DIGITAL IMAGE PROCESSING

Lecture 10

Machine Learning: Supervised Learning

Tammy Riklin Raviv

Electrical and Computer Engineering

Ben-Gurion University of the Negev

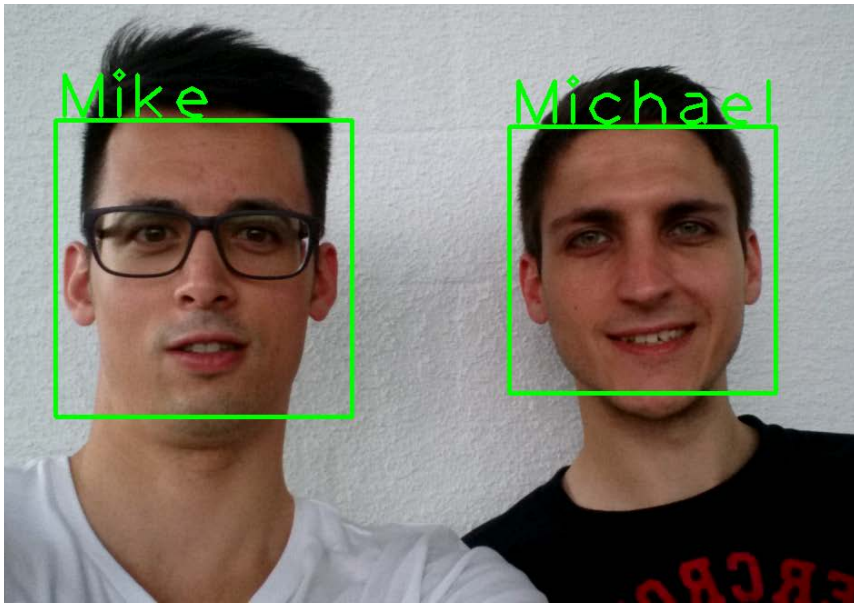
Machine Learning

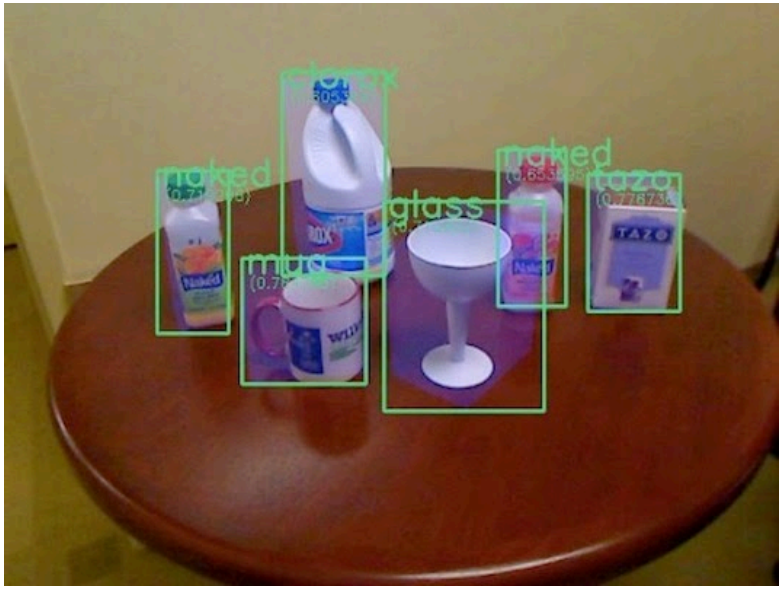
- Learn from and make predictions on data.
- Arguably the greatest export from computing to other scientific fields.
- Statisticians might disagree with Computer Scientists on the true origins…

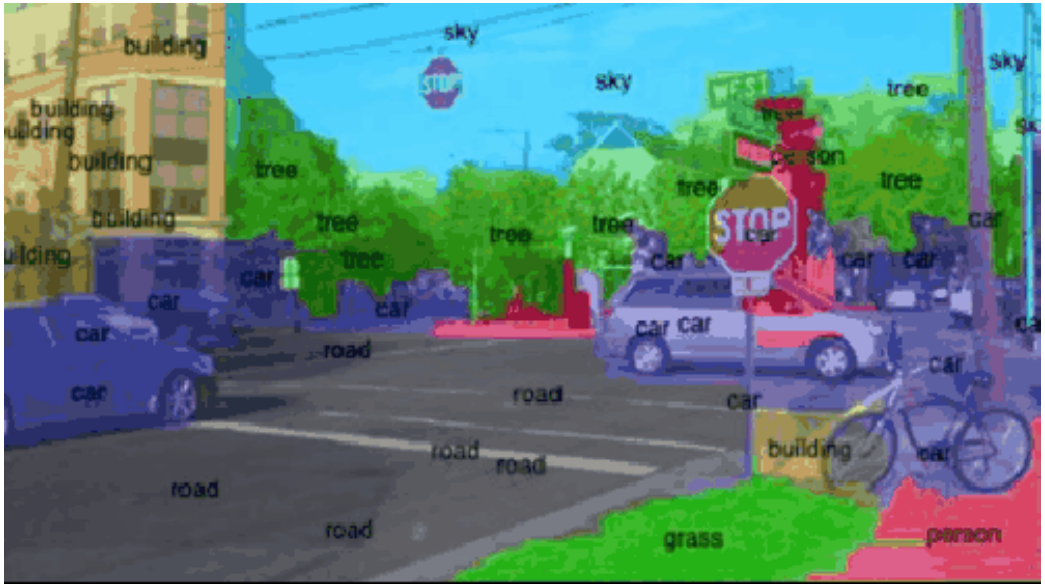
Most slides are taken (or adapted) from Michael Black's course in Brown

ML for Computer Vision

- Face Recognition
- Object Classification
- Scene Segmentation



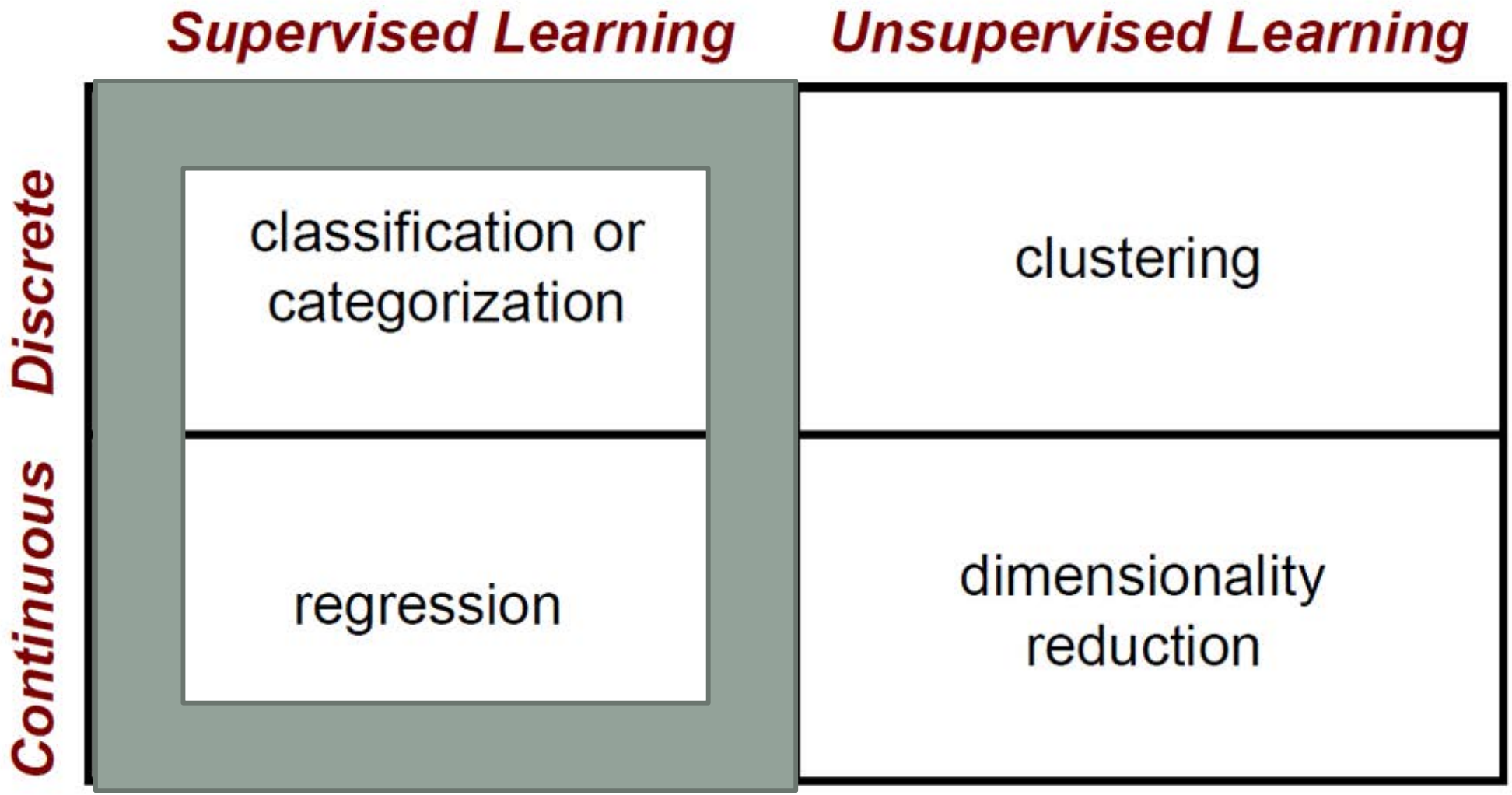




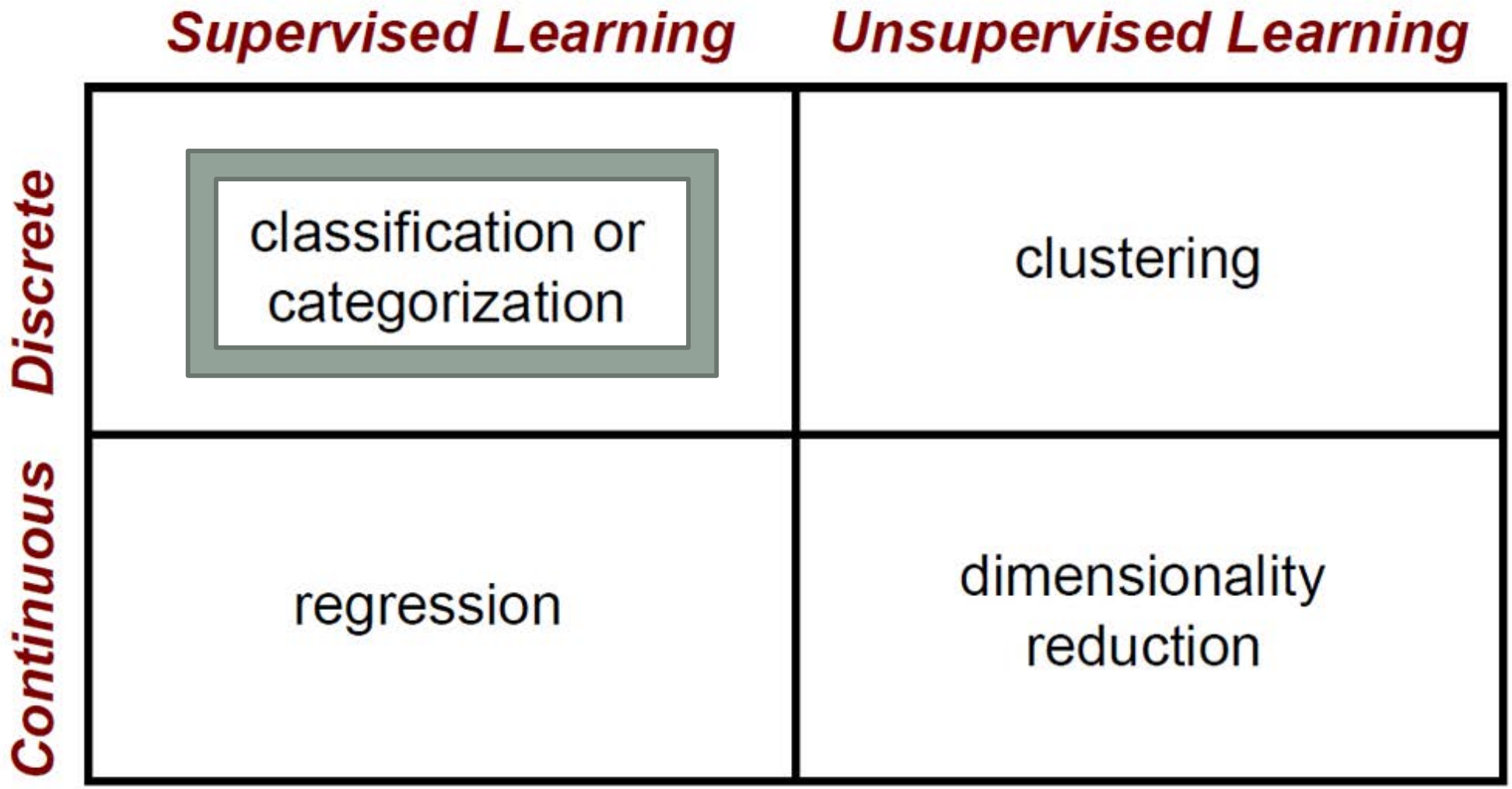
Machine Learning Problems

	Supervised Learning	*Unsupervised Learning*
Discrete	classification or categorization	clustering
Continuous	regression	dimensionality reduction

Machine Learning Problems



Machine Learning Problems



The machine learning framework

- Apply a prediction function to a feature representation of the image to get the desired output:

f() = “apple”

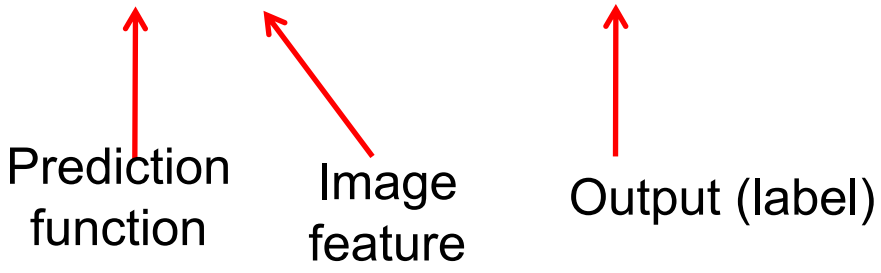
f() = “tomato”

f() = “cow”

Slide credit: L. Lazebnik

The machine learning framework

$$f(\mathbf{x}) = y$$



Training: Given a *training set* of labeled examples:

$$\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$$

Estimate the prediction function f by minimizing the prediction error on the training set.

Testing: Apply f to a unseen *test example* $\mathbf{x}$ and output the predicted value $y = f(\mathbf{x})$ to *classify* $\mathbf{x}$.

Slide credit: L. Lazebnik

Image Categorization

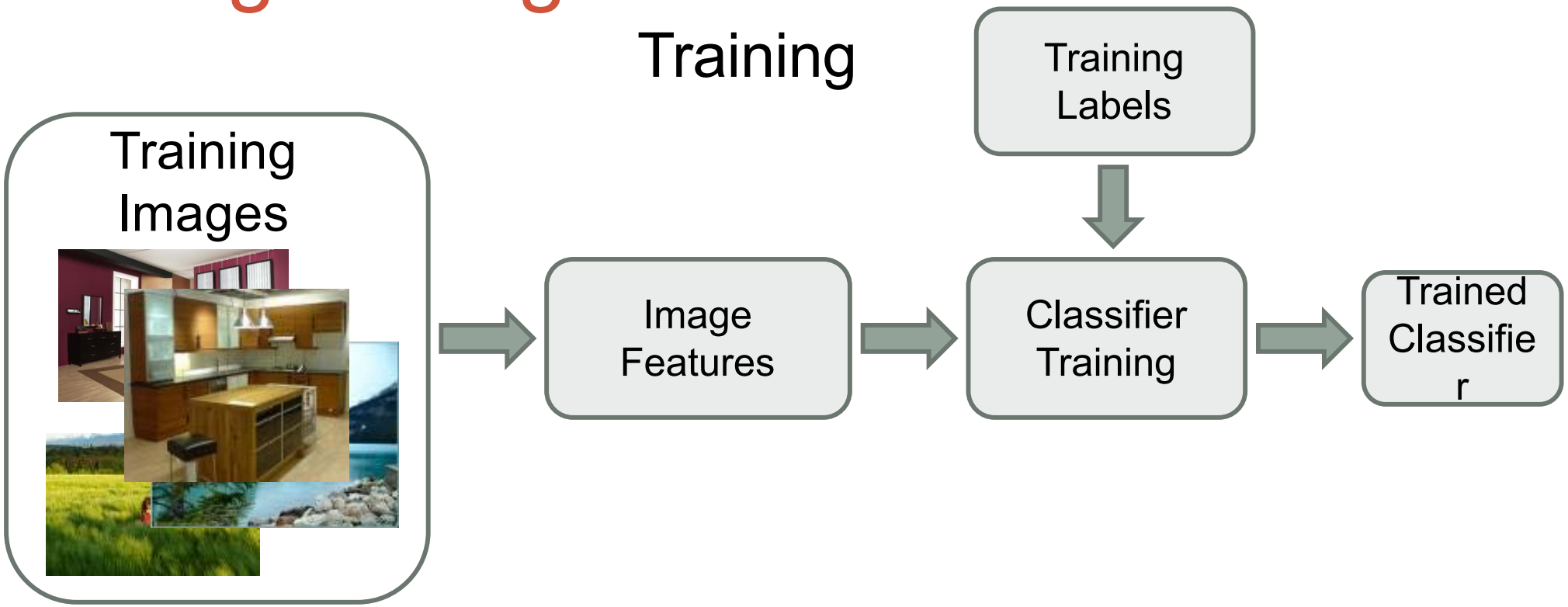


Image Categorization

Training

Training Images

Training Labels

Image Features

Classifier Training

Trained Classifier

Testing

Test Image

Image Features

Trained Classifier

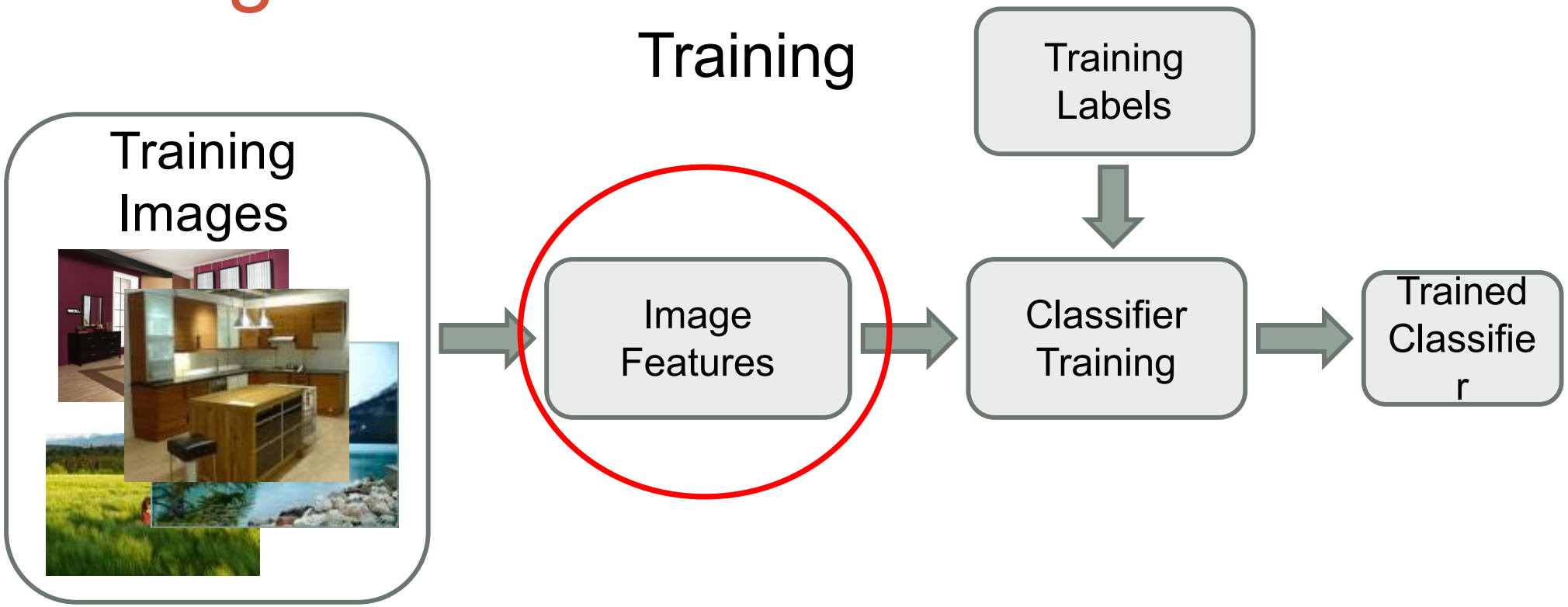
Prediction

Outdoor

Example: Scene Categorization

- Is this a kitchen?

Image features



General Principles of Representation

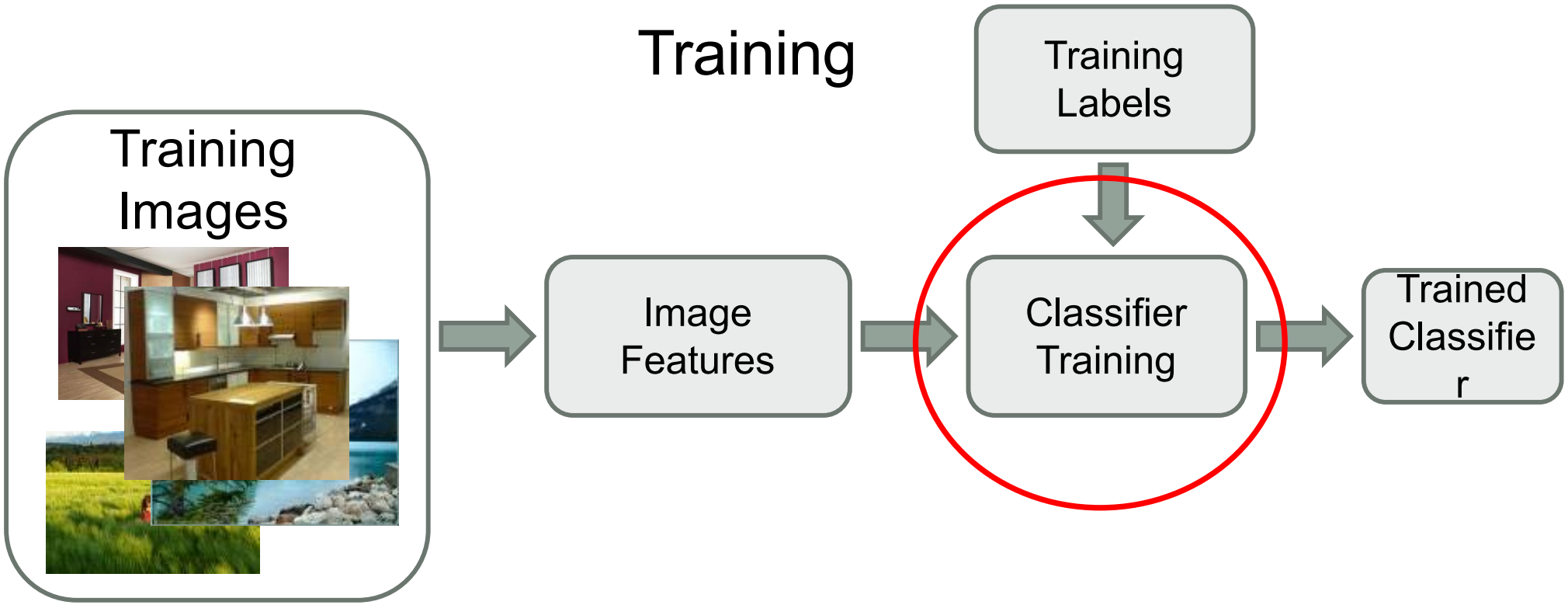
- Coverage
  - Ensure that all relevant info is captured
- Concision
  - Minimize number of features without sacrificing coverage
- Directness
  - Ideal features are independently useful for prediction

Image representations

- Templates
  - Intensity, gradients, etc.

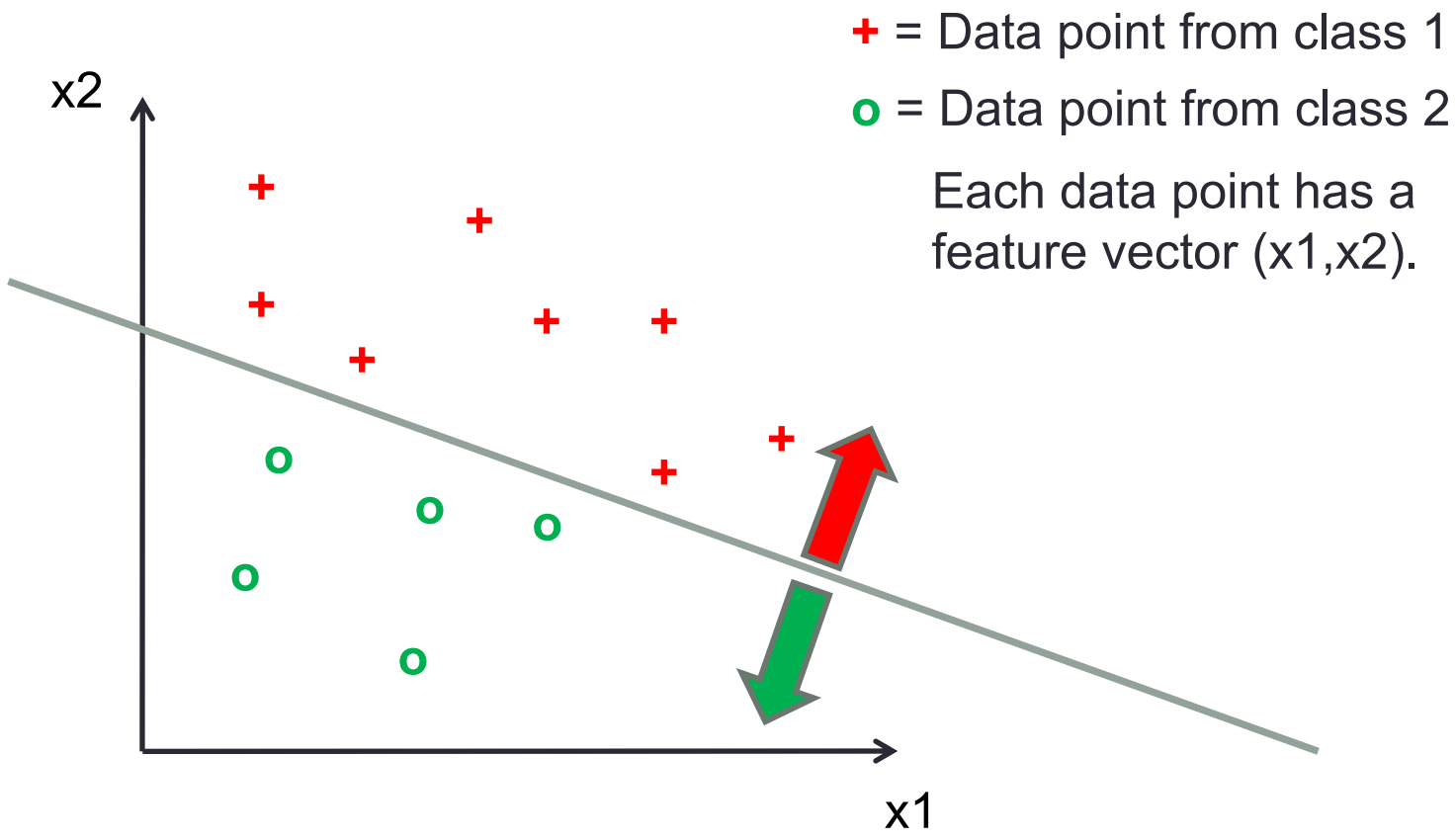
- Histograms
  - Color, texture, SIFT descriptors, etc.

