Supervised Learning

Supervised Learning Overview

- Main usages of supervised learning:
  - **Classification**: The expected output is one of several categories. Such an agent / program is called a **classifier**.
  - **Regression**: The expected output is a numerical value. Example: temperature in weather forecast.
- Both involve some input/output mapping: $f : x \rightarrow y$

Classification Models

Two approaches to build classification models:

- **Generative models**: Build models of the individual classes, and classification of a new sample is by comparing the “similarity” or “compatibility” to the individual models.
  - Examples: Bayesian classifiers and their derivatives like naïve Bayes, k-nearest-neighbor, etc..
- **Discriminative models**: Models that do classification by identifying the differences or boundaries between classes.
  - Examples: Logistic regression, SVM, decision trees, etc..

Supervised Learning Overview

Training and testing:

- Training data: Collect samples in the form of $(x \rightarrow y)$.
  - For such a pair of $(x \rightarrow y)$, y is often called the ground-truth or target output of x.
  - If y represents a category (class), then it is also called the class label (or simply the label) of x.
- Training: Derive an estimated model of the mapping f using the training data.
- Testing: Using the derived model, determine y for some given x (normally not in the training data).

Supervised Learning Overview

- Stages of training a model for $f : x \rightarrow y$:

Model Type

Polynomial?
Mixture of Gaussians?
Neural networks?
...

Model Design
Hyper-parameters

Polynomial: degree?
Mixture of Gaussians: #Gaussians?
Neural networks: #neurons?
...

Parameter Estimation

Polynomial: coefficients
Mixture of Gaussians: means/covars
Neural networks: network weights
...

Bias-Variance Dilemma

Desired properties of the model:

- Low **Bias** (goodness of fitting):
  - The estimated mapping should fit to the training data as well as possible.
  - Goal: To capture the behaviors of the underlying mapping as well as possible.
- Low **Variance** (high confidence of prediction):
  - Assume that we have several estimated mappings, each derived from a different set of training samples.
  - For a given $\boldsymbol{x}$, each mapping gives an estimation for $\boldsymbol{y}$.
  - The estimations for $\boldsymbol{y}$ should by as similar to each other as possible.
  - Goal: To be confident of the estimation of $\boldsymbol{y}$.

Bias-Variance Dilemma

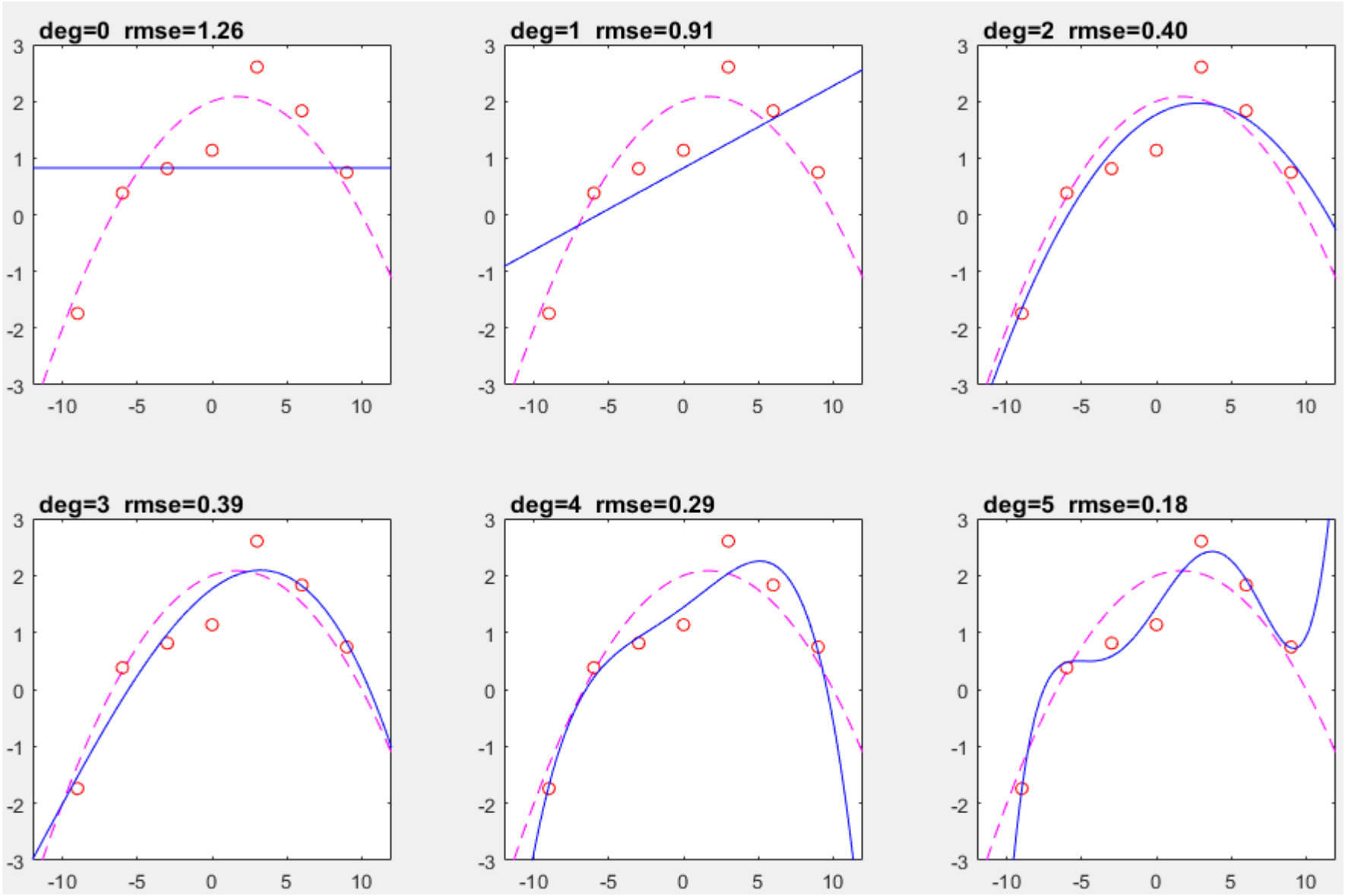
Now, difficulties for supervised learning:

- To reduce **Bias**
  - ➔ Use more complex models
  - ➔ BOTH signal and noise are learned
  - ➔ Overfitting
- To reduce **Variance**
  - ➔ Try to avoid learning noise (variance is caused by noise)
  - ➔ Use less complex models
  - ➔ Signal is not sufficiently learned
  - ➔ Underfitting

Here we use the term "noise" to represent variations in data that are not relevant to the prediction.

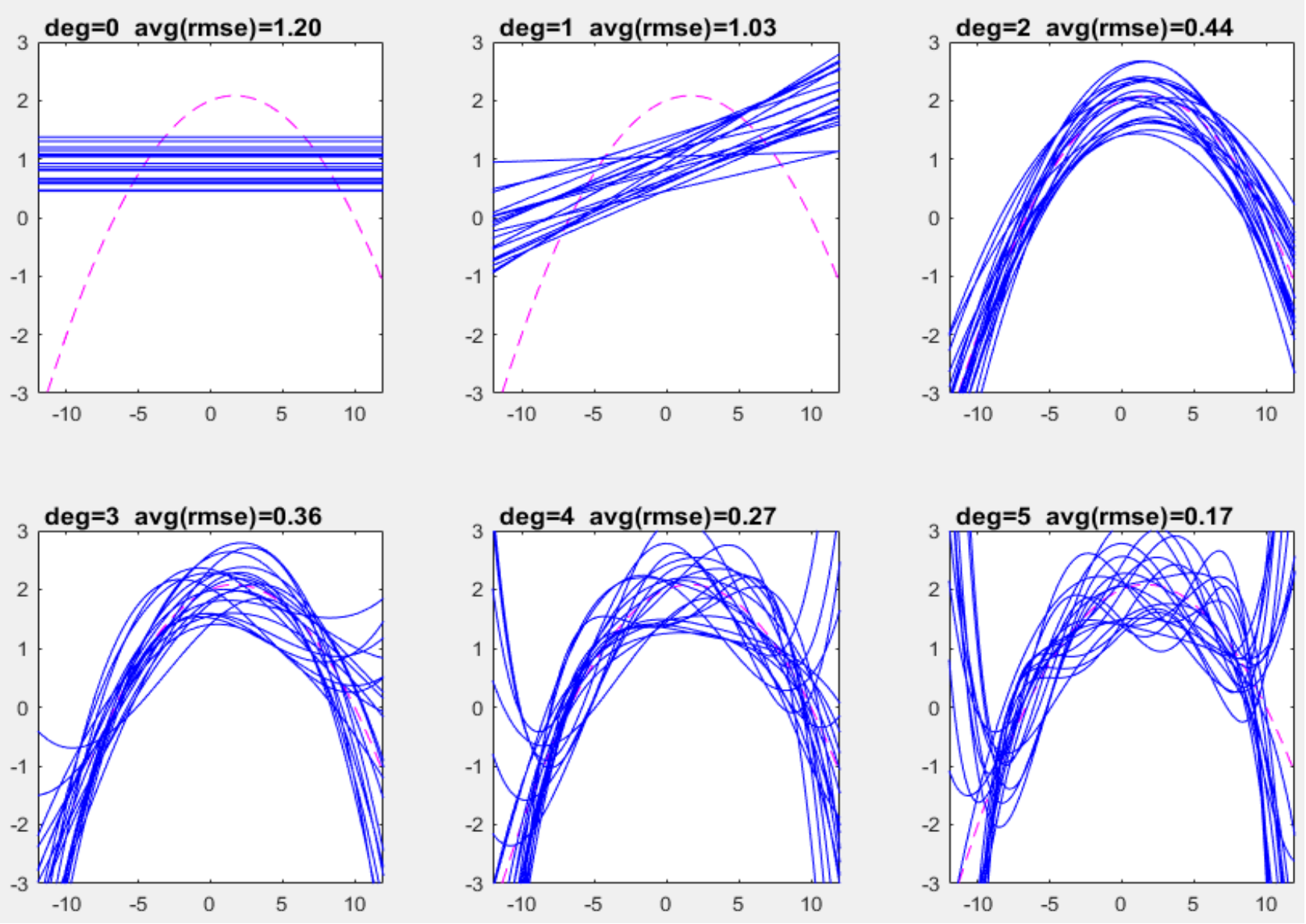
Fitting and Model Complexity

Using polynomial regression as an example:



Fitting and Model Complexity

Illustration of corresponding variances:



Each degree is sampled 20 times.

