CS57300
PURDUE UNIVERSITY

JANUARY 29, 2019

DATA MINING

DIMENSIONALITY REDUCTION

WHAT DIMENSION CAN BE DROPPED?

- Suppose we have a data matrix ***D*** of *n* rows and *p* columns (i.e., we have *n* data points, each data point is measured on *p* dimensions)
- If we want to decrease *p*, which dimensions can we drop?
  - Constant dimensions: 1, 1, …, 1
  - Constant dimensions with some noise: 1.001, 0.998, …, 1.003
  - Dimensions that is linearly dependent on other dimensions: Z=aX+bY

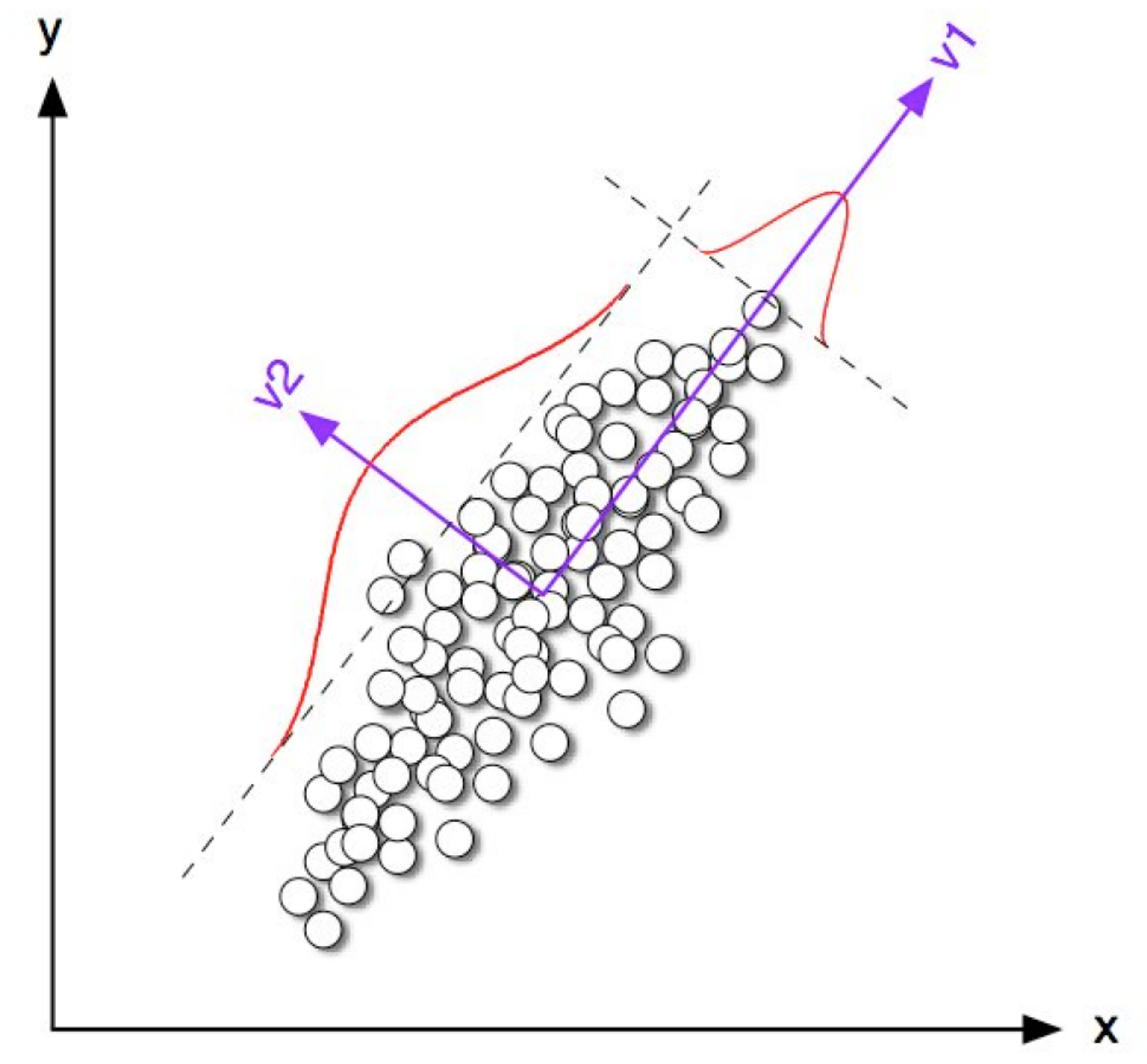
HIGH VARIANCE!

LOW COVARIANCE!

