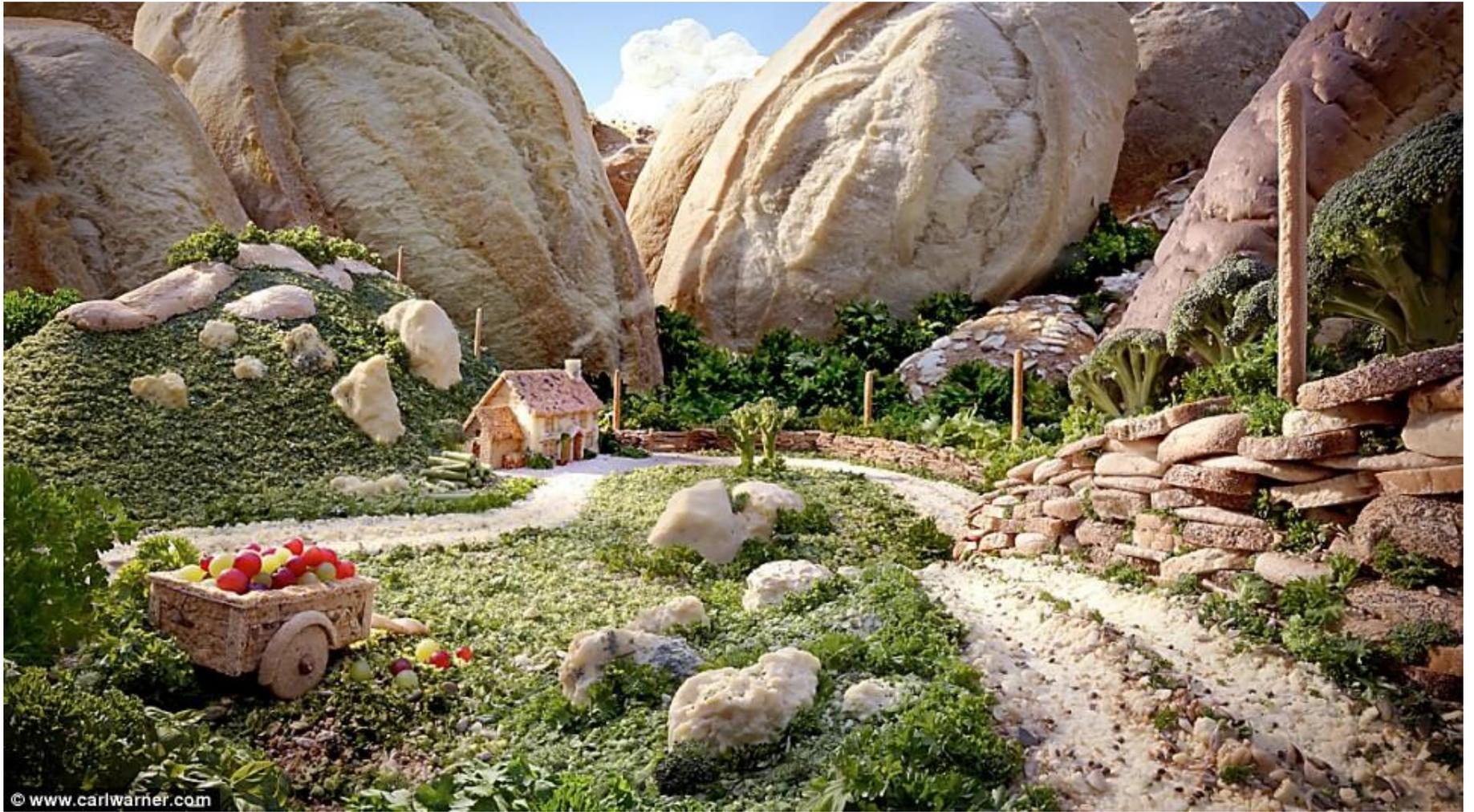




© www.carlwarner.com



© www.carlwarner.com

Photo by Carl Warner



© www.carlwarner.com

Photo by Carl Warner

# Machine Learning: Classification

Computer Vision

CS 143, Brown

James Hays

# The machine learning framework

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

$f(\text{apple image}) = \text{"apple"}$

$f(\text{tomato image}) = \text{"tomato"}$

$f(\text{cow image}) = \text{"cow"}$

# The machine learning framework

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

output      prediction function      Image feature

- **Training:** given a *training set* of labeled examples  $\{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$ , estimate the prediction function  $f$  by minimizing the prediction error on the training set
- **Testing:** apply  $f$  to a never before seen *test example*  $\mathbf{x}$  and output the predicted value  $y = f(\mathbf{x})$

# Steps

## Training

Training Images

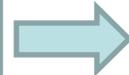
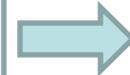


Image Features



Training



Learned model

Training Labels



## Testing



Test Image

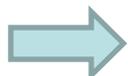


Image Features



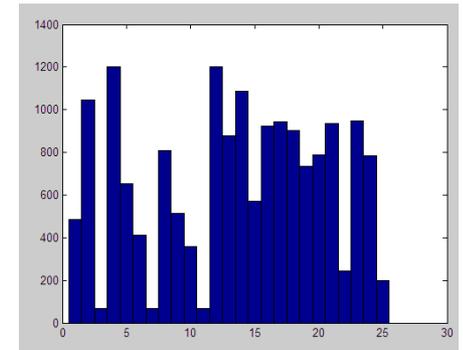
Learned model



Prediction

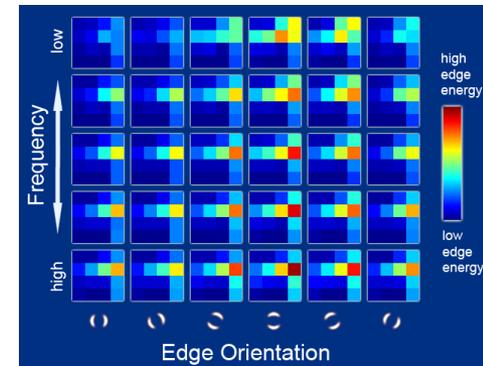
# Features

- Raw pixels



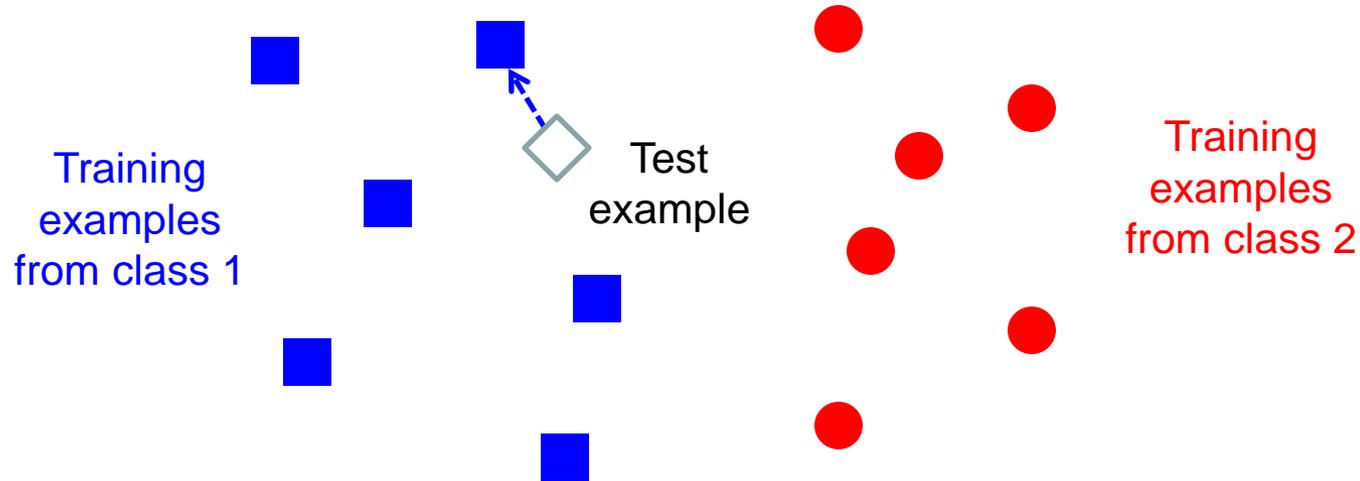
- Histograms

- GIST descriptors



- ...

