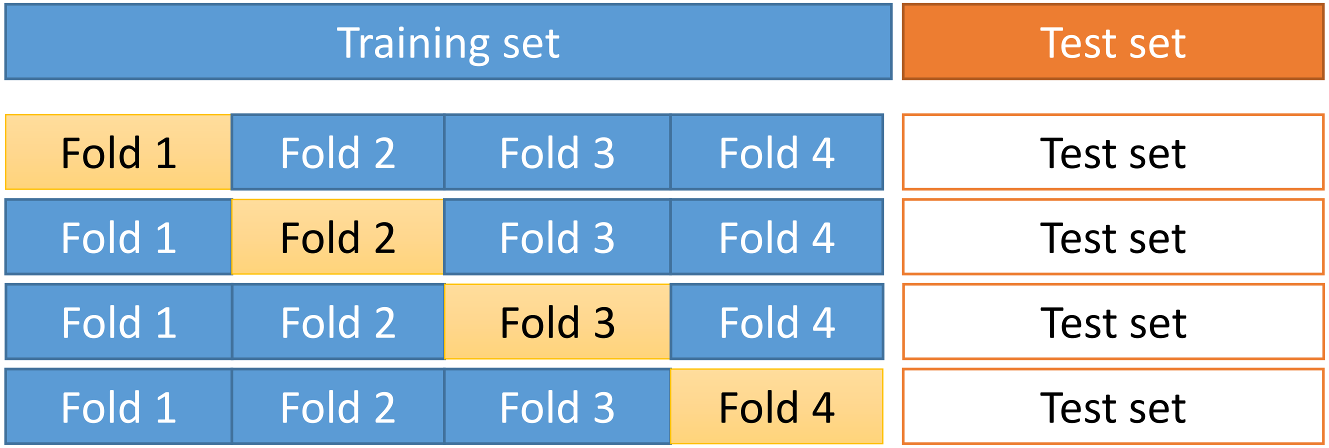


Training set | Validation set | Test set

Training set | Validation set | Test set

How to Determine Hyperparameters? (cont’d)

- Idea #3.5
  - What if only training and test sets are given, not the validation set?
  - Cross-validation (or *k-fold* cross validation)
    - Split the training set into k folds with a hyperparameter choice
    - Keep 1 fold as validation set and the remaining k-1 folds for training
    - After each of k folds is evaluated, report the average validation performance.
    - Choose the hyperparameter(s) which result in the highest average validation performance.
  - Take a 4-fold cross-validation as an example…



Training set				Test set
Fold 1	Fold 2	Fold 3	Fold 4	Test set
Fold 1	**Fold 2**	Fold 3	Fold 4	Test set
Fold 1	Fold 2	**Fold 3**	Fold 4	Test set
Fold 1	Fold 2	Fold 3	**Fold 4**	Test set