Classifiers



Learning a classifier

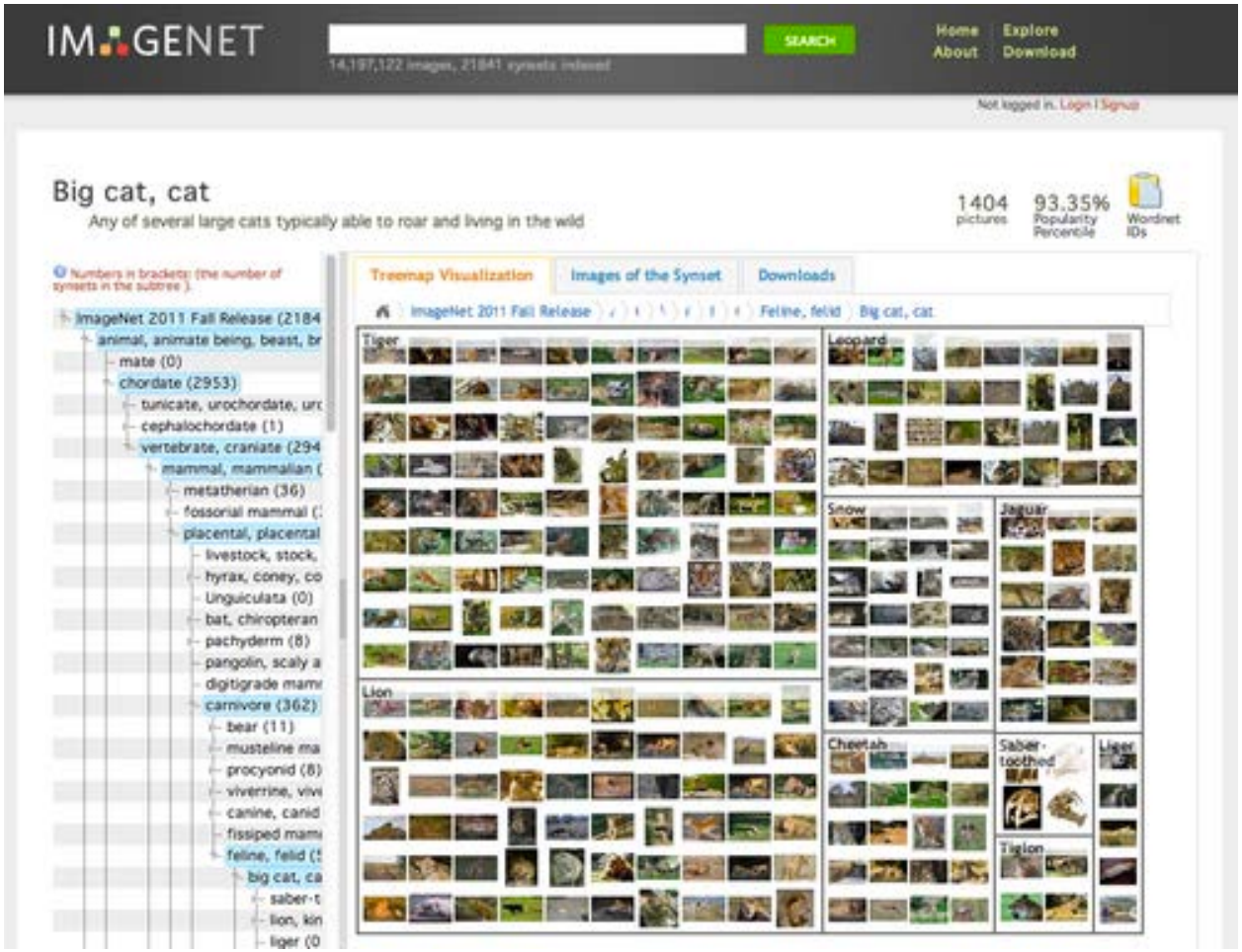
Given a set of features with corresponding labels, learn a function to predict the labels from the features.



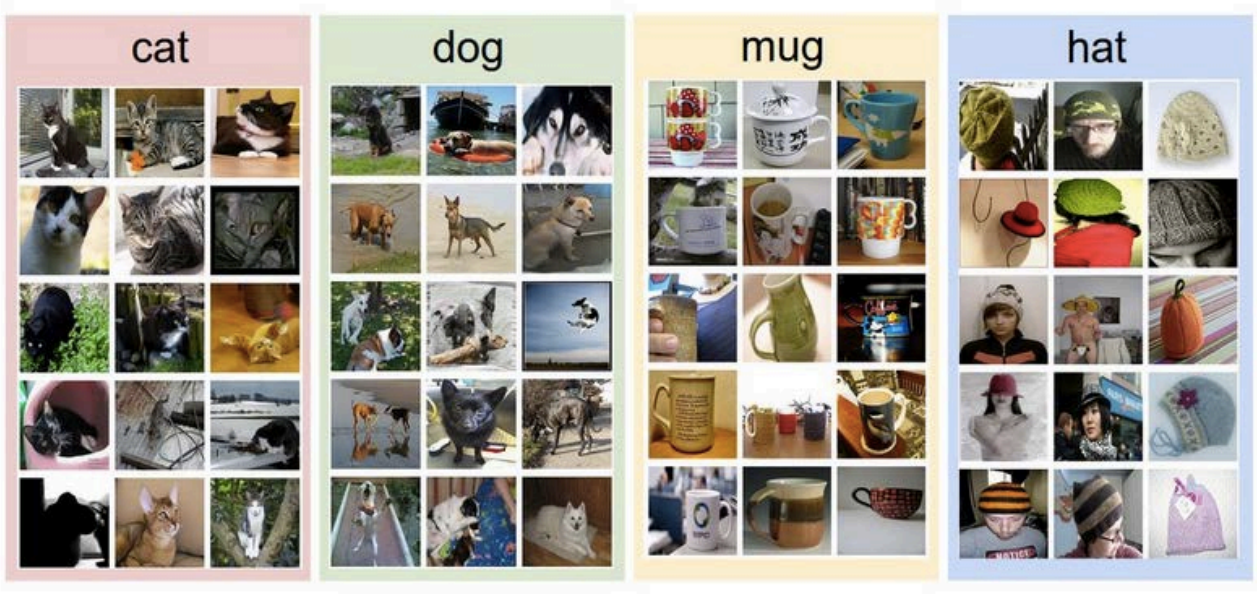
ImageNet

- Images for each category of WordNet
- 1000 classes
- 1.2mil images
- 100k test

- Top 5 error

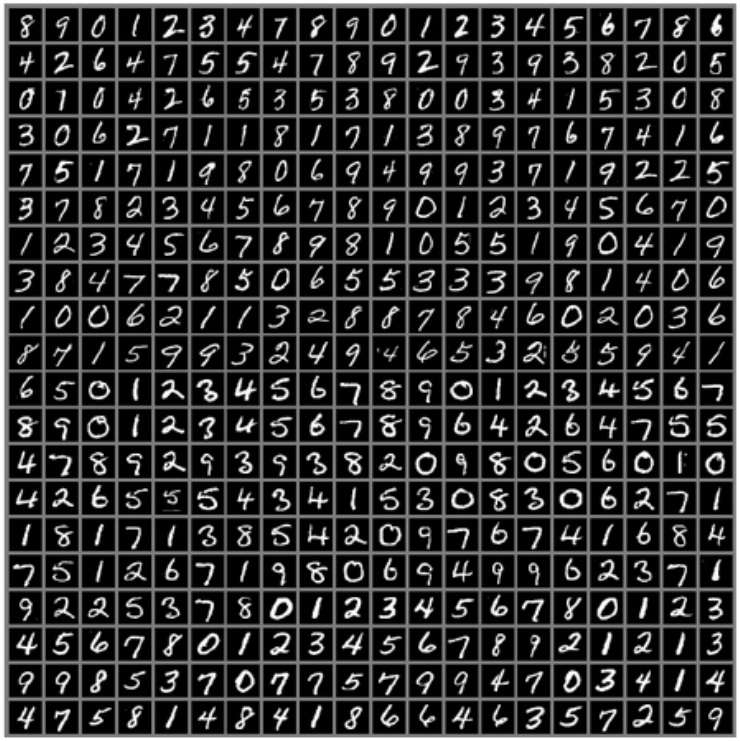


Labeled database



An example training set for four visual categories. In practice we may have thousands
of categories and hundreds of thousands of images for each category.

MNIST database



Dataset split

Training Images

- Train classifier

Validation Images

- Measure error
- Tune model hyperparameters

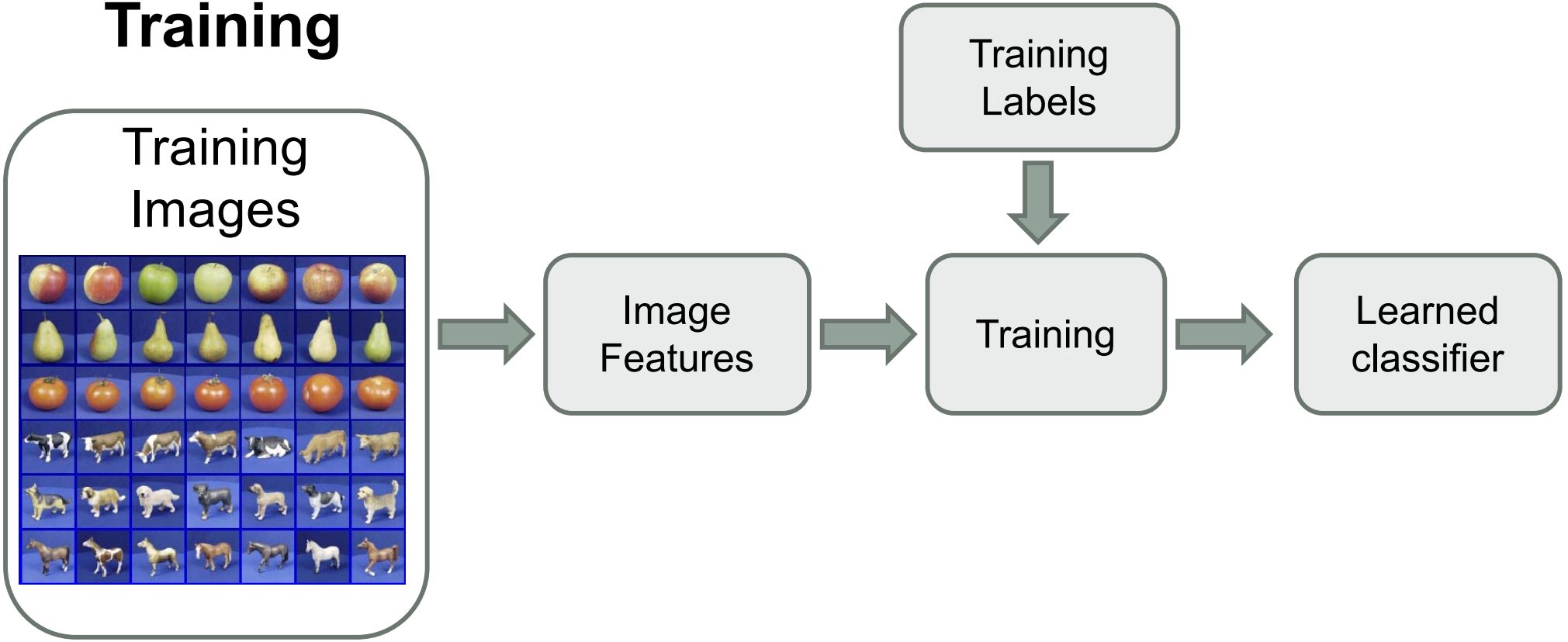
Testing Images

- Secret labels
- Measure error

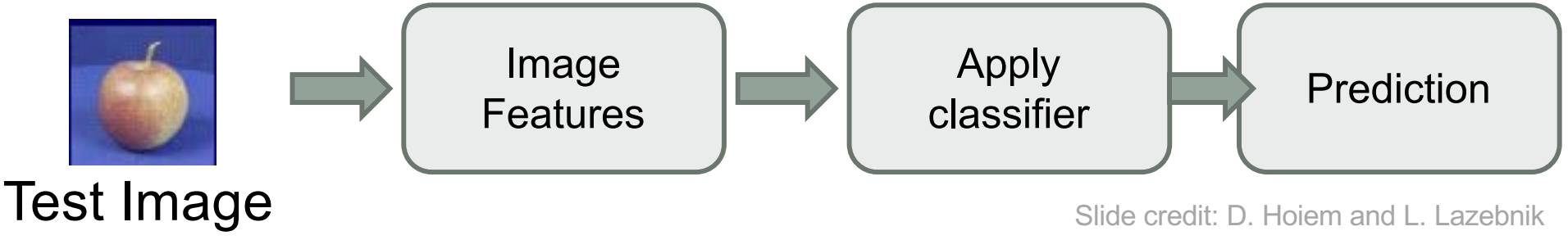
Random train/validate splits = cross validation

Steps

Training



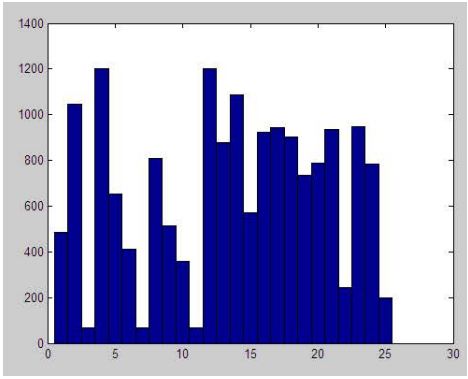
Testing

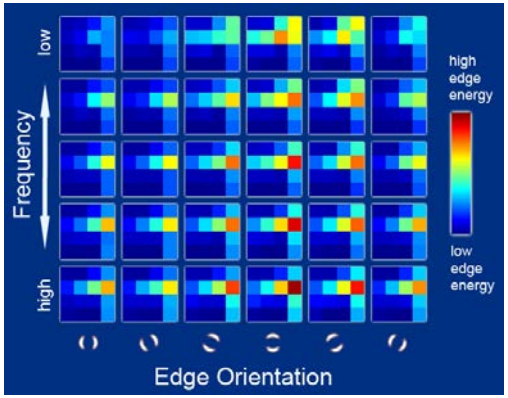


Slide credit: D. Hoiem and L. Lazebnik

Features

- Raw pixels
- Histograms
- other descriptors
- …





Slide credit: L. Lazebnik

One way to think about it…

- Training labels dictate that two examples are the same or different, in some sense.
- Features and distance measures define visual similarity.
- Classifiers try to learn weights or parameters for features and distance measures so that visual similarity predicts label similarity.

Many classifiers to choose from…

- SVM
- Neural networks
- Naïve Bayes
- Bayesian network
- Logistic regression
- Randomized Forests
- Boosted Decision Trees
- K-nearest neighbor
- Restricted Boltzmann Machines
- Deep Convolutional Network
- …

Which is the best?

Claim:

The decision to use machine learning is more important than the choice of a particular learning method.

*Deep learning seems to be an exception to this, currently, because it learns the feature representation.

Claim:

It is more important to have more or better labeled data than to use a different supervised learning technique.

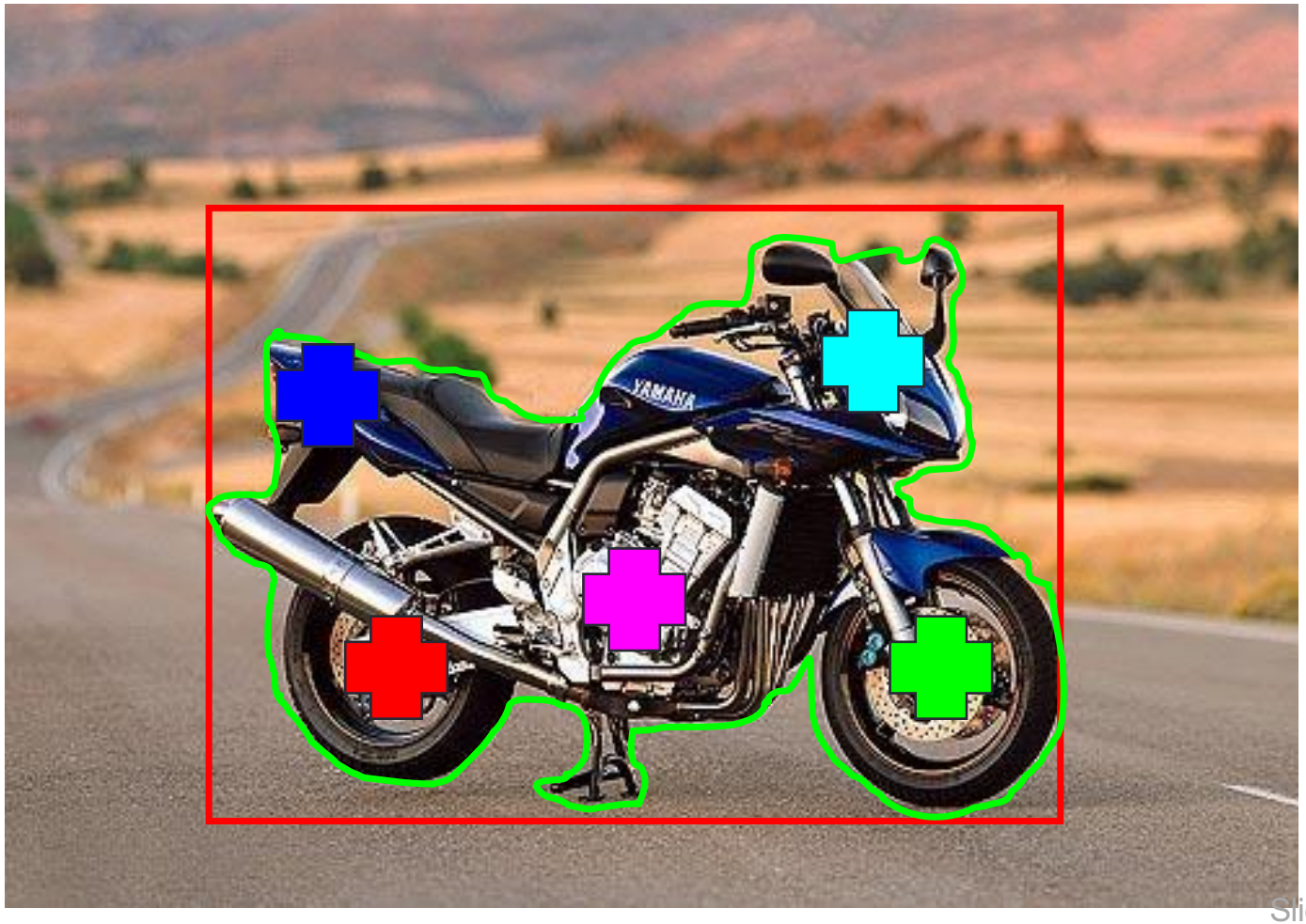
*Again, deep learning may be an exception here for the same reason, but deep learning _needs_ a lot of labeled data in the first place.

“The Unreasonable Effectiveness of Data” - Norvig

Recognition task and supervision

- Images in the training set must be annotated with the "correct answer" that the model is expected to produce

Contains a motorbike



Slide credit: L. Lazebnik

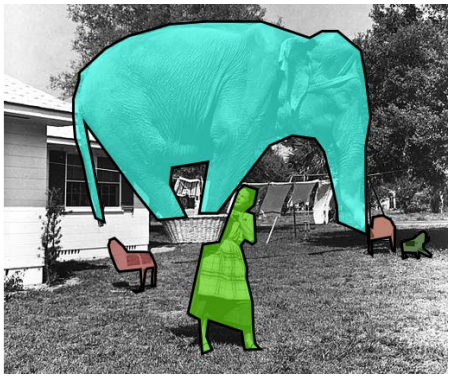
Spectrum of supervision

Less

More

E.G., ImageNet

E.G., MS Coco



Unsupervised

“Weakly” supervised

Fully supervised

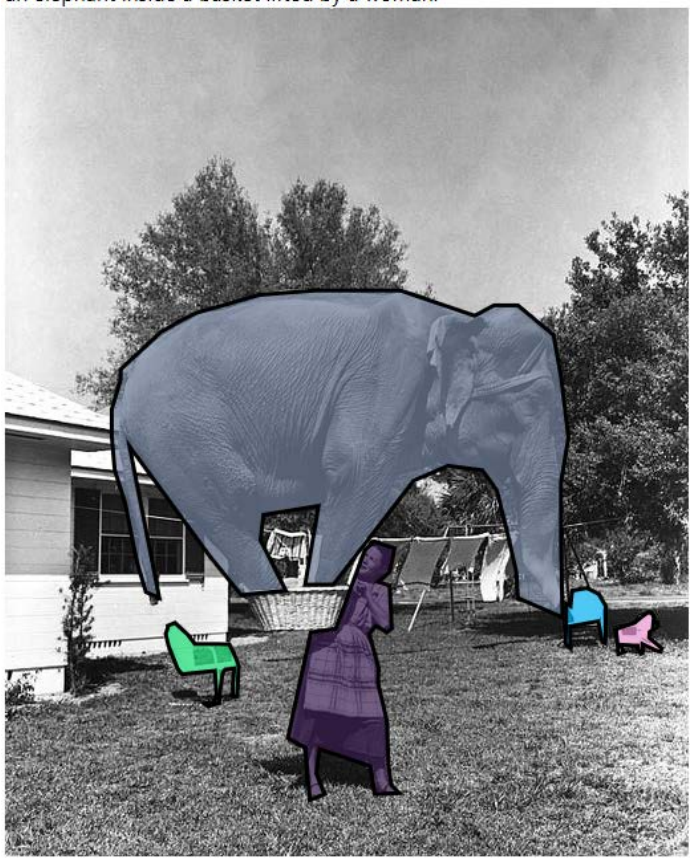
Fuzzy; definition depends on task

Good training data?

Good training data?

URL

an elephant standing on top of a basket being held by a woman.
a woman standing holding a basket with an elephant in it.
a lady holding an elephant in a small basket.
a lady holds an elephant in a basket.
an elephant inside a basket lifted by a woman.



http://mscoco.org/explore/?id=134918

Google guesses from the 1st caption

Generalization

Training set (labels known)

Test set (labels unknown)

- How well does a learned model generalize from the data it was trained on to a new test set?

Slide credit: L. Lazebnik

Generalization Error

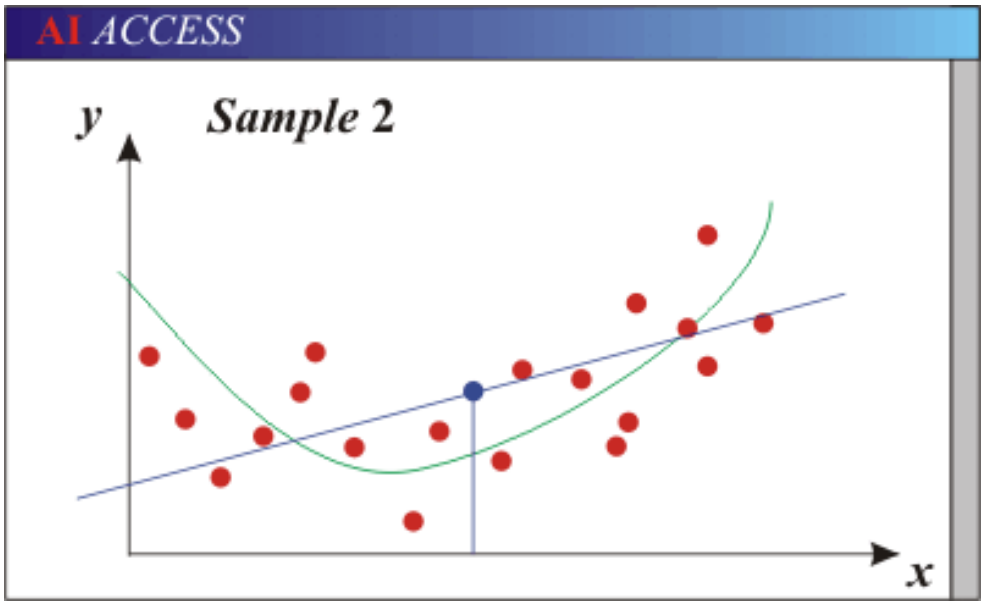
- **Bias:** how much the average model over all training sets differs from the true model.
  - Error due to inaccurate assumptions/simplifications made by the model.
- **Variance:** how much models estimated from different training sets differ from each other.

- **Underfitting:** model is too "simple" to represent all the relevant class characteristics
  - High bias (few degrees of freedom) 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 (many degrees of freedom) and high variance
  - Low training error and high test error

Slide credit: L. Lazebnik

Generalization Error Effects

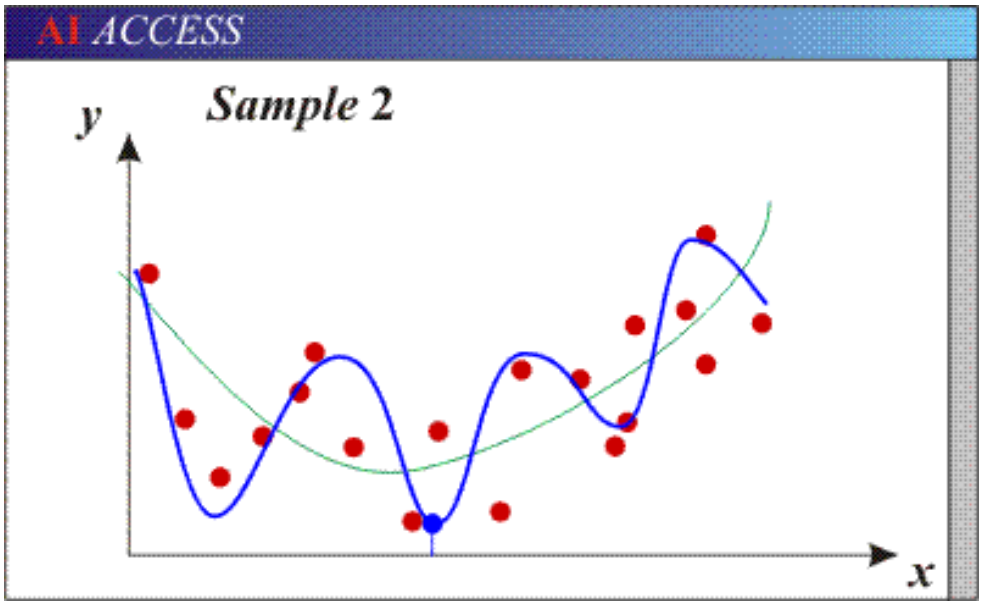
- **Underfitting:** model is too “simple” to represent all the relevant class characteristics
  - High bias (few degrees of freedom) and low variance
  - High training error and high test error



Slide credit: L. Lazebnik

Generalization Error Effects

- **Overfitting:** model is too “complex” and fits irrelevant characteristics (noise) in the data
  - Low bias (many degrees of freedom) and high variance
  - Low training error and high test error



Slide credit: L. Lazebnik

No Free Lunch Theorem

Slide credit: D. Hoiem

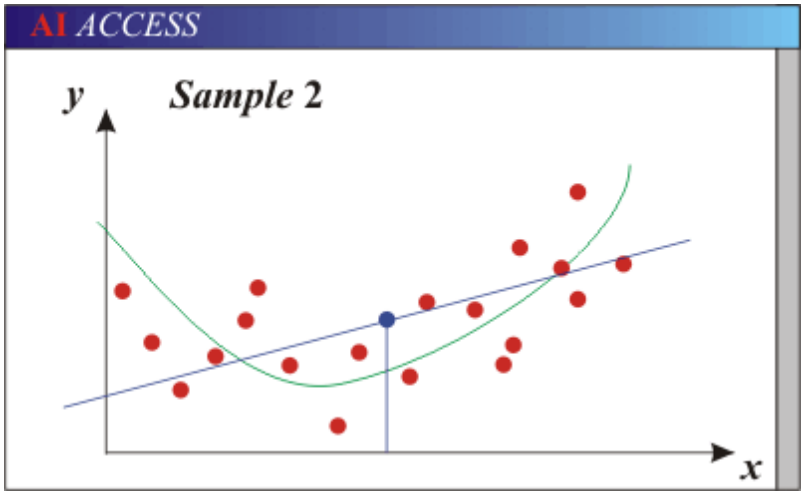
No free lunch theorem

- *Averaged over all possible data generating distributions, every classification algorithm has the same error rate when classifying previously unobserved points.*

No free lunch theorem

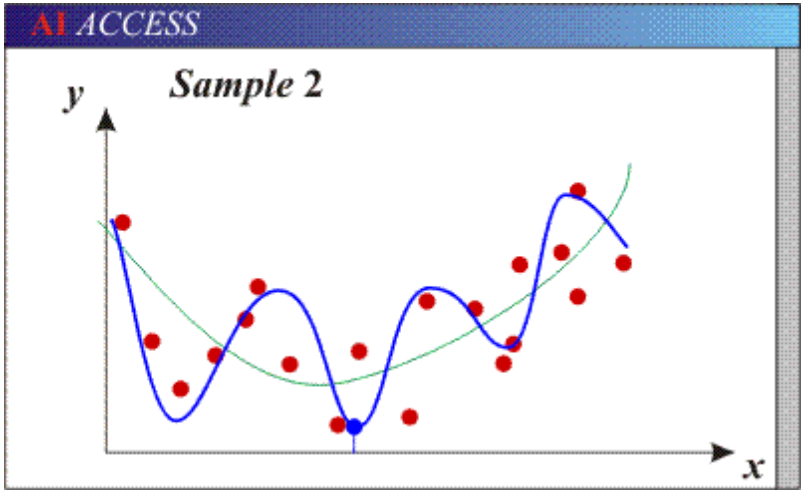
- *Averaged over all possible data generating distributions, every classification algorithm has the same error rate when classifying previously unobserved points.*
- This means that the goal of machine learning research is not to seek a universal learning algorithm or the absolute best learning algorithm. Instead, our goal is to understand what kinds of distributions are relevant to the “real world” that an AI agent experiences, and what kinds of machine learning algorithms perform well on data drawn from the kinds of data generating distributions we care about.