What to Do Now?

Several representative approaches:

- Increasing the number of training samples.
- Trying to determine the suitable level of model complexity.
- Regularization (the "suppression" of model complexity during training)
- Consensus-based methods (ensembles)

Amount of Training Data

- The usual case: The more training data, the better.
- Possible difficulties:
  - Data availability
  - Increased computational cost
- When there are few training samples, some classifier types (e.g., support vector machines) generally work better than others.
- **Data augmentation** and **resampling**: Produce new training data through the perturbation or combination of one or multiple samples of the same class.

Regularization

- Include the **bias-variance** trade-off during the learning process:
  - **Regularization**: Modify the learning objective to minimize BOTH the bias and the model complexity. (Normally, the learning process only attempts to minimize bias.)
  - Example **cost function** (to be minimized) for polynomial regression:

To minimize:

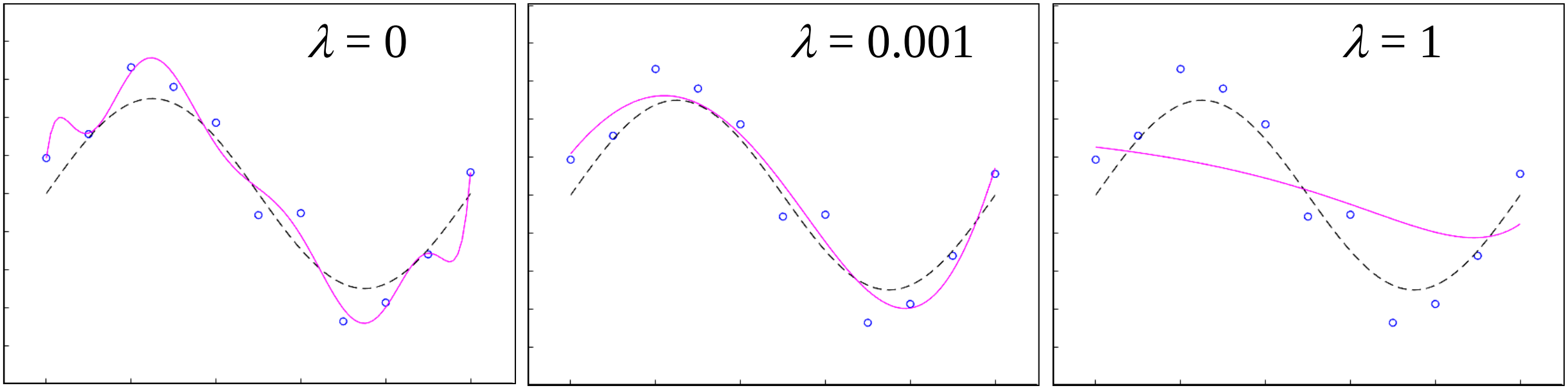
$$E = \sum_i \left| y_i - \sum_{k=0}^{d} a_k x_i^k \right|^2 + \lambda \sum_{k=0}^{d} a_k^2$$

Fitting Error (reducing bias)

Regularization (reducing complexity)

Regularization

- Example (polynomial regression):



N=11, degree=9

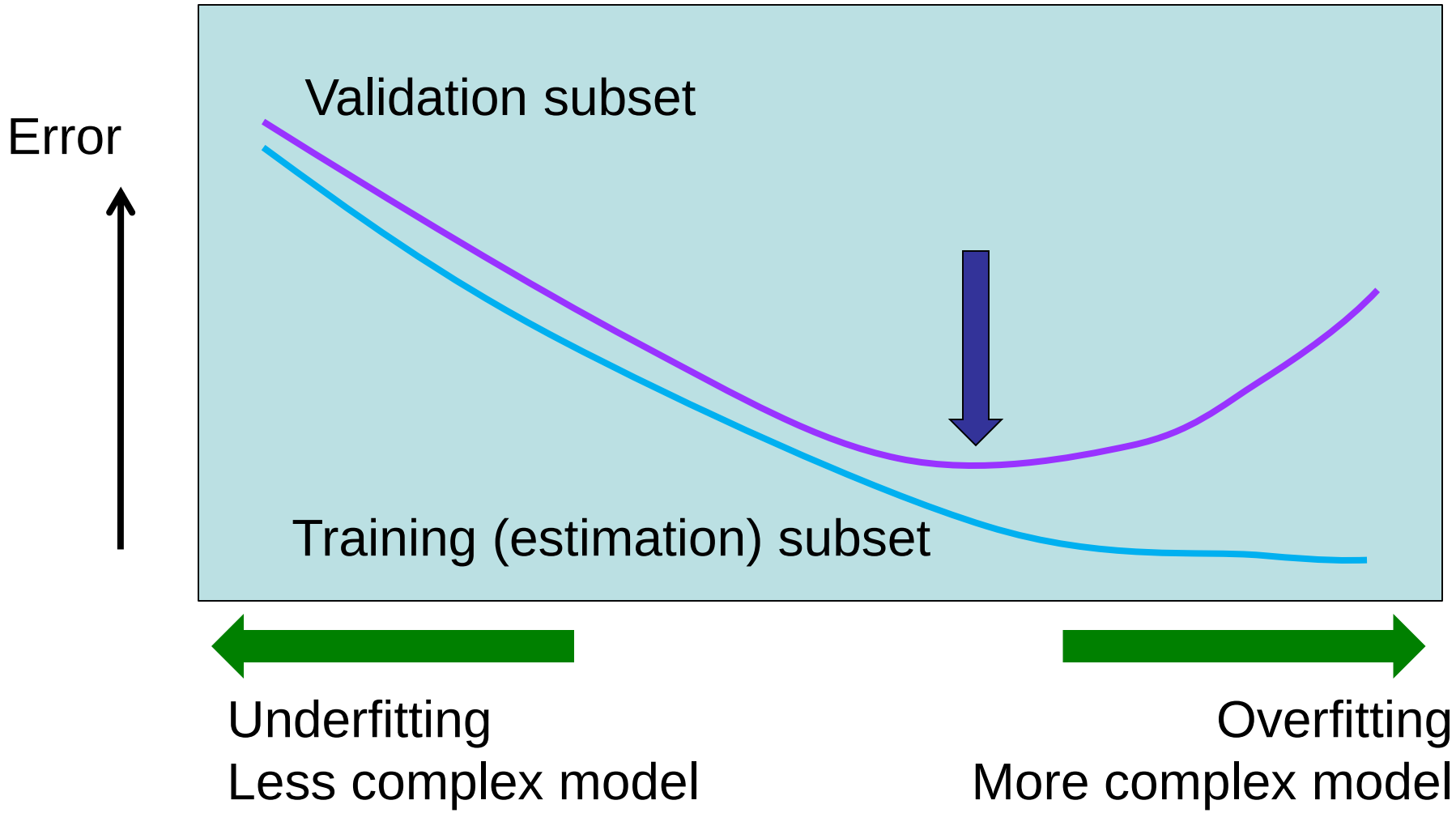
Consensus Based Methods

- Build many somewhat different models (an **ensemble**) for the same underlying mapping.
- The consensus should better represent the "signal" part, and the "noise" part is more likely to be averaged out. This leads to the reduction of variance with minimal effect on bias.
- The models in an ensemble need to be diverse:
  - Random subset of attributes/features
  - Resampling (bootstrapping) of training samples

How Much Model Complexity?

- **Ockham's Razor**: Among different ways to explain the data, choose the simplest one.
- Model complexity is usually controlled / adjusted using some **hyper-parameters**.
- Example methods: **cross-validation**, post-processing model pruning.
- Many of these methods utilize **validation data** (training samples not used for model parameter estimation) to check the **generalization** ability of the built model.

Validation


Error
Validation subset
Training (estimation) subset
Underfitting
Less complex model
Overfitting
More complex model

Cross-Validation

- The N labeled samples are divided into K subsets of approximately equal sizes ($K>1$). They should have similar distributions.
- The training process (model parameter estimation) is run K times (K trials).
- In the Kth trial, the Kth subset is used for validation, and the other subsets are used for training.
- The overall performance is the combination of the performance on all the validation subsets.

