CSCI 631
Foundations of Computer Vision

Ifeoma Nwogu
ion@cs.rit.edu

Lecture 8 - Classifiers

Schedule

- Last class
  - Segmentation and clustering
- Today
  - Start classifiers
- Readings (optional):
  - Forsyth and Ponce chapter 15;

Classifiers

- Given a feature representation for images, how do we learn a model for distinguishing features from different classes?
- Take a measurement x, predict a bit
  - binary classifiers: (yes/no; 1/-1; 1/0; etc.)
- Today:
  - Nearest neighbor classifiers
  - Linear classifiers: support vector machines
- Other techniques:
  - Logistic regression, Boosting, Decision trees and forests, Deep neural networks

Image classification

- The big problems
  - Image classification
    - e.g. this picture contains a parrot
  - Object detection
    - e.g. in this box in the picture is a parrot
- Strategy
  - Generate features from image (there are many quite complex strategies)
  - Put in one or more classifiers

Image classification - scenes

FIGURE 16.2: Some scenes are easily identified by humans. These are examples from the SUN dataset (Xiao *et al.* 2010) of scene categories that people identify accurately from images; the label above each image gives its scene type. *This figure was originally published as Figure 2 of "SUN database: Large-scale Scene Recognition from Abbey to Zoo," by J. Xiao, J. Hays, K. Ehinger, A. Oliva, and A. Torralba, Proc. IEEE CVPR 2010, © IEEE, 2010.*

Image classification - material

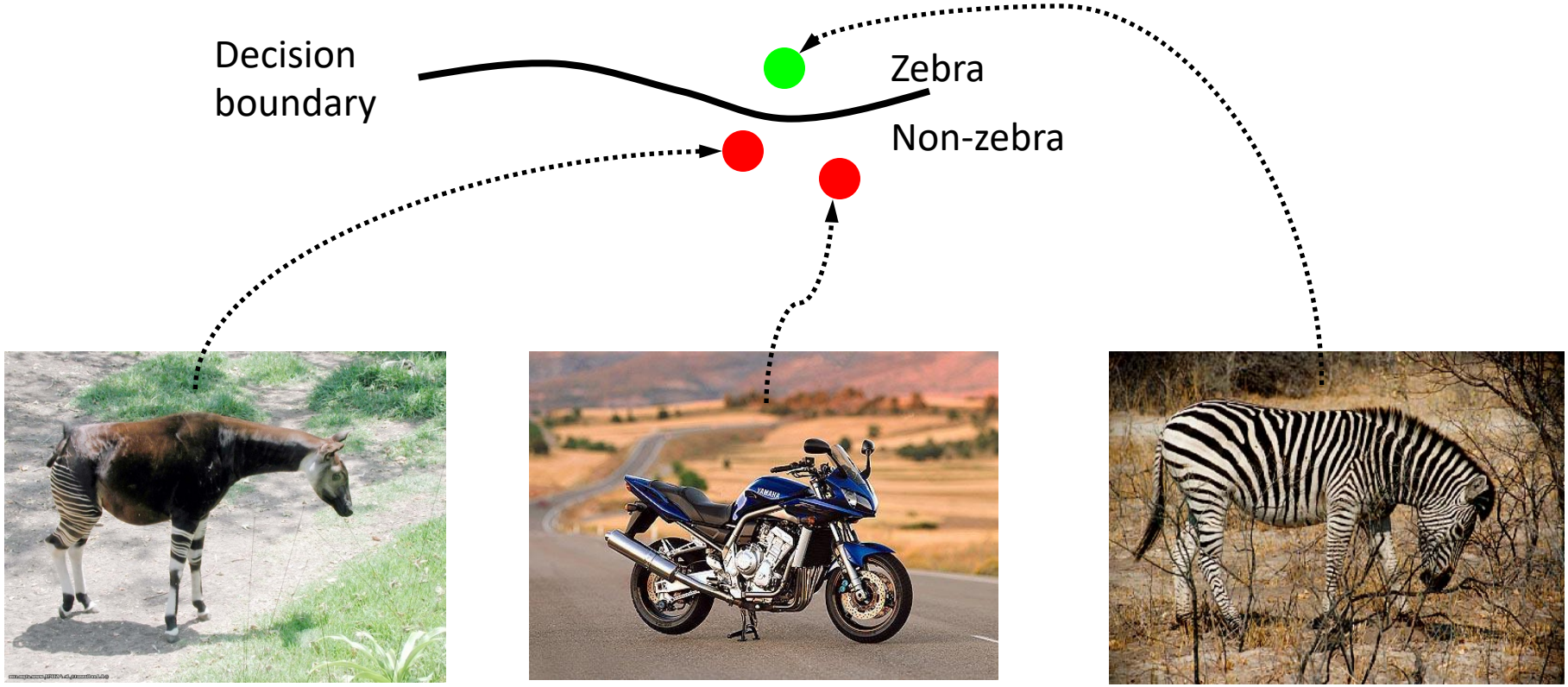
FIGURE 16.1: Material is not the same as object category (the three cars on the **top** are each made of different materials), and is not the same as texture (the three checkered objects on the **bottom** are made of different materials). Knowing the material that makes up an object gives us a useful description, somewhat distinct from its identity and its texture. *This figure was originally published as Figures 2 and 3 of "Exploring Features in a Bayesian Framework for Material Recognition," by C. Liu, L. Sharan, E. Adelson, and R. Rosenholtz Proc. CVPR 2010, 2010* © *IEEE, 2010.*

Troubleshooting

- Trained images of truck outdoors at multiple poses and scales
- Training yielded almost zero error
- Testing yielded almost 50% error!
- Possible reasons for failure?

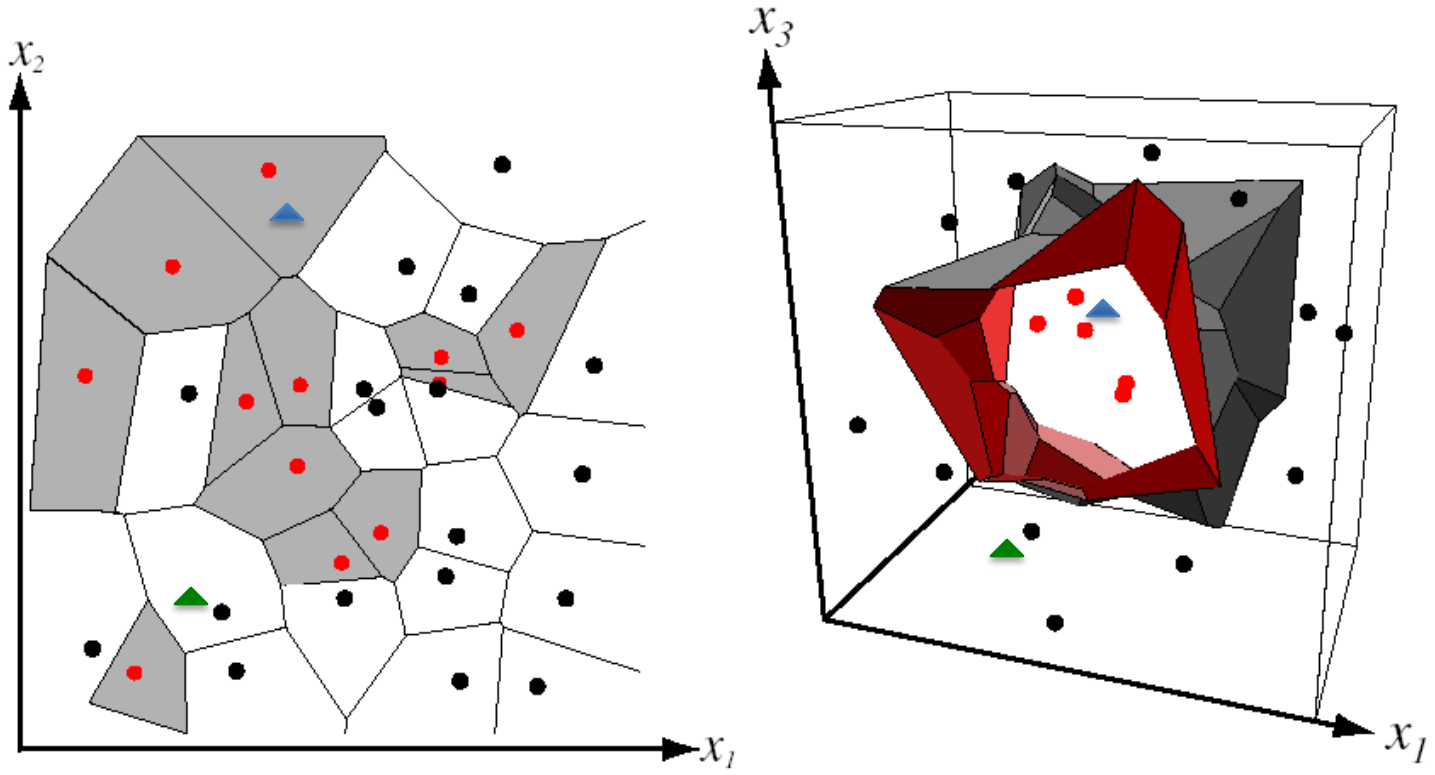
Classifiers

Given a feature representation for images, how do we learn a model for distinguishing features from different classes?



Nearest Neighbor Classifier

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



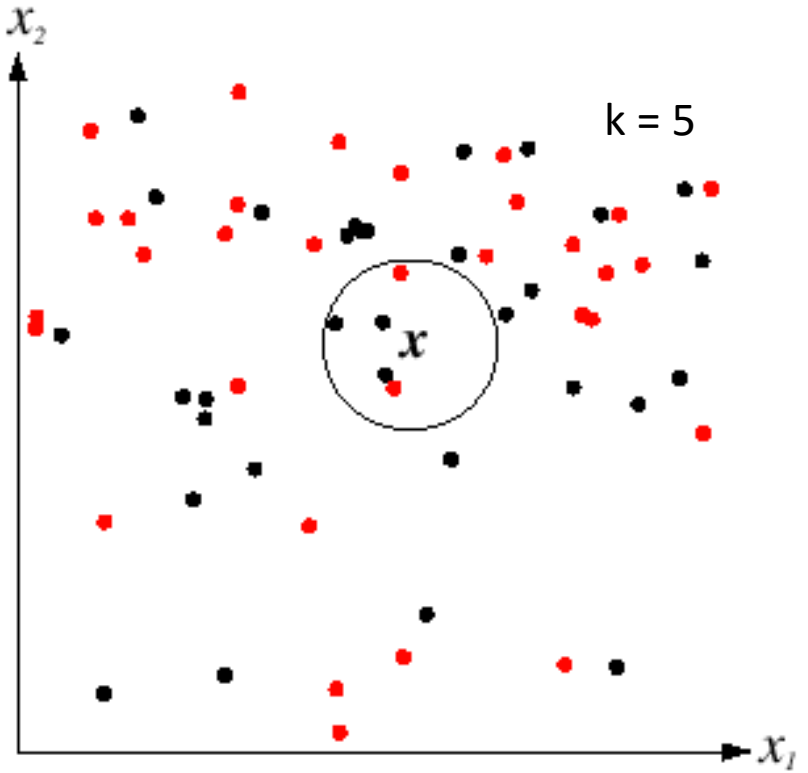
from Duda *et al.*

Classifiers

- Take a measurement x, predict a bit (yes/no; 1/-1; 1/0; etc)
- Strategies:
  - non-parametric
    - nearest neighbor
  - probabilistic
    - histogram
    - logistic regression
  - decision boundary
    - SVM

K-Nearest Neighbors

- For a new point, find the k closest points from training data
- Labels of the k points “vote” to classify



Histogram based classifiers

- Represent class-conditional densities with histogram
- Advantage:
  - estimates become quite good
    - (with enough data!)
- Disadvantage:
  - Histogram becomes big with high dimension
    - but maybe we can assume feature independence?

Histogram classifier for scene recognition

- When is scene identification just texture recognition?
  - Paper by *Laura Renninger and Jitendra Malik (Journal of Vision, 2004)*

Distance functions for bags of features

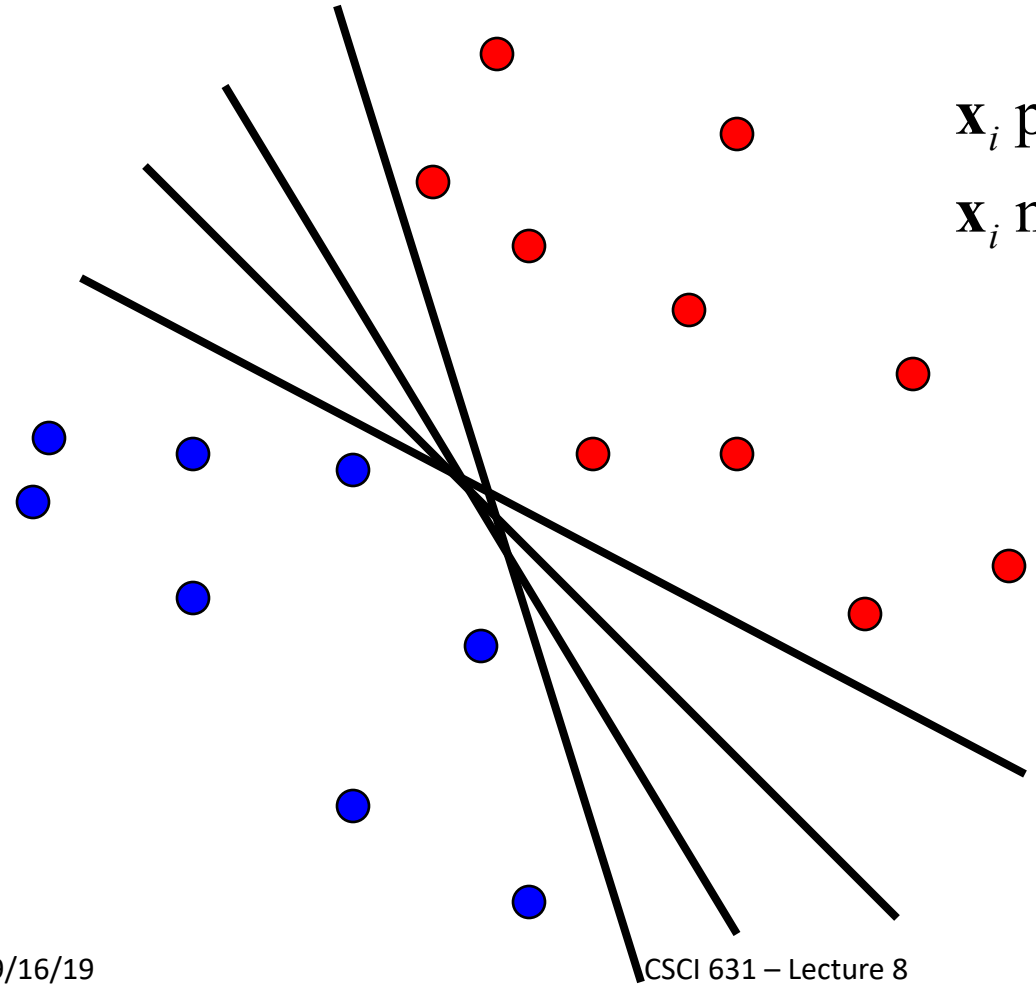
- Euclidean distance: $D(\mathbf{h}_1, \mathbf{h}_2) = \sqrt{\sum_{i=1}^{N} (\mathbf{h}_1(i) - \mathbf{h}_2(i))^2}$
- L1 distance: $D(\mathbf{h}_1, \mathbf{h}_2) = \sum_{i=1}^{N} | \mathbf{h}_1(i) - \mathbf{h}_2(i) |$
- χ^2 distance: $D(\mathbf{h}_1, \mathbf{h}_2) = \sum_{i=1}^{N} \frac{(\mathbf{h}_1(i) - \mathbf{h}_2(i))^2}{\mathbf{h}_1(i) + \mathbf{h}_2(i)}$
- Histogram intersection (similarity): $I(\mathbf{h}_1, \mathbf{h}_2) = \sum_{i=1}^{N} \min(\mathbf{h}_1(i), \mathbf{h}_2(i))$
- Hellinger kernel (similarity): $K(\mathbf{h}_1, \mathbf{h}_2) = \sum_{i=1}^{N} \sqrt{\mathbf{h}_1(i)\, \mathbf{h}_2(i)}$

Curse of dimension

- Hard to build a histogram based classifier for when feature space is high-dimensional
  - try R, G, B, and some texture features
  - Fails when there are too many histogram buckets
- Walk-around
  - Build prototypes from training images as shown in texture recognition example

Linear classifiers

- Find linear function (*hyperplane*) to separate positive and negative examples



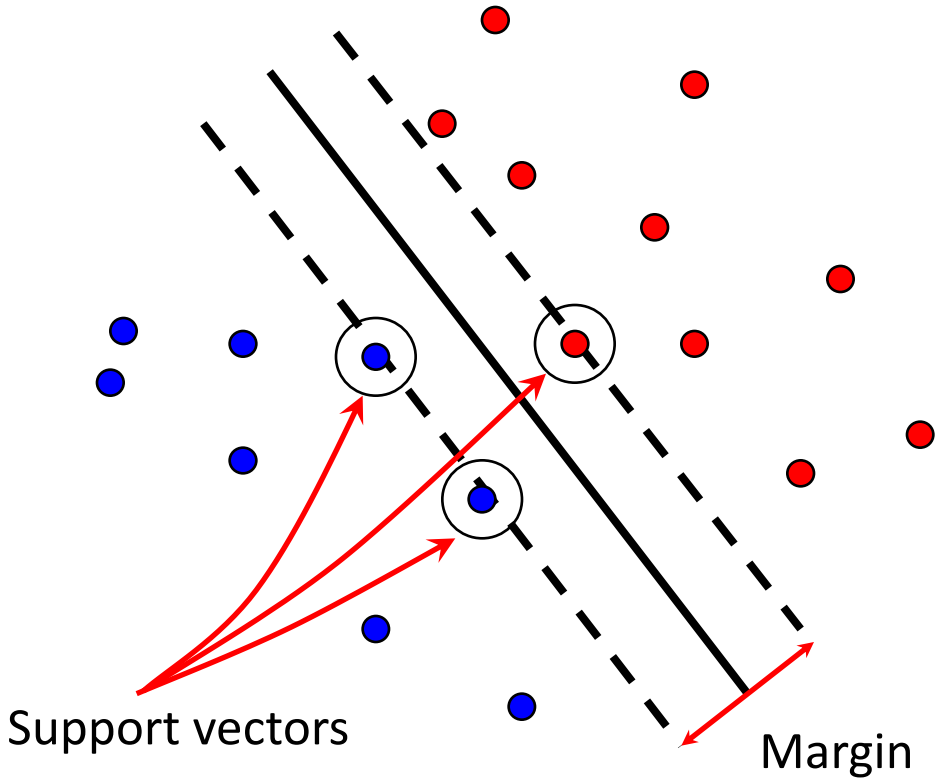
$$\mathbf{x}_i \text{ positive}: \quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 0$$

$$\mathbf{x}_i \text{ negative}: \quad \mathbf{x}_i \cdot \mathbf{w} + b < 0$$

Which hyperplane is best?

Support vector machines

- Find hyperplane that maximizes the *margin* between the positive and negative examples



$\mathbf{x}_i$ positive $(y_i = 1)$: $\mathbf{x}_i \cdot \mathbf{w} + b \geq 1$

$\mathbf{x}_i$ negative $(y_i = -1)$: $\mathbf{x}_i \cdot \mathbf{w} + b \leq -1$

For support vectors, $\mathbf{x}_i \cdot \mathbf{w} + b = \pm 1$

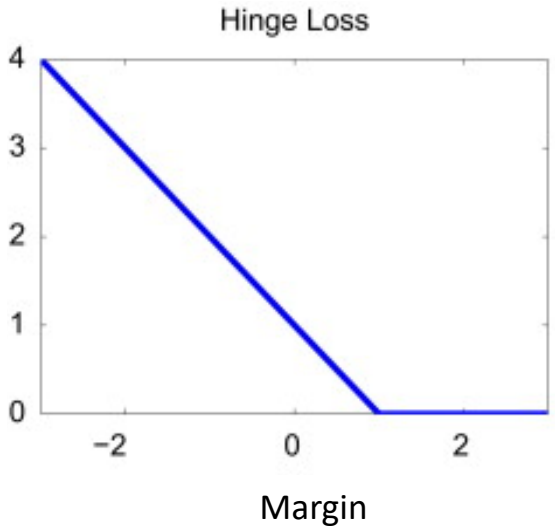
Distance between point and hyperplane: $\dfrac{|\mathbf{x}_i \cdot \mathbf{w} + b|}{\|\mathbf{w}\|}$

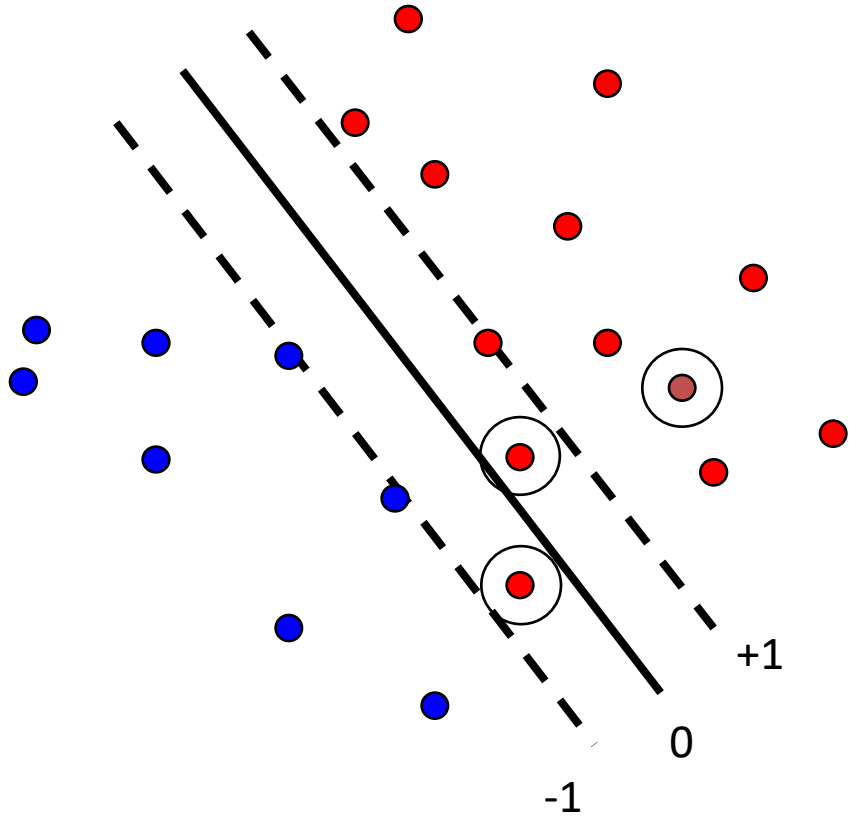
Therefore, the margin is $2 / \|\mathbf{w}\|$

C. Burges, A Tutorial on Support Vector Machines for Pattern Recognition, Data Mining and Knowledge Discovery, 1998

What if the data is not linearly separable?

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\max\left(0,1-y_i(\mathbf{w}\cdot\mathbf{x}_i+b)\right)$$

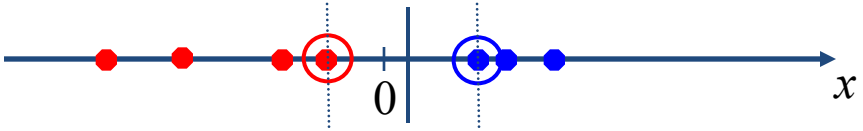




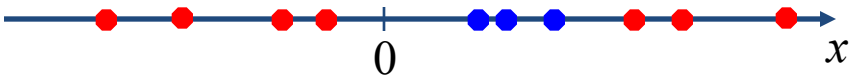
Demo: http://cs.stanford.edu/people/karpathy/svmjs/demo

Nonlinear SVMs

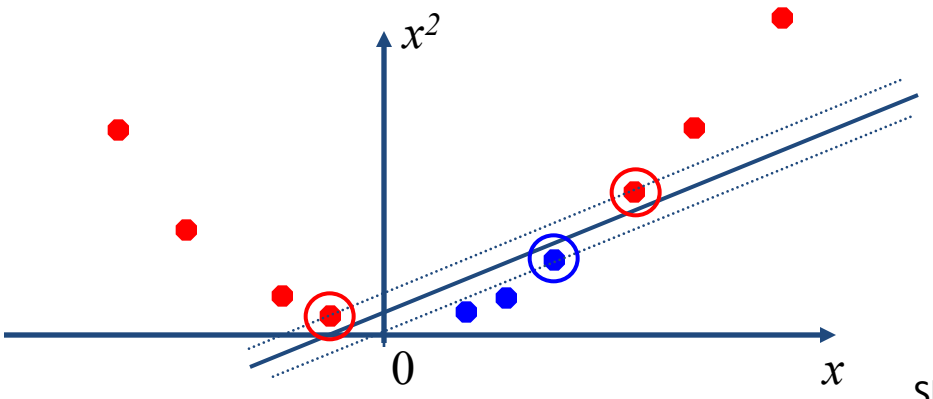
- Datasets that are linearly separable work out great:



- But what if the dataset is just too hard?



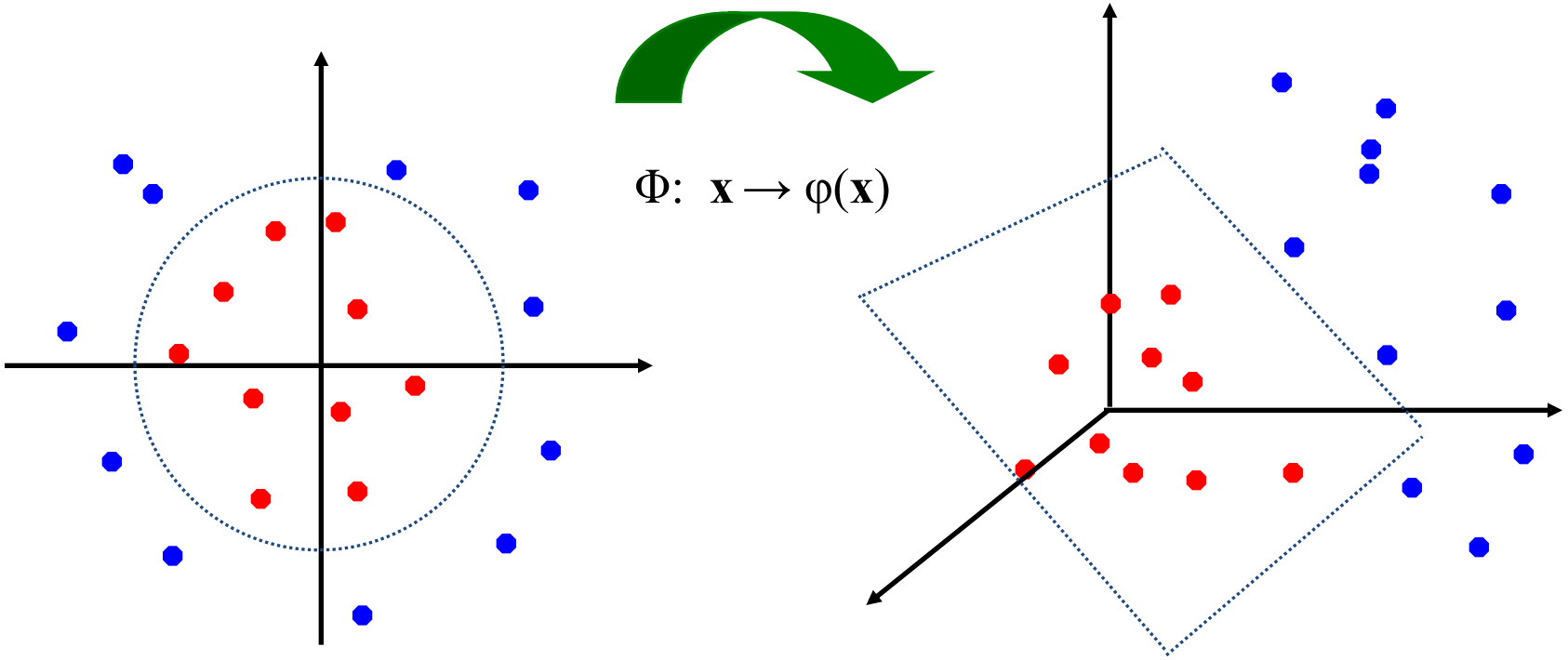
- We can map it to a higher-dimensional space:



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 SVMs

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

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

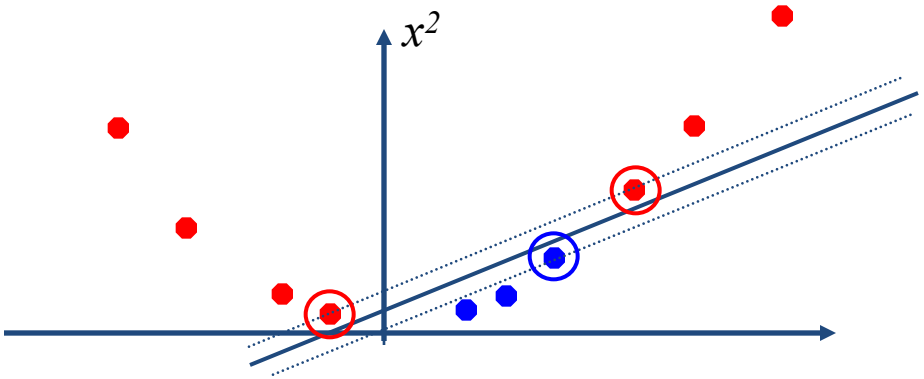
- This gives a nonlinear decision boundary in the original feature space:

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

C. Burges, A Tutorial on Support Vector Machines for Pattern Recognition, Data Mining and Knowledge Discovery, 1998

Nonlinear kernel: Example

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



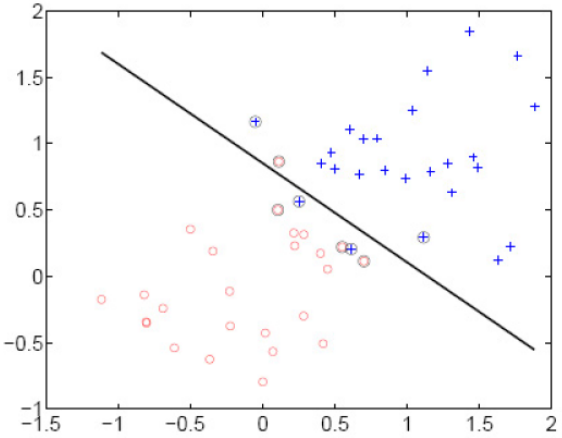
$$\varphi(x) \cdot \varphi(y) = (x, x^2) \cdot (y, y^2) = xy + x^2 y^2$$

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

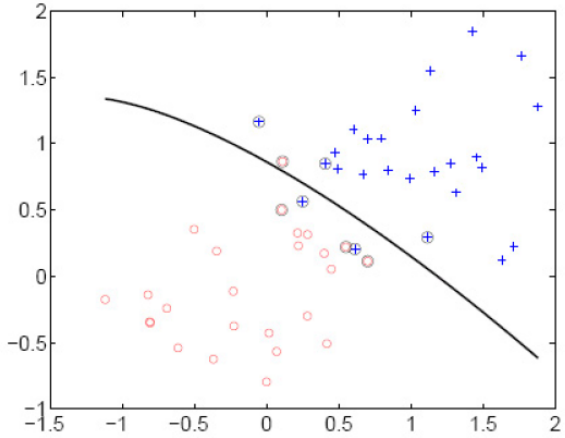
Excuse the abuse of notation

The kernel function computes the inner-product between two projected vectors

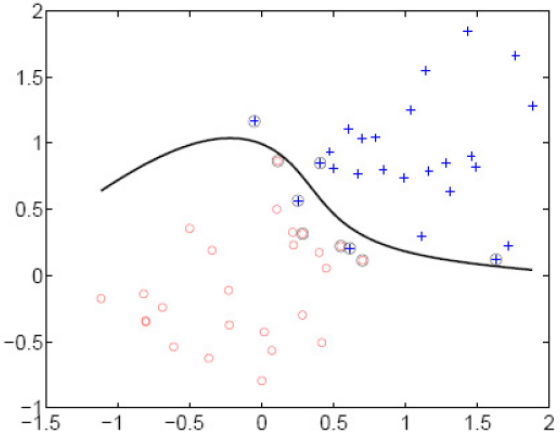
Polynomial kernel: $K(\mathbf{x}, \mathbf{y}) = (c + \mathbf{x} \cdot \mathbf{y})^d$



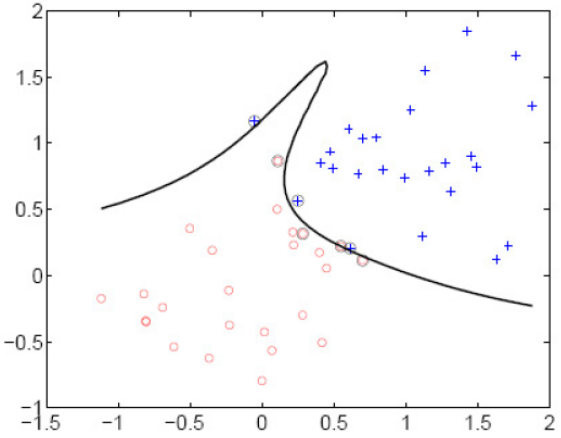
linear



2^{nd} order polynomial



4^{th} order polynomial



8^{th} order polynomial

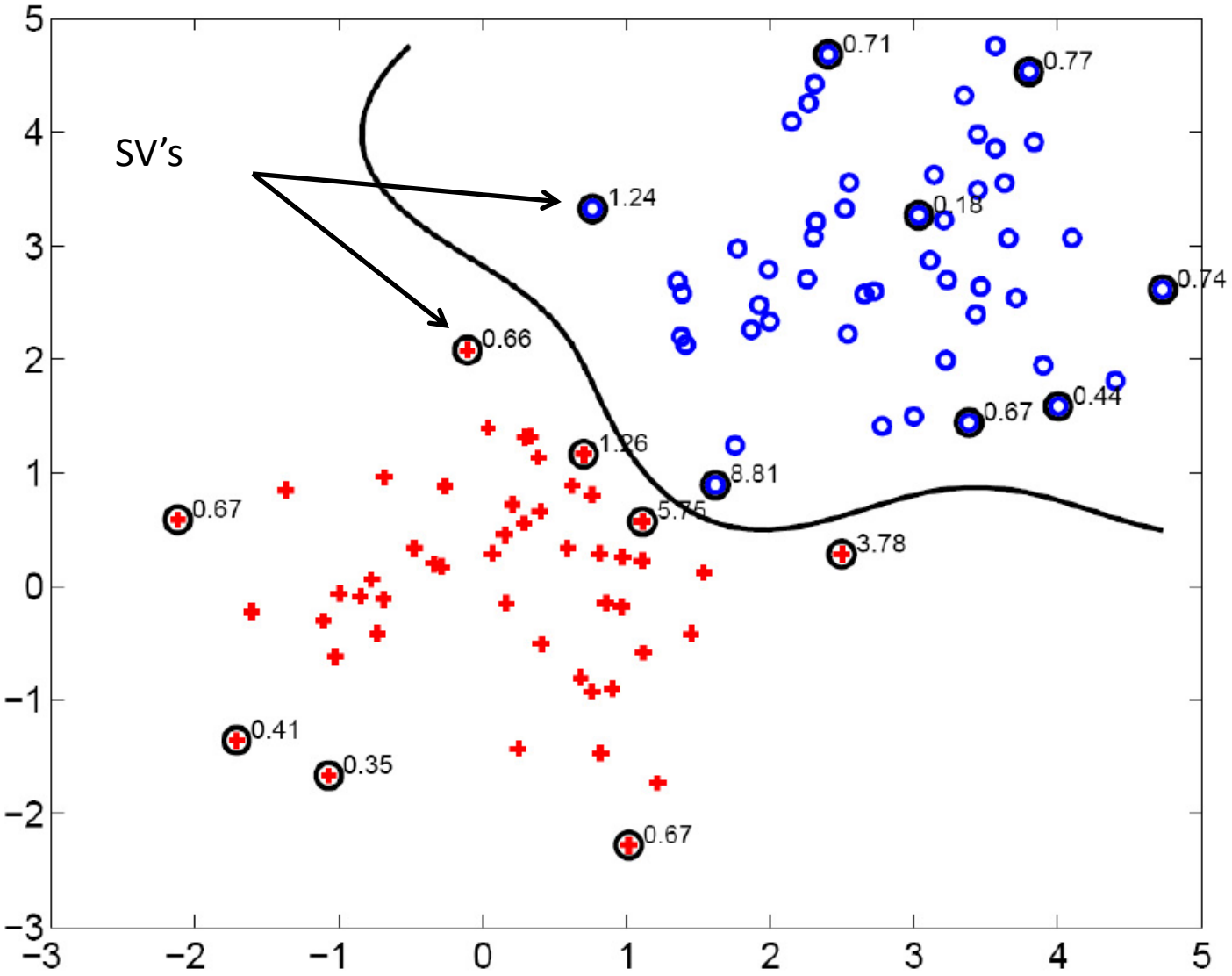
Gaussian kernel

- Also known as the radial basis function (RBF) kernel:

$$K(\mathbf{x},\mathbf{y}) = \exp\left(-\frac{1}{\sigma^2}\|\mathbf{x}-\mathbf{y}\|^2\right)$$

- The corresponding mapping $\varphi(\mathbf{x})$ is infinite-dimensional!

Gaussian kernel



Why use kernels ?

- Classifiers can be learnt for high dimensional features spaces
  - the points need not necessarily be mapped into the high dimensional space.
- Data may be linearly separable in the high dimensional space
  - the points need not be linearly separable in the original feature space

Why use kernels ? cont'd

- Kernels can be used for an SVM because of the scalar product in the dual form
  - Kernels can also be used elsewhere – they are not tied to the SVM formalism
- Kernels apply also to objects that are not vectors, e.g.

$$k(h, h') = \sum_k \min(h_k, h'_k)$$
for histograms with bins h_k, h'_k

Summary: SVMs for image classification

1. Pick an image representation
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
  - Faster in training, fewer support vectors
- 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

SVMs for large-scale datasets

- Efficient *linear* solvers
  - LIBLINEAR, PEGASOS
- Explicit approximate embeddings: define an explicit mapping $\boldsymbol{\varphi}(\mathbf{x})$ such that $\boldsymbol{\varphi}(\mathbf{x}) \cdot \boldsymbol{\varphi}(\mathbf{y})$ approximates $K(\mathbf{x}, \mathbf{y})$ and train a linear SVM on top of that embedding
  - Random Fourier features for the Gaussian kernel (Rahimi and Recht, 2007)
  - Embeddings for additive kernels, e.g., histogram intersection (Maji et al., 2013, Vedaldi and Zisserman, 2012)

Slide Credits

- Svetlana Lazebnik – UIUC
- David Forsyth - UIUC

Questions