Bias-Variance Trade-off



Models with too few parameters are inaccurate because of a large bias.

- Not enough flexibility!



Models with too many parameters are inaccurate because of a large variance.

- Too much sensitivity to the sample.

Slide credit: D. Hoiem

Bias-Variance Trade-off

$$E(MSE) = noise^2 + bias^2 + variance$$

Unavoidable error

Error due to incorrect assumptions

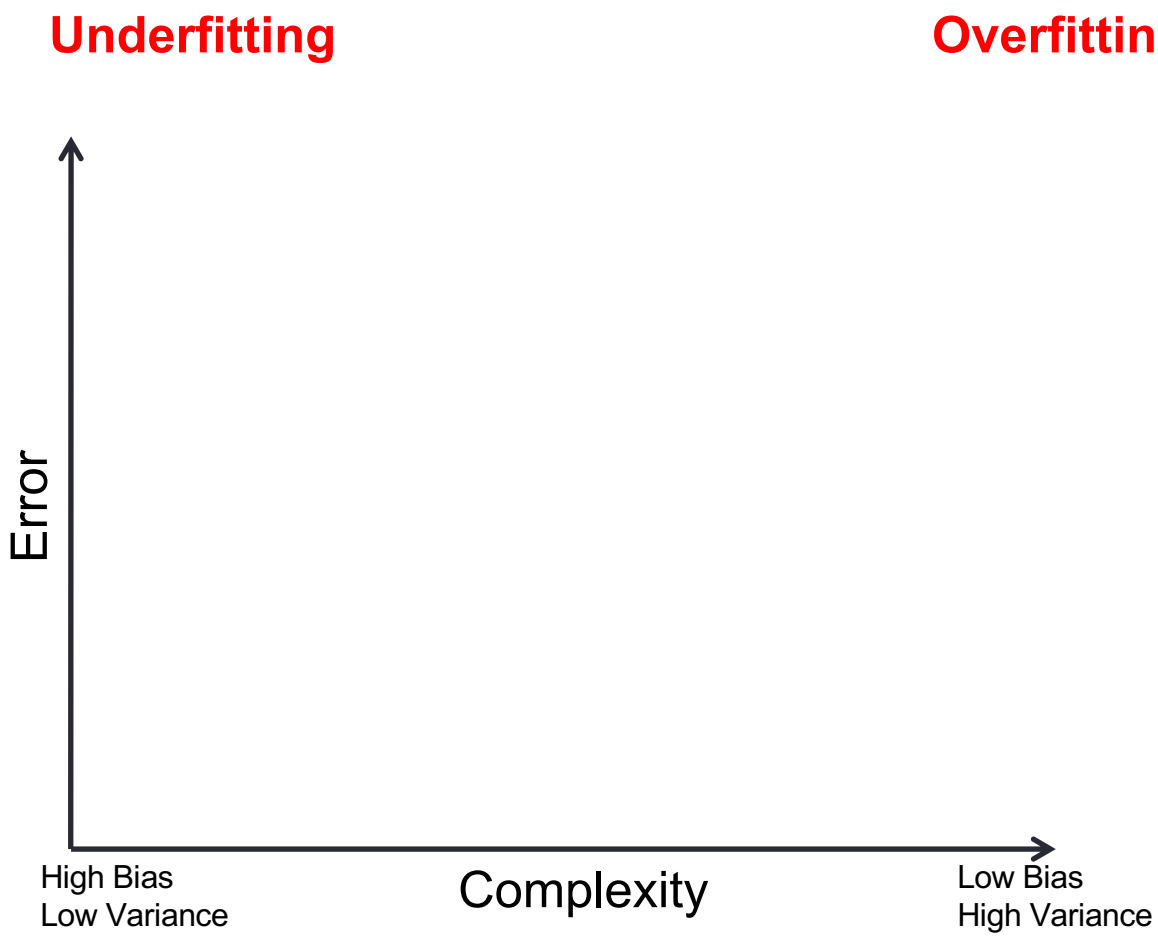
Error due to variance of training samples

For explanations of bias-variance (also Bishop's "Neural Networks" book):

- http://www.inf.ed.ac.uk/teaching/courses/mlsc/Notes/Lecture4/BiasVariance.pdf

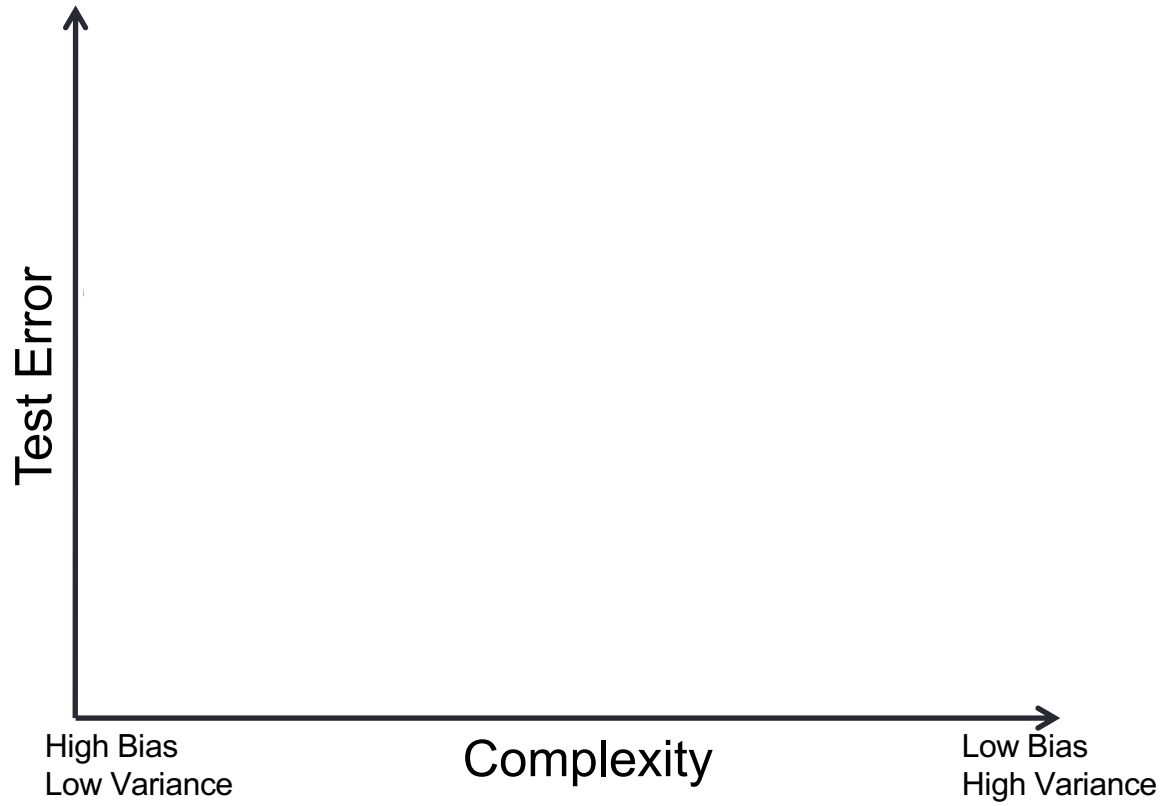
Slide credit: D. Hoiem

Bias-variance tradeoff



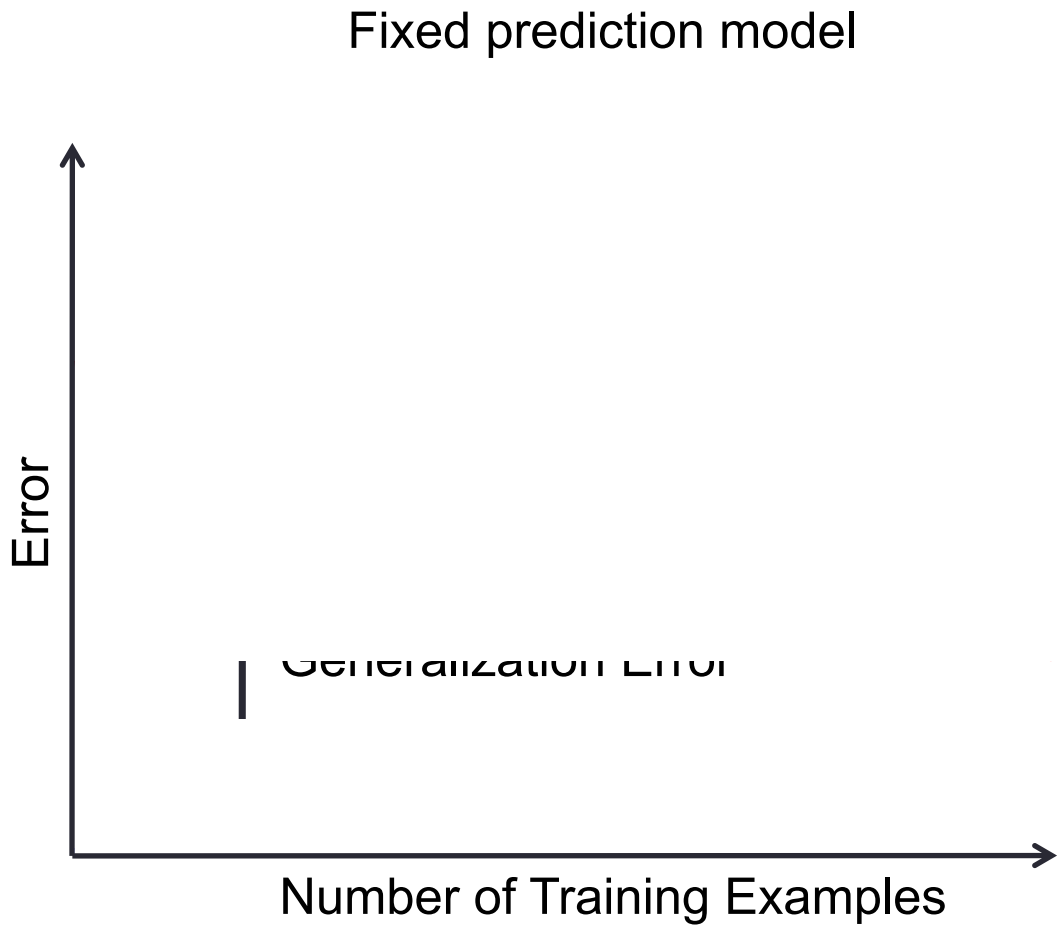
Slide credit: D. Hoiem

Bias-variance tradeoff



Slide credit: D. Hoiem

Effect of Training Size



Slide credit: D. Hoiem

The perfect classification algorithm

- Objective function: encodes the right loss for the problem
- Parameterization: makes assumptions that fit the problem
- Regularization: right level of regularization for amount of training data
- Training algorithm: can find parameters that maximize objective on training set
- Inference algorithm: can solve for objective function in evaluation

Slide credit: D. Hoiem

Remember…

- 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

Slide credit: D. Hoiem

How to reduce variance?

- Choose a simpler classifier
- Regularize the parameters
- Get more training data

Slide credit: D. Hoiem

Very brief tour of some classifiers

- **K-nearest neighbor**
- **SVM**
- Boosted Decision Trees
- Neural networks (+CNNs)
- Naïve Bayes
- Bayesian network
- Logistic regression
- Randomized Forests
- Restricted Boltzmann Machines
- ...

Generative vs. Discriminative Classifiers

Discriminative Models

- Learn to directly predict the labels from the data
- Often, assume a simple boundary (e.g., linear)
- Examples
  - Logistic regression
  - SVM
  - Boosted decision trees

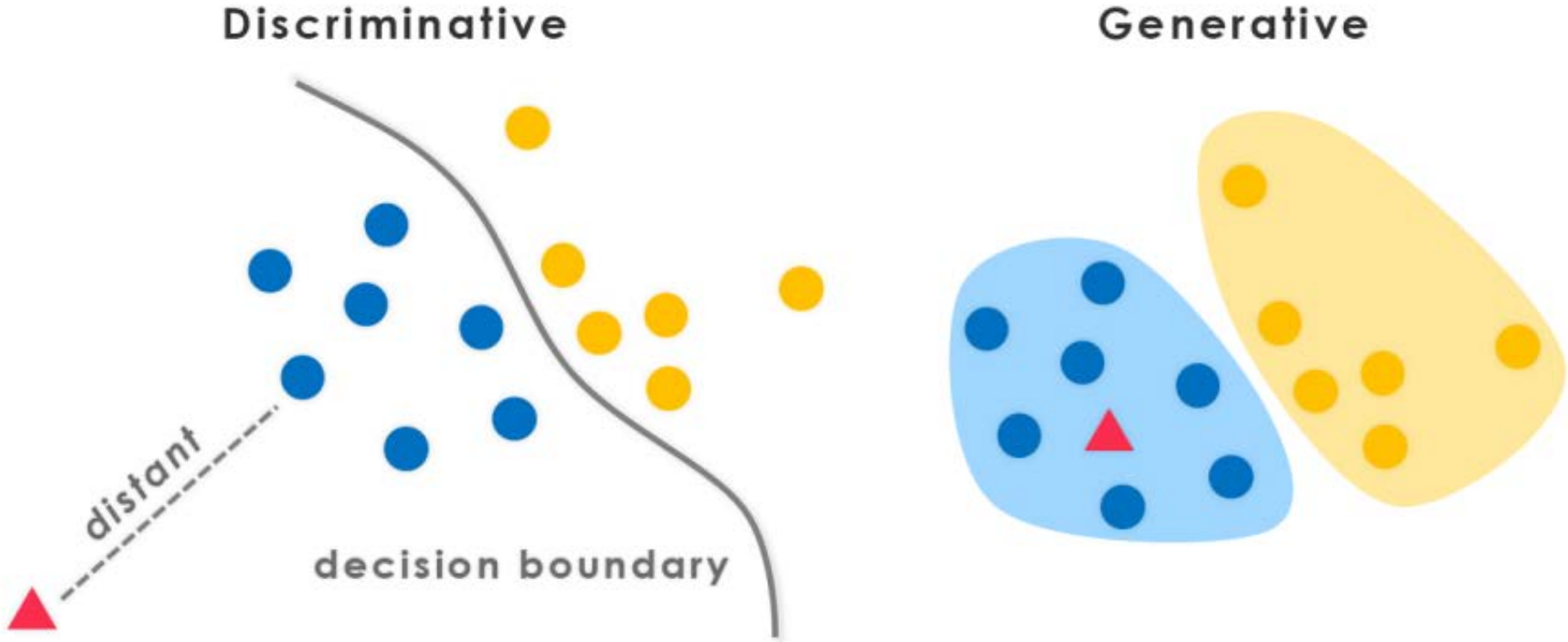
- Often easier to predict a label from the data than to model the data

Generative Models

- Represent both the data and the labels
- Often, makes use of conditional independence and priors
- Examples
  - Naïve Bayes classifier
  - Bayesian network

- Models of data may apply to future prediction problems

Slide credit: D. Hoiem



"Learn the data boundary"

$$P(Y|X = x)$$

target variable (label)

Observable variable

"Represent the data + boundary"

$$P(X, Y)$$

$$\downarrow$$

$$P(X|Y = y)$$

Relationship between discriminative and generative models

Given a model of the joint distribution, $P(X, Y)$

the distribution of the individual variables can be computed as the marginal distributions:

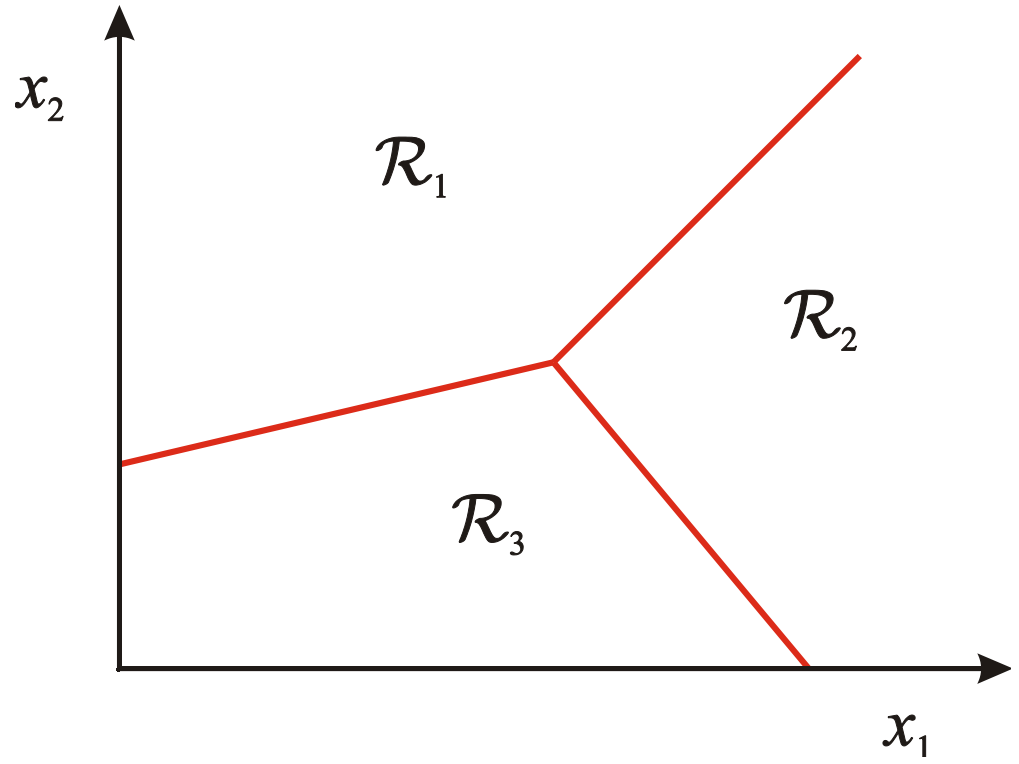
$$P(X) = \sum_y P(X, Y = y) \quad P(Y) = \int_x P(Y, X = x)$$

$$P(X|Y) = P(X, Y)/P(Y) \qquad P(Y|X) = P(X, Y)/P(X)$$

Bayes Rule: $P(Y|X)P(X) = P(X|Y)P(Y)$

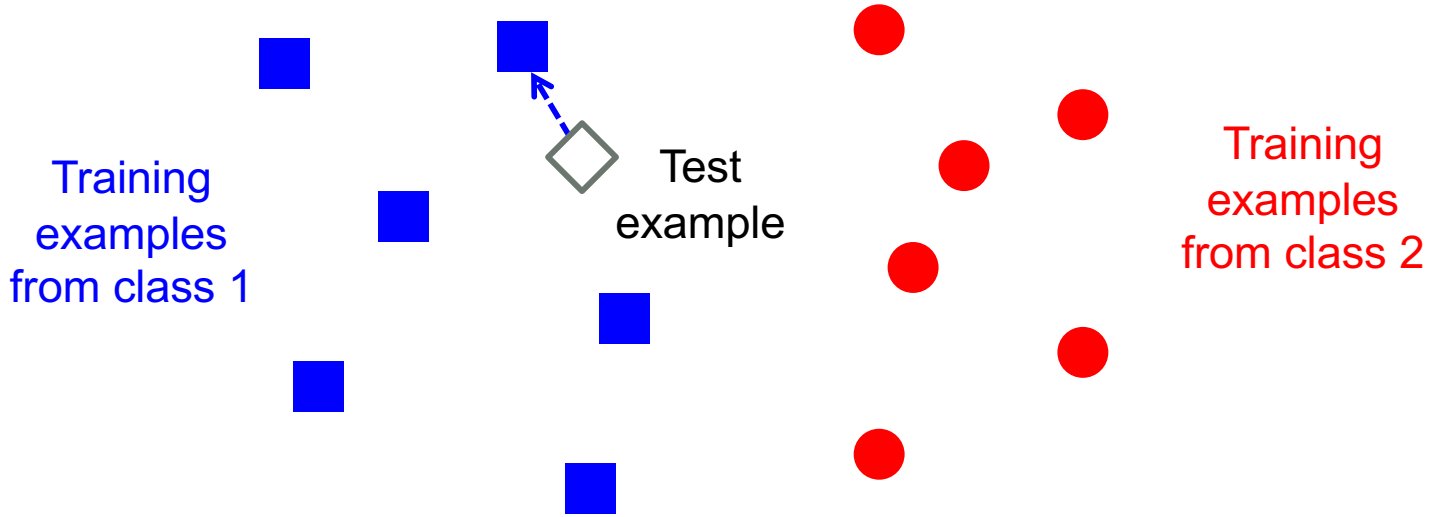
Classification

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



Slide credit: L. Lazebnik

Classifiers: Nearest neighbor

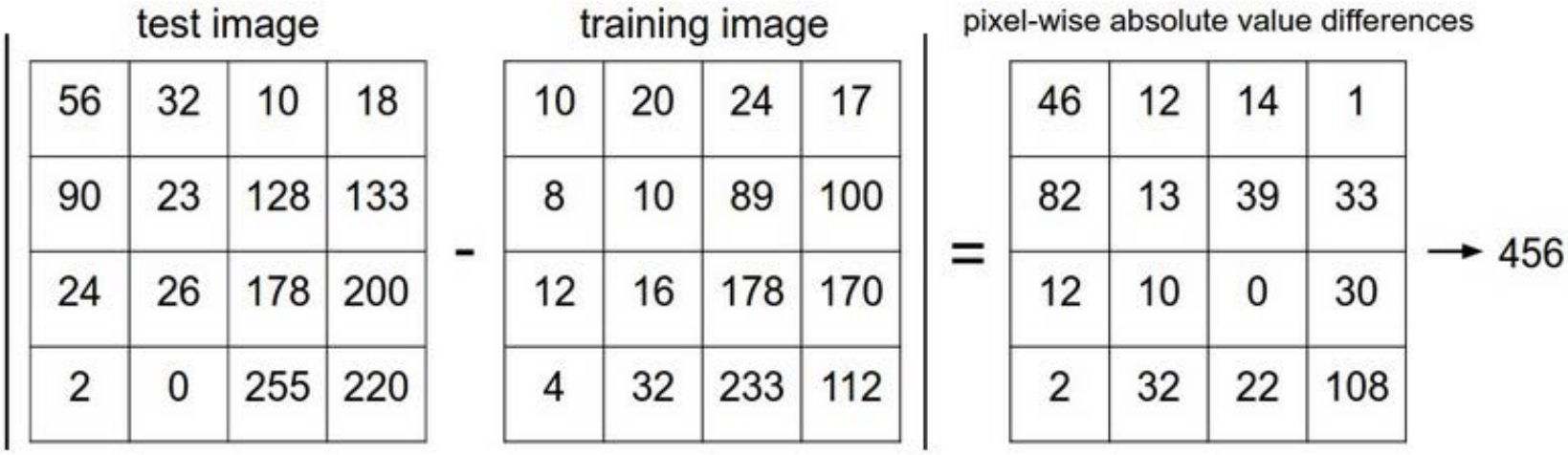


f(**x**) = label of the training example nearest to **x**

- All we need is a distance function for our inputs
- No training required!

Slide credit: L. Lazebnik

Nearest neighbor – L1 distance

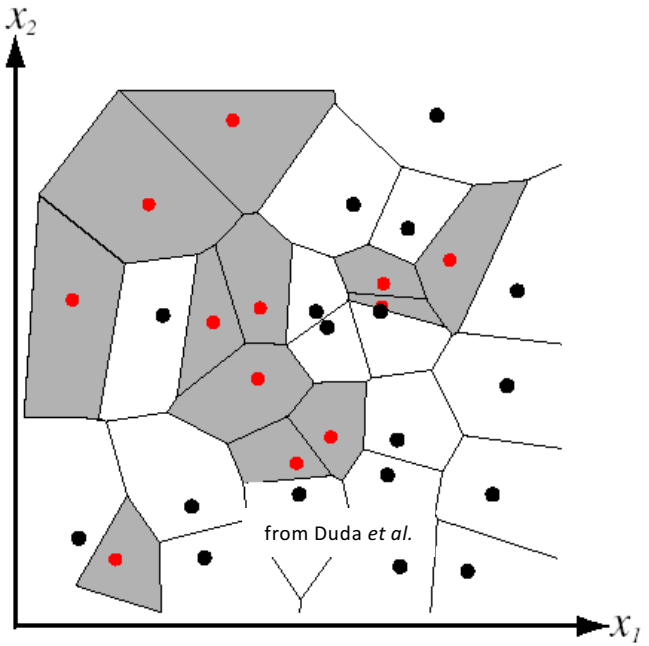


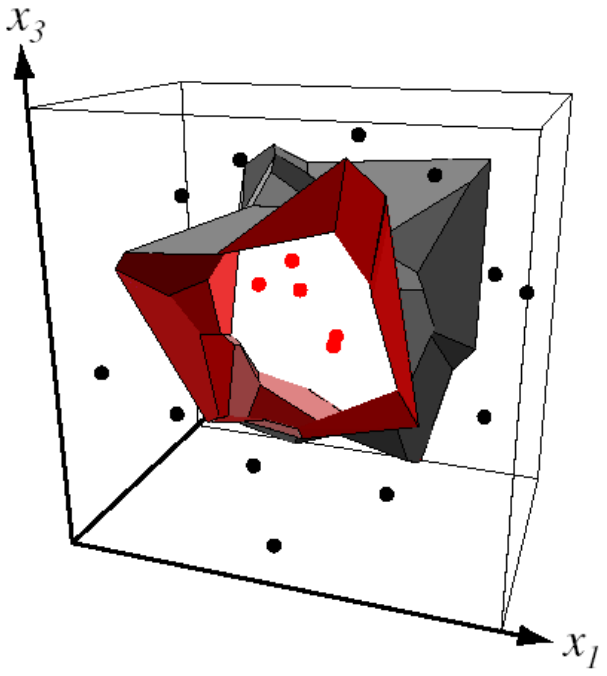
An example of using pixel-wise differences to compare two images with L1 distance (for one color channel in this example). Two images are subtracted elementwise and then all differences are added up to a single number.
If two images are identical the result will be zero. But if the images are very different the result will be large.

Nearest Neighbor Classifier

Assign label of nearest training data point to each test data point.