Cross-Validation

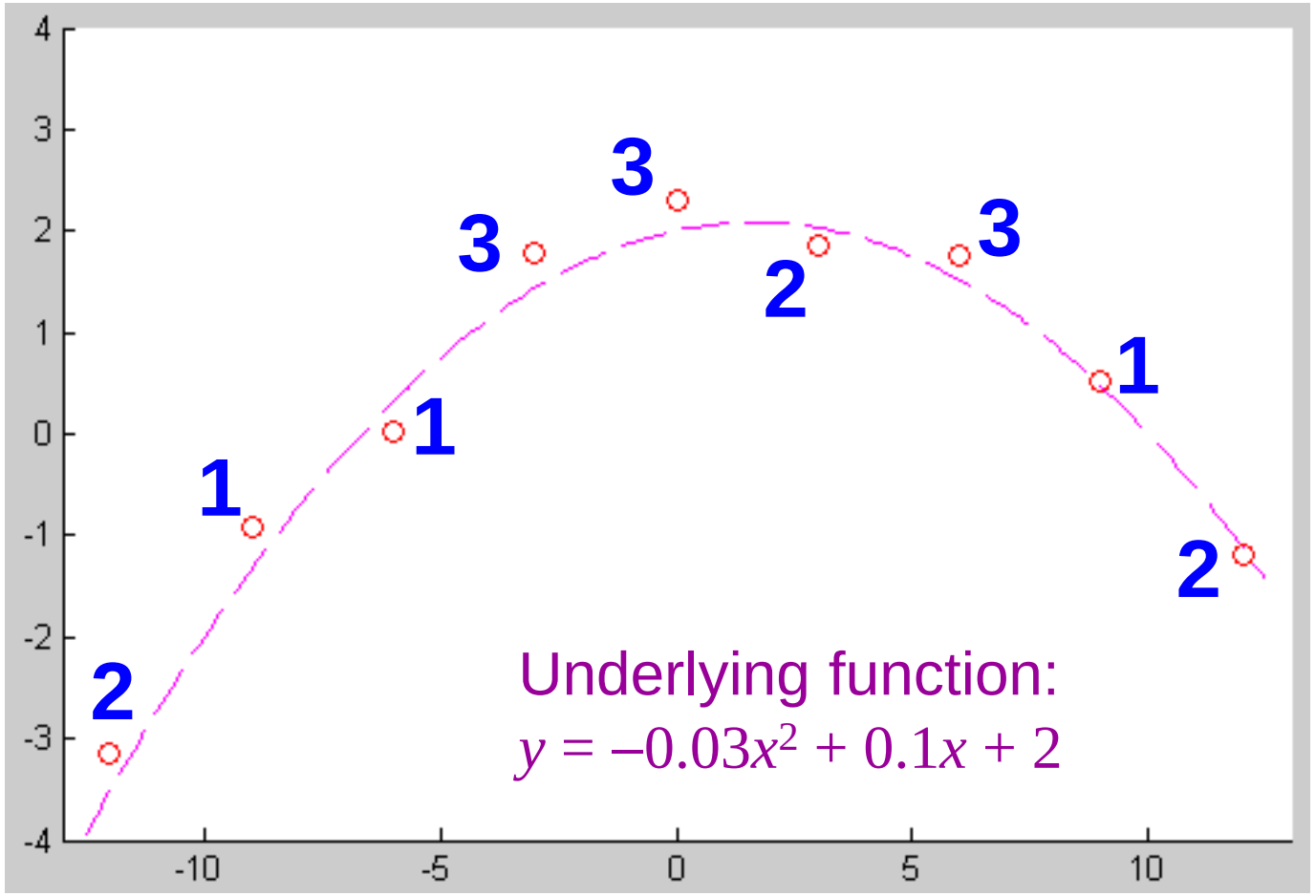
4-fold cross-validation: parameter estimation validation

Extreme case: The leave-one-out method ($K=N$).

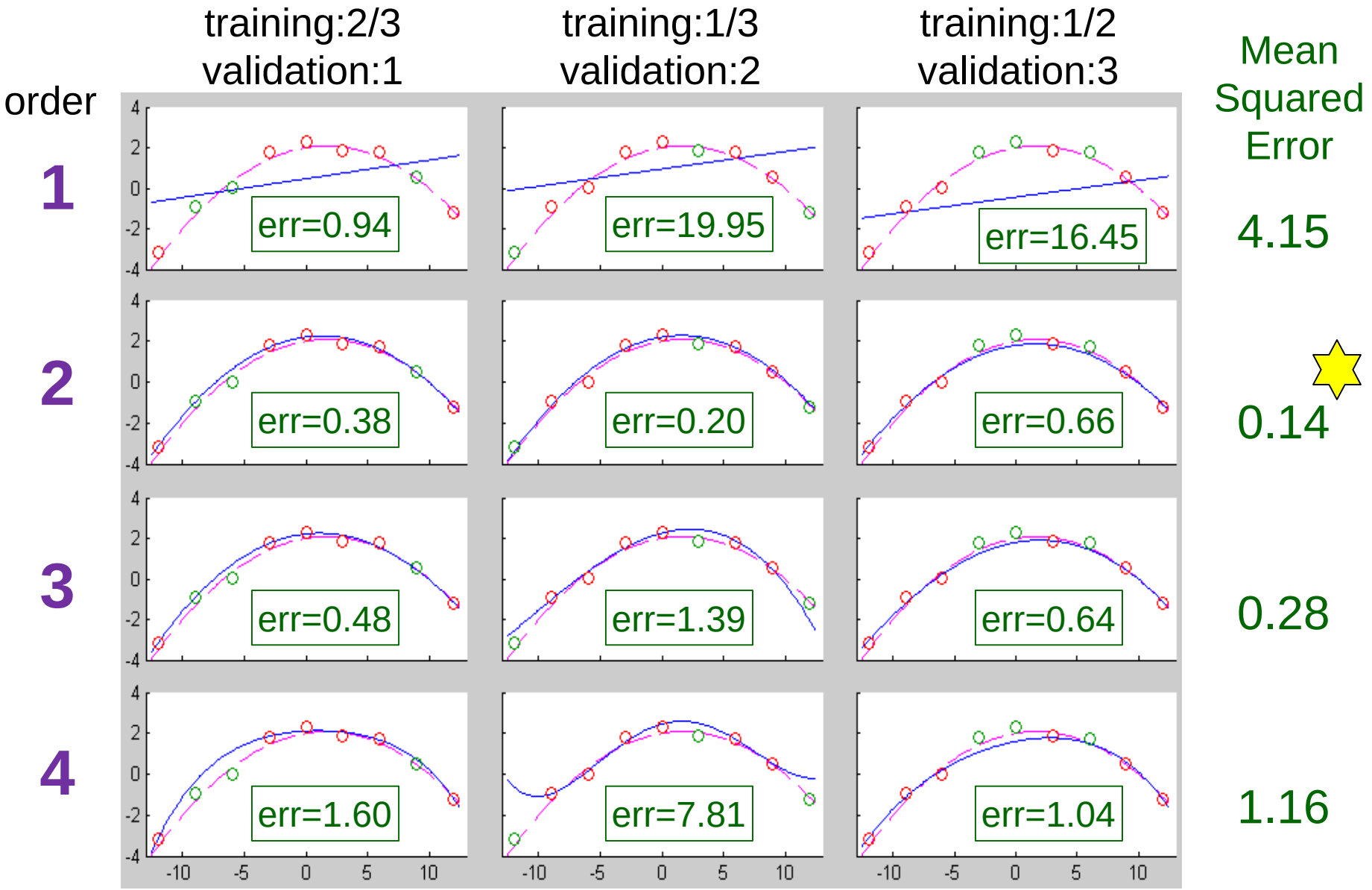
Model Selection by Cross-Validation

Example: polynomial regression:

9 training samples → 3 subsets

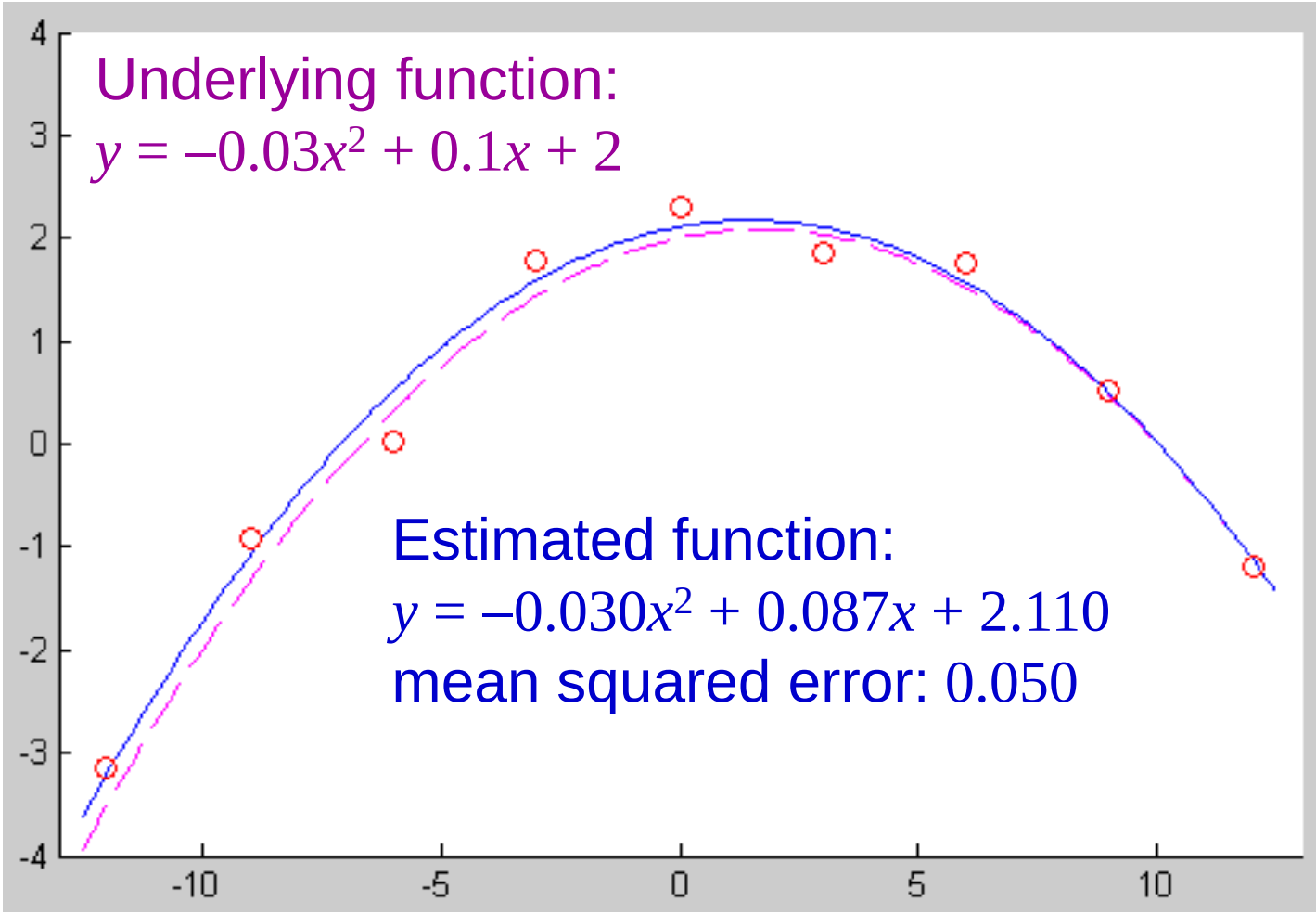


Model Selection by Cross-Validation



Model Selection by Cross-Validation

Finally, use the selected model and ALL the training samples for parameter optimization:

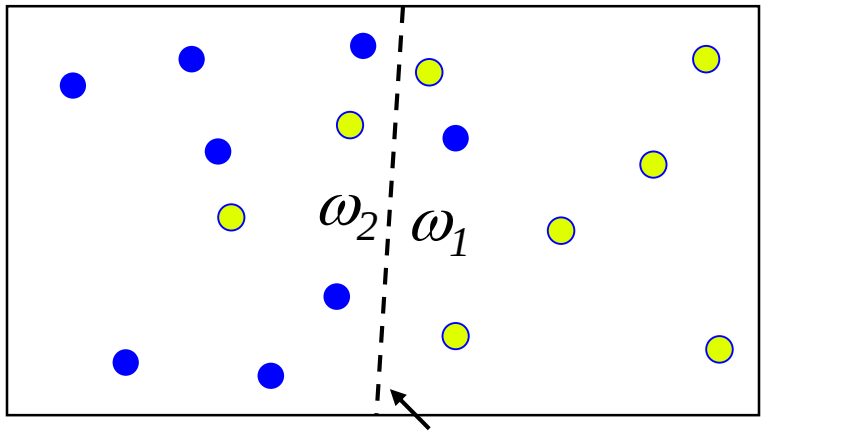


Classifier Evaluation

The most simple way to evaluate a classifier is to see the number of correct and incorrect classifications.

For a M-class problem with N samples to be classified, the ***confusion matrix*** is a MxM matrix, whose (i,j) element is the number of vectors that are actually in class ω_i and classified to class ω_j.

Example (2-class): ω_1 ω_2



Confusion Matrix: $\begin{pmatrix} 6 & 2 \\ 1 & 7 \end{pmatrix}$

Correct classification rate
= trace(Confusion Matrix)/N

Confusion Matrix

A typical confusion matrix for the Iris dataset:

50	0	0
0	46	4
0	0	50

Clothing color classification:

Real \ Pred	Red	Orange	Yellow	Green	Blue	Pink	Purple	Brown	Gray	Black	White
Red	167	17	1	0	4	23	8	4	3	9	2
Orange	4	37	13	0	2	0	0	2	0	1	0
Yellow	3	1	87	5	0	3	0	5	3	1	3
Green	0	0	9	100	7	2	0	3	8	8	3
Blue	0	0	0	13	450	10	6	0	42	114	21
Pink	16	2	2	0	2	124	6	3	5	2	9
Purple	9	0	1	1	23	21	70	1	7	15	2
Brown	3	2	8	12	0	7	0	66	14	22	7
Gray	4	0	1	23	21	15	1	14	289	38	38
Black	10	1	0	15	44	15	15	5	49	903	9
White	1	0	2	7	29	26	2	4	52	9	322

Two-Class Confusion Matrix

Many two-class problems can be considered as "detection" problems where the classifier is expected to answer a "Yes/No" question for each sample, such as in a medical screening test.

Let ω_1 be "No", ω_2 be "Yes", confusion matrix be $\begin{pmatrix} TN & FP \\ FN & TP \end{pmatrix}$

Common metrics (in pairs) derived from the confusion matrix:

PD (probability of correct detection) = $TP / (TP + FN)$
FA (probability of false positive/alarm) = $FP / (TN + FP)$
Recall = PD
Precision = $TP / (TP + FP)$
Sensitivity = PD
Specificity = $TN / (TN + FP) = 1 - \mathbf{FA}$
PPV (positive predictive value) = **Precision**
NPV (negative predictive value) = $TN / (TN + FN)$

Two-Class Confusion Matrix

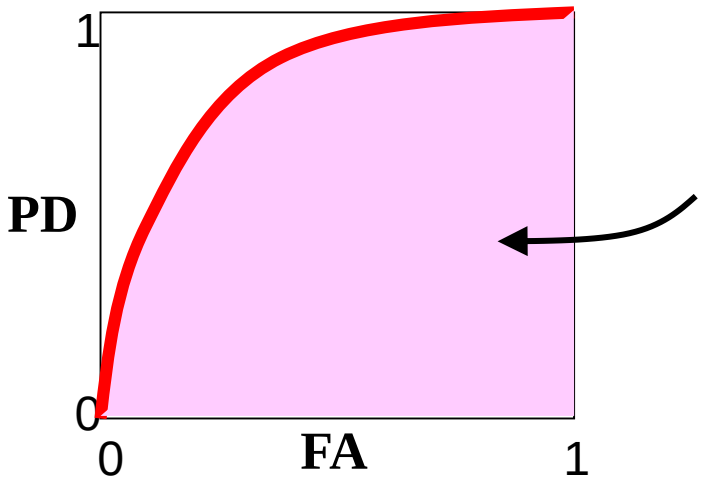
- A threshold / bias can be used to adjust how sensitive the classifier is to identify positive cases.
- This adjustment, while increasing one metric in a pair (e.g., recall), is likely to decrease the other (e.g., precision).
- **F1 measure** is one combined metric to allow for easier comparison between such paired classification results. It can also be used to select a “proper” threshold / bias.

$$F1 = \frac{2 * precision * recall}{precision + recall}$$

ROC Curves

Receiver Operating Characteristics (ROC) Curve is the plot of **PD** vs. **FA** at different threshold (bias) values.

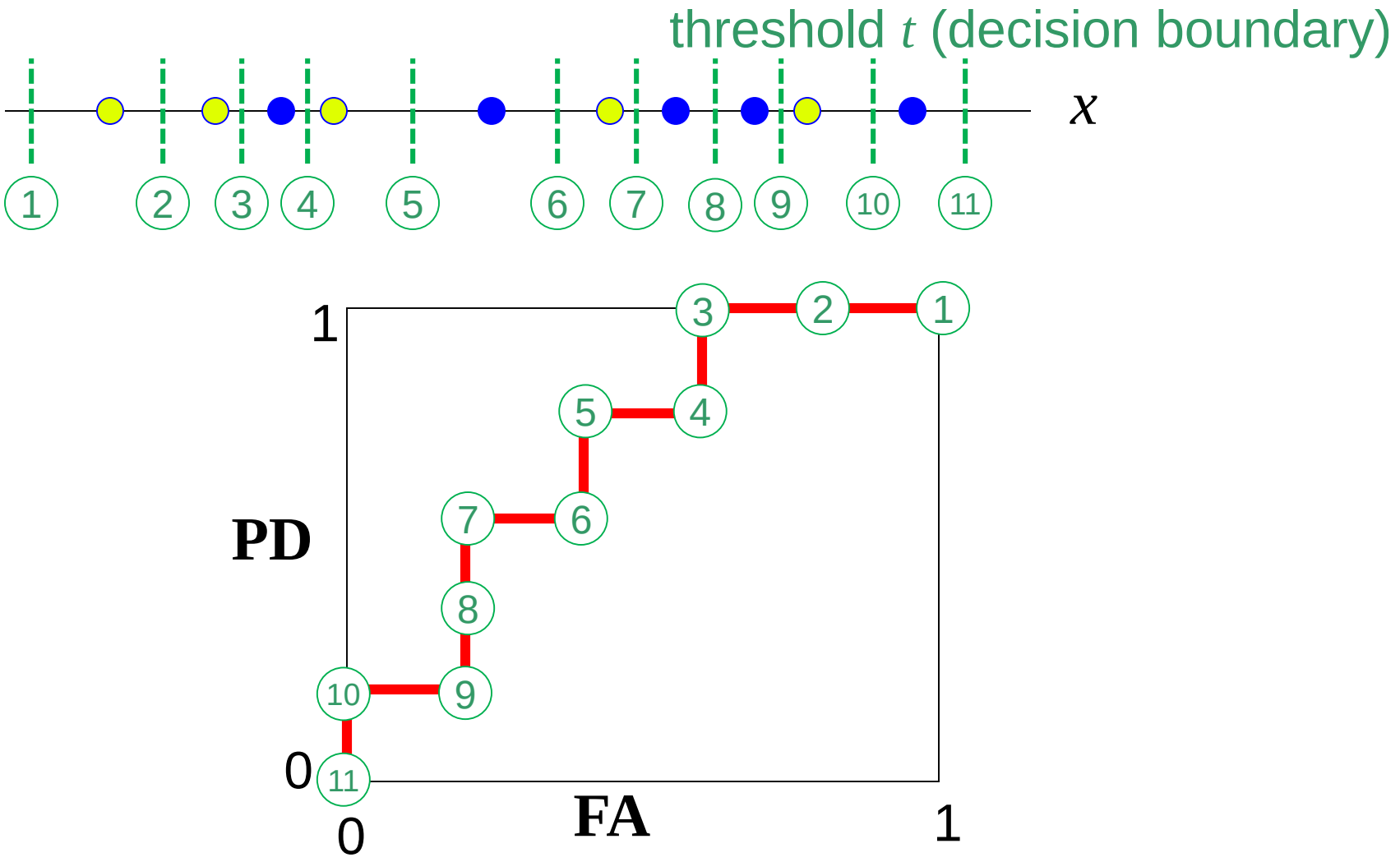
It allows the separation of the evaluation of different classification methods and/or settings from the choice of the threshold.