CHANGE OF BASIS

https://www.youtube.com/watch?v=FgakZw6K1QQ

- But the dimension with highest variance may not necessarily be the dimension that we have measured
- Need change of basis such that:
  - The largest amount of variability of the data can be reflected by projecting the data to some basis vector in the new basis
  - After projecting the data to the new basis (or "new dimensions"), the covariances between new dimensions are low



PRINCIPLE COMPONENT ANALYSIS (PCA)

- Input: the $n*p$ data matrix ***D***
- Preprocess **D** so that the mean of each dimension is 0, call this matrix **X**
- Goal: Find a $p*p$ orthogonal transformation matrix ***A*** to conduct basis change, i.e., ***Y***=***XA***, such that under the new basis, the covariances between the new dimensions are low
  - The $p*p$ covariance matrix $\boldsymbol{Y^TY}$ is a diagonal matrix.

PRINCIPLE COMPONENT ANALYSIS (PCA)

$$\begin{aligned} Y^T Y &= (XA)^T (XA) \\ &= A^T X^T X A \end{aligned}$$

- Notice $\Sigma = X^T X$ is the covariance matrix under the current basis, it is a symmetric square matrix!
- So, we can conduct eigendecomposition for Σ

$$Y^T Y = A^T (Q \Lambda Q^T) A = (A^T Q) \Lambda (A^T Q)^T$$

PRINCIPLE COMPONENT ANALYSIS (PCA)

$$Y^TY = A^T(Q\Lambda Q^T)A = (A^TQ)\Lambda(A^TQ)^T$$

- Let $A^TQ = I$, we get $A = (Q^{-1})^T = (Q^T)^T = Q$
  - By doing so, $Y^TY = I\Lambda I = \Lambda$
- In other words, the transformation matrix A is Q, where each column is the eigenvector of the covariance matrix $\Sigma = X^TX$!
- The column vectors of A (or Q) are thus called the **principle component vectors**!

NOT DONE YET…

- So far we have only changed basis, i.e., we project the data to another *p* dimensions
  - The covariance on these *p* new dimensions is 0!
  - But we don't know which dimensions we can drop, i.e., which dimensions have smaller variance yet…
- Recall that $Y^TY = \Lambda$, this is the covariance matrix after projecting data to the *p* new dimensions!
  - λ_i is the variance of new dimension *i* (i.e., the *i*-th column of A), $\sum_{j=1}^{p} \sigma_j^2 = \sum_{j=1}^{p} \lambda_j$

APPLYING PCA

- Order principal components according to the corresponding eigenvalues. New data vectors are formed by projecting the data onto the first few principal components (i.e., top *m* eigenvectors)

$$\mathbf{x} = [x_1, x_2, \ldots, x_p] \quad \text{(original instance)}$$

$$\mathbf{A} = [\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_p] \quad \text{(principal components)}$$

$$x'_1 = \mathbf{a}_1\mathbf{x} = \sum_{j=1}^{p} a_{1j}x_j$$

$$\ldots$$

$$x'_m = \mathbf{a}_m\mathbf{x} = \sum_{j=1}^{p} a_{mj}x_j \quad \boxed{for\ m < p}$$