Divides input space into *decision regions* separated by *decision boundaries – Voronoi.*

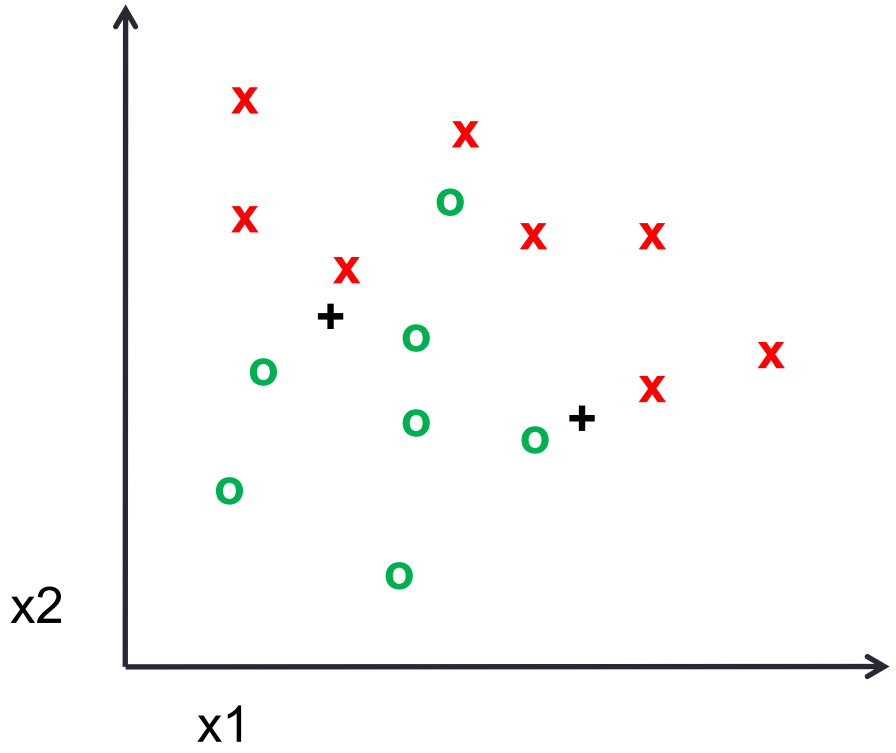




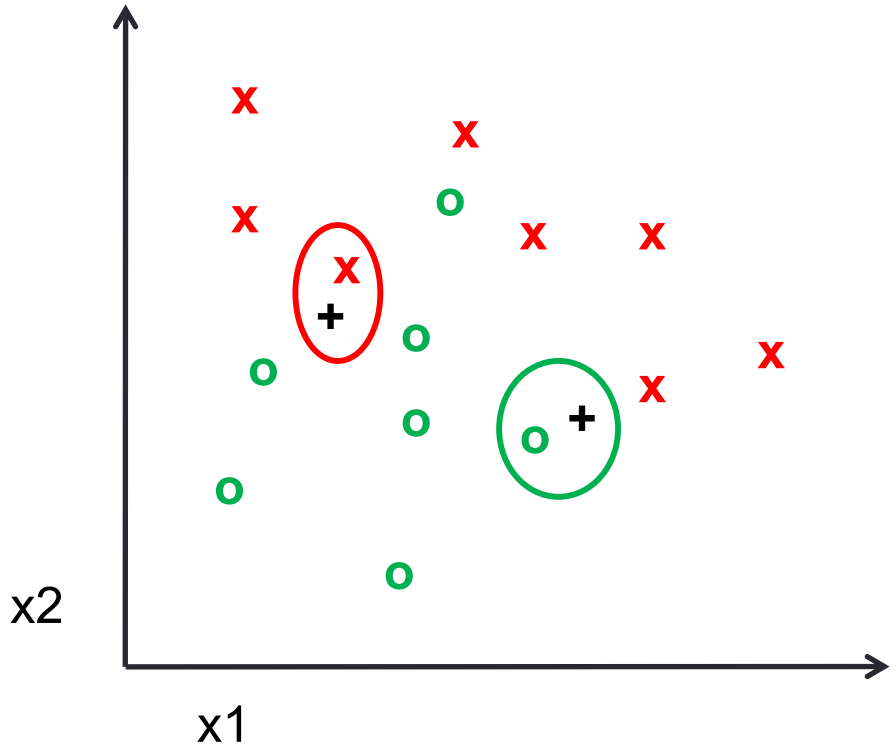
Voronoi partitioning of feature space for two-category 2D and 3D data

Source: D. Lowe

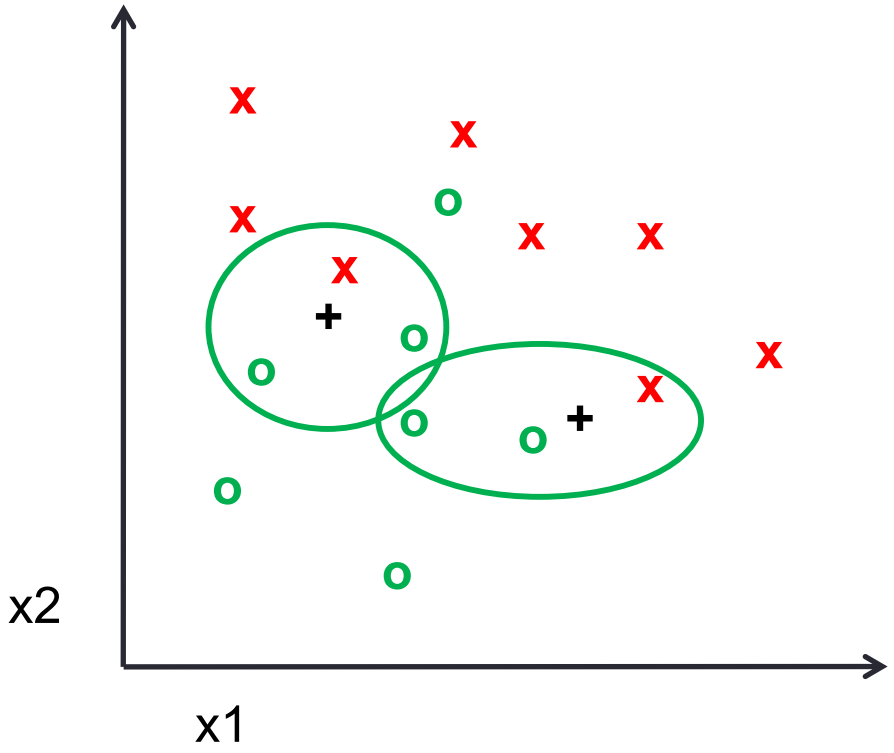
K-nearest neighbor



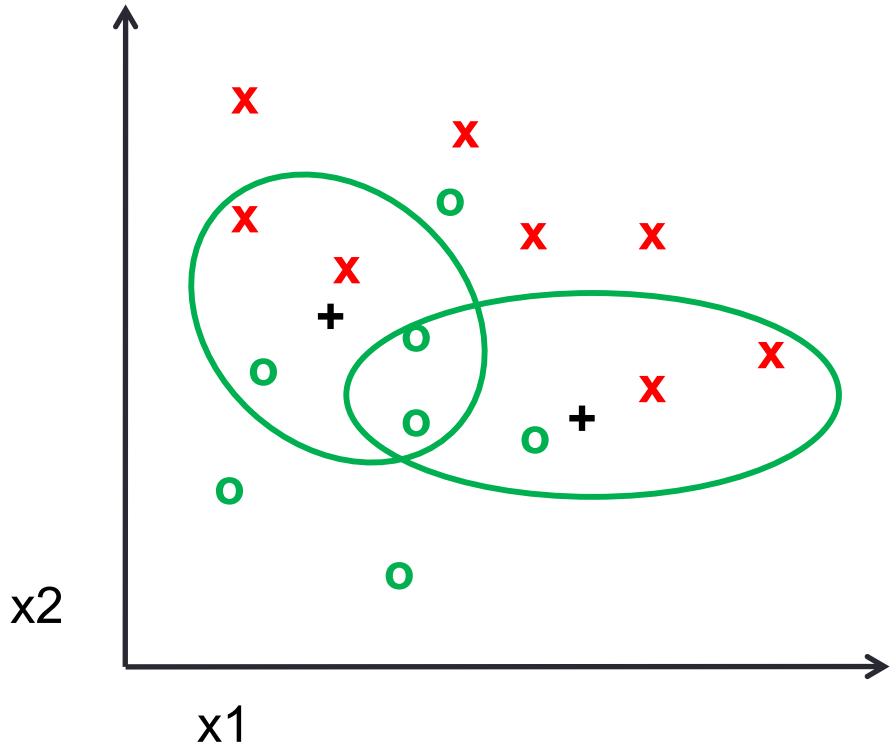
1-nearest neighbor



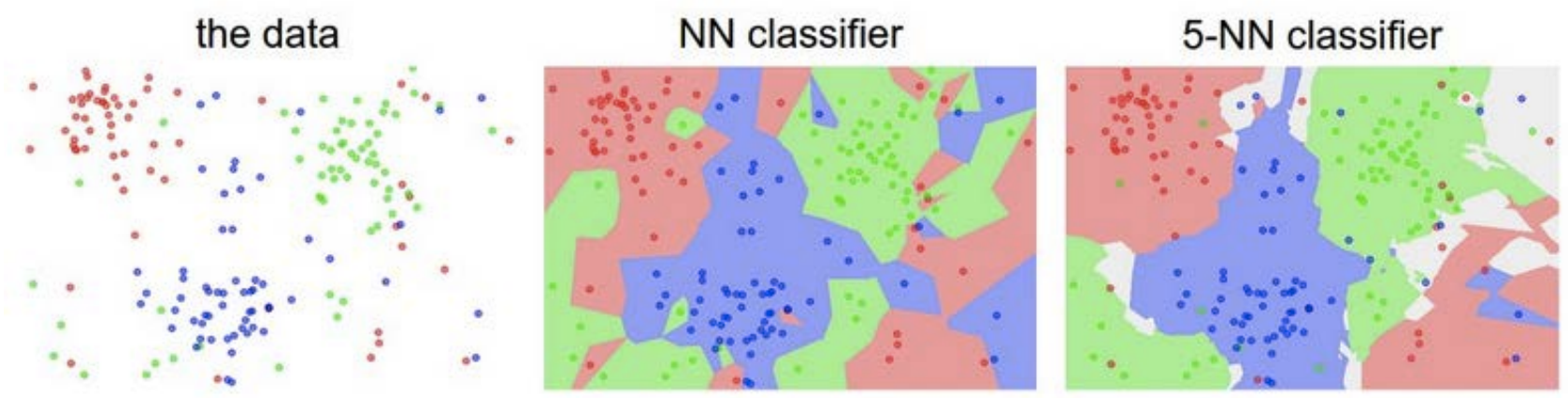
3-nearest neighbor



5-nearest neighbor



K- nearest neighbors



Find the top **k** closest images, and have them vote on the label of the test image.

Validation sets for Hyperparameter tuning

K is an hyperparameter

How can we determine K ?

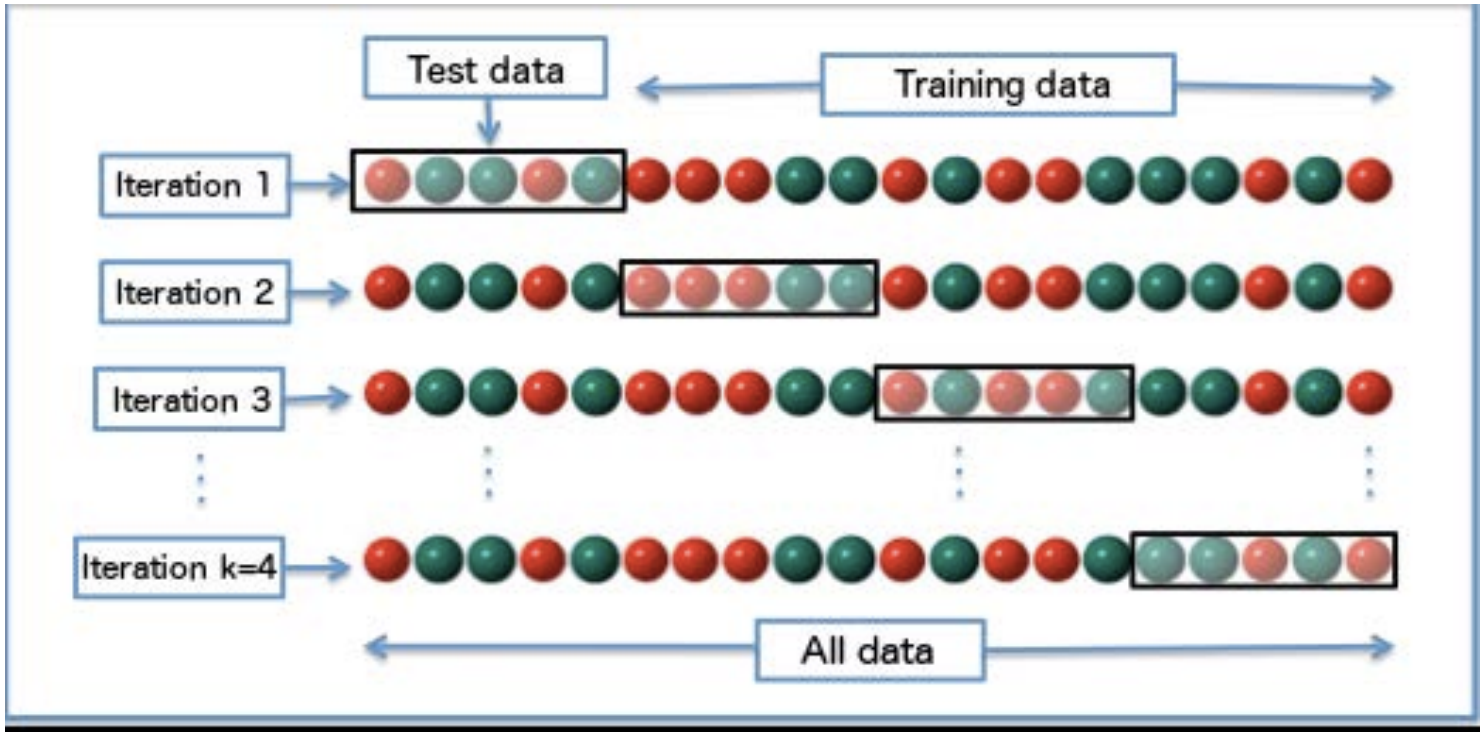
We CANNOT find K by tweaking it for the test set

Use *validation set* instead

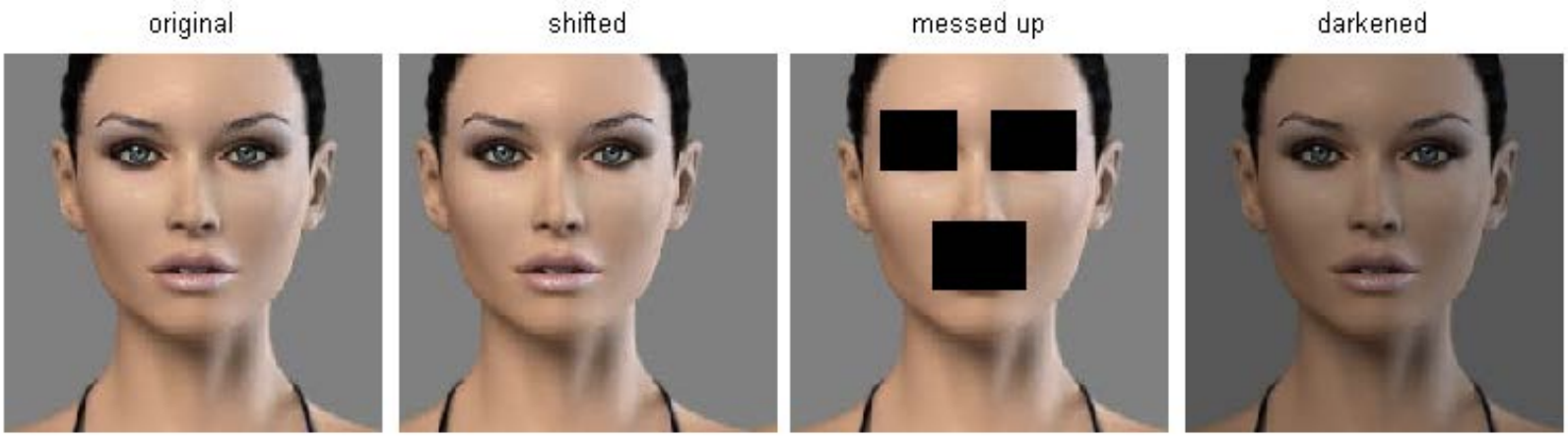
Split your training set into training set and a validation set.
Use validation set to tune all hyperparameters.
At the end run a single time on the test set and report performance.

Cross validation

Insufficient data ?
Split your training set into training set and a validation set.
Use validation set to tune all hyperparameters.
At the end run a single time on the test set and report performance.



Pros and Cons of Nearest Neighbor classifier



Pixel-based distances on high-dimensional data (and images especially) can be very unintuitive. An original image (left) and three other images next to it that are all equally far away from it based on L2 pixel distance. Clearly, the pixel-wise distance does not correspond at all to perceptual or semantic similarity.

K-NN Disadvantages

- 1) The classifier must *remember* all of the training data and store it for future comparisons with the test data. This is space inefficient because datasets may easily be gigabytes in size.

2) Classifying a test image is expensive since it requires a comparison to all training images.

Parameterized mapping from images to label scores

- Define a *score function* that maps the pixel values of an image to confidence scores for each class:
- Training data-set: N examples, K categories

$$f : \mathbb{R}^D \rightarrow \mathbb{R}^K$$

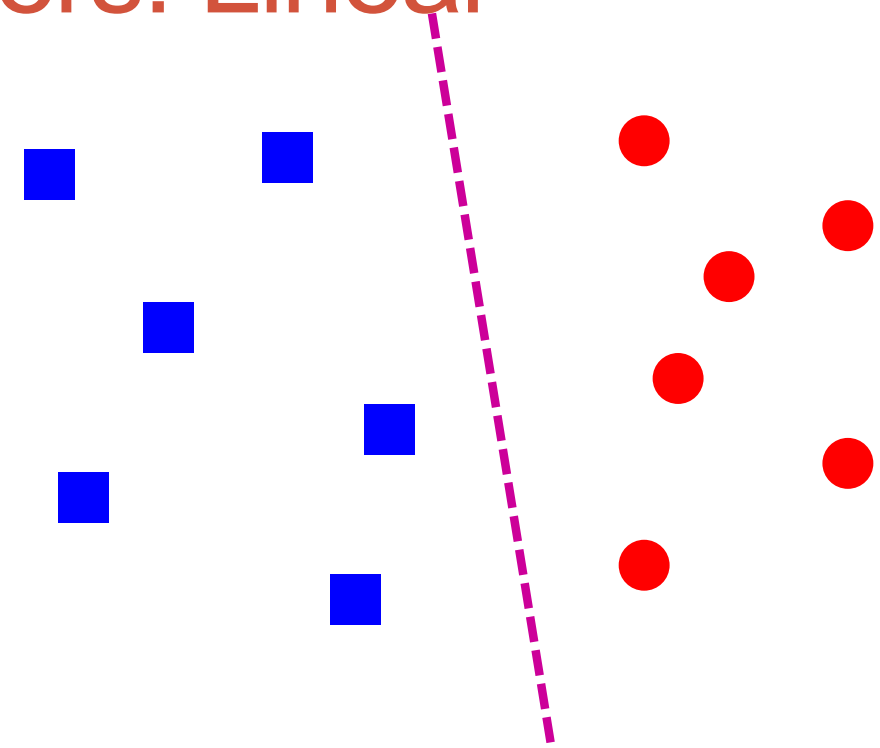
$$\{x_i, y_i\}_{i=1}^N, \quad x_i \in \mathbb{R}^D, y_i \in \{1, \ldots K\}$$

$$f(x) = Wx_i + b$$

Linear classifier:

The matrix **W** (of size [K x D]), and the vector **b** (of size [K x 1]) are the *parameters (weights)* of the function.

Classifiers: Linear

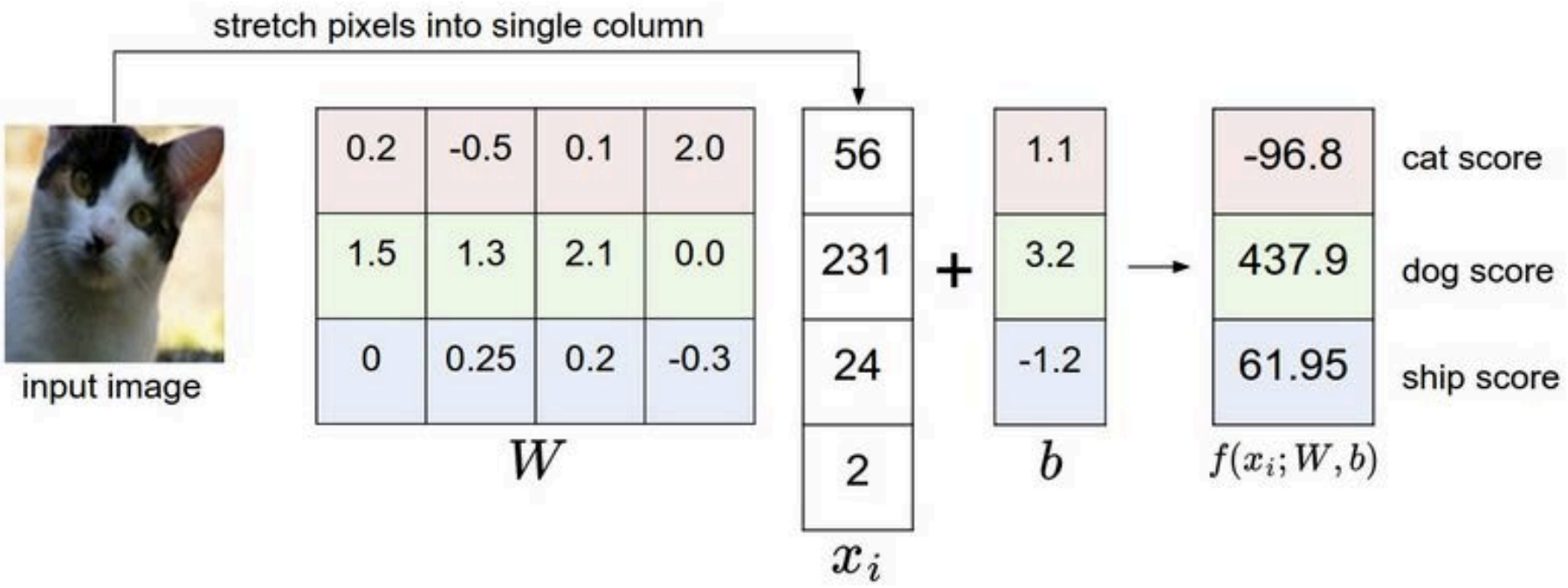


- Find a *linear function* to separate the classes:

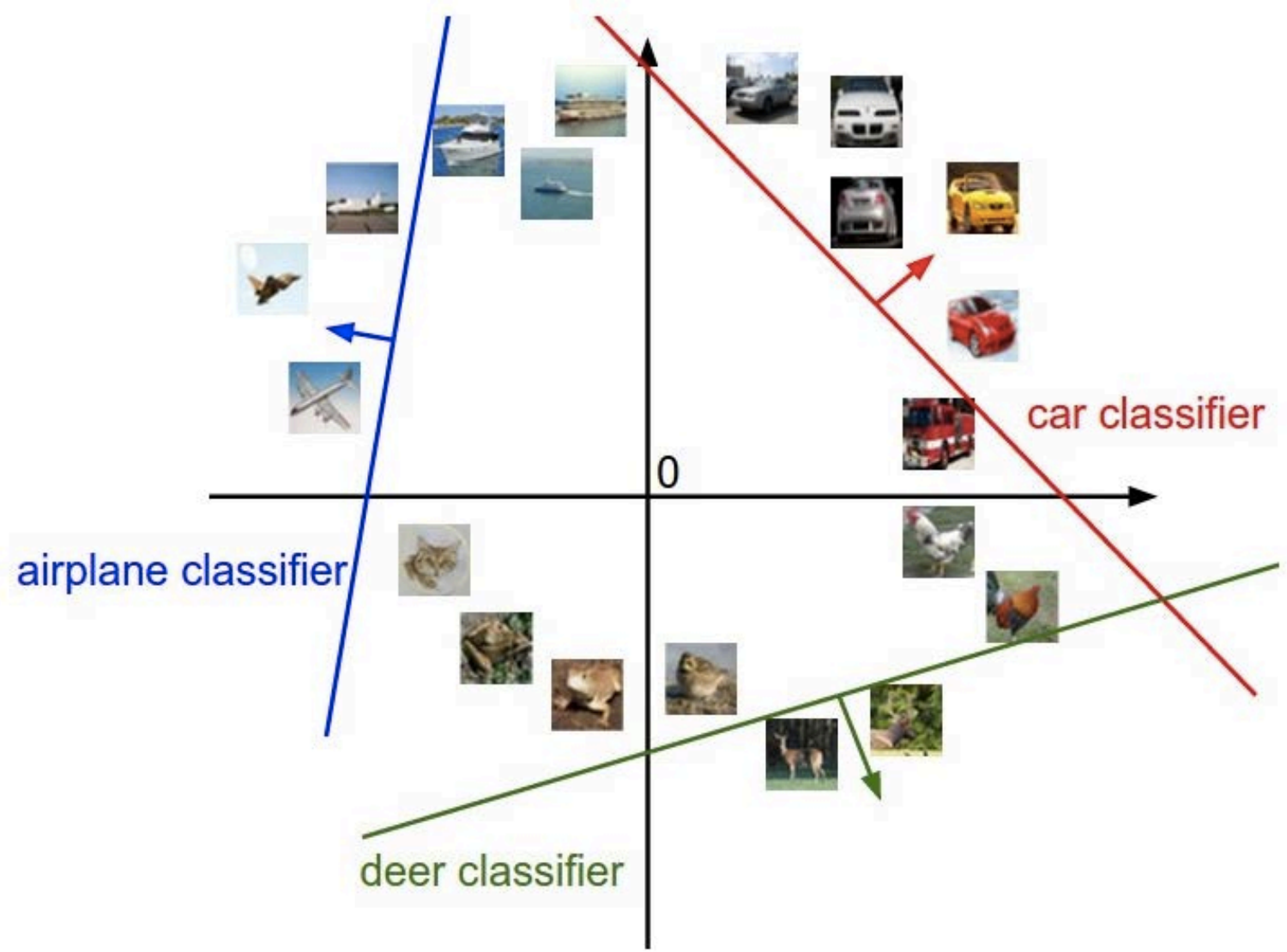
$$f(\mathbf{x}) = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b)$$

Slide credit: L. Lazebnik

Interpreting a linear classifier



Analogy of images as high-dimensional points.



Classifiers: Linear Support Vector Machine (SVM)

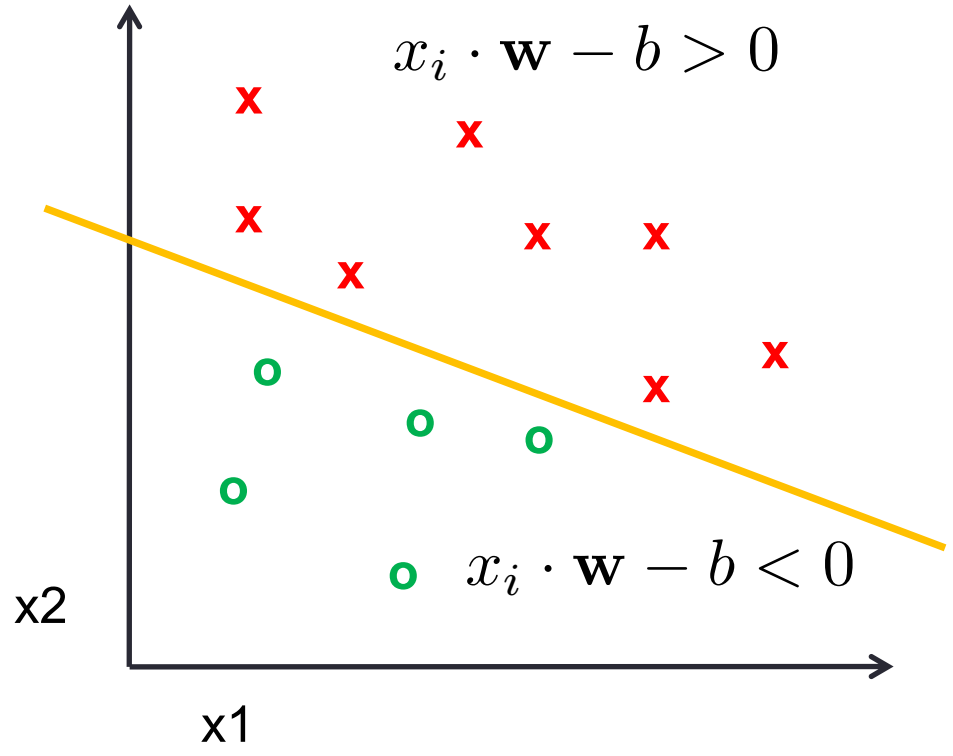
$$x_i \in \mathbb{R}^n \quad i = 1, \ldots, m$$

$$y_i = \{-1, 1\}$$

$$\mathbf{w} \in \mathbb{R}^n$$

$$b \in \mathbb{R}$$

$$y_i(x_i \cdot \mathbf{w} - b) > 0$$



APNIK, V., and A. LERNER, 1963. Pattern recognition using generalized portrait method. *Automation and Remote Control*, **24**, 774–780.

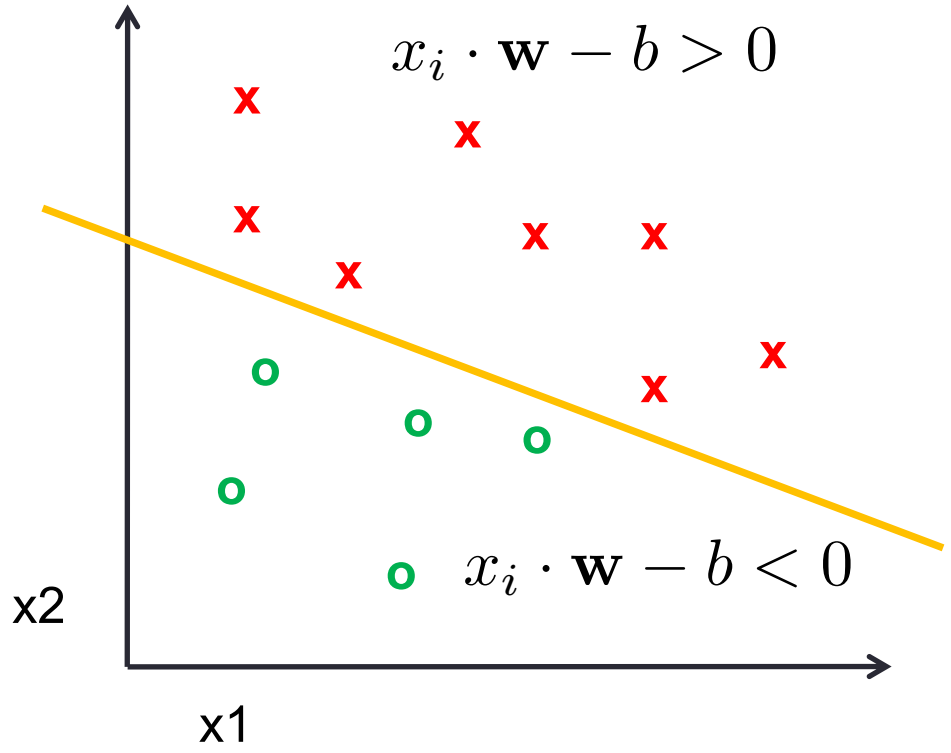
Classifiers: Linear Support Vector Machine (SVM)

$$x_i \in \mathbb{R}^n \quad i = 1, \ldots, m$$

$$y_i = \{-1, 1\}$$

$$\mathbf{w} \in \mathbb{R}^n$$

$$b \in \mathbb{R}$$



- Find a *linear function* to separate the classes:

$$f(\mathbf{x}) = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b)$$

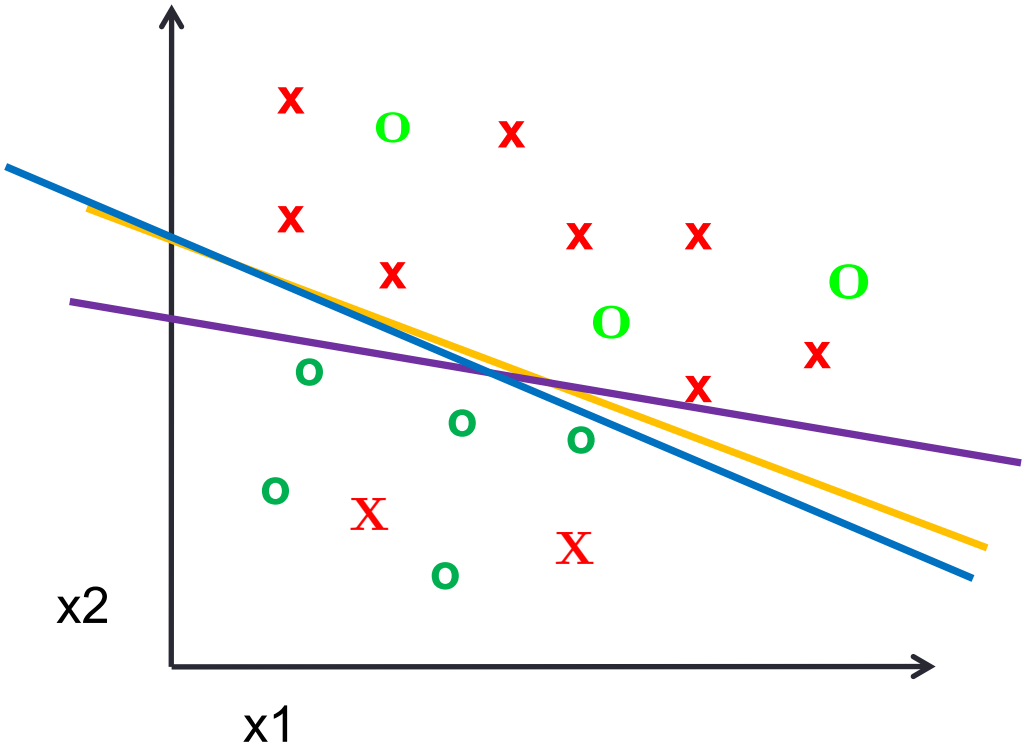
Classifiers: Linear Support Vector Machine (SVM)

$$x_i \in \mathbb{R}^n \quad i = 1, \ldots, m$$

$$y_i = \{-1, 1\}$$

$$\mathbf{w} \in \mathbb{R}^n$$

$$b \in \mathbb{R}$$



Challenges:

- The classes may not be linearly separable
- Suppose the classes are linearly separable the hyperplane defined by $b, \mathbf{w}$ is not unique.

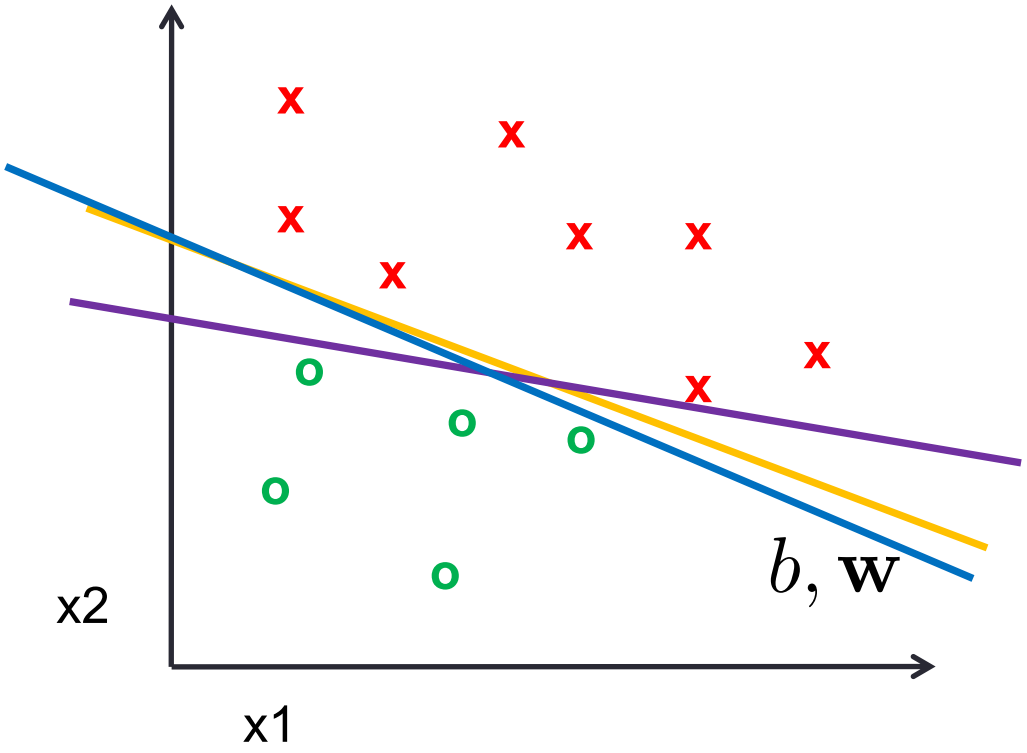
Classifiers: Linear Support Vector Machine (SVM)

$$x_i \in \mathbb{R}^n \quad i = 1, \ldots, m$$

$$y_i = \{-1, 1\}$$

$$\mathbf{w} \in \mathbb{R}^n$$

$$b \in \mathbb{R}$$



Among all possible hyperplanes $b, \mathbf{w}$ which of them has the best generalization properties?

The goal is to construct a learning machine is to maximize the performance on the test data (the instances we haven't seen),

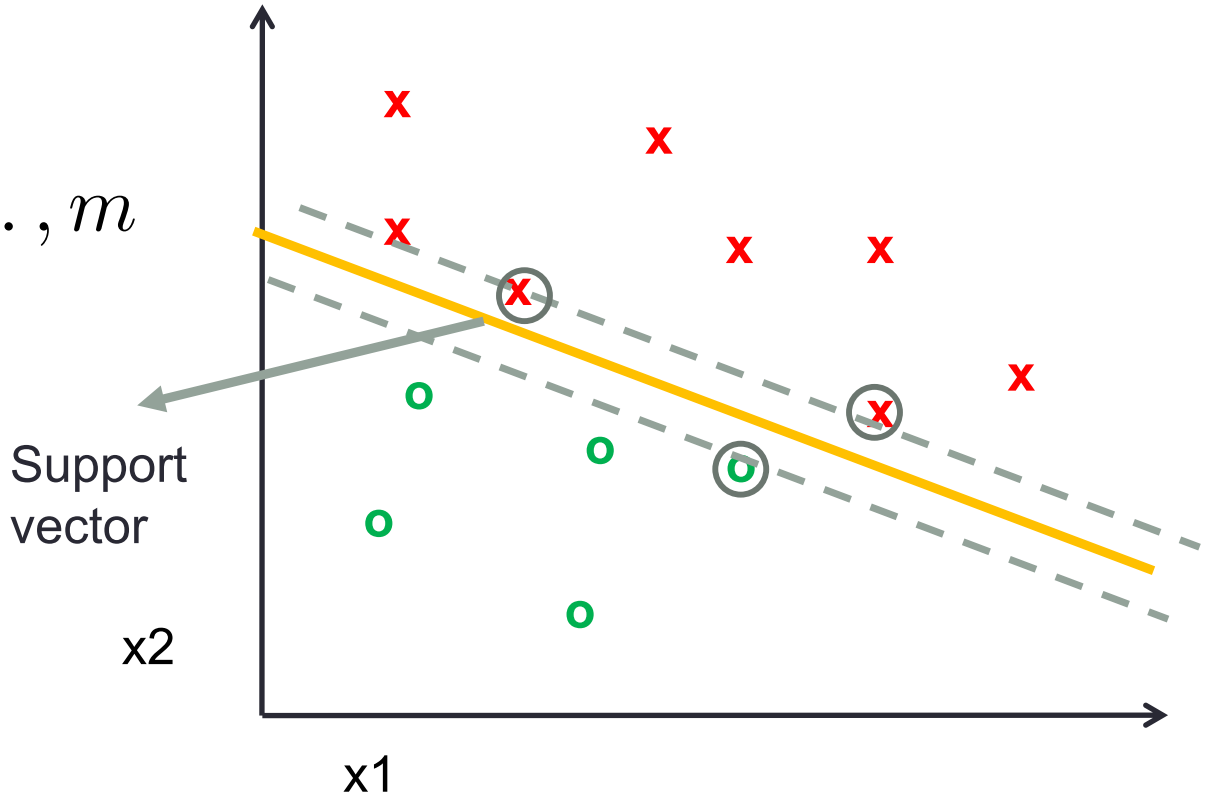
Classifiers: Linear SVM - Margin

$$x_i \in \mathbb{R}^n \quad i = 1, \ldots, m$$

$$y_i = \{-1, 1\}$$

$$\mathbf{w} \in \mathbb{R}^n$$

$$b \in \mathbb{R}$$



C_γ -a subset of all hyperplanes which have a fixed margin γ

A margin is defined as the distance of the closest training point to the hyperplane:

$$\gamma = \min_i \left\{ \frac{y_i(\mathbf{w}^T \mathbf{x}_i - b)}{||\mathbf{w}||} \right\}$$

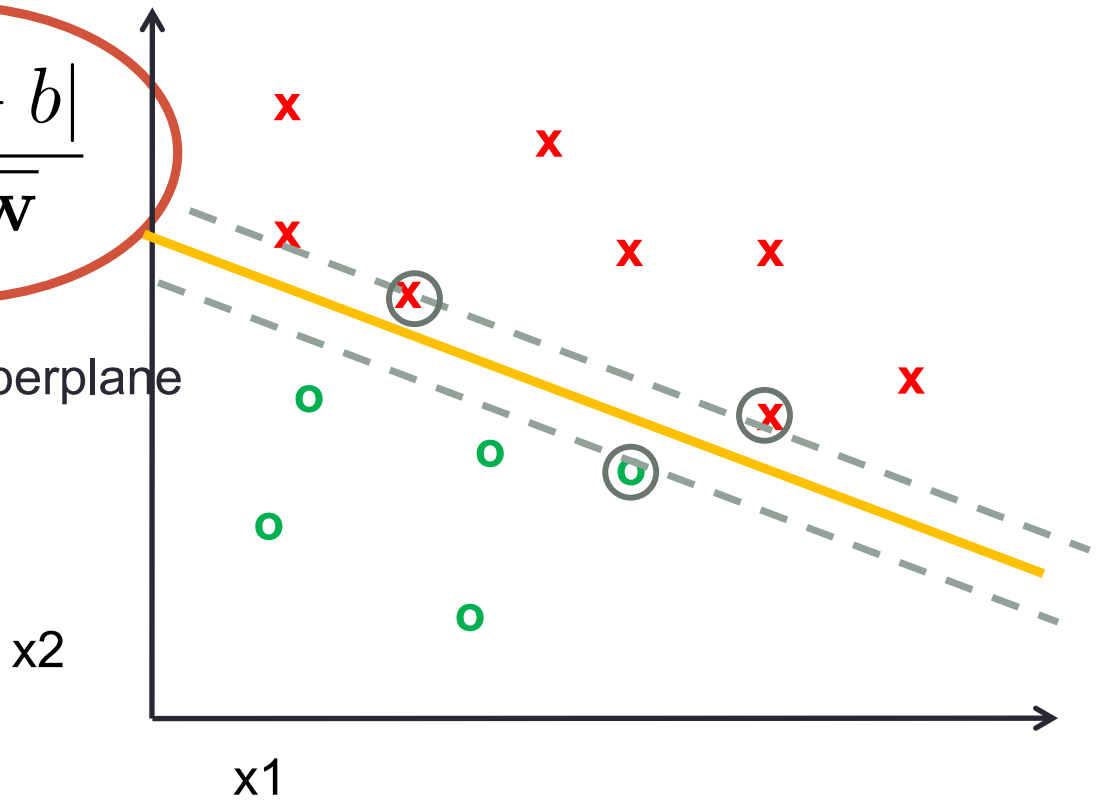
Classifiers: Linear SVM

We look for $\arg\max_{\mathbf{w}} \frac{|\mathbf{w} \cdot \mathbf{x} - b|}{\sqrt{\mathbf{w} \cdot \mathbf{w}}}$

Distance of a point $\mathbf{x}$ from an hyperplane

We can normalize $b, \mathbf{w}$ such that

$$|\mathbf{w} \cdot \mathbf{x} - b| = 1$$



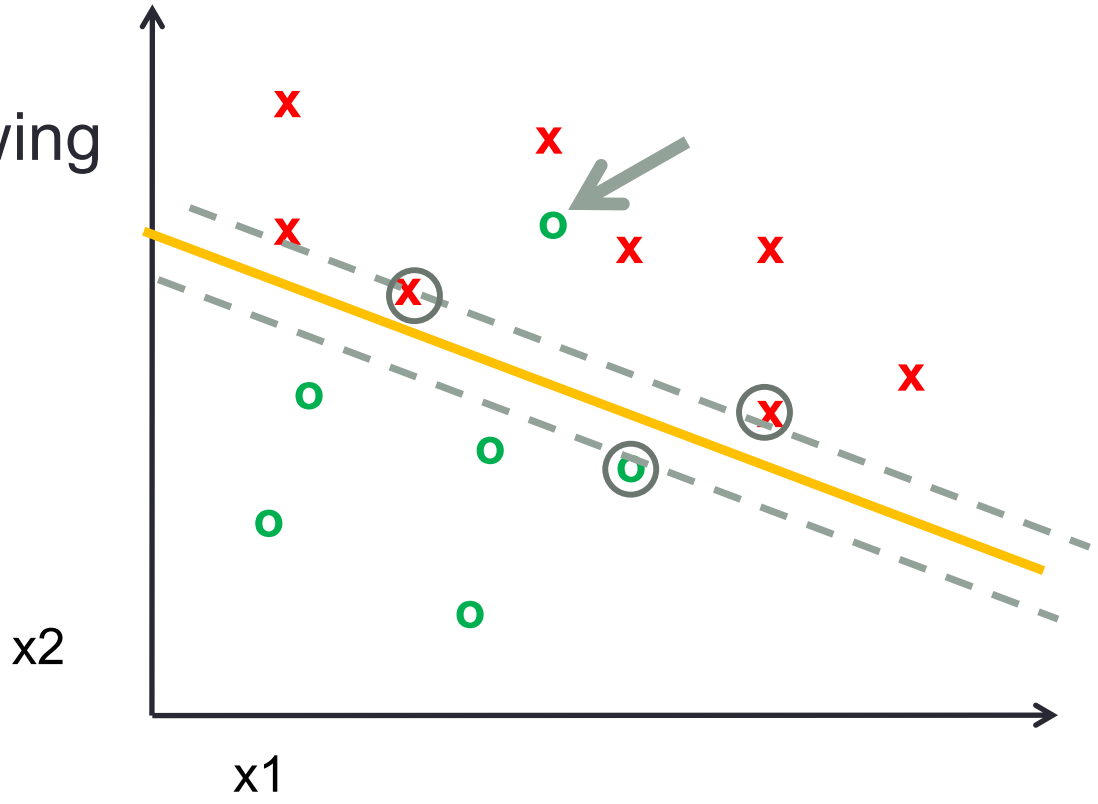
We can equivalently solve for

$$\arg\min_{\mathbf{w}} \frac{1}{2} \mathbf{w} \cdot \mathbf{w}$$

(Hard)

Classifiers: Linear SVM

We can define the following optimization problem:



$$\min_{\mathbf{w},b} \quad \frac{1}{2}\mathbf{w}\cdot\mathbf{w}$$

$$\text{subject to}$$

$$y_i(\mathbf{w}\cdot x_i - b) - 1 \geq 0 \quad 1 = 1,\ldots,m$$

Quadratic Linear Programming (QP) problem

but this is valid only when the classes are linearly separable

Classifiers: Linear SVM

$b, \mathbf{w}$ maximum-margin hyperplane

Support vectors are defined by $\mathbf{x}$ such that:

$$|\mathbf{w} \cdot \mathbf{x} - b| = 1$$

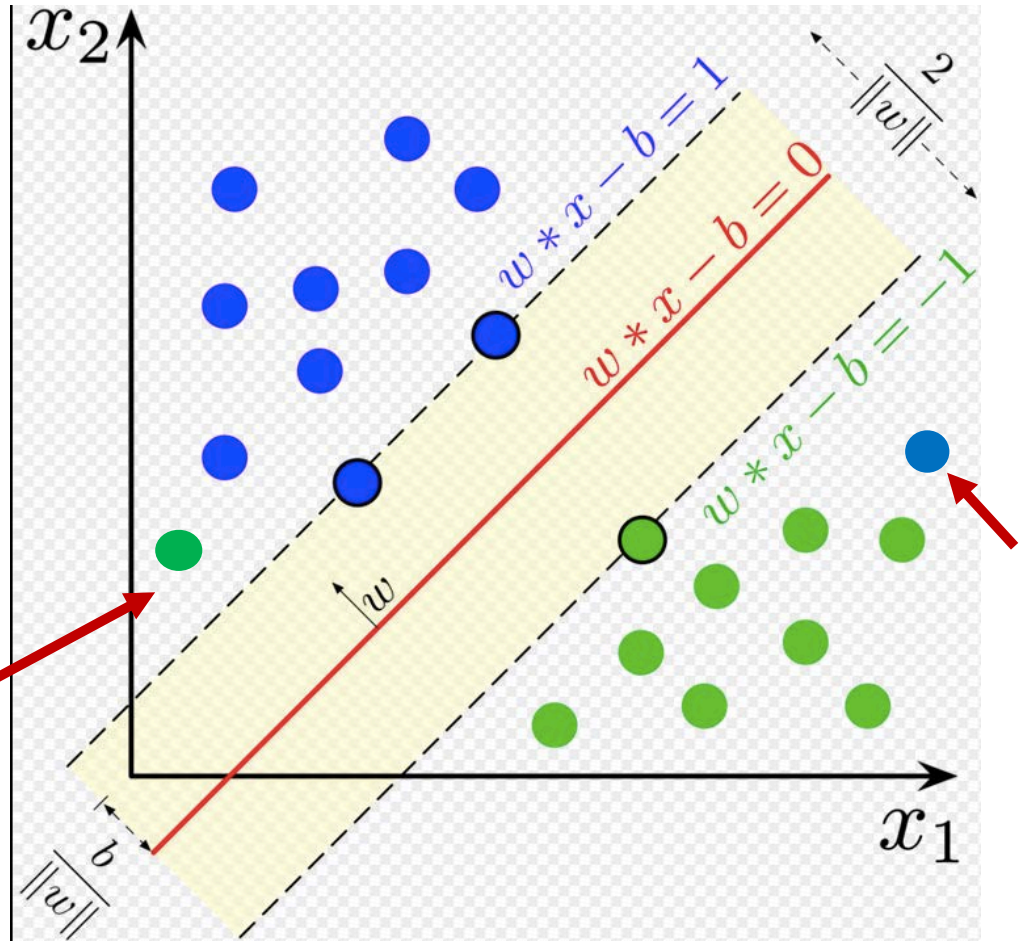
For linearly separated points

$$y_i(\mathbf{w} \cdot \mathbf{x}_i - b) \geq 1$$

$$1 - y_i(\mathbf{w} \cdot \mathbf{x}_i - b) \leq 0$$

otherwise $\mathbf{x}_j$

$$1 - y_i(\mathbf{w} \cdot \mathbf{x}_j - b) \geq 0$$



(Wikipedia)

Hinge Loss: $\max(0, 1 - y_i(\mathbf{w} \cdot \mathbf{x}_j - b))$

SVM Classifier: Soft Margin

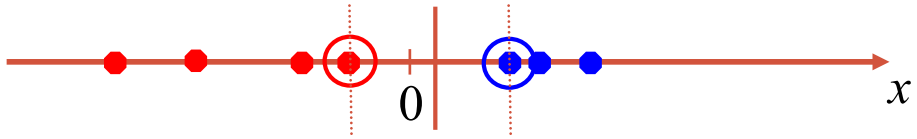
We look for $b, \mathbf{w}$ to minimize

$$\left[\sum_{i=1}^{n} \max(0, 1 - y_i(\mathbf{w} \cdot \mathbf{x}_j - b))\right] + \lambda ||\mathbf{w}||$$

margin/
regularization

Nonlinear SVMs

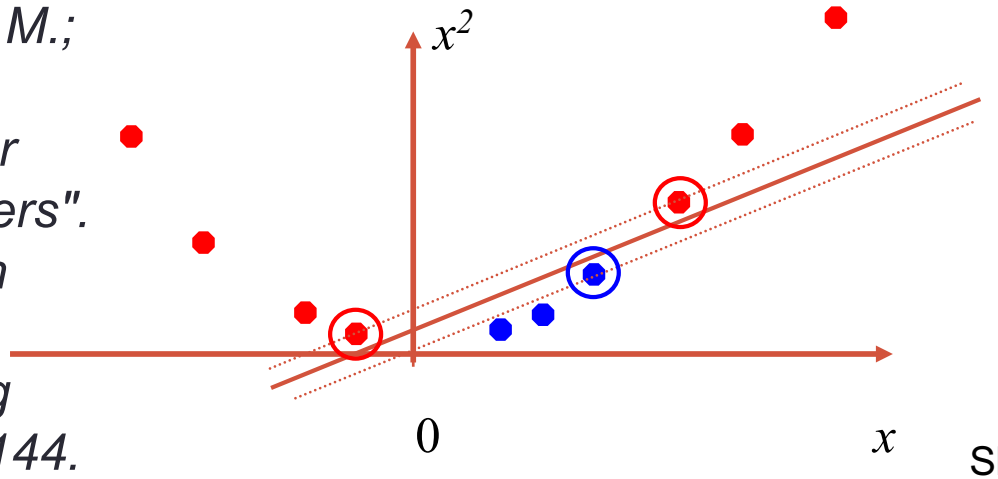
- Datasets that are linearly separable work out great:



- But what if the dataset is just too hard?

- We can map it to a higher-dimensional space:

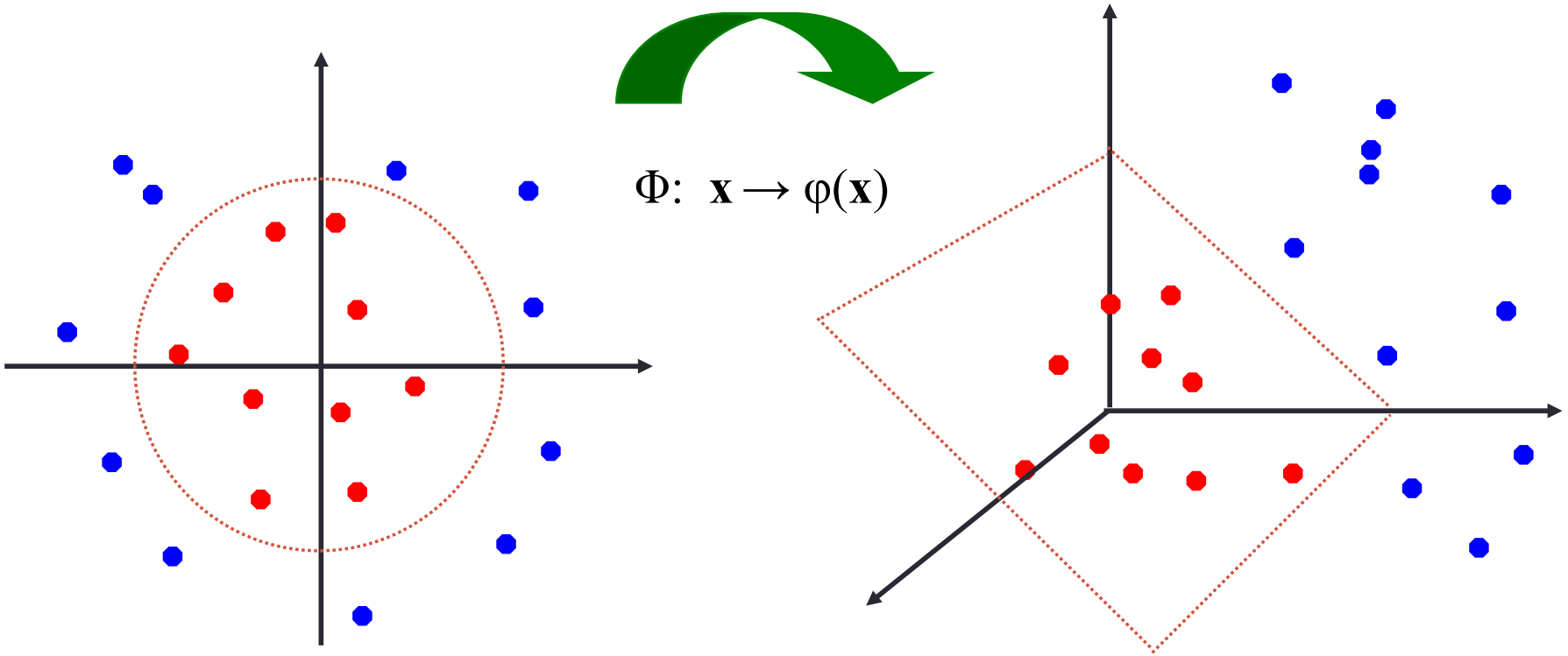
Boser, B. E.; Guyon, I. M.; Vapnik, V. N. (1992). "A training algorithm for optimal margin classifiers". Proceedings of the fifth annual workshop on Computational learning theory – COLT '92. p. 144.



Slide credit: Andrew Moore

Nonlinear SVMs

- General idea: the original input space can always be mapped to some higher-dimensional feature space where the training set is separable:



Slide credit: Andrew Moore

Nonlinear SVM: kernel trick

- We would like to learn a nonlinear classification rule which corresponds to a linear classification rule for the transformed data points $\rho(\mathbf{x}_i)$.
- Instead of explicitly calculating the transformation, we define a kernel function as follows:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \rho(\mathbf{x}_i) \cdot \rho(\mathbf{x}_j)$$

and use the Kernel Trick.

Nonlinear SVM: common kernel functions

- Polynomial (homogeneous): $k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)^d$
- Polynomial (inhomogeneous):

$$k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j + 1)^d$$

- Gaussian radial basis function:
-

$k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma ||\mathbf{x}_i - \mathbf{x}_j||^2)$ sometimes $\gamma = \frac{1}{2}\sigma^2$

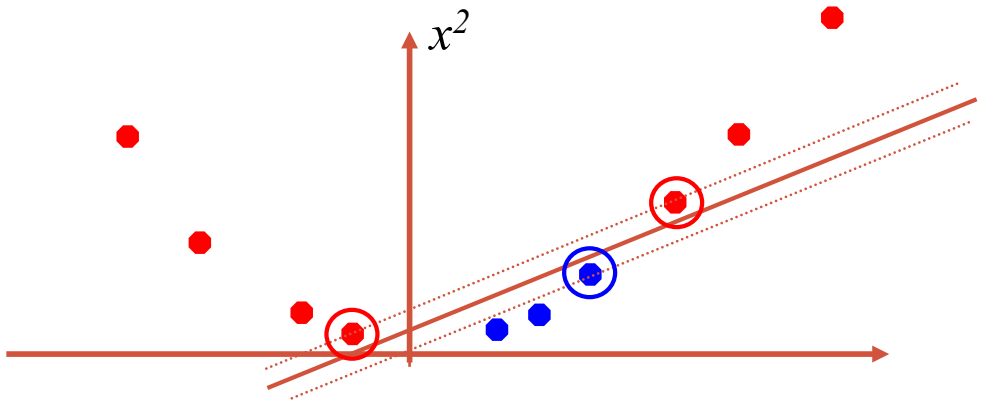
- Hyperbolic tangent: $k(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\kappa \mathbf{x}_i \cdot \mathbf{x}_j + c)$

$$\kappa > 0, \quad c > 0$$

$K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ is positive semi-definite

Nonlinear kernel: Example

- Consider the mapping $\varphi(x) = (x, x^2)$



$$\varphi(x) \cdot \varphi(y) = (x, x^2) \cdot (y, y^2) = xy + x^2 y^2$$

$$K(x, y) = xy + x^2 y^2$$

Nonlinear SVM: Kernel trick

- We can formulate the problem such that only inner products of the input vectors (i.e., $\mathbf{x}_i \cdot \mathbf{x}_j$) are used.
- The kernel is related to the transform by definition:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \rho(\mathbf{x}_i) \cdot \rho(\mathbf{x}_j)$$

- The hyperplane is also defined in the transformed space as follows: $\mathbf{w} = \sum_i \alpha_i y_i \rho(\mathbf{x}_i)$
- Dot products with $\mathbf{w}$ for classification can again be computed by the kernel trick, i.e.,

$$\mathbf{w} \cdot \rho(\mathbf{x}_i) = \sum_i \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}_j)$$

Non-linear SVM algorithm

- The non-linear SVM algorithm is formally similar to the linear SVM, except that every dot product is replaced by a nonlinear kernel function.
- This allows the algorithm to fit the maximum-margin hyperplane in a transformed feature space.
- The transformation may be nonlinear and the transformed space high dimensional; although the classifier is a hyperplane in the transformed feature space, it may be nonlinear in the original input space.

Summary: SVMs for image classification

1. Pick an image representation (in our case, bag of features)
2. Pick a kernel function for that representation
3. Compute the matrix of kernel values between every pair of training examples
4. Feed the kernel matrix into your favorite SVM solver to obtain support vectors and weights
5. At test time: compute kernel values for your test example and each support vector, and combine them with the learned weights to get the value of the decision function

Slide credit: L. Lazebnik

What about multi-class SVMs?

- Unfortunately, there is no “definitive” multi-class SVM formulation
- In practice, we have to obtain a multi-class SVM by combining multiple two-class SVMs
- One vs. others
  - Traning: learn an SVM for each class vs. the others
  - Testing: apply each SVM to test example and assign to it the class of the SVM that returns the highest decision value
- One vs. one
  - Training: learn an SVM for each pair of classes
  - Testing: each learned SVM “votes” for a class to assign to the test example

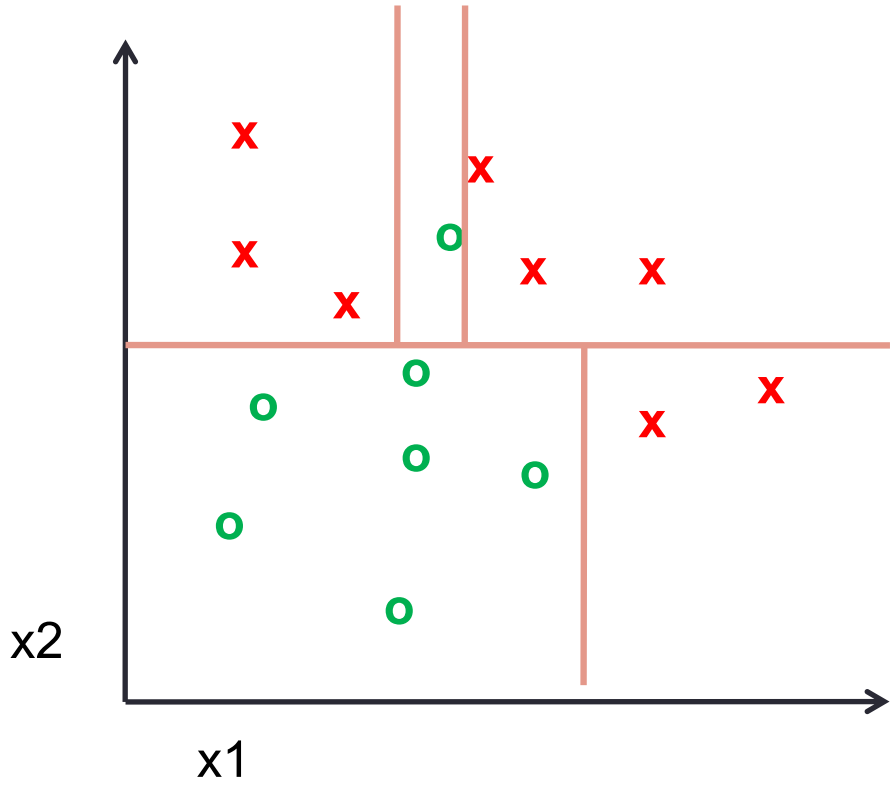
Slide credit: L. Lazebnik

SVMs: Pros and cons

- Pros
  - Many publicly available SVM packages: http://www.kernel-machines.org/software
  - Kernel-based framework is very powerful, flexible
  - SVMs work very well in practice, even with very small training sample sizes

- Cons
  - No “direct” multi-class SVM, must combine two-class SVMs
  - Computation, memory
    - During training time, must compute matrix of kernel values for every pair of examples
    - Learning can take a very long time for large-scale problems

Classifiers: Decision Trees



Random forests L. Breiman.. 1999, 2001
Discrete Adaboost Freund and Shapire 1996

Forests and trees

A forest is an ensemble of trees. The trees are all slightly different from one another.

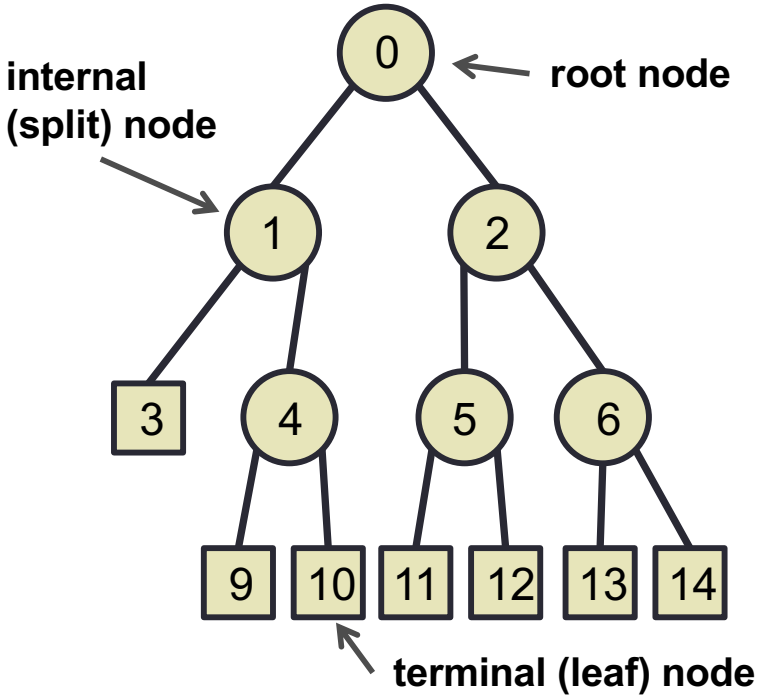
[Y. Amit and D. Geman. **Shape quantization and recognition with randomized trees**. Neural Computation. 9:1545--1588, 1997]
[L. Breiman. **Random forests**. Machine Learning. 45(1):5--32, 2001]

taken from: A. Criminisi, J. Shotton and E. Konukoglu

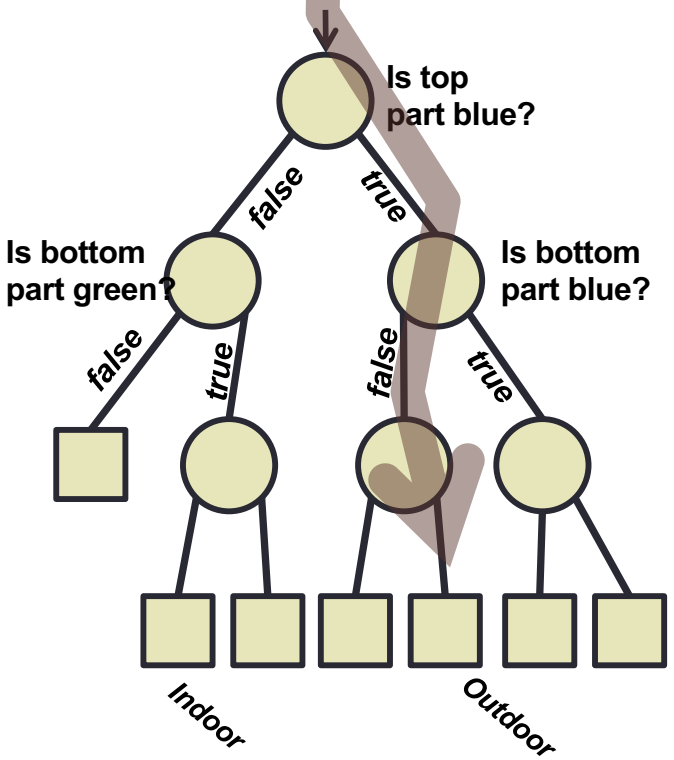
Generic trees and decision trees

taken from: A. Criminisi, J. Shotton and E. Konukoglu

A general tree structure

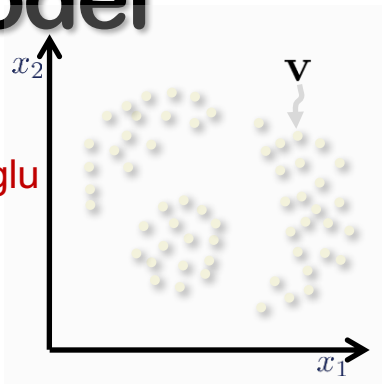


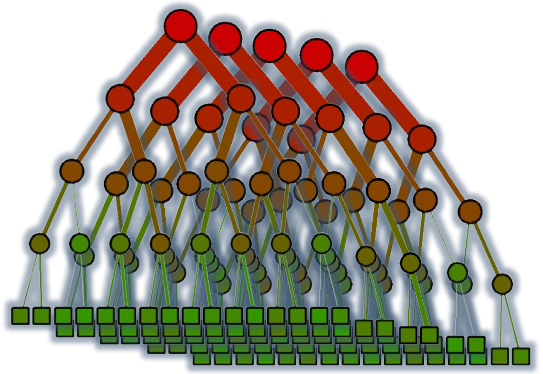
A decision tree



The decision forest model

taken from: A. Criminisi, J. Shotton and E. Konukoglu





Basic notation

Input data point	*e.g.*	$\mathbf{v} = (x_1, \cdots, x_d) \in \mathbb{R}^d$	*Collection of feature responses* x_i. *d=?*
Output/label space	*e.g.*	$\in \{c_k\}$? $\mathbb{R}$?	*Categorical, continuous?*
Feature response selector		ϕ	*Features can be e.g. wavelets? Pixel intensities? Context?*

Forest model

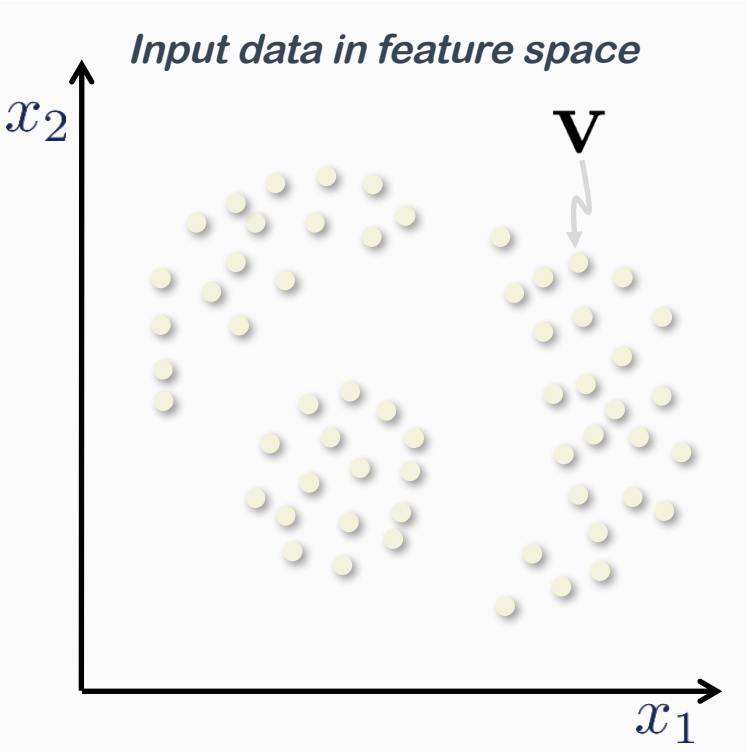
tree

Node test parameters		$\boldsymbol{\theta} \in \mathcal{T}$	*Parameters related to each split node: i) which features, ii) what geometric primitive, iii) thresholds.*
Node objective function (train.)	*e.g.*	$I = I(\mathcal{S}_j, \boldsymbol{\theta})$	*The "energy" to be minimized when training the j-th split node*
Node weak learner	*e.g.*	$h(\mathbf{v}, \boldsymbol{\theta}_j) \in \{\texttt{true}, \texttt{false}\}$	*The test function for splitting data at a node j.*
Leaf predictor model	*e.g.*	$p(c\|\mathbf{v})$	*Point estimate? Full distribution?*
Randomness model (train.)	*e.g.*	*1. Bagging, 2. Randomized node optimization*	*How is randomness injected during training? How much?*
Stopping criteria (train.)	*e.g.*	*max tree depth =* D	*When to stop growing a tree during training*

ensemble

Forest size		T	*Total number of trees in the forest*
The ensemble model	*e.g.*	$p(c\|\mathbf{v}) = \frac{1}{T}\sum_t^T p_t(c\|\mathbf{v})$	*How to compute the forest output from that of individual trees?*

Decision tree testing (runtime)

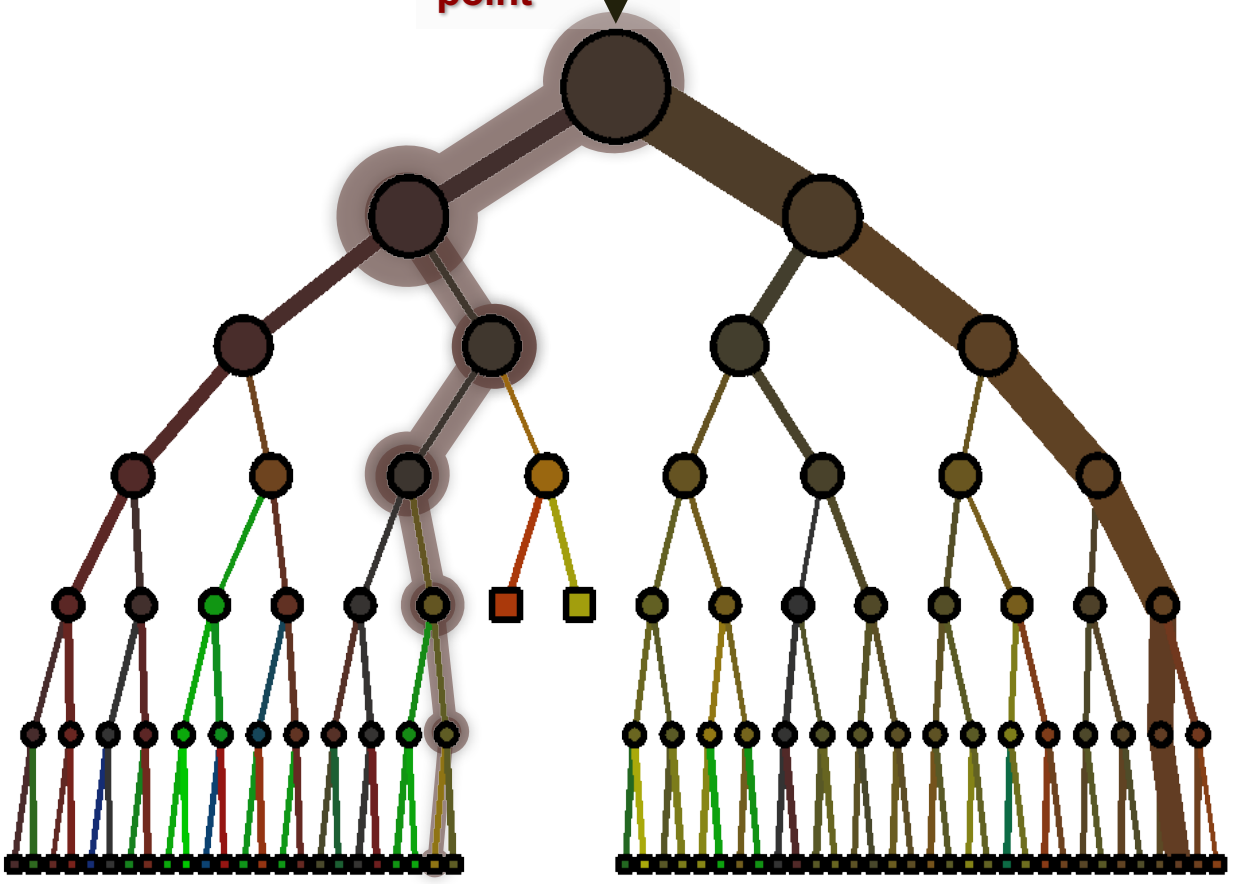


$$\mathbf{v} = (x_1, \cdots, x_d) \in \mathbb{R}^d$$

Input test point $\mathbf{v}$

Split the data at node

$$h(\mathbf{v}, \boldsymbol{\theta}) \in \{\texttt{true}, \texttt{false}\}$$

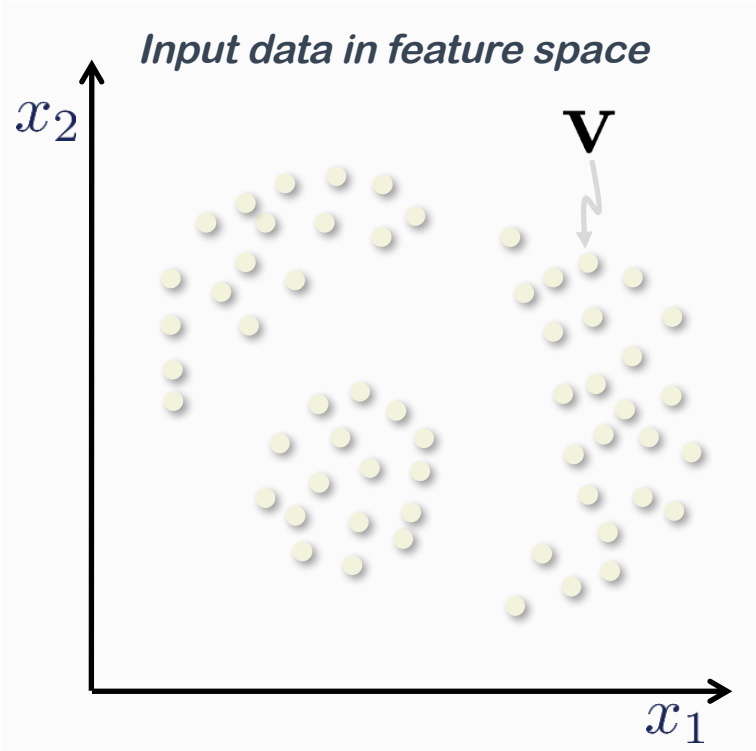


Prediction at leaf

$$p(c|\mathbf{v}) = \sum_j p(c|j)p(j|\mathbf{v})$$

taken from: A. Criminisi, J. Shotton and E. Konukoglu

Decision tree training (off-line)



$$\mathbf{v} = (x_1, \cdots, x_d) \in \mathbb{R}^d$$

Input training data

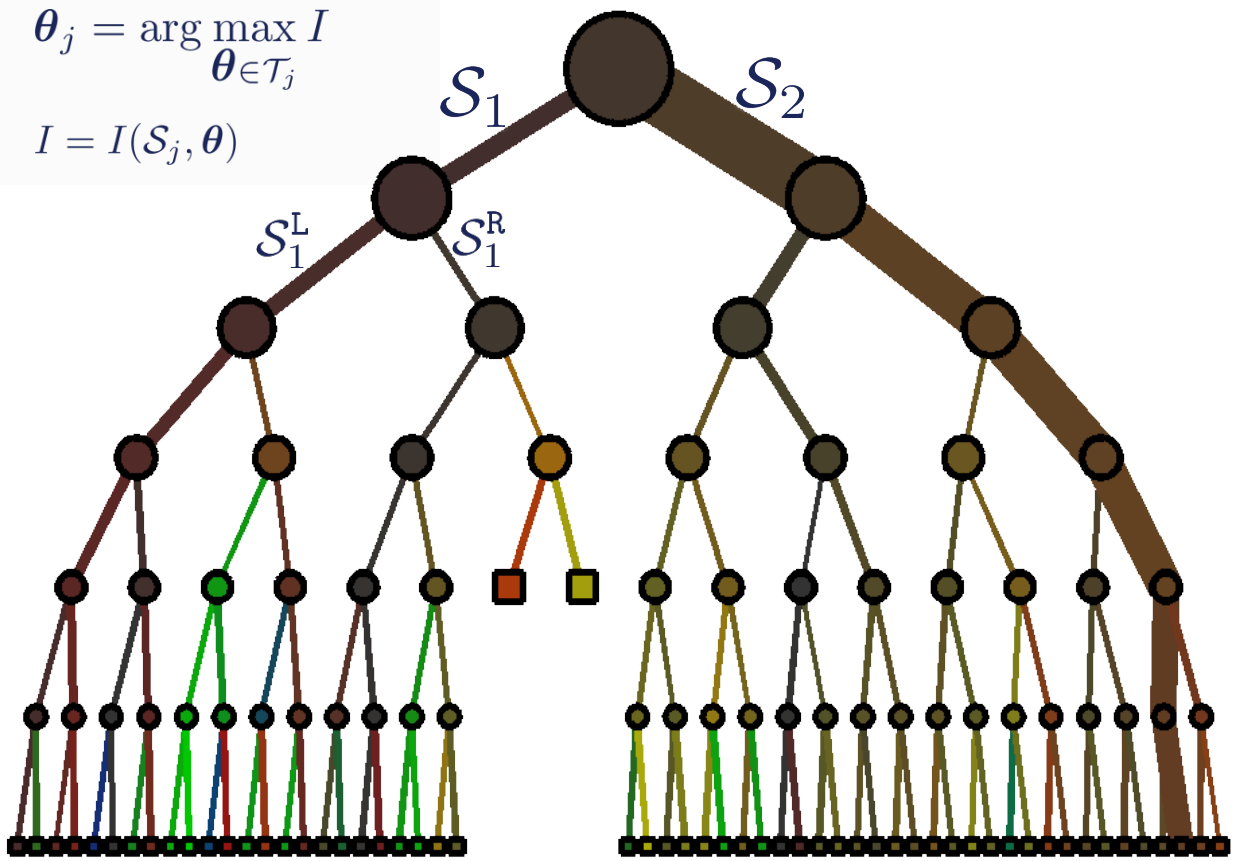
$$\mathcal{S}_0 = \{\mathbf{v}, c\}$$

How to split the data?

$$h(\mathbf{v}, \boldsymbol{\theta}) \in \{\texttt{true}, \texttt{false}\}$$

$$\boldsymbol{\theta}_j = \arg\max_{\boldsymbol{\theta} \in \mathcal{T}_j} I$$

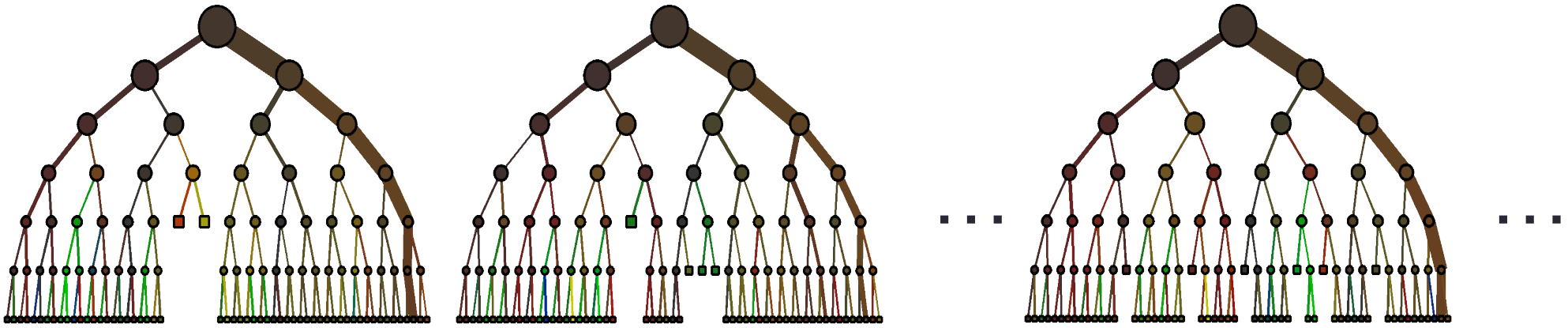
$$I = I(\mathcal{S}_j, \boldsymbol{\theta})$$



Binary tree? Ternary?
How big a tree?
What tree structure?

taken from: A. Criminisi, J. Shotton and E. Konukoglu

Decision forest training (off-line)



How many trees?
How different?
How to fuse their outputs?

taken from: A. Criminisi, J. Shotton and E. Konukoglu

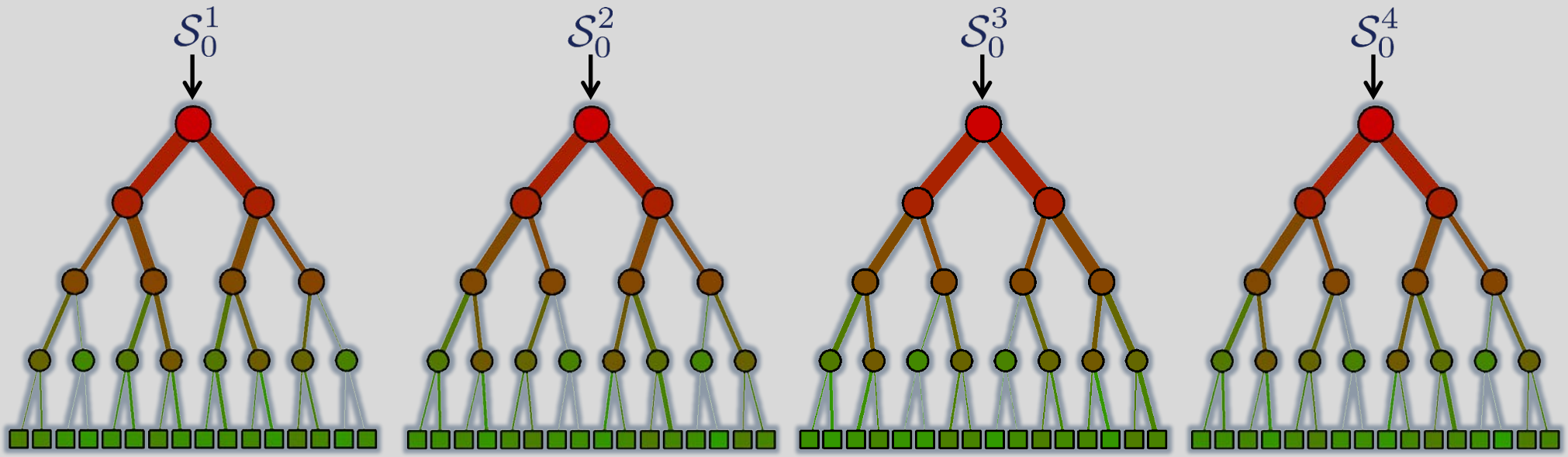
Decision forest model: the randomness model

1) Bagging (randomizing the training set)

$\mathcal{S}_0$ The full training set

$\mathcal{S}_0^t \subset \mathcal{S}_0$ The randomly sampled subset of training data made available for the tree *t*

Forest training

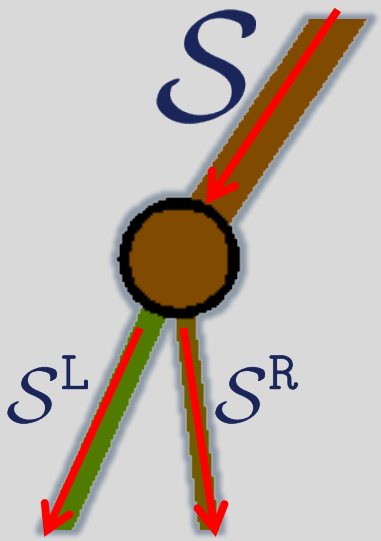


taken from: A. Criminisi, J. Shotton and E. Konukoglu

Efficient training

Decision forest model: training and information gain

(for categorical, non-parametric distributions)



Information gain

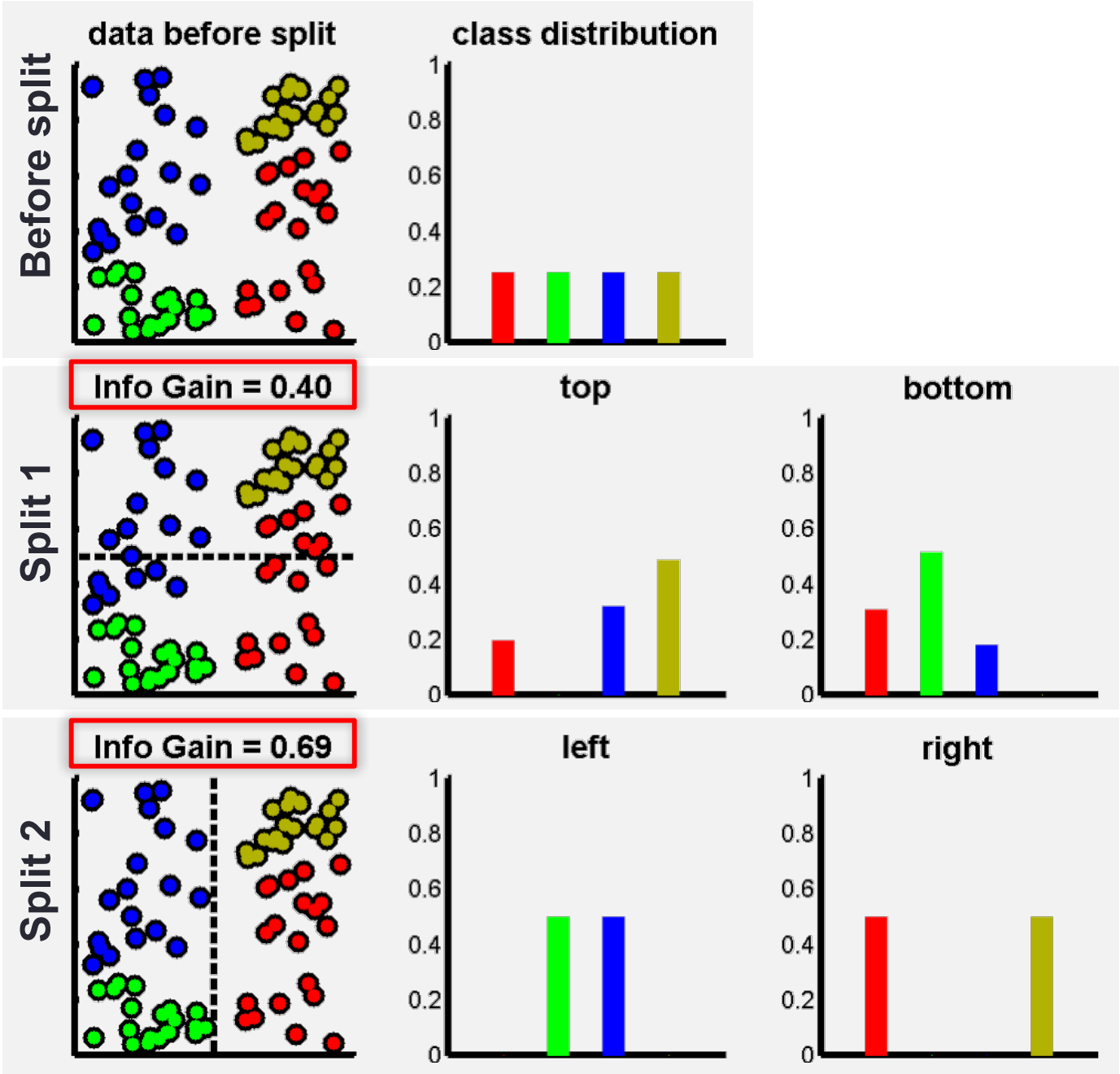
$$I(\mathcal{S},\boldsymbol{\theta}) = H(\mathcal{S}) - \sum_{i\in\{\mathrm{L},\mathrm{R}\}} \frac{|\mathcal{S}^i|}{|\mathcal{S}|} H(\mathcal{S}^i)$$

Shannon's entropy

$$H(\mathcal{S}) = -\sum_{c\in\mathcal{C}} p(c)\log(p(c))$$

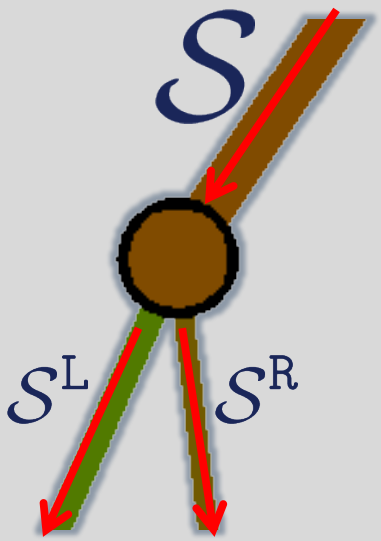
Node training

$$\boldsymbol{\theta} = \arg\max_{\boldsymbol{\theta}\in\mathcal{T}_j} I(\mathcal{S}_j,\boldsymbol{\theta})$$



Decision forest model: training and information gain

(for continuous, parametric densities)



Information gain

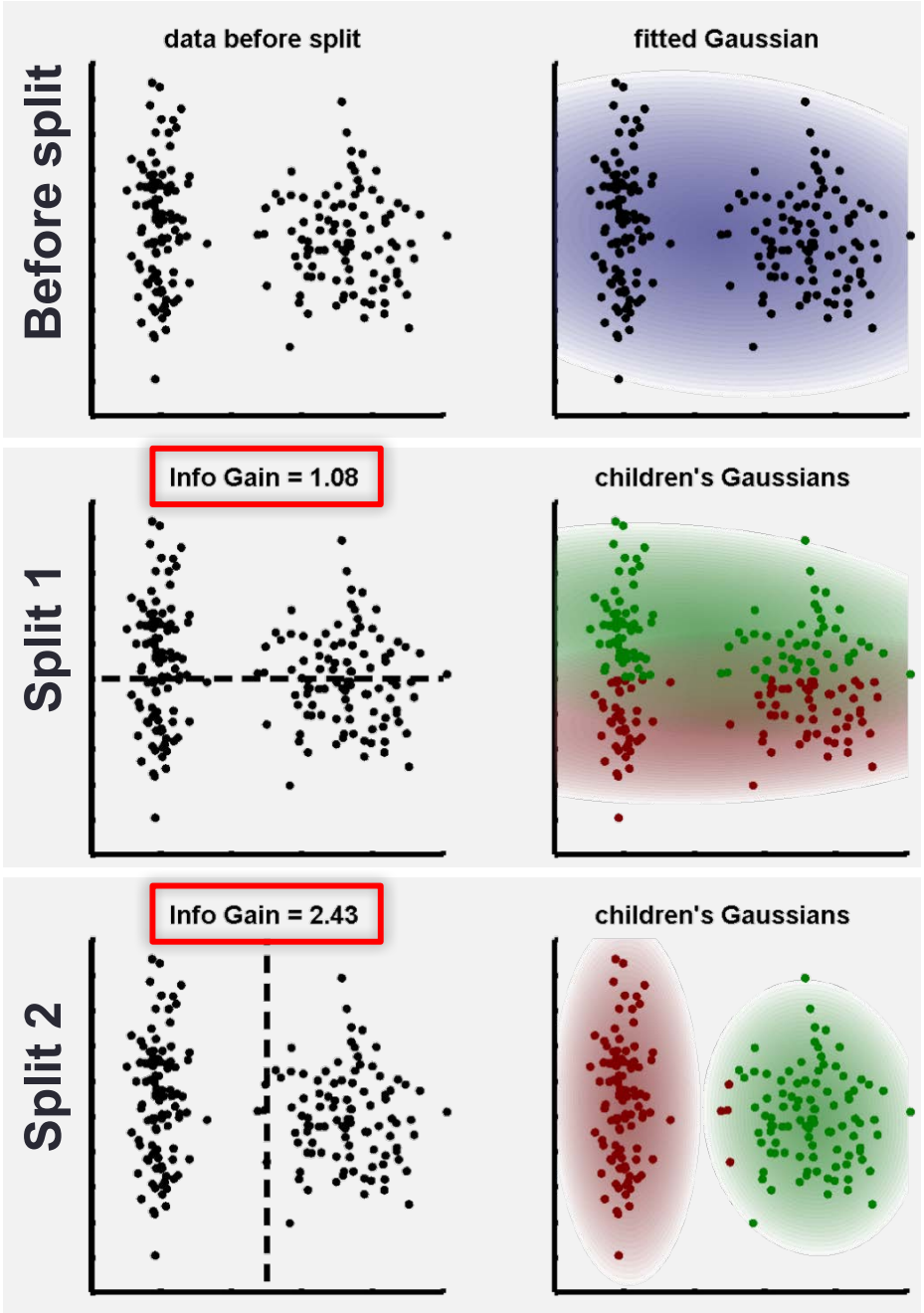
$$I(\mathcal{S},\boldsymbol{\theta}) = H(\mathcal{S}) - \sum_{i\in\{\mathrm{L},\mathrm{R}\}} \frac{|\mathcal{S}^i|}{|\mathcal{S}|} H(\mathcal{S}^i)$$

Differential entropy of Gaussian

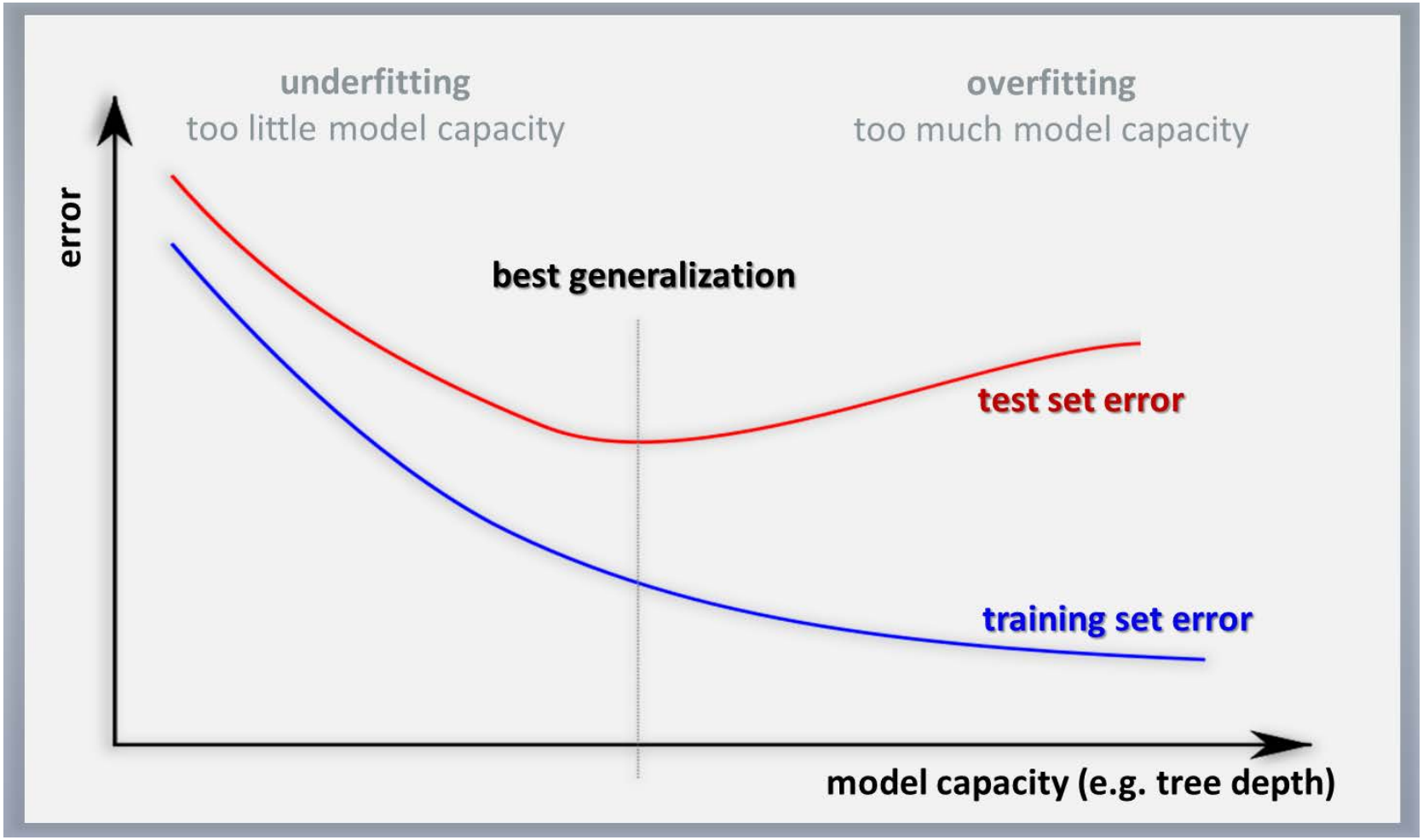
$$H(\mathcal{S}) = \frac{1}{2}\log\left((2\pi e)^d |\Lambda(\mathcal{S})|\right)$$

Node training

$$\boldsymbol{\theta} = \arg\max_{\boldsymbol{\theta}\in\mathcal{T}_j} I(\mathcal{S}_j,\boldsymbol{\theta})$$

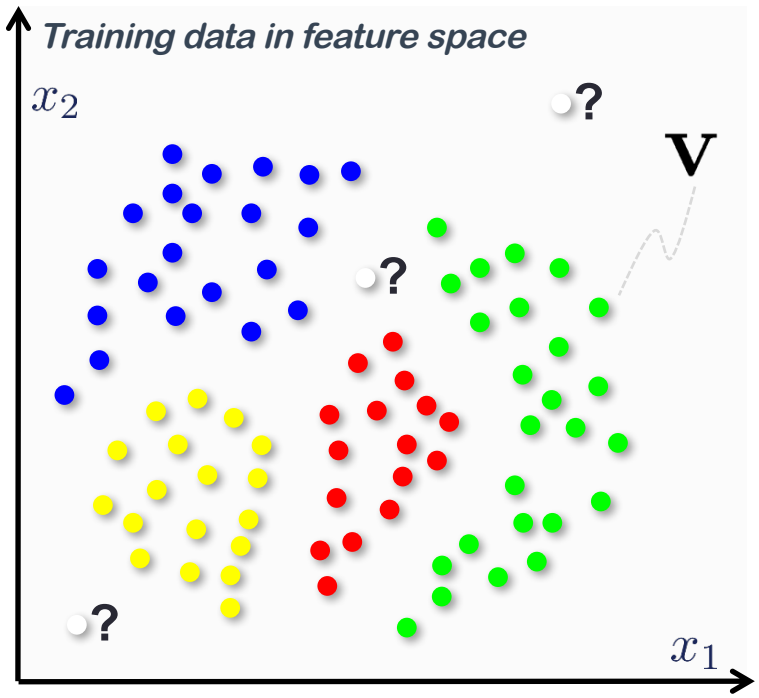


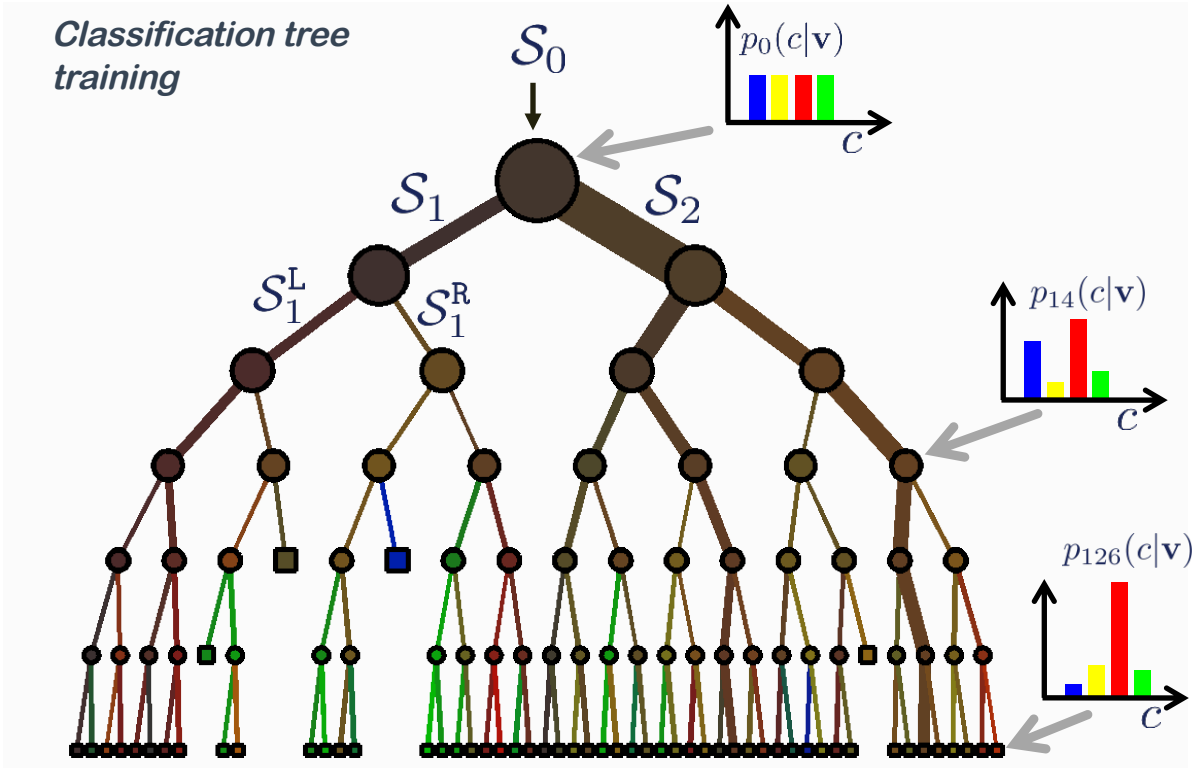
Background: overfitting and underfitting



taken from: A. Criminisi, J. Shotton and E. Konukoglu

Classification forest





Model specialization for classification

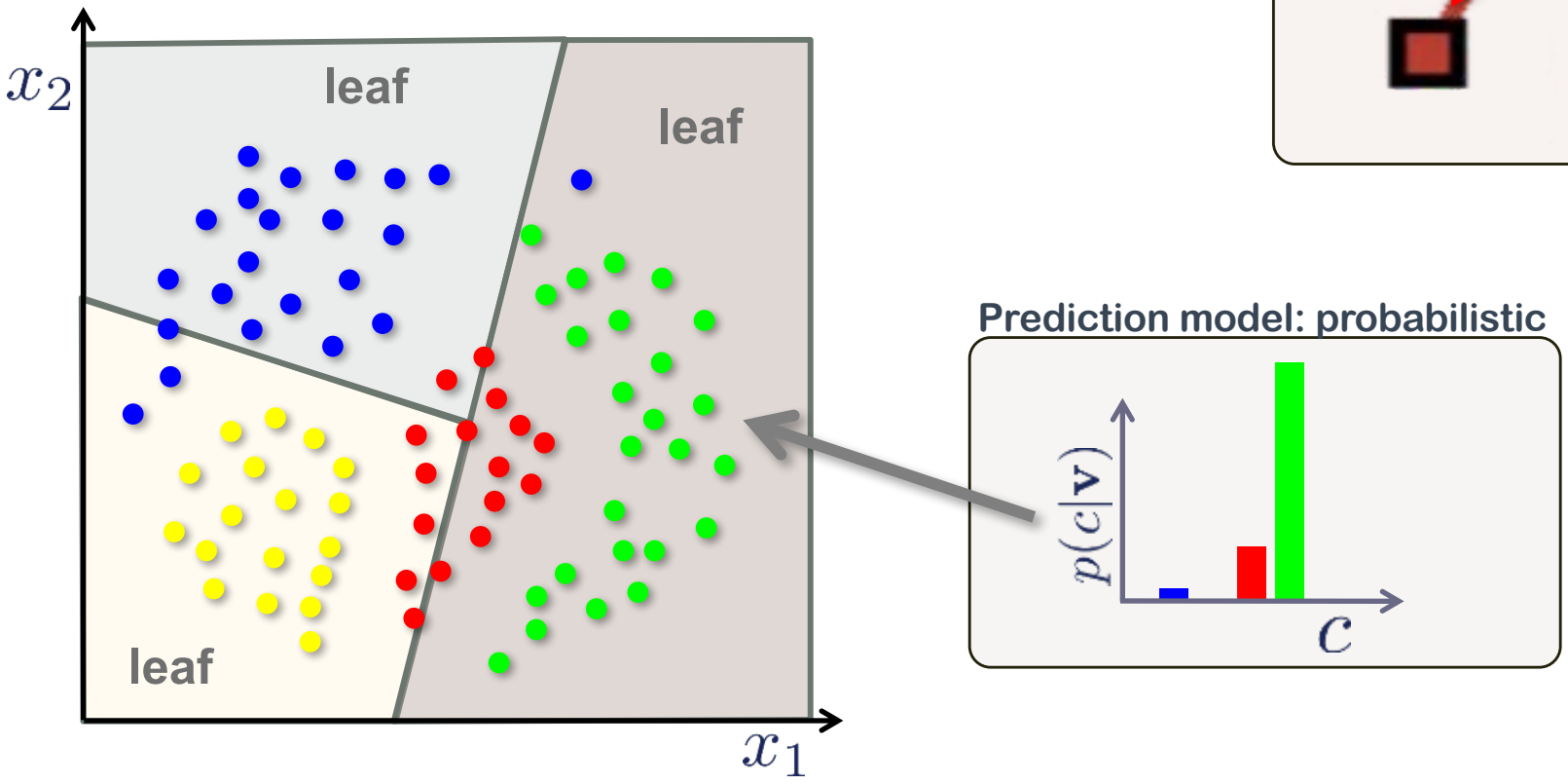
Input data point	$\mathbf{v} = (x_1, \cdots, x_d) \in \mathbb{R}^d$	(x_i is feature response)
Output is categorical	$c \in \mathcal{C}$ with $\mathcal{C} = \{c_k\}$	(discrete set)
Node weak learner	$h(\mathbf{v}, \boldsymbol{\theta}) \in \{\mathtt{true}, \mathtt{false}\}$	
Obj. funct. for node j	$I = H(\mathcal{S}_j) - \sum_{i=\mathrm{L,R}} \frac{\lvert\mathcal{S}_j^i\rvert}{\lvert\mathcal{S}_j\rvert} H(\mathcal{S}_j^i)$	(information gain)
Training node j	$\boldsymbol{\theta}_j = \arg\max_{\boldsymbol{\theta} \in \mathcal{T}_j} I(\mathcal{S}_j, \boldsymbol{\theta})$	
Predictor model	$p(c\|\mathbf{v}) = \sum_j p(c\|j)p(j\|\mathbf{v})$	(class posterior)

Entropy of a discrete distribution

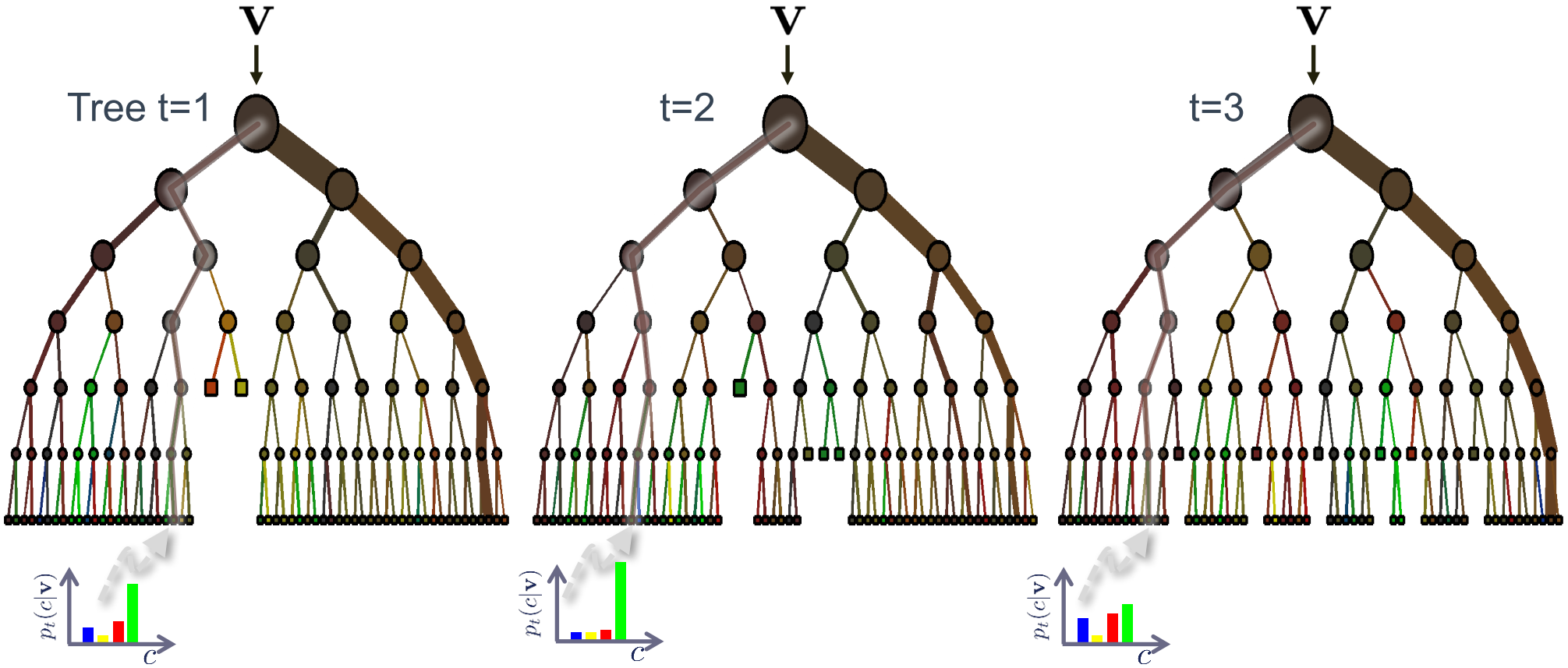
$$H(\mathcal{S}) = -\sum_{c \in \mathcal{C}} p(c) \log(p(c))$$

with $c(\mathbf{v}) : \mathbb{R}^d \rightarrow \mathcal{C}$

Classification forest: the prediction model

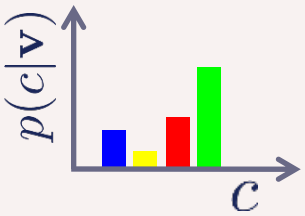


Classification forest: the ensemble model



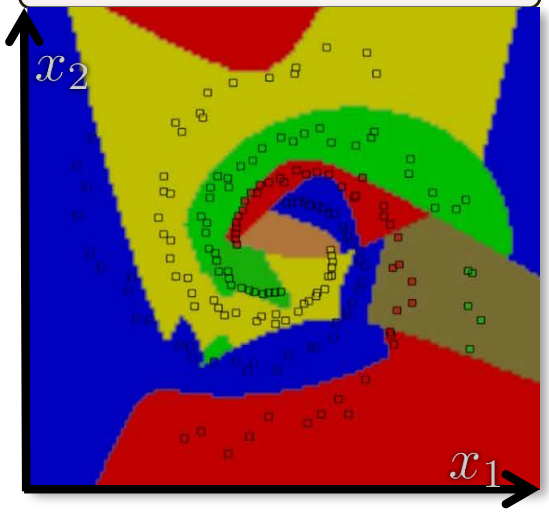
The ensemble model

Forest output probability $$p(c|\mathbf{v}) = \frac{1}{T}\sum_{t}^{T} p_t(c|\mathbf{v})$$

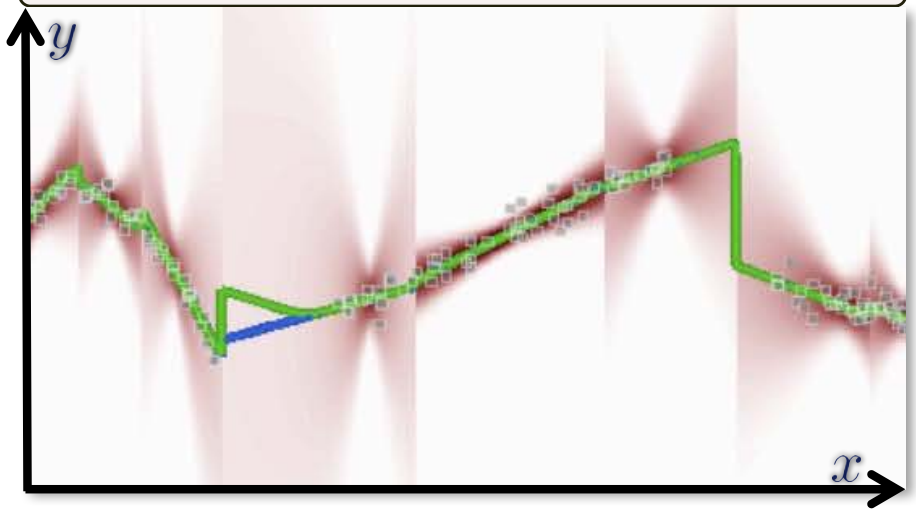


What can decision forests do? tasks

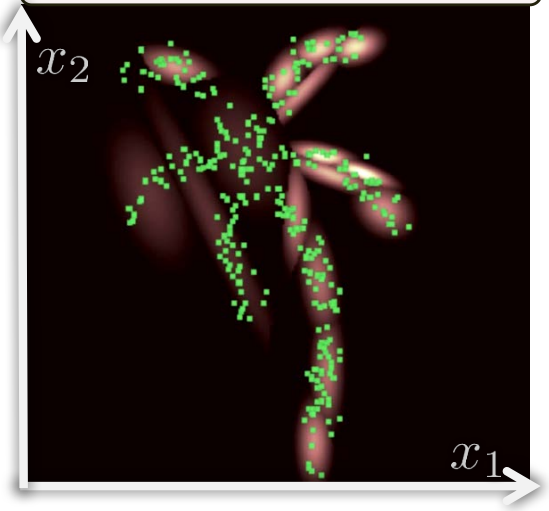
Classification forests



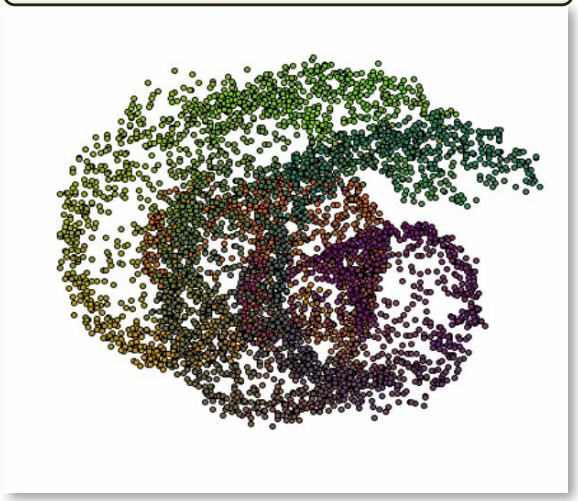
Regression forests



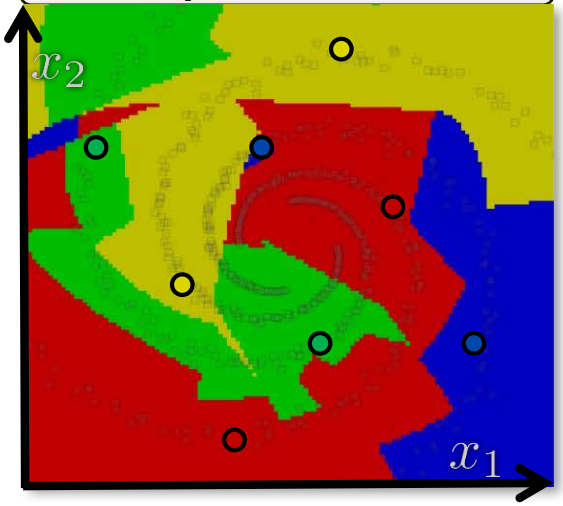
Density forests



Manifold forests



Semi-supervised forests



What can decision forests do? applications

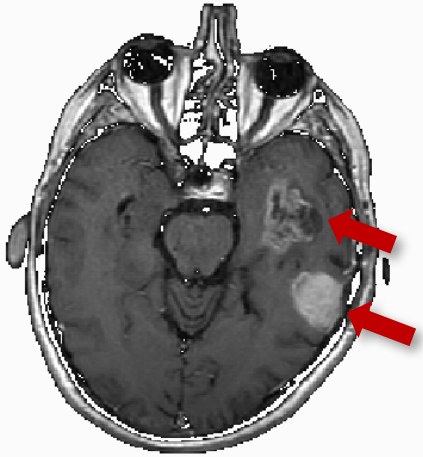
Classification forests

e.g. semantic segmentation

Regression forests

e.g. object localization

Density forests



e.g. novelty detection

Manifold forests

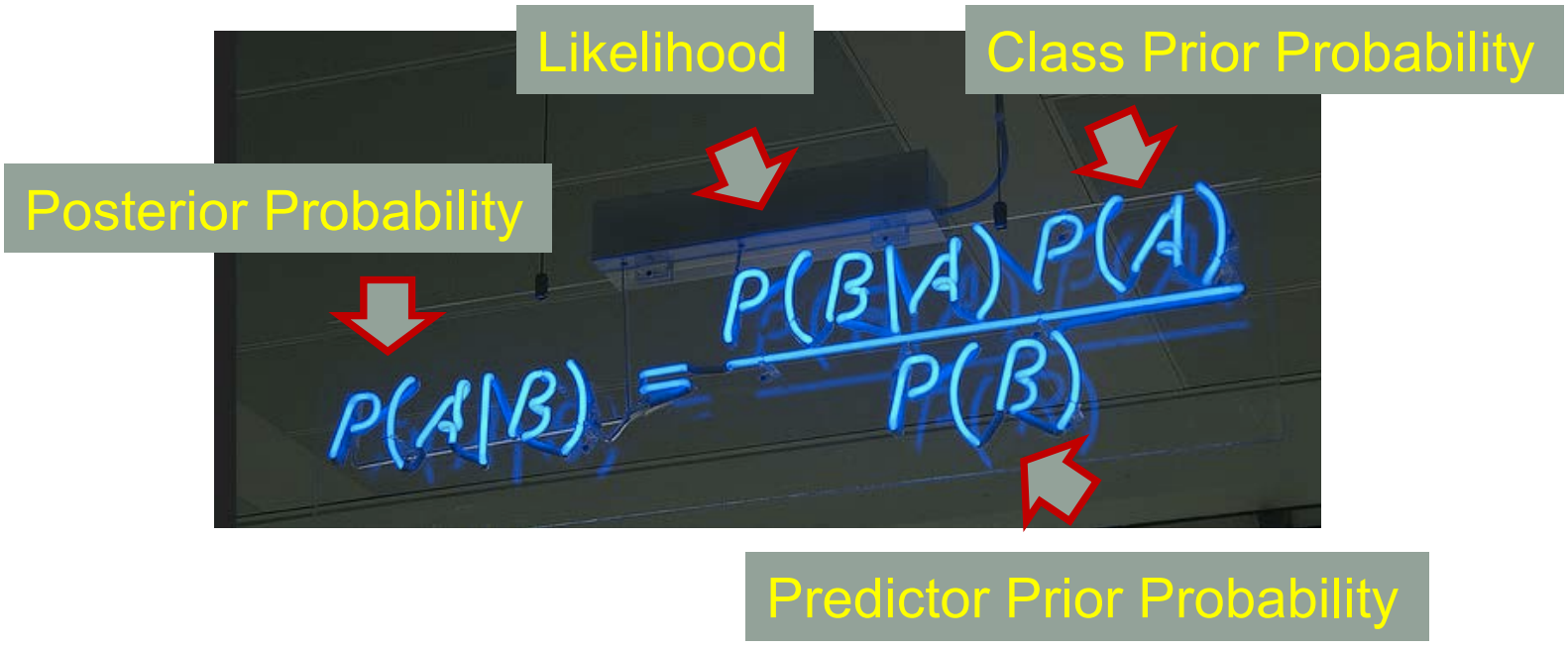
e.g. dimensionality reduction

Semi-supervised forests

e.g. semi-sup. semantic segmentation

Naïve Bayes

- A family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features.



Naïve Bayes

Naive Bayes is a conditional probability model.

Given a problem instance to be classified, represented by a vector $\mathbf{x} = (x_1, \ldots, x_n)$ representing some n features (independent variables), it assigns to this instance probabilities

$$p(C_k|x_1, \ldots, x_n)$$

for each of *K* possible outcomes or *classes* C_k

Naïve Bayes

When n is large or $\mathbf{x} = (x_1, \ldots, x_n)$ can take on a large number of values, then basing this model

$$p(C_k | x_1, \ldots, x_n)$$

on probability tables is infeasible.

Making the model more tractable using Bayes' theorem, the conditional probability can be decomposed as

$$p(C_k|\mathbf{x}) = \frac{p(C_k)p(\mathbf{x}|C_k)}{p(\mathbf{x})} \qquad \text{posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}}$$

effectively constant, does not depend on C_k

Naïve Bayes

The numerator is equivalent to the joint probability model

$$p(C_k, x_1, \ldots, x_n)$$

It can be rewritten as follows, using the chain rule for repeated applications of the definition of conditional probability:

$$\begin{aligned}
p(C_k, x_1, \ldots, x_n) &= p(x_1, \ldots, x_n, C_k) \\
&= p(x_1 \mid x_2, \ldots, x_n, C_k) p(x_2, \ldots, x_n, C_k) \\
&= p(x_1 \mid x_2, \ldots, x_n, C_k) p(x_2 \mid x_3, \ldots, x_n, C_k) p(x_3, \ldots, x_n, C_k) \\
&= \ldots \\
&= p(x_1 \mid x_2, \ldots, x_n, C_k) p(x_2 \mid x_3, \ldots, x_n, C_k) \ldots p(x_{n-1} \mid x_n, C_k) p(x_n \mid C_k) p(C_k)
\end{aligned}$$

Naïve Bayes

``Naive" conditional independence assumptions:

Assume that each feature x_i is conditionally independent of every other feature x_j for $i \neq j$ given the category C

This means that: $$p(x_i \mid x_{i+1}, \ldots, x_n, C_k) = p(x_i \mid C_k)$$

Thus

$$\begin{aligned} p(C_k \mid x_1, \ldots, x_n) &\propto p(C_k, x_1, \ldots, x_n) \\ &\propto p(C_k)\, p(x_1 \mid C_k)\, p(x_2 \mid C_k)\, p(x_3 \mid C_k) \cdots \\ &\propto p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)\,. \end{aligned}$$

Naïve Bayes

This means that under the above independence assumptions, the conditional distribution over the class variable C is: $p(C_k \mid x_1, \ldots, x_n) = \frac{1}{Z} p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)$

where the evidence Z is a scaling factor dependent only on $x_1, x_2, \ldots, x_n$

that is, a constant if the values of the feature variables are known.

Maximum a Posteriori (MAP)

The naive Bayes classifier combines independent feature model model with a decision rule. One common rule is to pick the hypothesis that is most probable; this is known as the *maximum a posteriori* or *MAP* decision rule. The (corresponding) Bayes classifier, is the function that assigns a class label $\hat{y} = C_k$ for some k as follows:

$$\hat{y} = \underset{k \in \{1,\ldots,K\}}{\operatorname{argmax}} \, p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k).$$

Parameter estimation and event models

A class's prior may be calculated by assuming equiprobable classes

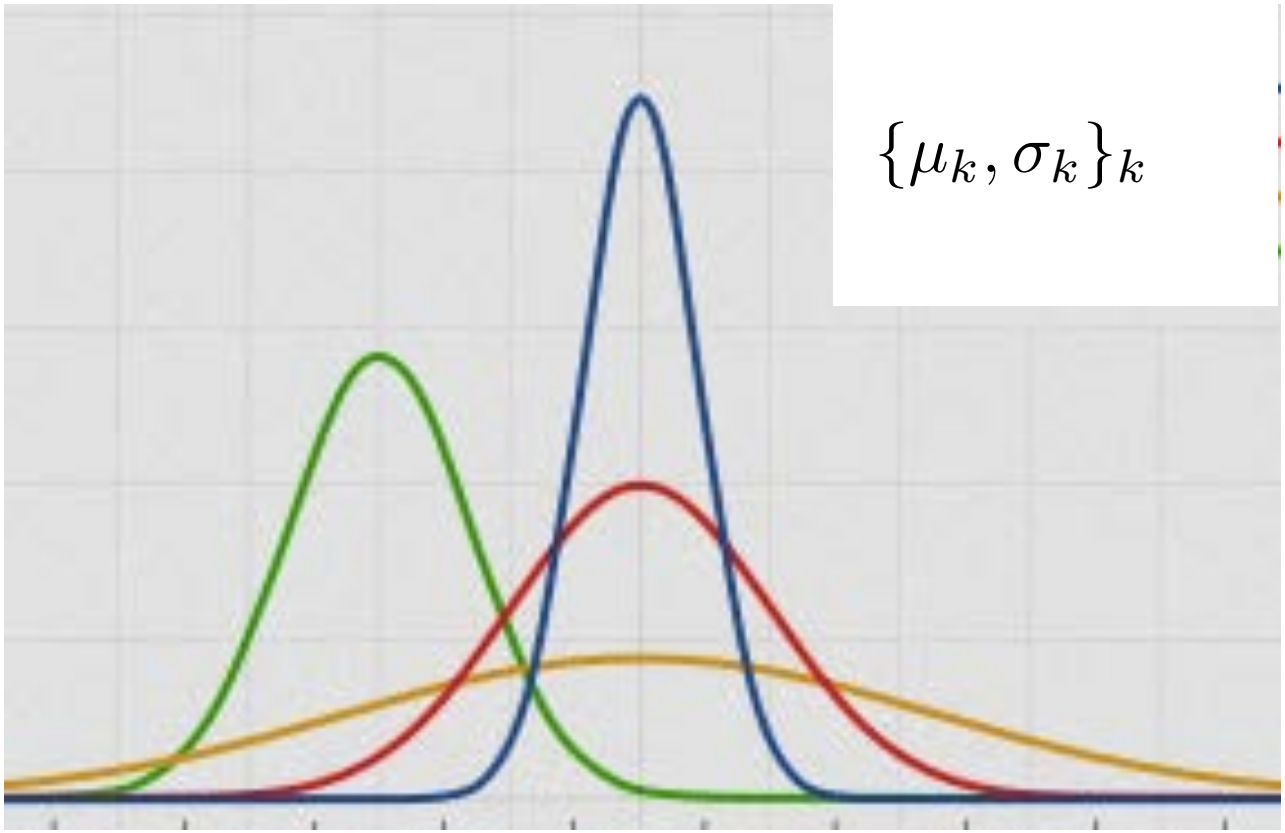
$$\text{prior} = \frac{1}{\text{number of classes}}$$

or by calculating an estimate for the class probability from the training set

$$\text{prior for } C_k = \frac{\text{number of samples in } C_k}{\text{number of samples in } \sum_{k=1}^{K} C_k}$$

To estimate the parameters for a feature's distribution, one must assume a distribution or generate nonparametric models for the features from the training set.

Example: Gaussian naive Bayes



$\{\mu_k, \sigma_k\}_k$

likelihood term

$$p(x = v \mid C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{-\frac{(v-\mu_k)^2}{2\sigma_k^2}}$$

Using Naïve Bayes

- Simple thing to try for categorical data
- Very fast to train/test

Naïve Bayes and Logistic Regression

In the case of discrete inputs (indicator or frequency features for discrete events), naive Bayes classifiers form a *generative-discriminative* pair with (multinomial) logistic regression classifiers: each naive Bayes classifier can be considered a way of fitting a probability model that optimizes the joint likelihood $p(C, \mathbf{x})$ while logistic regression fits the same probability model to optimize the conditional $p(C|\mathbf{x})$

Naïve Bayes and Logistic Regression

The decision function for naive Bayes (in the binary case) can be rewritten as "predict class C_1 if the odds of $p(C_1|\mathbf{x})$ exceed those of $p(C_2|\mathbf{x})$ ":

$$\log \frac{p(C_1 \mid \mathbf{x})}{p(C_2 \mid \mathbf{x})} = \log p(C_1 \mid \mathbf{x}) - \log p(C_2 \mid \mathbf{x}) > 0$$

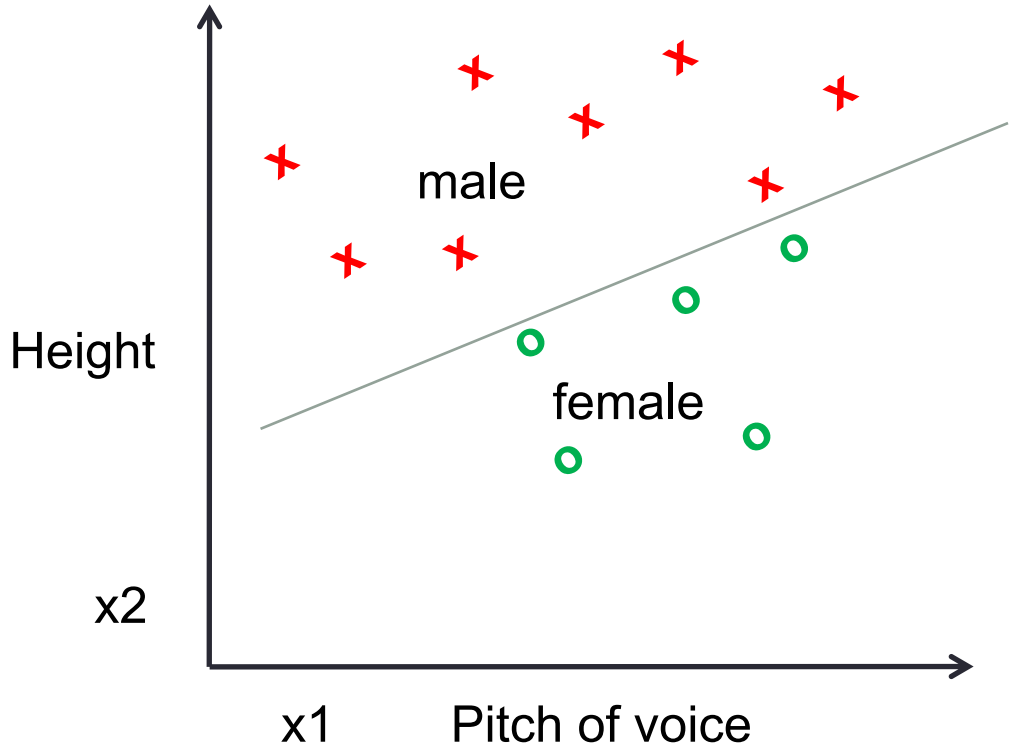
Logit function

Since naive Bayes is also a linear model for the two "discrete" event models, it can be reparametrised as a linear function

$$y = b + \mathbf{w}^T \mathbf{x}$$

Classifiers: Logistic Regression

Maximize likelihood of label given data, assuming a log-linear model



$$\log \frac{P(x_1, x_2 \mid y = 1)}{P(x_1, x_2 \mid y = -1)} = \mathbf{w}^T \mathbf{x}$$

$$P(y = 1 \mid x_1, x_2) = 1 / \left(1 + \exp\left(-\mathbf{w}^T \mathbf{x}\right)\right)$$

Using Logistic Regression

- Quick, simple classifier (try it first)
- Outputs a probabilistic label confidence
- Use L2 or L1 regularization
  - L1 does feature selection and is robust to irrelevant features but slower to train

Multinomial logistic regression function – SoftMax function

Ideals for a classification algorithm

- Objective function: encodes the right loss for the problem
- Parameterization: takes advantage of the structure of the problem
- Regularization: good priors on the parameters
- Training algorithm: can find parameters that maximize objective on training set
- Inference algorithm: can solve for labels that maximize objective function for a test example

Two ways to think about classifiers

1. What is the objective? What are the parameters? How are the parameters learned? How is the learning regularized? How is inference performed?

2. How is the data modeled? How is similarity defined? What is the shape of the boundary?

Slide credit: D. Hoiem

Comparison

assuming x in {0 1}

	Learning Objective	Training	Inference
Naïve Bayes	$\text{maximize} \sum_i \left[\sum_j \log P(x_{ij} \mid y_i; \theta_j) + \log P(y_i; \theta_0) \right]$	$\theta_{kj} = \frac{\sum_i \delta(x_{ij} = 1 \wedge y_i = k) + r}{\sum_i \delta(y_i = k) + Kr}$	$\boldsymbol{\theta}_1^T \mathbf{x} + \boldsymbol{\theta}_0^T (1 - \mathbf{x}) > 0$ where $\theta_{1j} = \log \frac{P(x_j = 1 \mid y = 1)}{P(x_j = 1 \mid y = 0)}$, $\theta_{0j} = \log \frac{P(x_j = 0 \mid y = 1)}{P(x_j = 0 \mid y = 0)}$
Logistic Regression	$\text{maximize} \sum_i \log(P(y_i \mid \mathbf{x}, \boldsymbol{\theta})) + \lambda \lVert \boldsymbol{\theta} \rVert$ where $P(y_i \mid \mathbf{x}, \boldsymbol{\theta}) = 1 / (1 + \exp(-y_i \boldsymbol{\theta}^T \mathbf{x}))$	Gradient ascent	$\boldsymbol{\theta}^T \mathbf{x} > 0$
Linear SVM	$\text{minimize } \lambda \sum_i \xi_i + \frac{1}{2} \lVert \boldsymbol{\theta} \rVert$ such that $y_i \boldsymbol{\theta}^T \mathbf{x} \geq 1 - \xi_i \quad \forall i$	Linear programming	$\boldsymbol{\theta}^T \mathbf{x} > 0$
Kernelized SVM	complicated to write	Quadratic programming	$\sum_i y_i \alpha_i K(\hat{\mathbf{x}}_i, \mathbf{x}) > 0$
Nearest Neighbor	most similar features → same label	Record data	y_i where $i = \underset{i}{\text{argmin}} \; K(\hat{\mathbf{x}}_i, \mathbf{x})$

Slide credit: D. Hoiem

What to remember about classifiers

- No free lunch: machine learning algorithms are tools, not dogmas
- Try simple classifiers first
- Better to have smart features and simple classifiers than simple features and smart classifiers
- Use increasingly powerful classifiers with more training data (bias-variance tradeoff)

Slide credit: D. Hoiem

Making decisions about data

- 3 important design decisions:
  1) What data do I use?
  2) How do I represent my data (what feature)?
  3) What classifier / regressor / machine learning tool do I use?
- These are in decreasing order of importance
- Deep learning addresses 2 and 3 simultaneously (and blurs the boundary between them).
- You can take the representation from deep learning and use it with any classifier.

Some Machine Learning References

- General
  - Tom Mitchell, *Machine Learning*, McGraw Hill, 1997
  - Christopher Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, 1995
- Adaboost
  - Friedman, Hastie, and Tibshirani, “Additive logistic regression: a statistical view of boosting”, Annals of Statistics, 2000
- SVMs
  - http://www.support-vector.net/icml-tutorial.pdf

Slide credit: D. Hoiem