AUC (area under the ROC curve):
The larger, the better.

ROC Curves

Example (1-D): negative positive



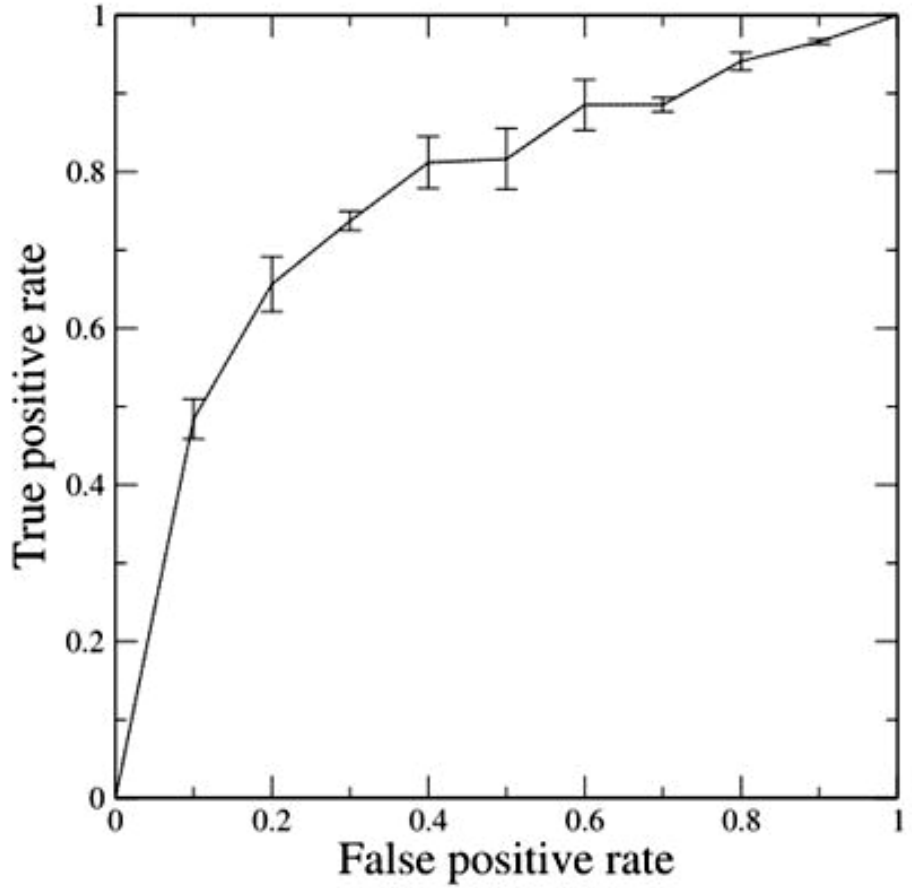
Classifier Evaluation w/ Cross-Validation

- Confusion matrix: Add the validation-subset confusion matrices from all the trials. Each sample is included exactly once. Metrics based on the confusion matrix, such as PD and FA, can then be computed.
- ROC curve (two-class problems):
  - Method 1: Just collect the outputs of the validation subsets from all the trials and draw a single ROC curve.
  - Method 2: Compute a ROC curve for each trial. We can average the curves at any given FA rate. The standard deviation gives us an estimation of the uncertainty of PD at any FA rate.
  - Method 3: Use a common set of thresholds to compute separate PD and FA values for all the validation subsets. This produces a single ROC curve with uncertainties for both PD and FA.

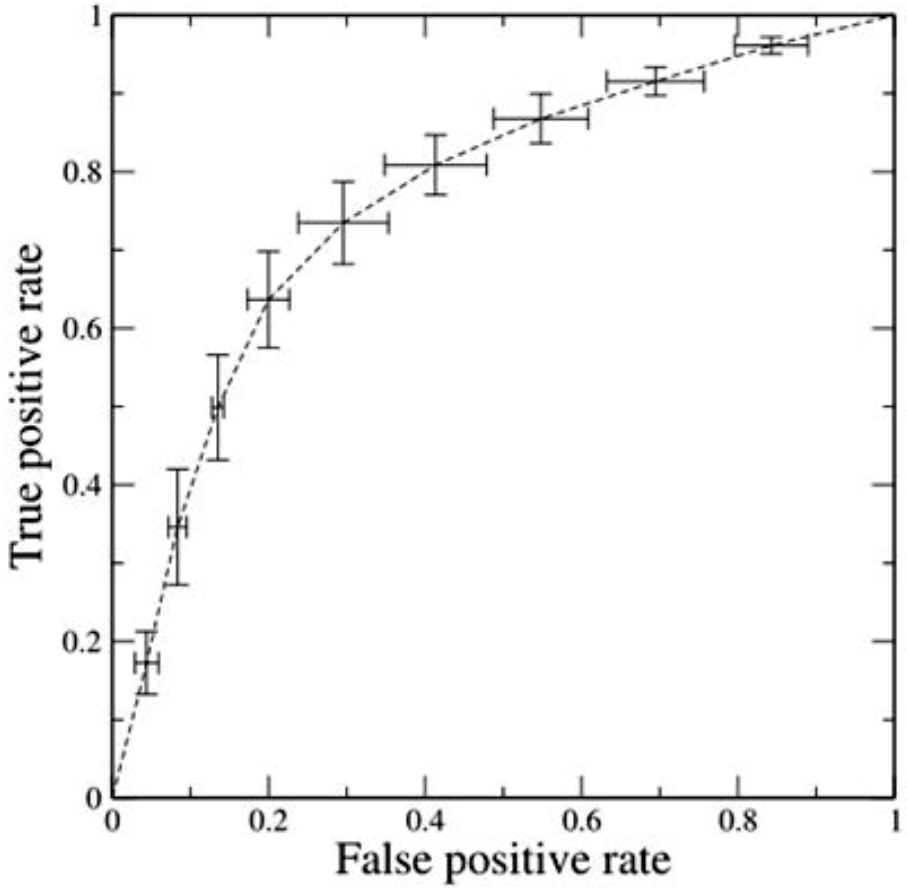
Classifier Evaluation w/ Cross-Validation

Examples of generating ROC curves with cross-validation:

(method 2)



(method 3)



https://slideplayer.com/slide/7028153/

A Look at Features

- Features are the set of numbers/categories we use to represent the samples.
- Features are domain specific.
- Using the very famous Iris dataset as an example:
  - Data: Each sample represents an iris flower.
  - Four features for each flower: sepal length, sepal width, petal length, petal width.

Iris sestosa

Iris versicolor

Iris virginica

Sepal Leng.	Sepal Width	Petal Leng.	Petal Width	Sepal Leng.	Sepal Width	Petal Leng.	Petal Width	Sepal Leng.	Sepal Width	Petal Leng.	Petal Width
5.1	3.5	1.4	0.2	7.0	3.2	4.7	1.4	6.3	3.3	6.0	2.5
4.9	3.0	1.4	0.2	6.4	3.2	4.5	1.5	5.8	2.7	5.1	1.9
4.7	3.2	1.3	0.2	6.9	3.1	4.9	1.5	7.1	3.0	5.9	2.1

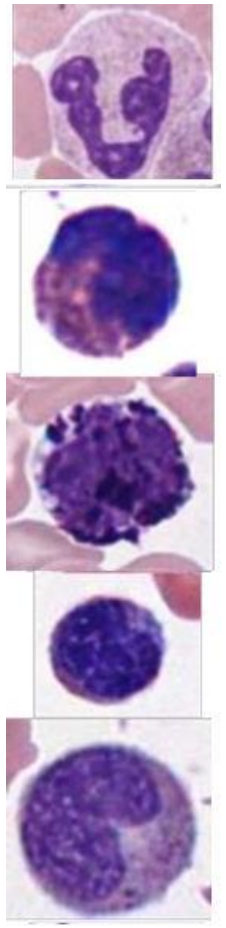
and more …

A Look at Features

- Some features are from direct measurements, or are otherwise straightforward.
  - Features for Iris
  - A person's age, gender, etc.
- For many problems, the "raw" features are difficult to use. Examples:
  - Images
  - Speech; audio signals
  - Trajectory, such as online handwriting recognition
  - Text
  - ...