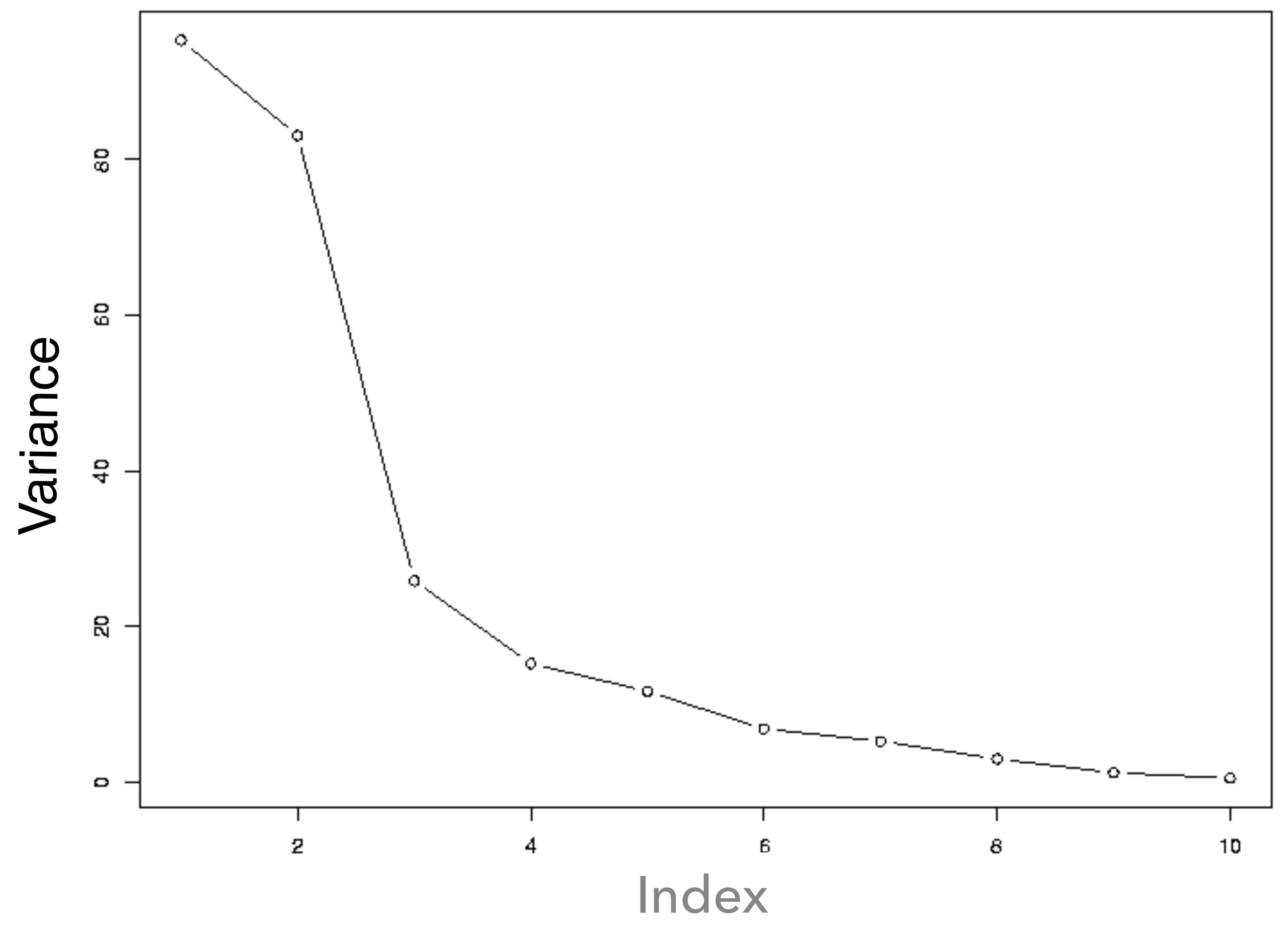
If ***m=p*** then data is transformed

If ***m<p*** then transformation is lossy and dimensionality is reduced

$$\mathbf{x}' = [x'_1, x'_2, \ldots, x'_m] \quad \text{(transformed instance)}$$

APPLYING PCA (CONT')

- Goal: Find a new (smaller) set of dimensions that captures most of the variability of the data
- Use **scree plot** to choose number of dimensions
  - Choose $m < p$ so projected data captures much of the variance of original data