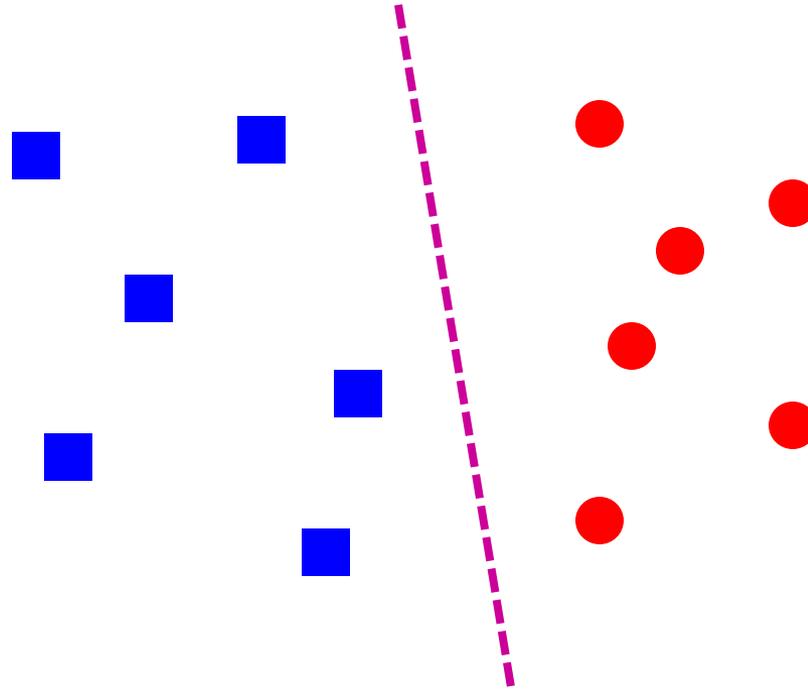
# Classifiers: Nearest neighbor



$f(\mathbf{x}) = \text{label of the training example nearest to } \mathbf{x}$

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

# Classifiers: Linear



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

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

# Many classifiers to choose from

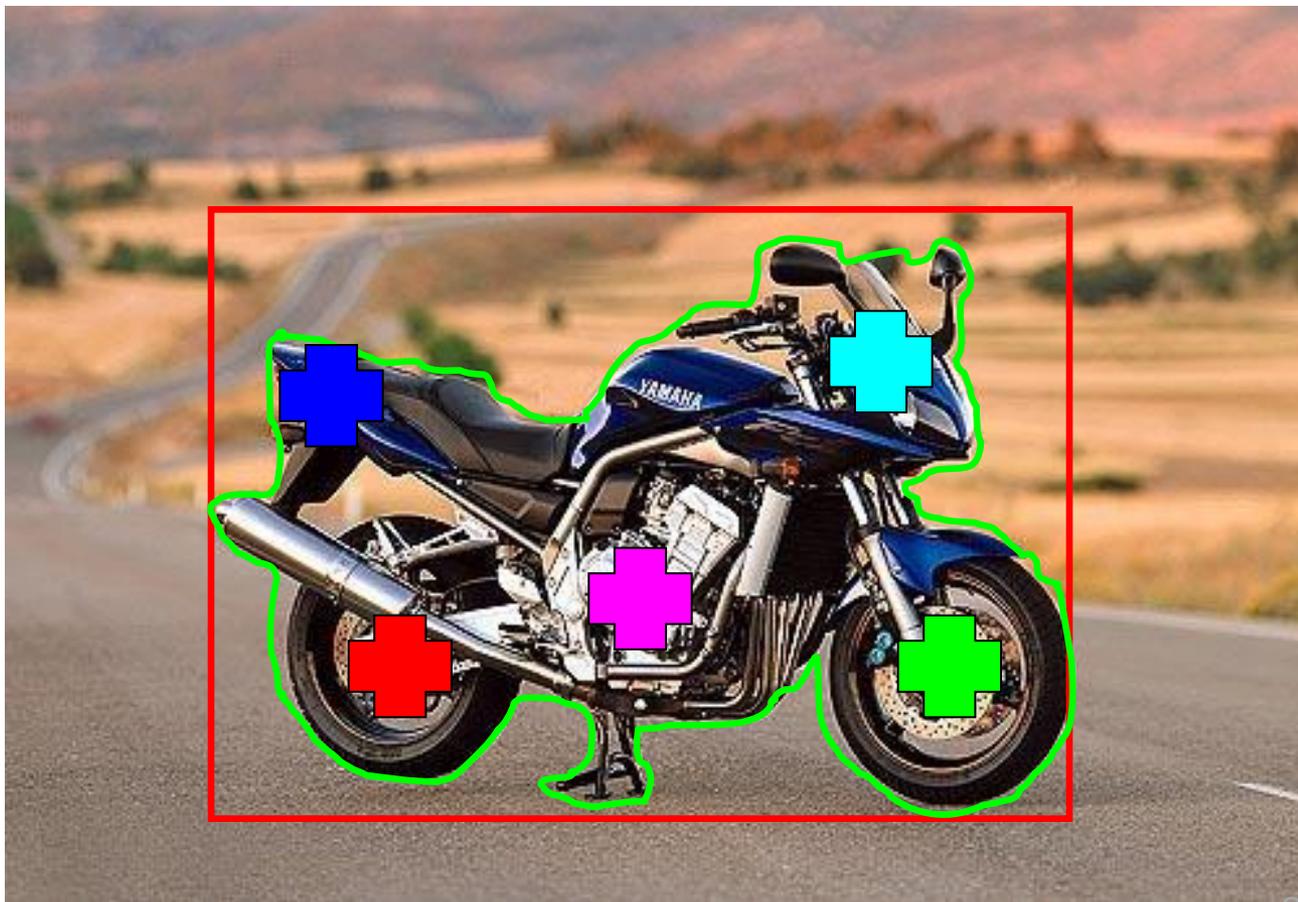
- SVM
- Neural networks
- Naïve Bayes
- Bayesian network
- Logistic regression
- Randomized Forests
- Boosted Decision Trees
- K-nearest neighbor
- RBMs
- Etc.

Which is the best one?

# 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



# Spectrum of supervision

Less

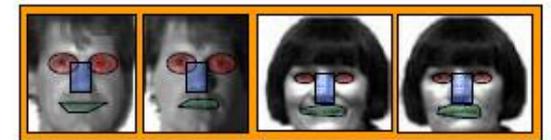
More



Unsupervised



“Weakly” supervised



Fully supervised

Definition depends on task

# Generalization



Training set (labels known)



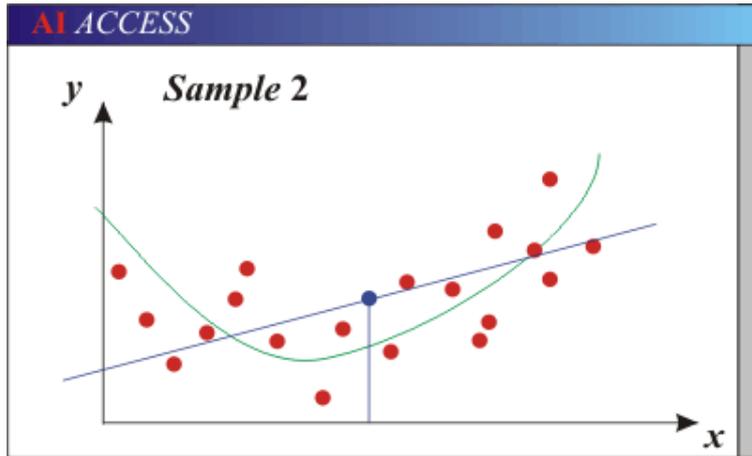
Test set (labels unknown)