A Look at Features

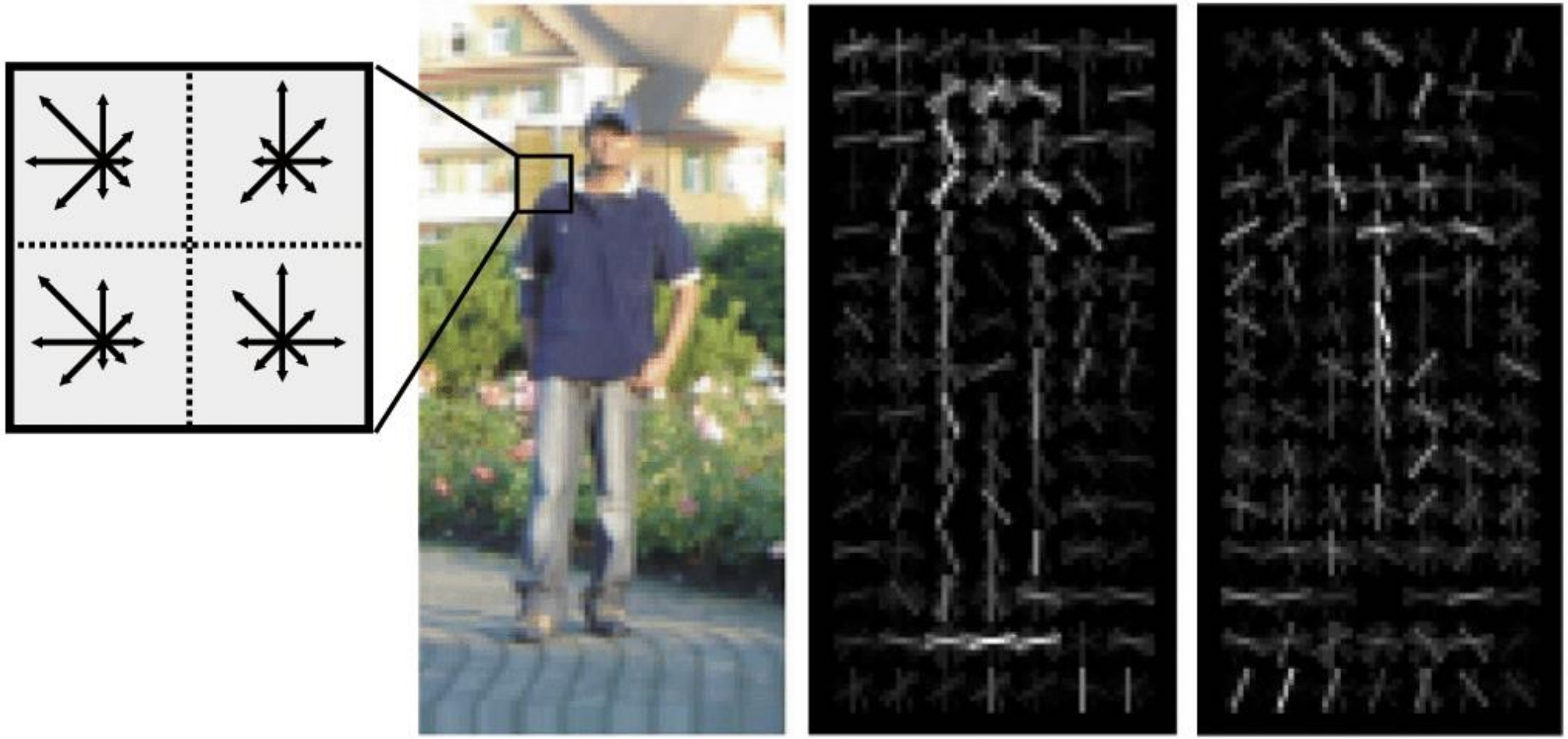
- For problems where it is impractical to use the "raw features", we need to find ways to extract meaningful and manageable "derived features" from the raw features, and then use these derived features for our classification / regression tasks.
- Example:



size (area)
ratio of nuclei
shape features
    moments
    etc.
texture features
    DCT
    etc.

A Look at Features

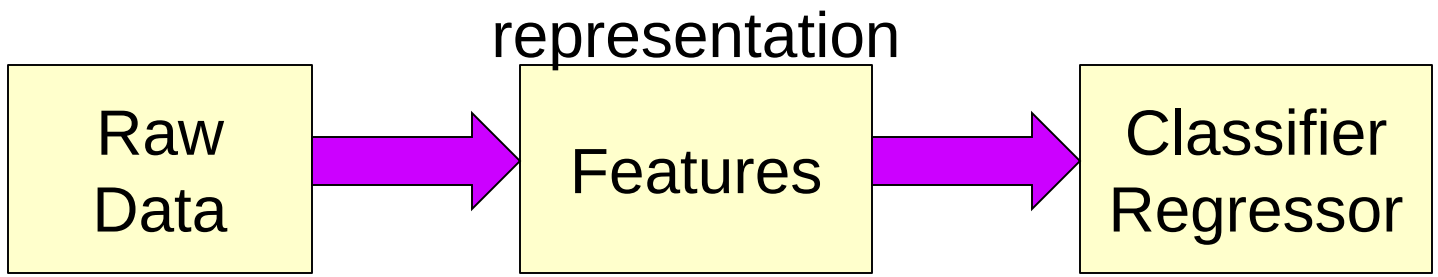
- Another classic example: The HOG (histogram of oriented gradients) features for pedestrian detection. This was the state-of-the-art before CNNs became practical.



http://doi.ieeecomputersociety.org/10.1109/TPAMI.2009.122

A Look at Features

- The standard approach:

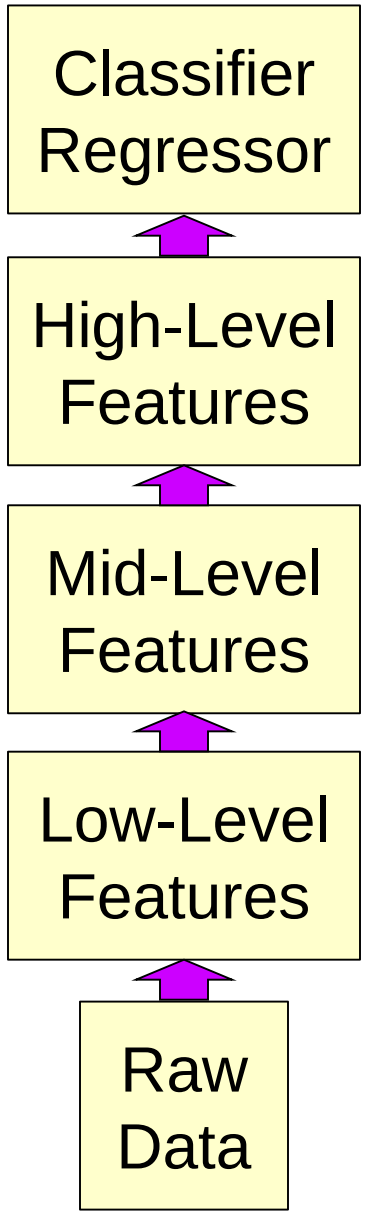


- For each type of data, there are numerous types of "derived features".
- Good features are usually more important than good classifiers/regressors for solving a particular problem. Examples:
  - Haar features for face detection
  - HOG for pedestrian detection
  - Bag of Words for text classification
  - MFCC for audios

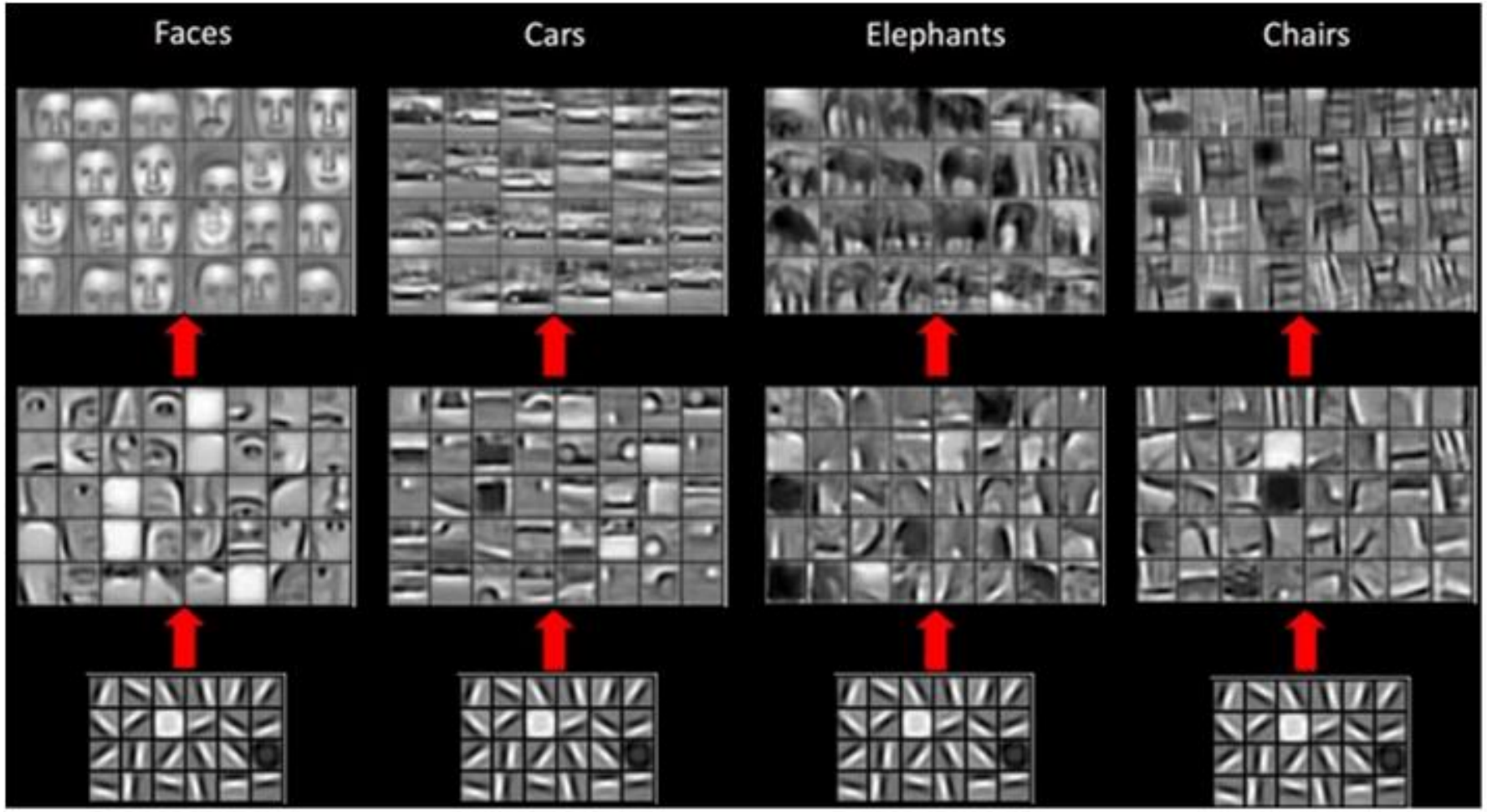
A Look at Features

- However, even good hand-crafted features are never good enough.
  - They can not capture all the useful information.
  - They can only represent the low-level information (easier to define and compute) well.
  - It is very difficult to design features for high-level and semantically rich information.
    - To understand this, consider the HOG features for pedestrian detection. They do not provide us with information regarding whether and where the head, arms, or legs are detected.