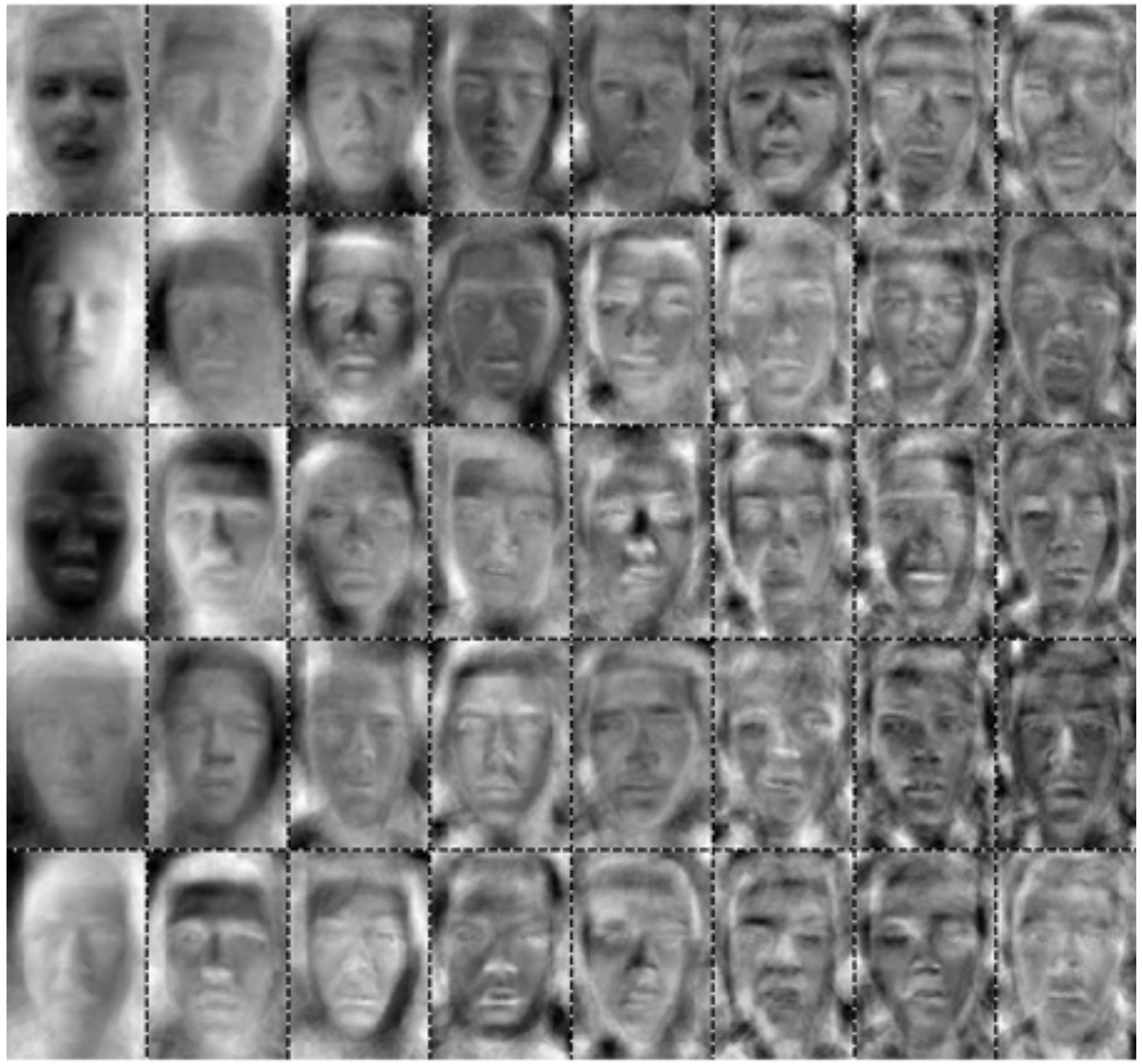
EXAMPLE: EIGENFACES

PCA applied to images of human faces.

Reduce dimensionality to set of basis images.

All other images are linear combo of these “eigenpictures”.

Used for facial recognition.



First 40 PCA dimensions

DIMENSIONALITY REDUCTION METHODS

- Principal component analysis (PCA)
  - Linear transformation, minimize unexplained variance
- Factor analysis
  - Linear combination of small number of **latent** variables
- Multidimensional scaling (MDS)
  - Project into low-dimensional subspace while preserving distance between points (can be non-linear)

PREDICTIVE MODELING

DATA MINING COMPONENTS

- Task specification: **Prediction**
- Knowledge representation
- Learning technique
- Prediction and/or interpretation

PREDICTIVE MODELING

- Data representation:
  - Paired attribute vectors and class labels $<y(i), \boldsymbol{x}(i)>$ or $n \times p$ tabular data with class label (y) and $p\text{-}1$ attributes ($\boldsymbol{x}$)

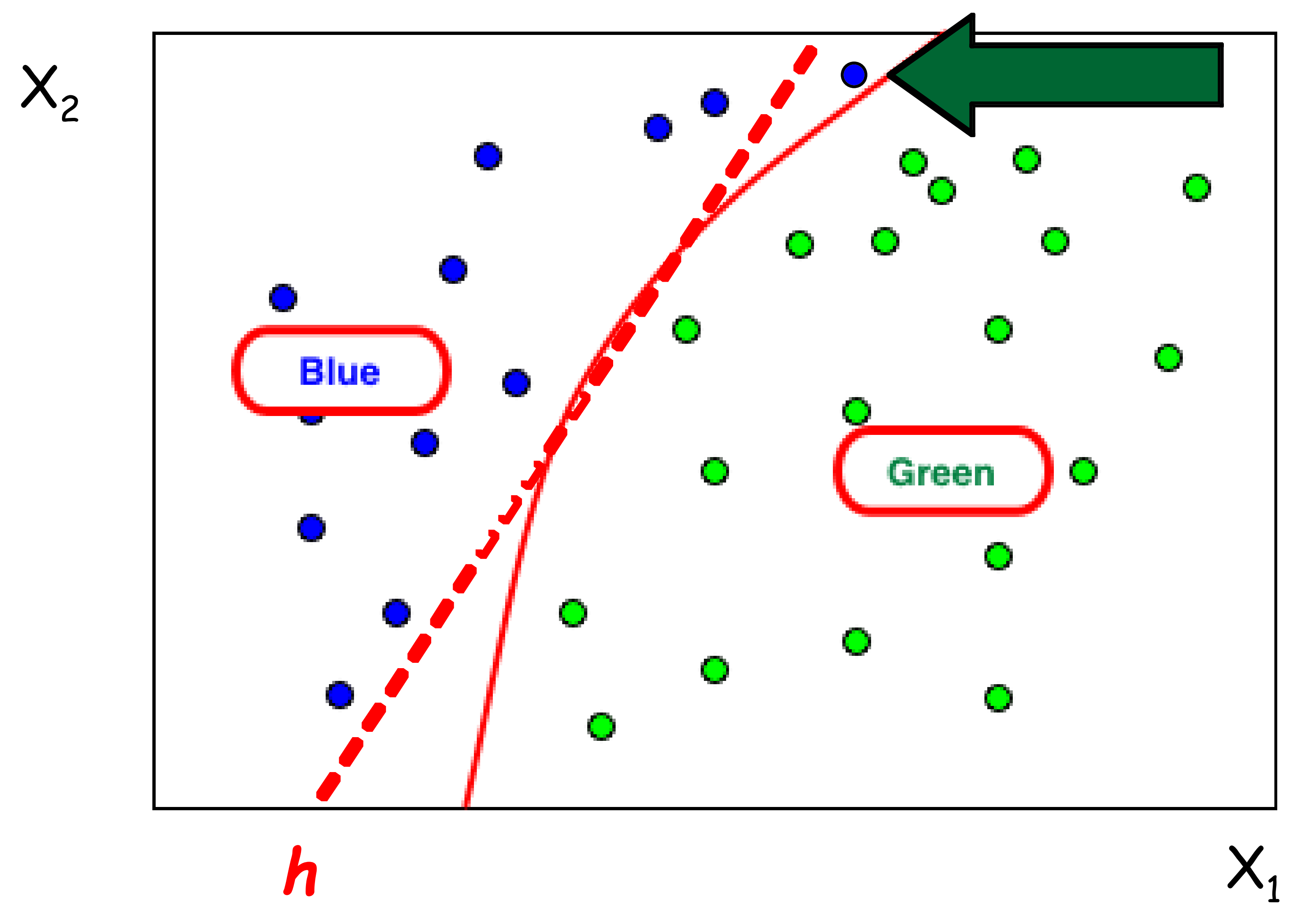
- Task: Estimate a predictive function $f(\boldsymbol{x};\theta)=y$
  - Assume that there is a function $y=f(\boldsymbol{x})$ that **maps** data instances ($\boldsymbol{x}$) to class labels (y)
  - Construct a model that approximates the mapping
    - Classification: if y is categorical ← **Focus of this course**
    - Regression: if y is real-valued