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

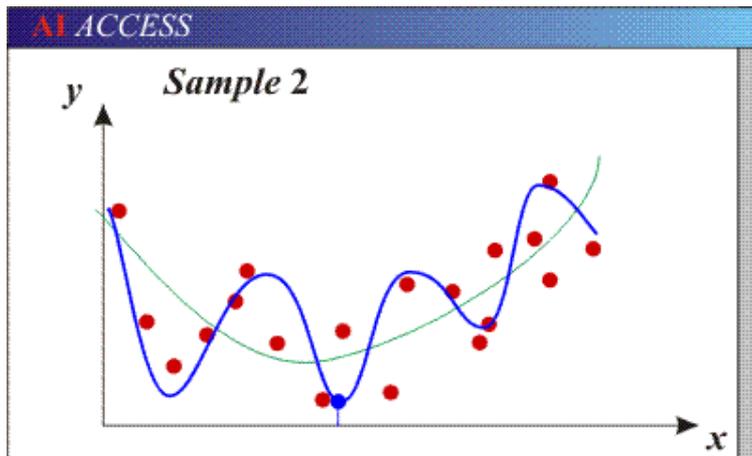
# Generalization

- Components of generalization error
  - **Bias:** how much the average model over all training sets differ 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 and low variance
  - High training error and high test error
- **Overfitting:** model is too “complex” and fits irrelevant characteristics (noise) in the data
  - Low bias and high variance
  - Low training error and high test error

# Bias-Variance Trade-off



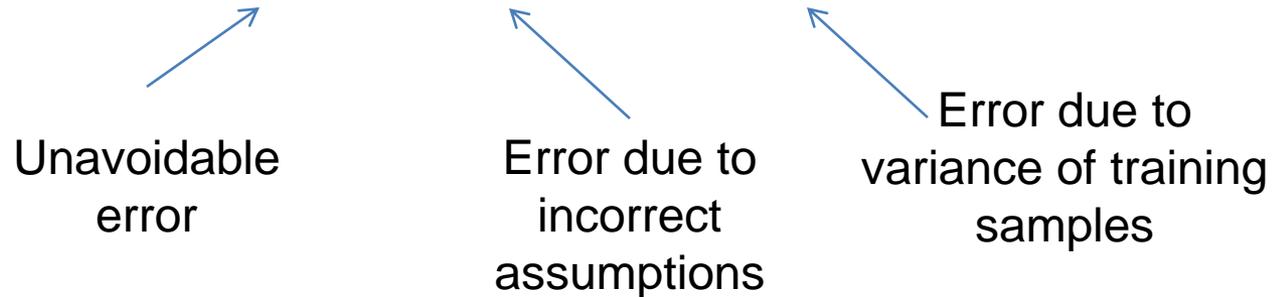
- Models with too few parameters are inaccurate because of a large bias (not enough flexibility).



- Models with too many parameters are inaccurate because of a large variance (too much sensitivity to the sample).

# Bias-Variance Trade-off

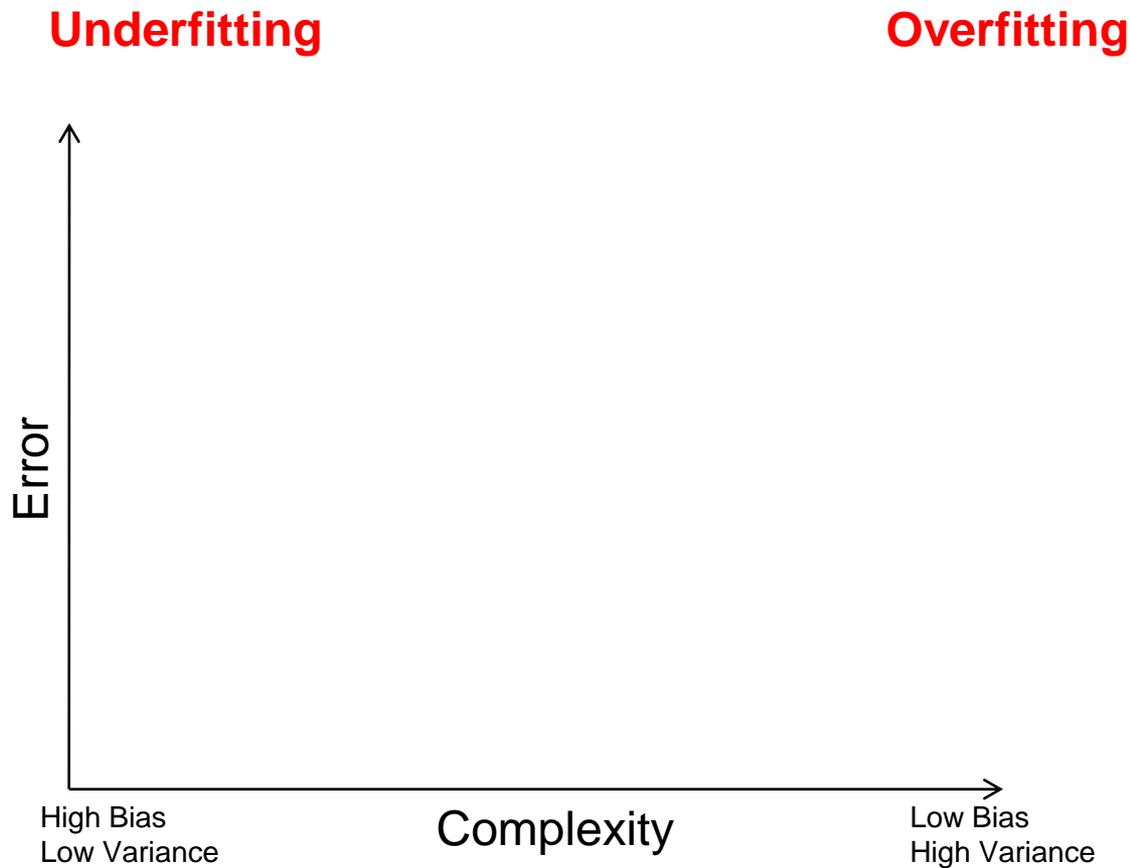
$$E(\text{MSE}) = \text{noise}^2 + \text{bias}^2 + \text{variance}$$



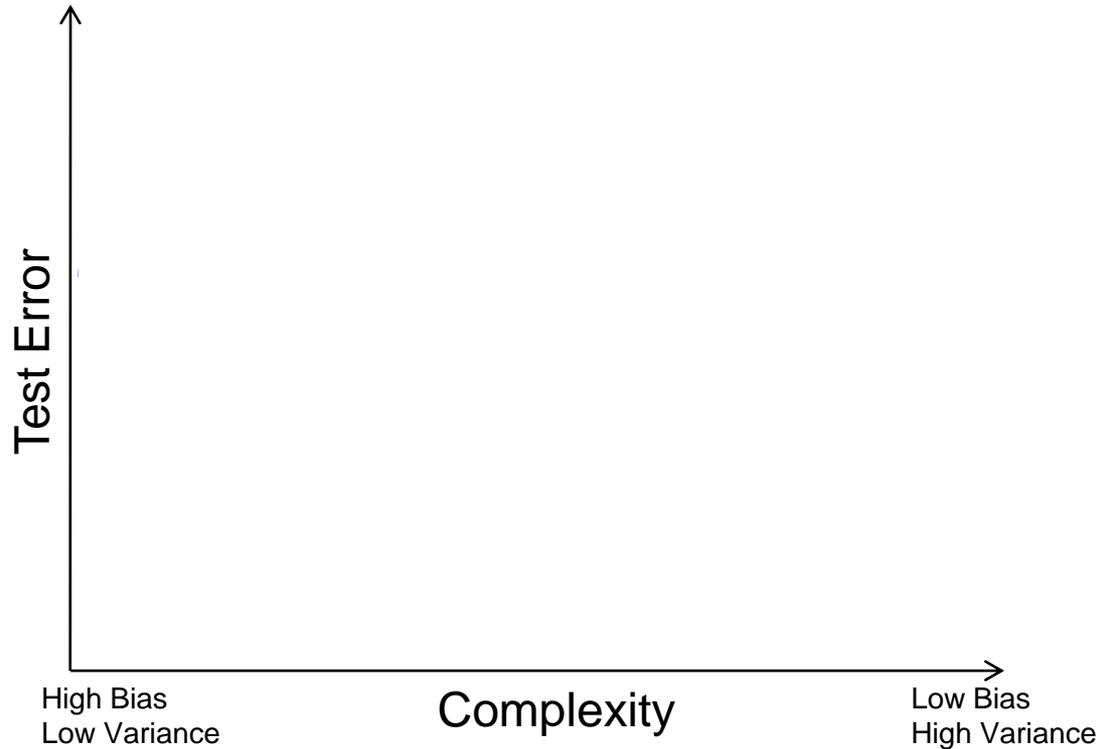
See the following for explanations of bias-variance (also Bishop's "Neural Networks" book):

- <http://www.stat.cmu.edu/~larry/=stat707/notes3.pdf>
- <http://www.inf.ed.ac.uk/teaching/courses/mlsc/Notes/Lecture4/BiasVariance.pdf>

# Bias-variance tradeoff

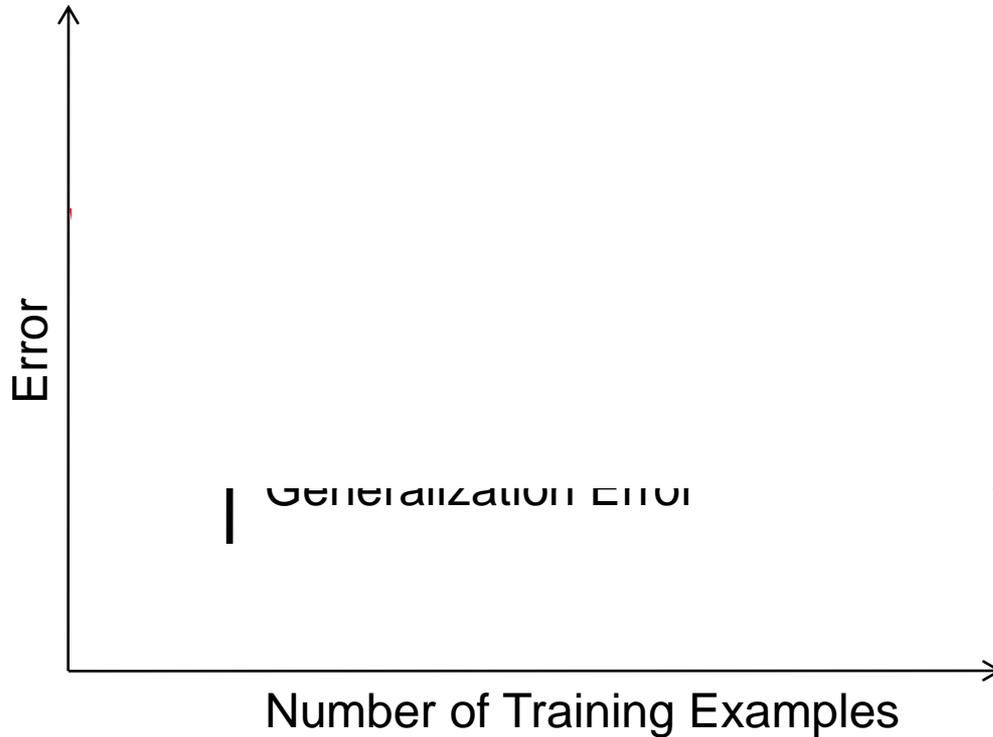


# Bias-variance tradeoff



# Effect of Training Size

Fixed prediction model



# 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



# How to reduce variance?

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

# Very brief tour of some classifiers

- **SVM**
- Neural networks
- **Naïve Bayes**
- Bayesian network
- Logistic regression
- Randomized Forests
- **Boosted Decision Trees**
- **K-nearest neighbor**
- RBMs
- Etc.

# Generative vs. Discriminative Classifiers

## Generative Models

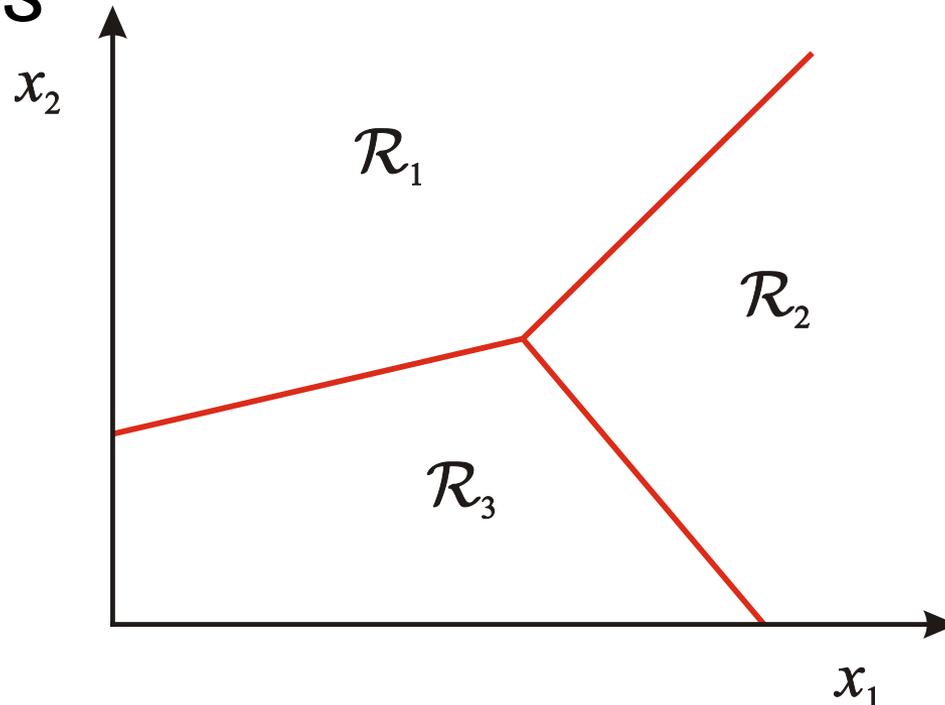
- Represent both the data and the labels
- Often, makes use of conditional independence and priors
- Examples
  - Naïve Bayes classifier
  - Bayesian network
- Models of data may apply to future prediction problems

## 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

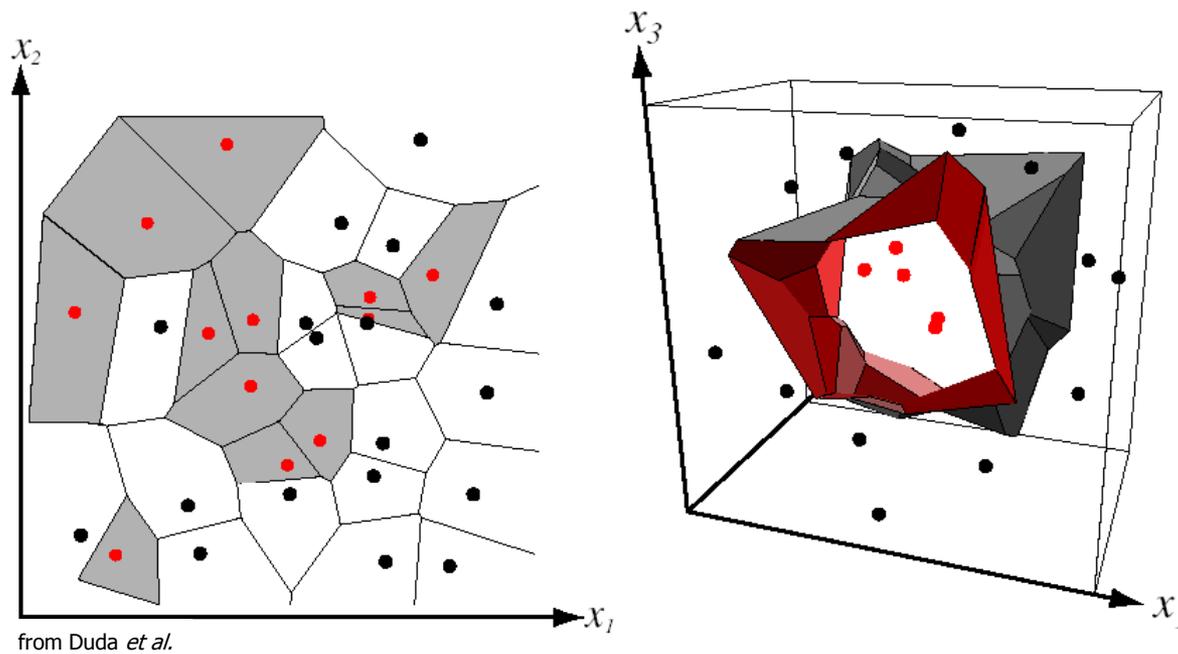
# Classification

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



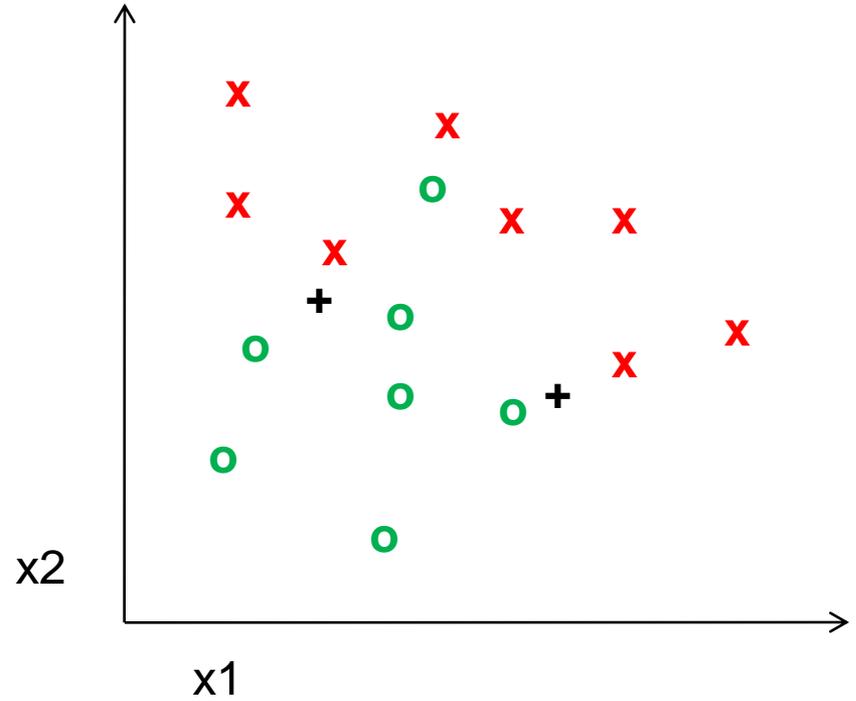
# Nearest Neighbor Classifier

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

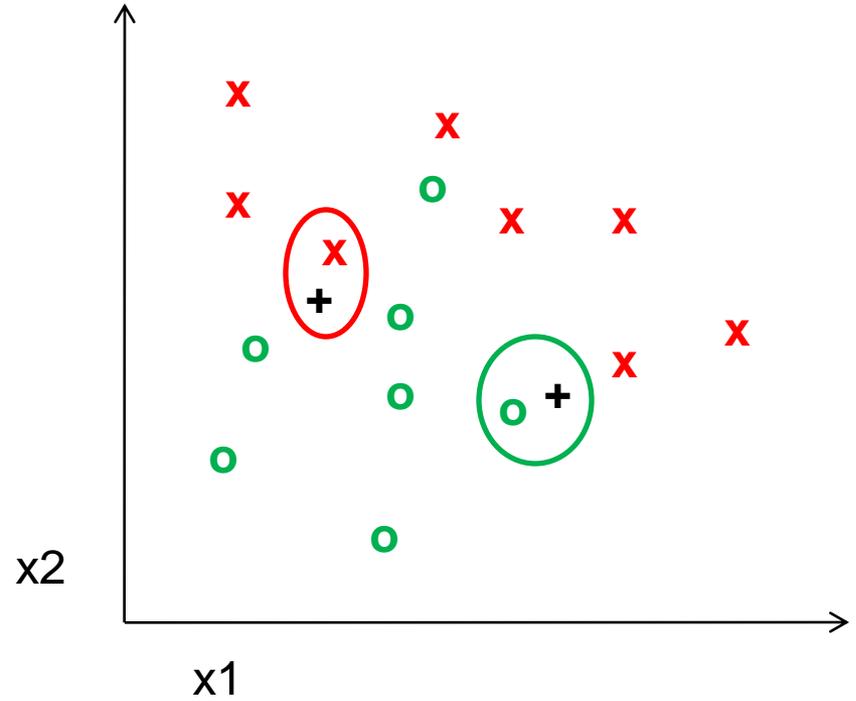


Voronoi partitioning of feature space  
for two-category 2D and 3D data

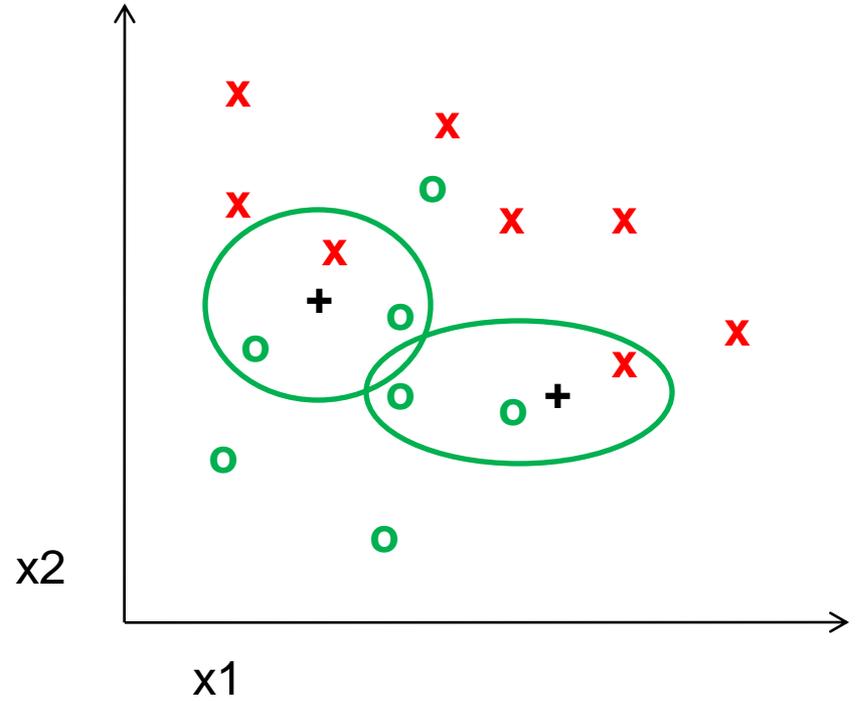
# K-nearest neighbor



# 1-nearest neighbor



# 3-nearest neighbor

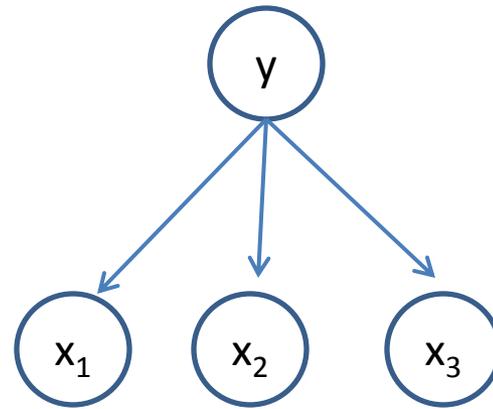




# Using K-NN

- Simple, a good one to try first
- With infinite examples, 1-NN provably has error that is at most twice Bayes optimal error

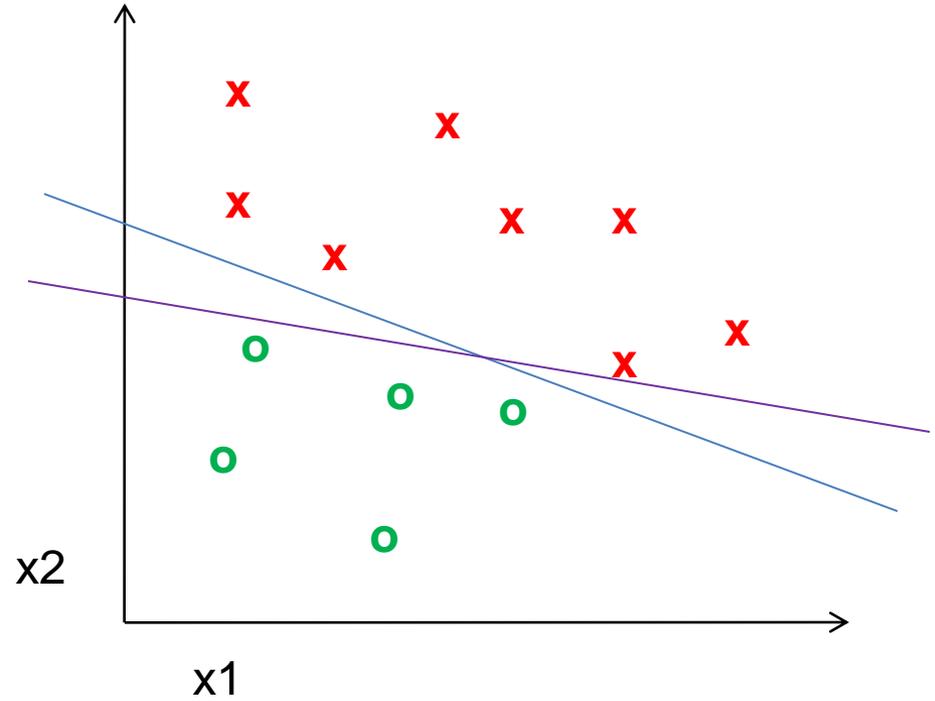
# Naïve Bayes



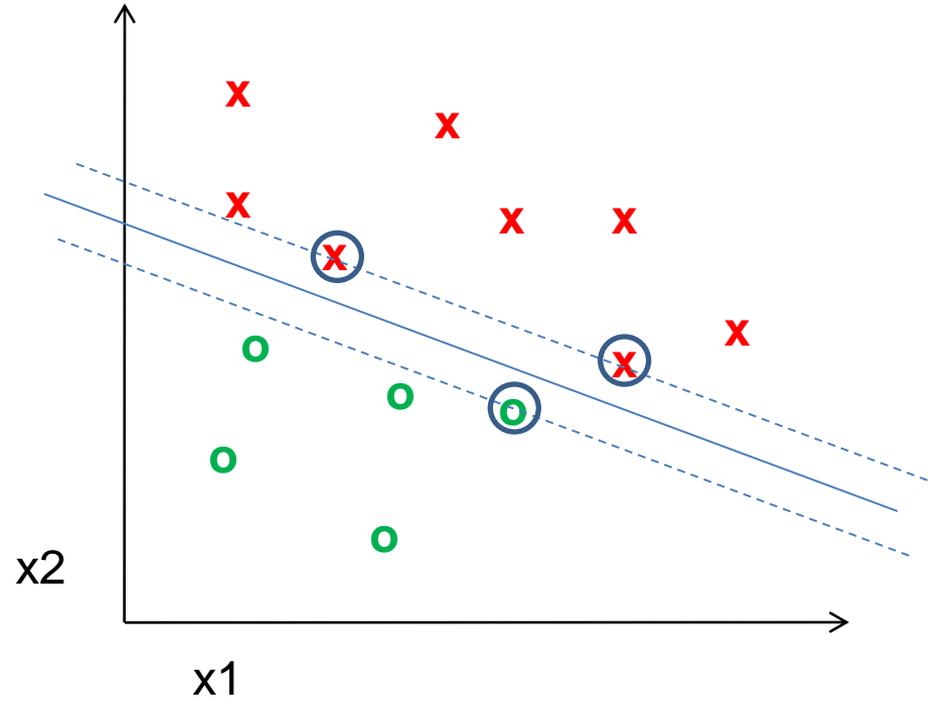
# Using Naïve Bayes

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

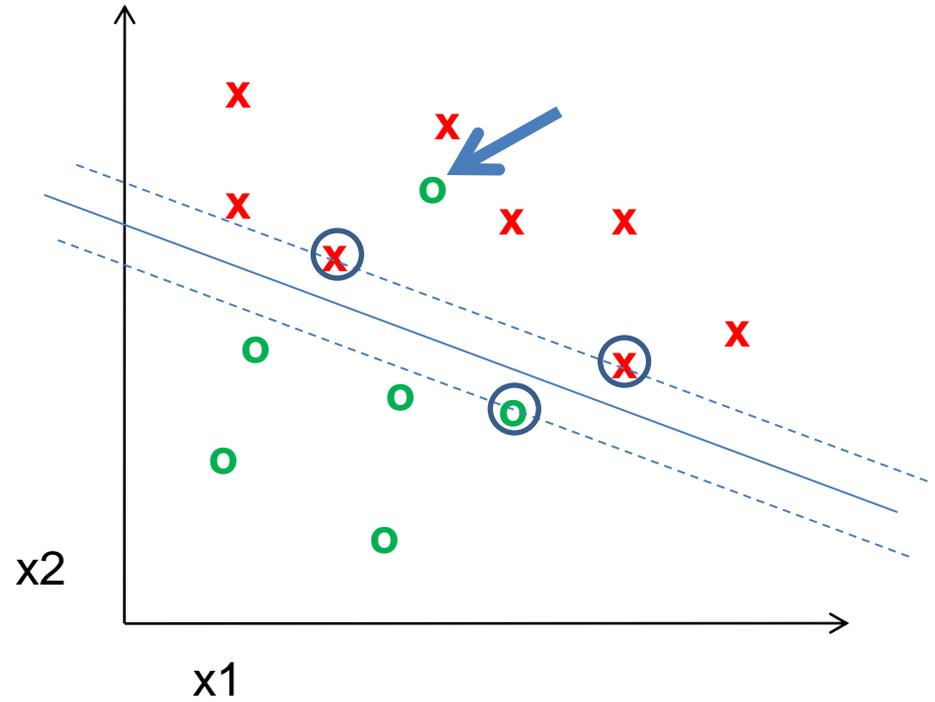
# Classifiers: Linear SVM



# Classifiers: Linear SVM



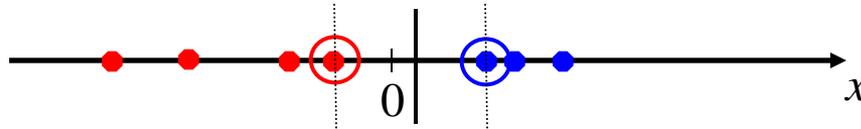
# Classifiers: Linear SVM



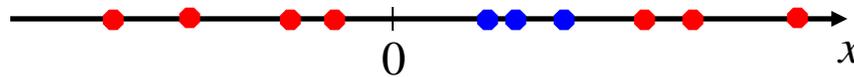
# Nonlinear SVMs

---

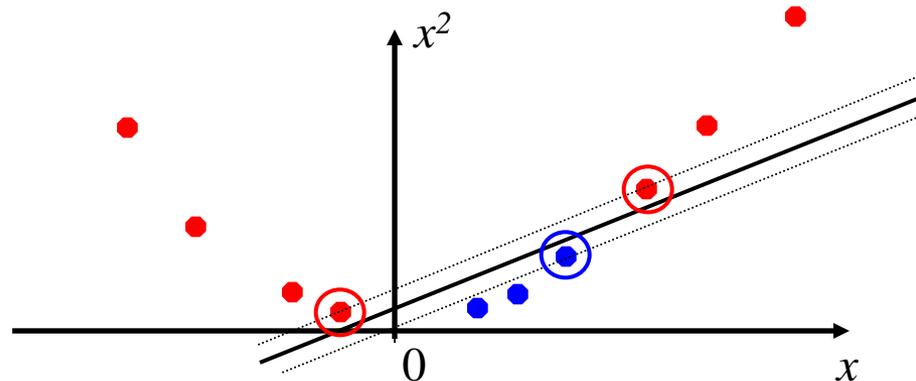
- Datasets that are linearly separable work out great:



- But what if the dataset is just too hard?



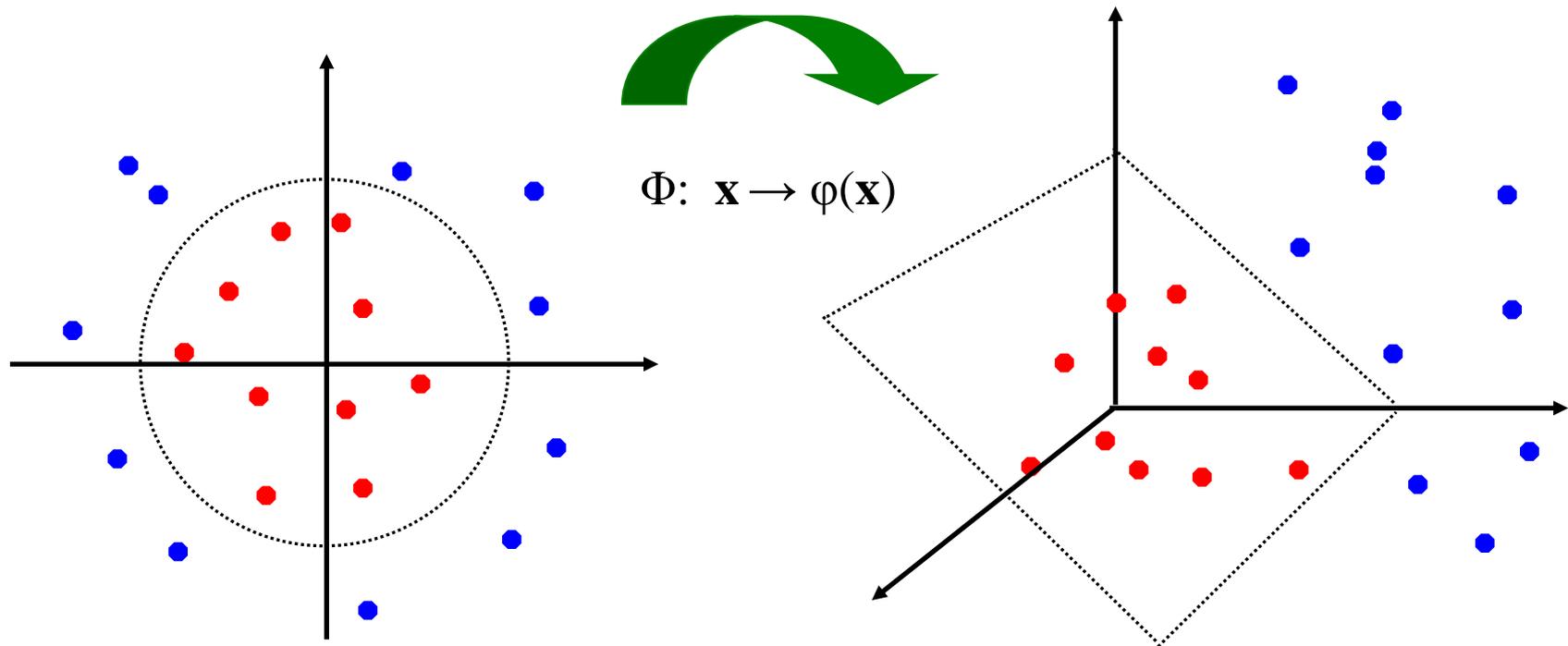
- We can map it to a higher-dimensional space:



# Nonlinear SVMs

---

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



# Nonlinear SVMs

---

- *The kernel trick*: instead of explicitly computing the lifting transformation  $\varphi(\mathbf{x})$ , define a kernel function  $K$  such that