Neural Networks as Feature Extractors



This is how our brain processes visual inputs.



https://www.researchgate.net/figure/Pictorial-representation-of-features-in-3-layers-of-a-CNN-34-Notice-the-increasing_fig2_308883811

Dimensionality Reduction with PCA

Motivation: When you have a large number of features for each sample, it is very likely that the features are correlated (and therefore somewhat redundant).

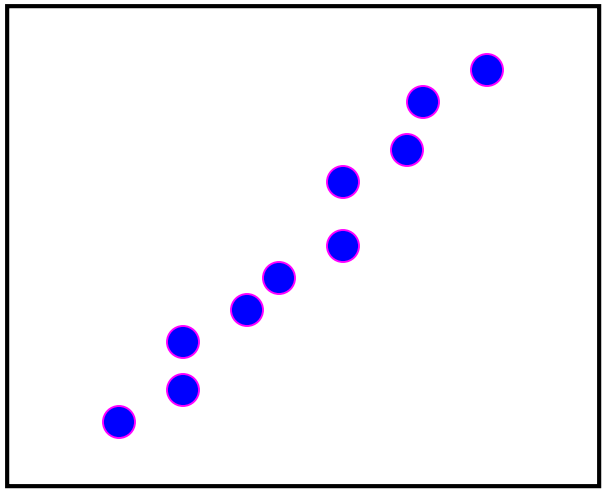
The goal of ***Principle Component Analysis*** (PCA) is to find a linear transform that minimizes the correlation between different components (features) in the transformed space. If we have the transform in the form

$$\boldsymbol{y} = A^T \boldsymbol{x}$$

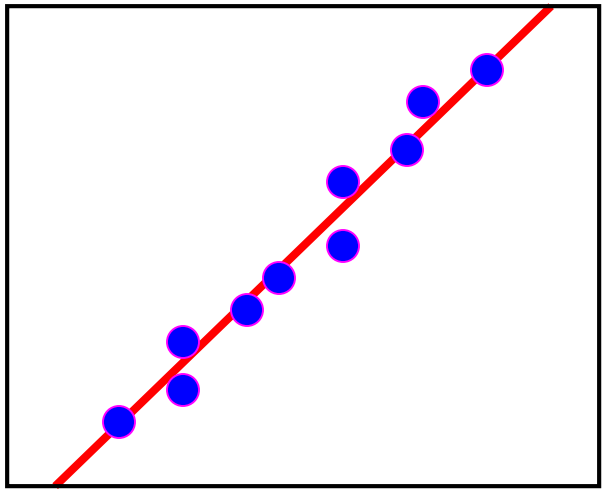
where $\boldsymbol{x}$ is the original feature vector, then the goal is to find A such that

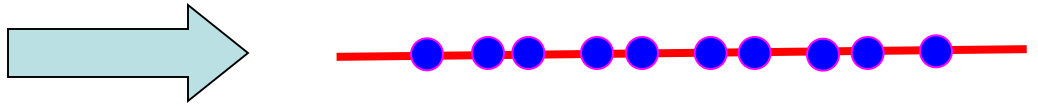
$$E[y_i y_j] = 0, \quad \forall i \neq j$$

PCA Idea



These 2-D feature vectors have very high correlation between the two features, x_1 and x_2.





We can get a pretty good approximation of the vectors by projecting them onto only one dimension.

PCA with Covariance Matrix

A covariance matrix is symmetric, and its eigenvectors are orthogonal.

$$\Sigma_y = A^T \Sigma_x A$$

where A now consists of the eigenvectors of Σ_x.

If A consists of only a subset of the eigenvectors of Σ_x, the transform now projects $\boldsymbol{x}$ onto a subspace of the feature space (with origin at the global mean $\boldsymbol{\mu}$) spanned by these selected eigenvectors.

PCA as Data Approximation

The next task is to determine which eigenvectors to include in the projection matrix. This brings us to another goal of PCA, which is to provide a minimum-squared-error approximation of the data in a lower-dimensional space.

Assume that we have subtracted $\boldsymbol{\mu}$ from all $\boldsymbol{x}$ (so that we have zero mean), and let $\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_l$ be the l orthonormal basis vectors of the subspace satisfying

$$\boldsymbol{e}_i{}^T \boldsymbol{e}_j = 0 \quad \text{for } \forall i \neq j \qquad \text{and} \quad \boldsymbol{e}_i{}^T \boldsymbol{e}_i = 1 \quad \text{for } \forall i$$

The projection of $\boldsymbol{x}$ onto this subspace is given by

$$\sum_{j=1}^{l} (\boldsymbol{x}_i^T \boldsymbol{e}_j) \boldsymbol{e}_j$$

PCA as Data Approximation

Then PCA minimizes the cost function:

$$J = \sum_{i=1}^{N} \left\| \boldsymbol{x}_i - \sum_{j=1}^{l} (\boldsymbol{x}_i^T \boldsymbol{e}_j)\boldsymbol{e}_j \right\|^2$$

Some manipulation gives the following Lagragian:

$$L = \sum_{i=1}^{N} \| \boldsymbol{x}_i \|^2 - \sum_{j=1}^{l} \left(\boldsymbol{e}_j{}^T \Sigma_x \boldsymbol{e}_j \right) - \sum_{j=1}^{l} \lambda_j \left(\boldsymbol{e}_j{}^T \boldsymbol{e}_j - 1 \right)$$

The last term is included based on the constraints (orthonormal basis vectors).

PCA as Data Approximation

Setting $\frac{\partial L}{\partial \boldsymbol{e}_j} = \boldsymbol{0}$ results in the eigenvalue problem:

$$\Sigma_x \boldsymbol{e}_j = \lambda_j \boldsymbol{e}_j$$

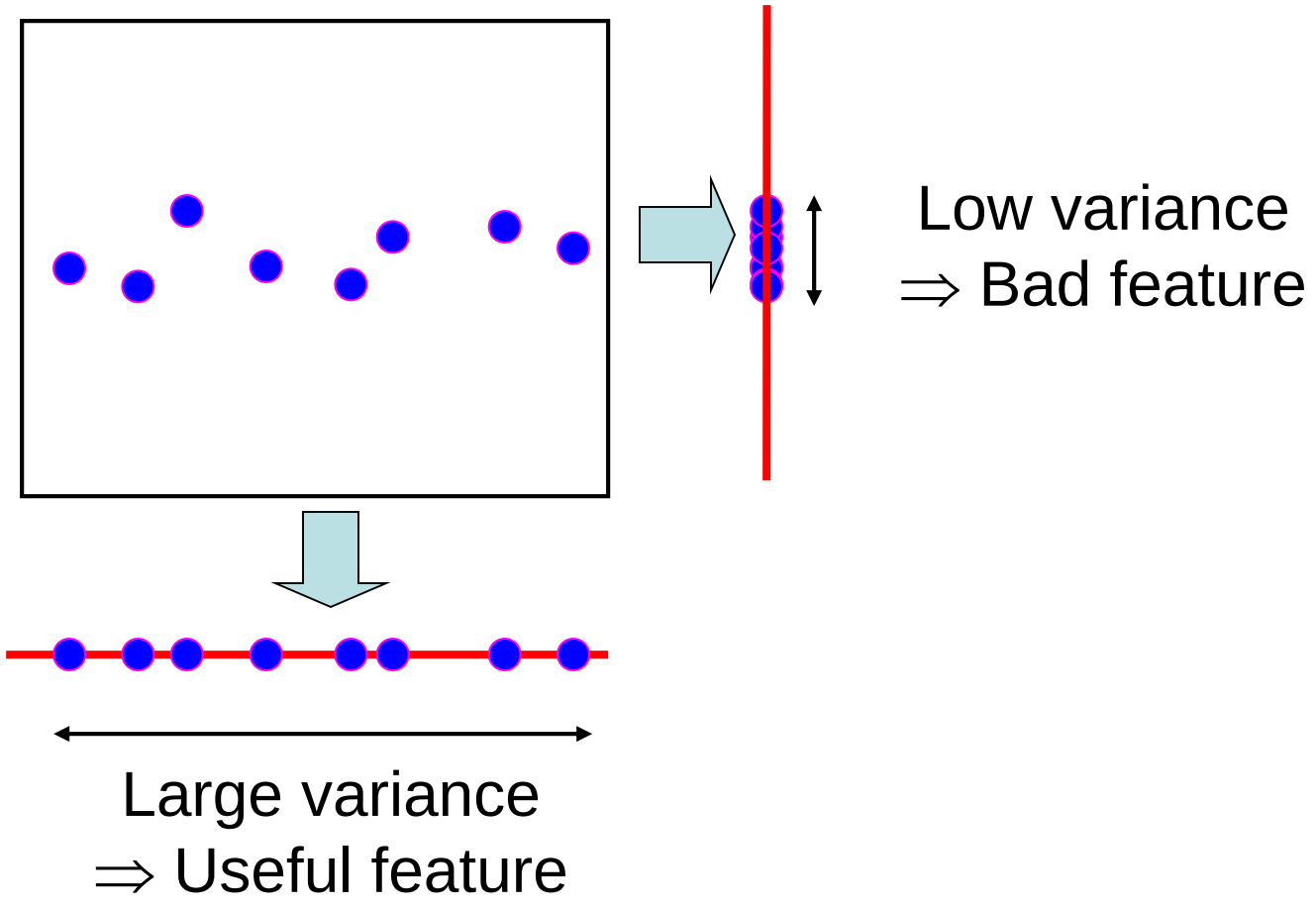
Putting this back into the original cost function gives

$$J = \sum_{i=1}^{N} \| \boldsymbol{x}_i \|^2 - \sum_{j=1}^{l} \lambda_j$$

Implication: To minimize the mean squared error after projecting the feature vectors onto a l-dimensional linear subspace, we choose the eigenvectors that correspond to the l largest eigenvalues of Σ_x to define the projection.