CLASSIFICATION OUTPUT

- Different classification tasks can require different kinds of output
- **Class labels** – Each instance is assigned a single label
  - *Model only need to decide on crisp class boundaries*
- **Ranking** – Instances are ranked according to their likelihood of belonging to a particular class
  - *Model implicitly explores many potential class boundaries*
- **Probabilities** – Instances are assigned class probabilities $p(y|\mathbf{x})$
  - *Allows for more refined reasoning about sets of instances*
- Each requires progressively more accurate models (e.g., a poor probability estimator can still produce an accurate ranking)

DISCRIMINATIVE CLASSIFICATION

- Output: Class Labels
  - Direct mapping from inputs ***x*** to class label *y*
  - No attempt to model probability distributions
- Model the decision boundary directly
- May seek a discriminant function *f(**x**;θ)* that maximizes measure of separation between classes
- Examples:
  - Perceptrons, decision trees, nearest neighbor classifiers, support vector machines



PROBABILISTIC CLASSIFICATION

- Output: Probabilities
  - Maps from inputs $\boldsymbol{x}$ to class label y indirectly through posterior class distribution $p(y|\mathbf{x})$
- Model the underlying probability distributions
  - Posterior class probabilities: $p(y|\mathbf{x})$
  - Class-conditional and class prior: $p(\mathbf{x}|y)$ and $p(y)$
- Examples:
  - Naive Bayes classifier, logistic regression

KNOWLEDGE REPRESENTATION

KNOWLEDGE REPRESENTATION

- Underlying structure of the model or patterns that we seek from the data

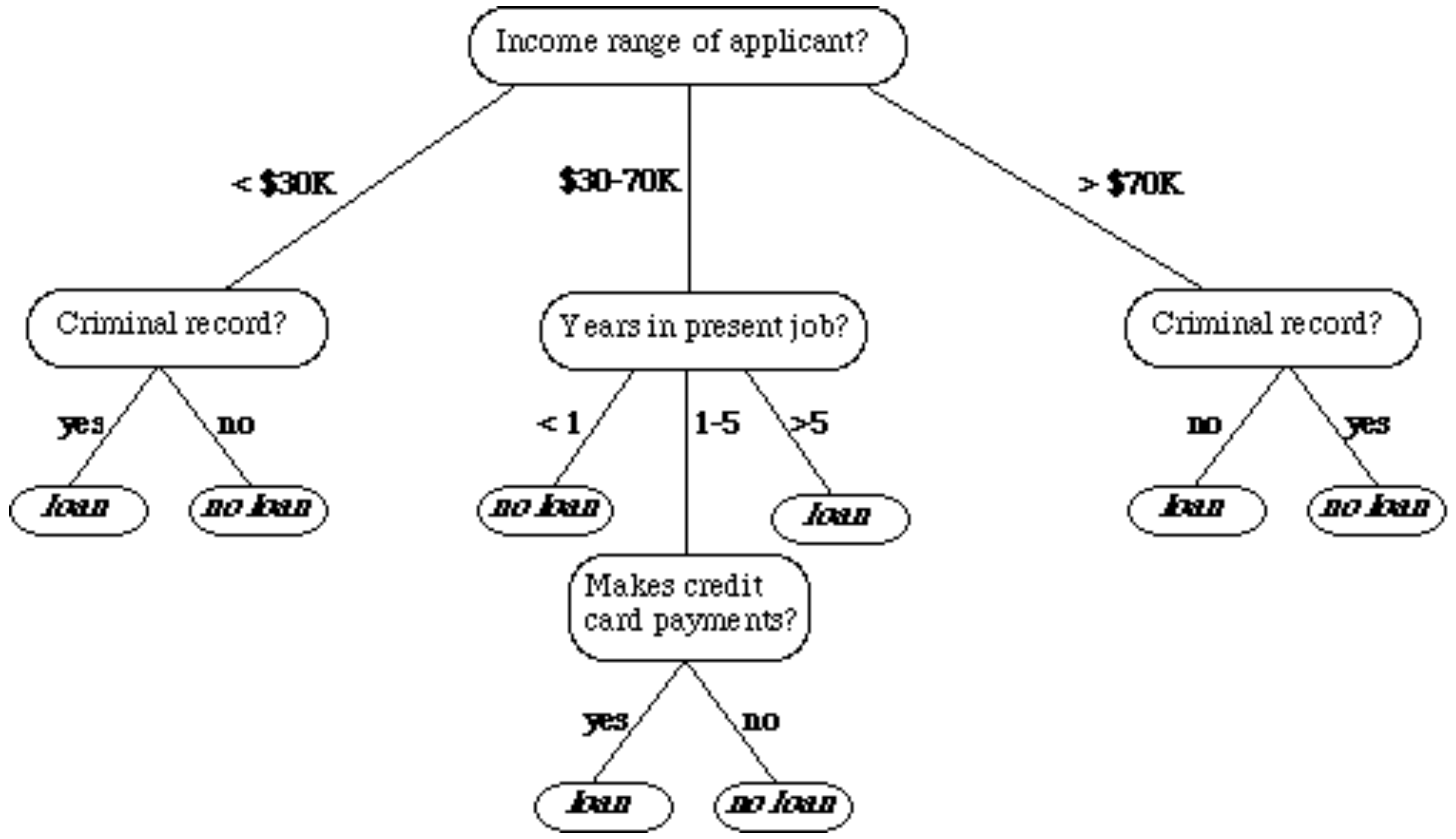
- Model: high-level global description of dataset
  - Choice of model family determines **space** of parameters and structure
  - Estimate model parameters and possibly model structure from data

PERCEPTRON

$$f(x) = \begin{cases} 1 & \sum w_j x_j > 0 \\ 0 & \sum w_j x_j \leq 0 \end{cases}$$

Model space:
weights *w*, for each of *j* attributes

DECISION TREE



Model space:

all possible decision trees

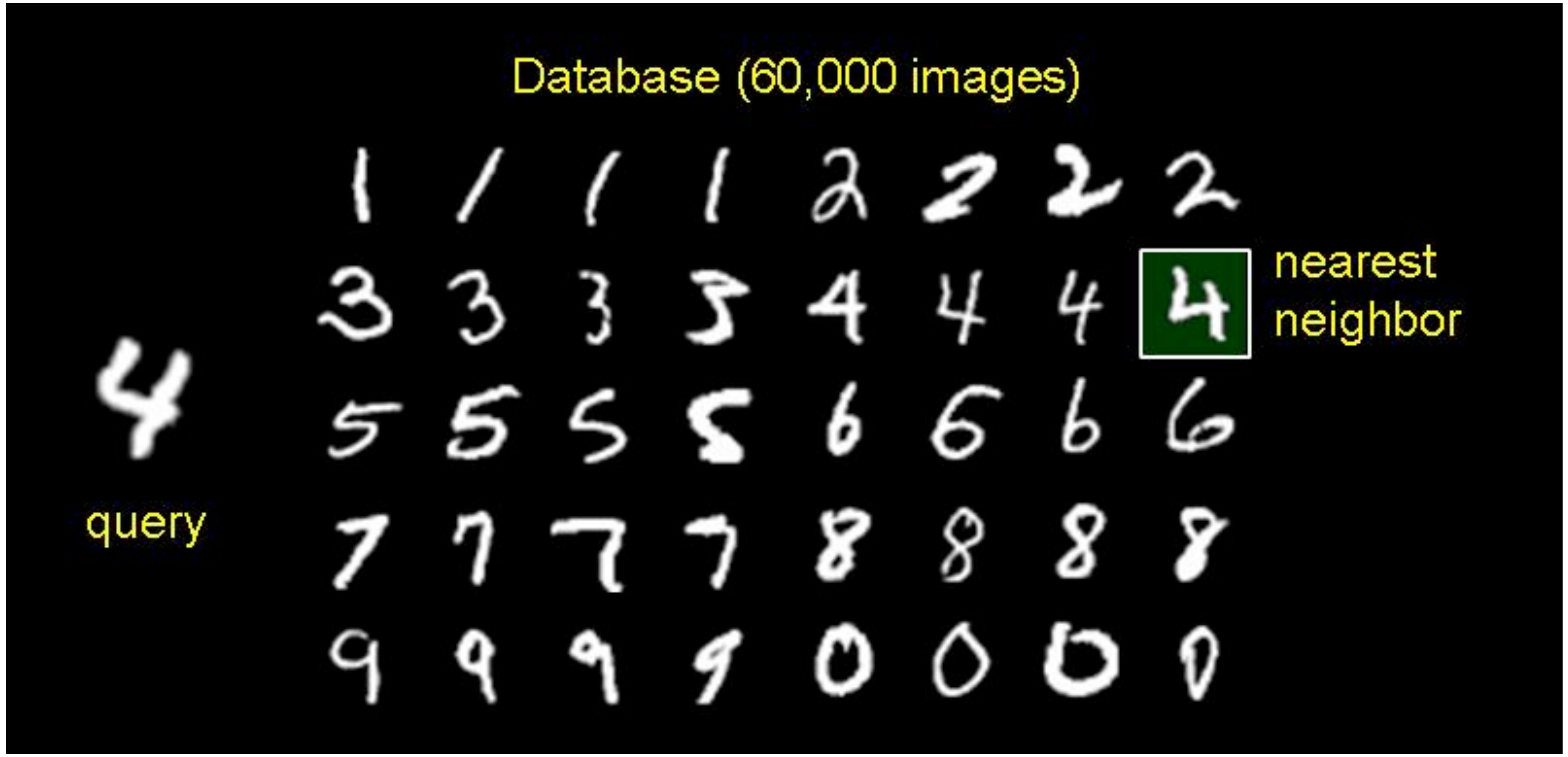
MODEL SPACE

- How large is the space?
- Simplifying assumptions
  - Binary tree
  - Fixed depth
  - 10 binary attributes
- Can we search exhaustively?

Tree depth	Number of trees
1	10
2	8×10^{2}
3	3×10^{6}
4	2×10^{13}
5	5×10^{25}

NEAREST NEIGHBOR

Rule: find *k* closest (training) points to the test instance and assign the most frequently occurring class



Model space:

Choice of *k*, definition of distance, etc.

NAIVE BAYES CLASSIFIER

$$p(y|\mathbf{x}) = \frac{p(\mathbf{x}|y)p(y)}{p(\mathbf{x})} = \frac{\prod_i p(x_i|y)\, p(y)}{\sum_j p(\mathbf{x}|y_j)p(y_j)}$$

Model space:

parameters in conditional distributions $p(x_i|y)$

parameters in prior distribution $p(y)$

PARAMETRIC VS. NON-PARAMETRIC MODELS

- Parametric
  - **Functional form of the model is defined by a finite set of parameters**
  - Particular functional form is assumed
  - Examples: Naive Bayes, perceptron
- Non-parametric
  - **Functional form of the model is determined from data**
  - Few assumptions are made about the functional form
  - Examples: decision tree, nearest neighbor

PREDICTIVE MODELING: LEARNING

LEARNING PREDICTIVE MODELS

- Select a **knowledge representation** (a “model”)
  - Defines a **space** of possible models $M=\{M_1, M_2, \ldots, M_k\}$
- Define **scoring functions** to “score" different models
- Use **search** to identify “best” model(s)
  - Search the space of models (i.e., with alternative structures and/or parameters)
  - Evaluate possible models with **scoring function** to determine the model which best fits the data
  - Score function can be used to search over ***parameters*** and/or ***model structure***

PREDICTIVE SCORING FUNCTIONS

- Assess the quality of predictions for a set of instances
  - Measures **difference** between the prediction *M* makes for an instance *i* and the true class label value of *i*

$$S(M) = \underbrace{\sum_{i=1}^{N}}_{\text{Sum over examples}} \underbrace{d}_{\text{Distance between predicted and true}}\big[\underbrace{f(x(i);M)}_{\text{Predicted class label for item } i}, \underbrace{y(i)}_{\text{True class label for item } i}\big]$$

PREDICTIVE SCORING FUNCTIONS

- Common score functions:
  - Zero-one loss

$$S_{0/1}(M) = \frac{1}{N} \sum_{i=1}^{N} I\big[f(x(i); M), y(i)\big]$$

$$\text{where } I(a,b) = \begin{cases} 1 & a \neq b \\ 0 & \text{otherwise} \end{cases}$$

  - Squared loss

$$S_{sq}(M) = \frac{1}{N} \sum_{i=1}^{N} \big[f(x(i); M) - y(i)\big]^2$$

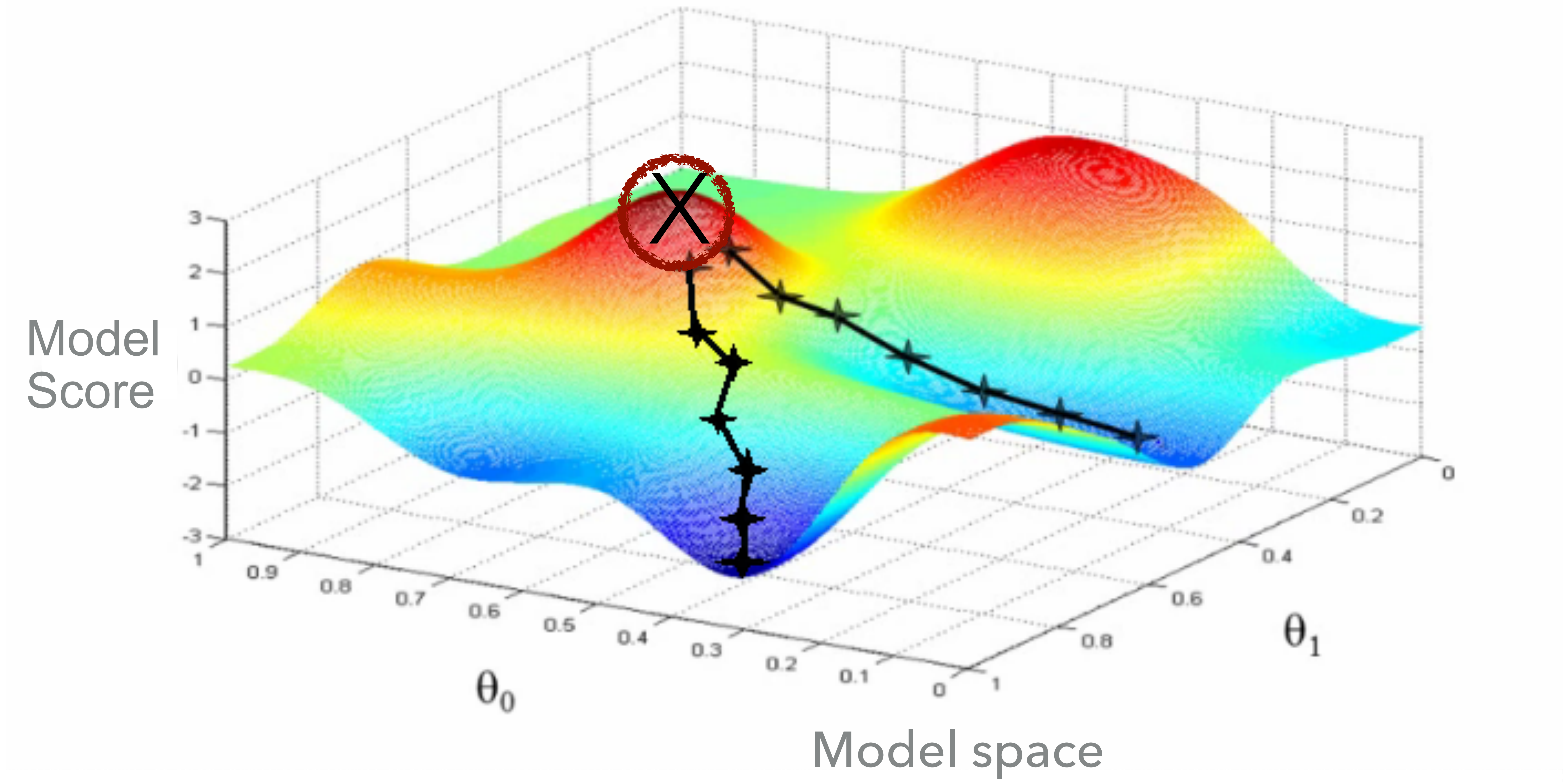
- Do we minimize or maximize these functions?

SEARCHING OVER MODELS

- Consider a **space** of possible models $M=\{M_1, M_2, \ldots, M_k\}$ with parameters θ
- Search could be over model structures or parameters, e.g.:
    - **Parameters**: In a perceptron, find the weights (**w**) that minimize 0-1 loss
    - **Model structure**: In decision trees, find the tree structure that maximizes accuracy on the training data

WHAT SPACE ARE WE SEARCHING?

$$\text{Learned model} \approx (\theta_0 = 0.8, \theta_1 = 0.4)$$



OPTIMIZATION OVER SCORE FUNCTIONS

- **Smooth** functions:
  - If a function is *smooth*, it is differentiable and the derivatives are continuous, then we can use gradient-based optimization
    - If function is *convex*, we can solve the minimization problem in closed form: $\nabla S(\theta)$ using **convex optimization**
    - If function is smooth but non-linear, we can use iterative search over the surface of S to find a local minimum (e.g., hill-climbing)
- **Non-smooth** functions:
  - If the function is *discrete*, then traditional optimization methods that rely on smoothness are not applicable. Instead we need to use **combinatorial optimization**