$$K(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i) \cdot \varphi(\mathbf{x}_j)$$

(to be valid, the kernel function must satisfy *Mercer's condition*)

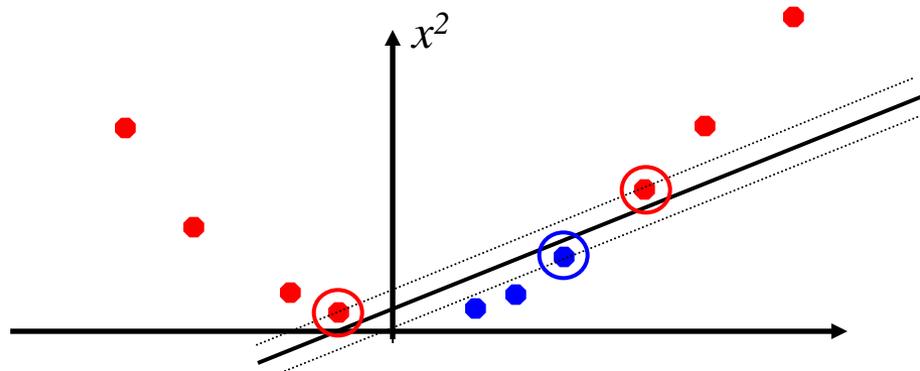
- This gives a nonlinear decision boundary in the original feature space:

$$\sum_i \alpha_i y_i \varphi(\mathbf{x}_i) \cdot \varphi(\mathbf{x}) + b = \sum_i \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b$$

# Nonlinear kernel: Example

---

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



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

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

# Kernels for bags of features

---

- Histogram intersection kernel:

$$I(h_1, h_2) = \sum_{i=1}^N \min(h_1(i), h_2(i))$$

- Generalized Gaussian kernel:

$$K(h_1, h_2) = \exp\left(-\frac{1}{A} D(h_1, h_2)^2\right)$$

- $D$  can be L1 distance, Euclidean distance,  $\chi^2$  distance, etc.

# 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

# 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
  - Training: 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

# SVMs: Pros and cons

---

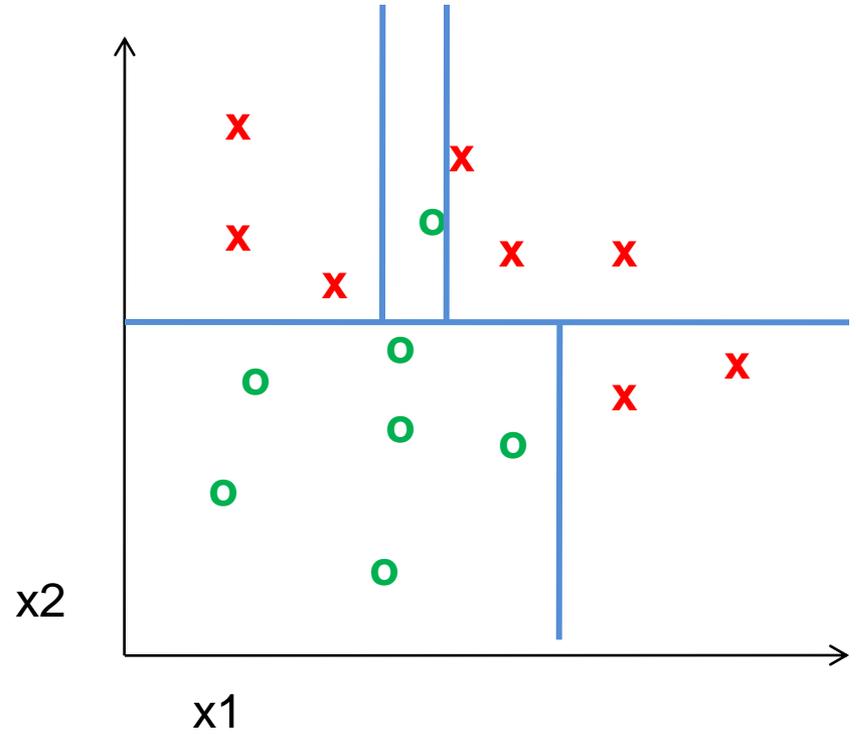
- Pros

- 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

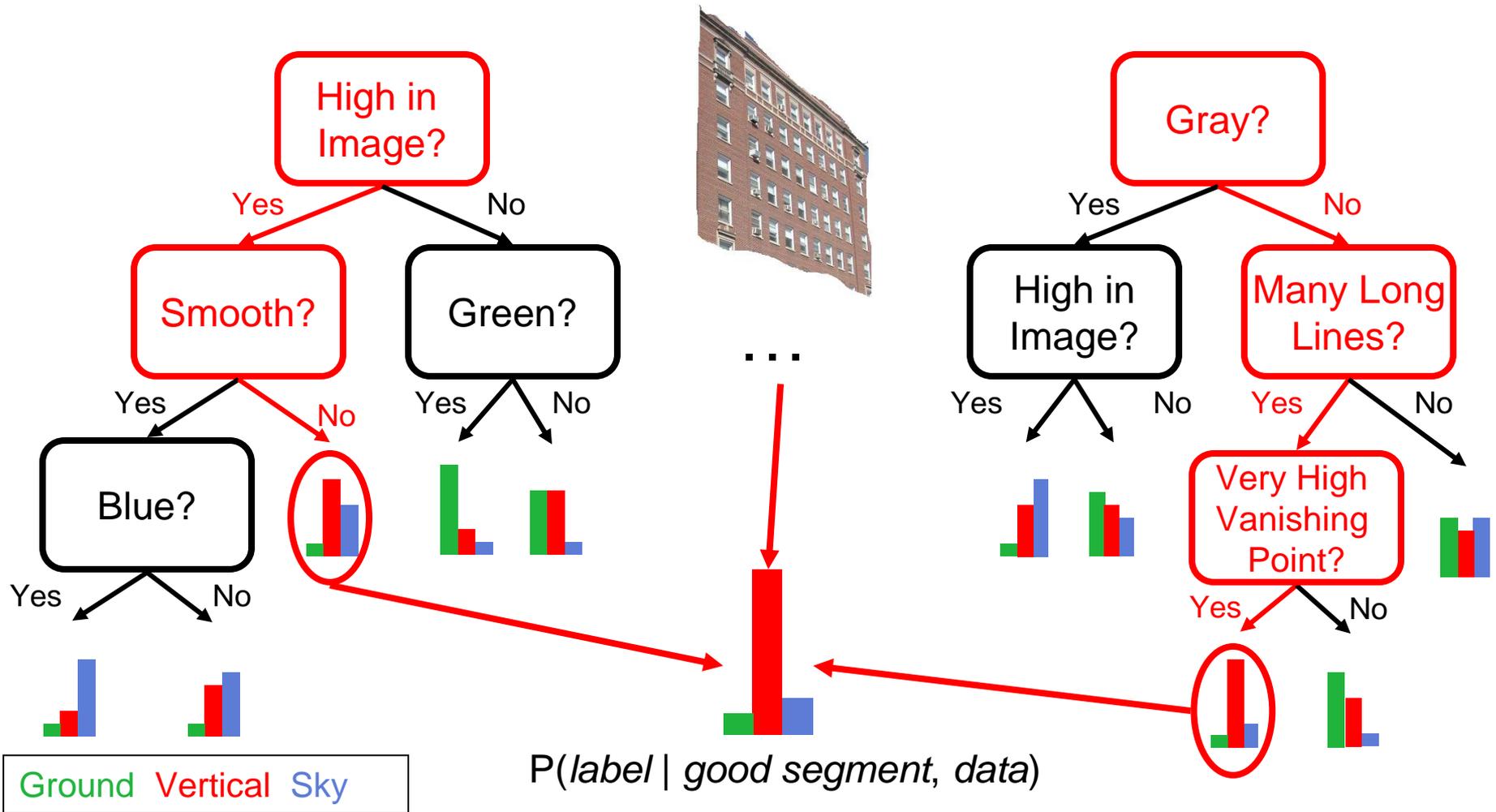


# Ensemble Methods: Boosting

## Discrete AdaBoost(Freund & Schapire 1996b)

1. Start with weights  $w_i = 1/N$ ,  $i = 1, \dots, N$ .
2. Repeat for  $m = 1, 2, \dots, M$ :
  - (a) Fit the classifier  $f_m(x) \in \{-1, 1\}$  using weights  $w_i$  on the training data.
  - (b) Compute  $\text{err}_m = E_w[1_{(y \neq f_m(x))}]$ ,  $c_m = \log((1 - \text{err}_m)/\text{err}_m)$ .
  - (c) Set  $w_i \leftarrow w_i \exp[c_m \cdot 1_{(y_i \neq f_m(x_i))}]$ ,  $i = 1, 2, \dots, N$ , and renormalize so that  $\sum_i w_i = 1$ .
3. Output the classifier  $\text{sign}[\sum_{m=1}^M c_m f_m(x)]$

# Boosted Decision Trees



# Using Boosted Decision Trees

- Flexible: can deal with both continuous and categorical variables
- How to control bias/variance trade-off
  - Size of trees
  - Number of trees
- Boosting trees often works best with a small number of well-designed features
- Boosting “stubs” can give a fast classifier

# 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?

# Comparison

assuming  $\mathbf{x}$  in  $\{0, 1\}$

## Learning Objective

## Training

## Inference

Naïve Bayes

$$\text{maximize } \sum_i \left[ \sum_j \log P(c_{ij} | y_i; \theta_j) + \log P(y_i; \theta_0) \right]$$

$$\theta_{kj} = \frac{\sum_i \delta(c_{ij} = 1 \wedge y_i = k) + r}{\sum_i \delta(c_{ij} = k) + Kr}$$

$$\theta_1^T \mathbf{x} + \theta_0^T \mathbf{1} - \mathbf{x} \geq 0$$

where  $\theta_j = \log \frac{P(c_j = 1 | y = 1)}{P(c_j = 1 | y = 0)}$

$$\theta_{0j} = \log \frac{P(c_j = 0 | y = 1)}{P(c_j = 0 | y = 0)}$$

Logistic Regression

$$\text{maximize } \sum_i \log P(c_i | \mathbf{x}, \boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|$$

where  $P(c_i | \mathbf{x}, \boldsymbol{\theta}) = 1 / (1 + \exp(-y_i \boldsymbol{\theta}^T \mathbf{x}))$

Gradient ascent

$$\boldsymbol{\theta}^T \mathbf{x} > 0$$

Linear SVM

$$\text{minimize } \lambda \sum_i \xi_i + \frac{1}{2} \|\boldsymbol{\theta}\|^2$$

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

$$\sum_i y_i \alpha_i K(c_i, \mathbf{x}) \geq 0$$

Nearest Neighbor

most similar features  $\rightarrow$  same label

Record data

$$y_i$$

where  $i = \operatorname{argmin}_i K(c_i, \mathbf{x})$

# 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)

# 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>