PCA and Variance

Variance is usually used as a measure of information contained in data. For example:



PCA and Variance

The "total variance" is the trace of the covariance matrix. This value is unchanged during an orthogonal transform (including the diagonalization), so it is given by

$$\sum_{j=1}^{m} \lambda_j$$

Similarly, the "total variance" in the projection space is

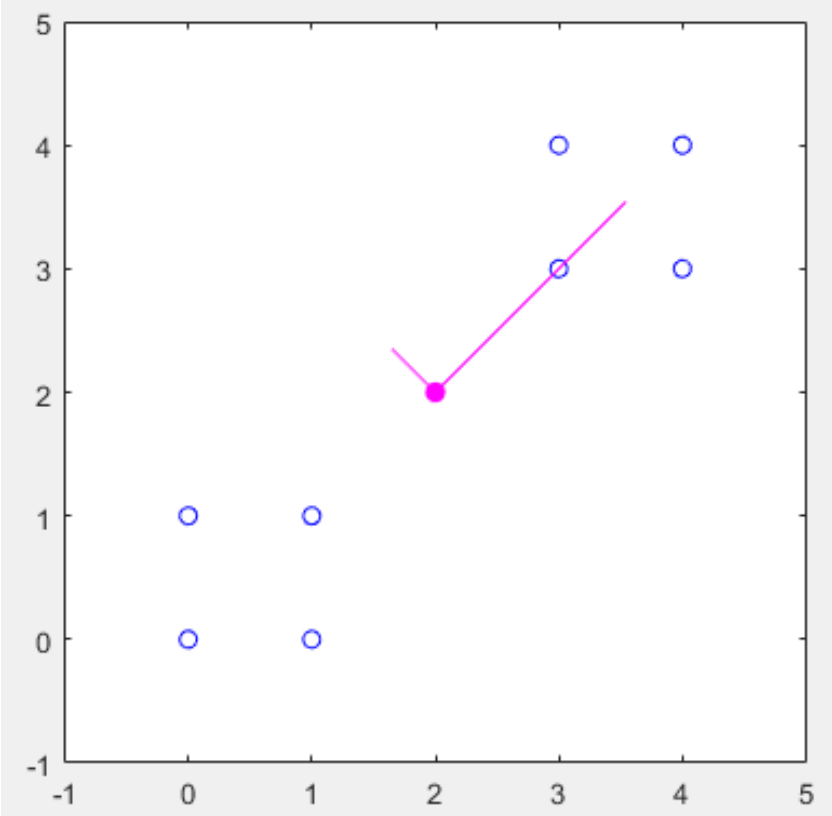
$$\sum_{j=1}^{l} \lambda_j$$

The amount of "variance accounted for", or more intuitively, the information preserved during the projection, is given by the ratio of these two numbers.

PCA Example

For the following 2-D data, let us find the 1-D representation using PCA:

$$X = \begin{bmatrix} 3 & 3 & 4 & 4 & 1 & 1 & 0 & 0 \\ 4 & 3 & 3 & 4 & 0 & 1 & 1 & 0 \end{bmatrix}$$



Eigenvalues: 38 and 2

Example PCA Application: Eigenfaces

PCA is widely used in applications where it is important to reduce the number of features.

An example is face recognition, where each input vector has the dimension of the number of pixels. A popular technique is "eigenface", where the most important eigenvectors obtained by PCA are used as the basis vectors. The recognition is then based on the projection of an input image to these "eigenfaces".

Example PCA Application: Eigenfaces

200 images (100 males + 100 females) + horizontally flipped images, size 40x40 ➔ 400 1600-D vectors

Example PCA Application: Eigenfaces

$$\Sigma_x \boldsymbol{e}_j = XX^T \boldsymbol{e}_j = \lambda_j \boldsymbol{e}_j$$

However, the covariance matrix (1600x1600) is rank deficient, given that we have only 400 samples. We then use the trick:

$$\Rightarrow X^T XX^T \boldsymbol{e}_j = \lambda_j X^T \boldsymbol{e}_j$$
$$\Rightarrow (X^T X)(X^T \boldsymbol{e}_j) = \lambda_j (X^T \boldsymbol{e}_j)$$

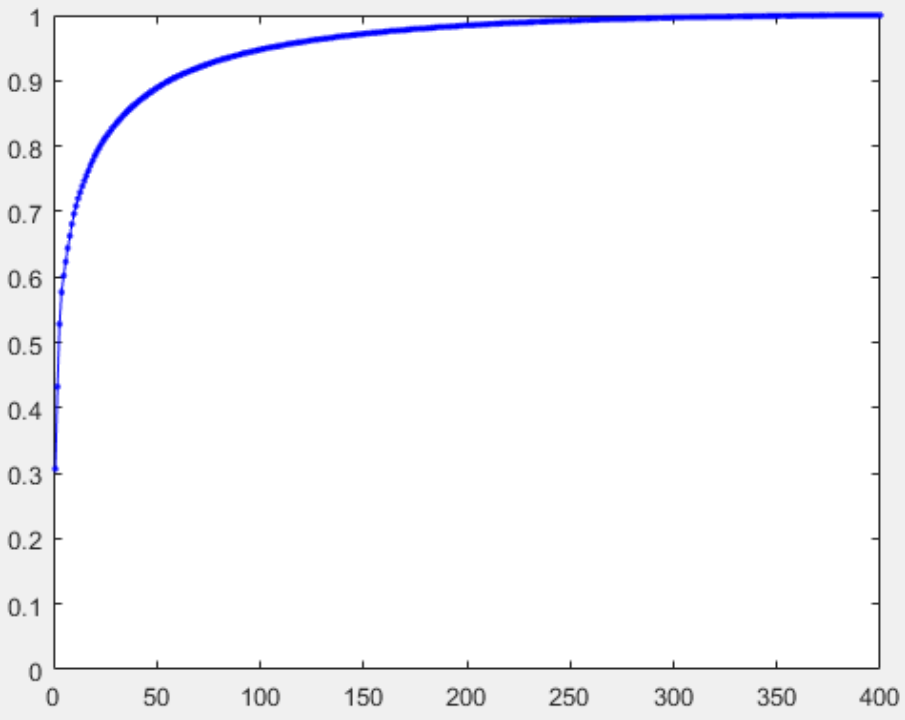
Here X^TX is only 400x400. It has the same eigenvalues as the original covariance matrix.

To obtain the eigenvectors in the original 1600-D space:

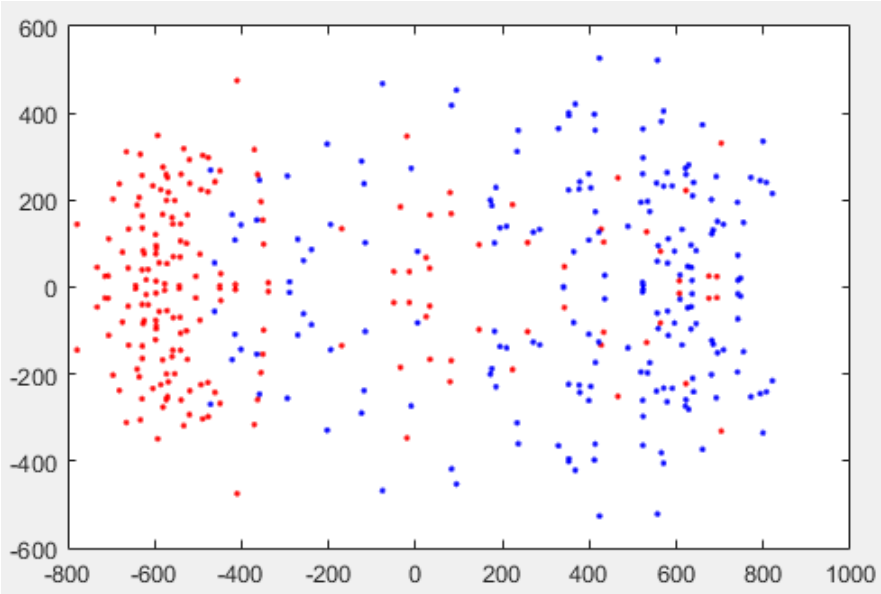
$$\boldsymbol{e}_j \leftarrow X(X^T \boldsymbol{e}_j)$$ plus normalization

Example PCA Application: Eigenfaces

Accumulated variances (normalized):

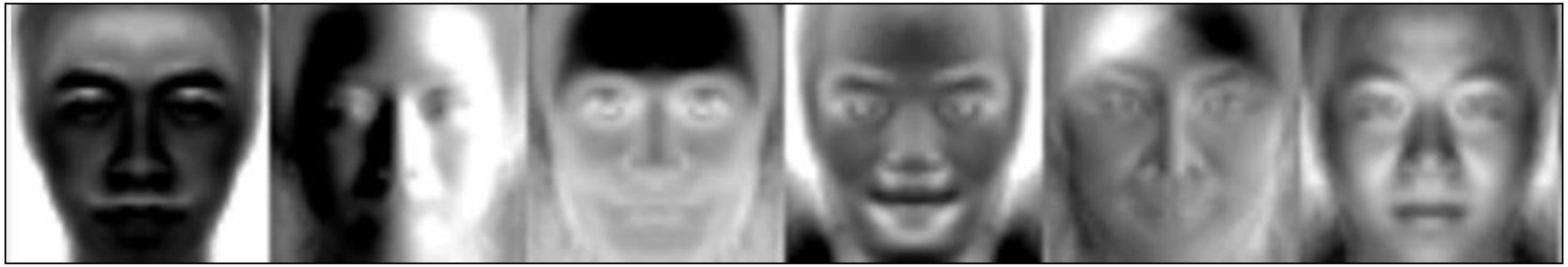


Projections in the first two PCA dimensions:



Example PCA Application: Eigenfaces

The first few eigenfaces:



Reconstructed faces along the first PCA dimension: