

*CS 1674: Intro to Computer Vision*

# **Visual Recognition**

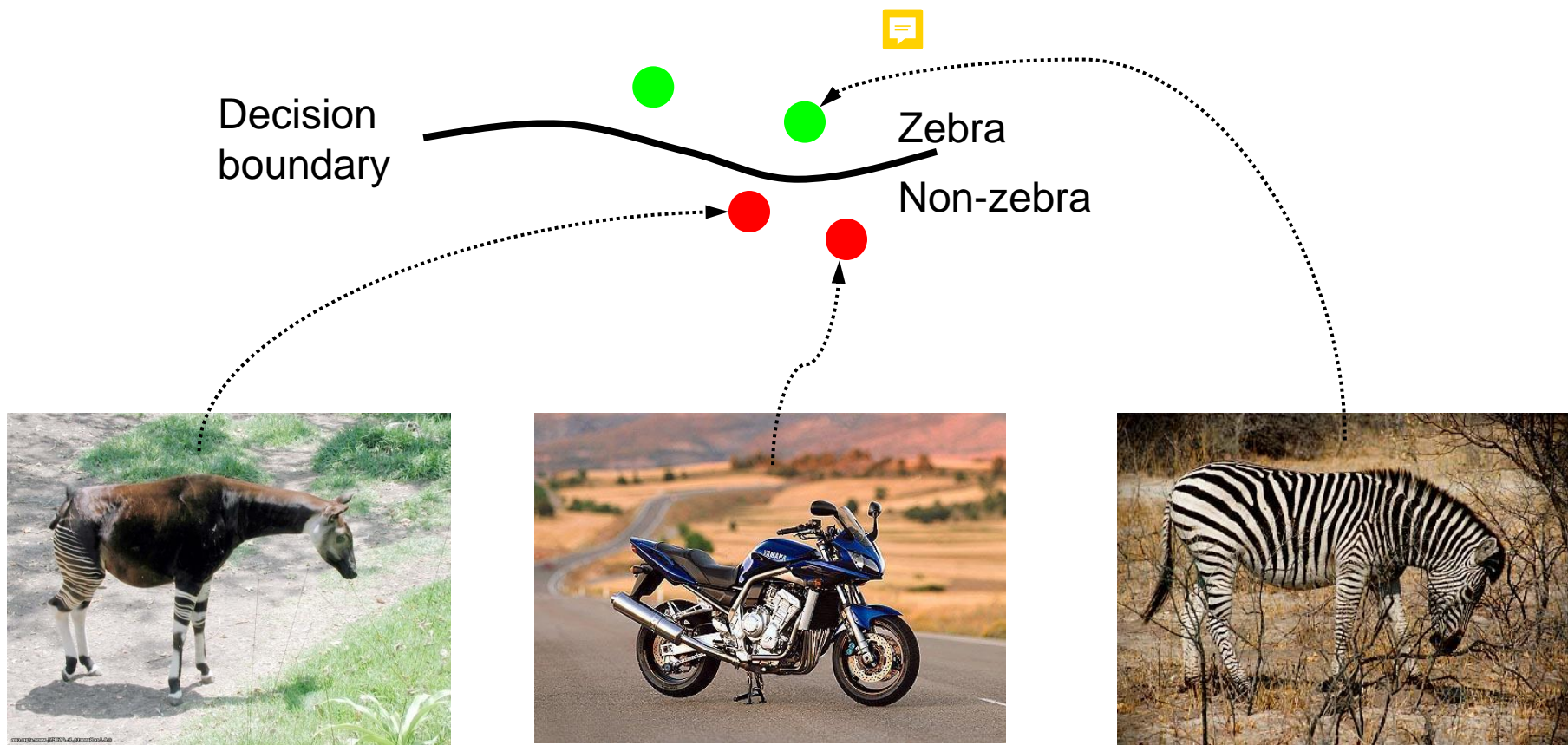
Prof. Adriana Kovashka  
University of Pittsburgh  
October 6, 2020

# Plan for this lecture

- What is recognition?
  - a.k.a. classification, categorization
- Support vector machines
  - Separable case / non-separable case
  - Linear / non-linear (kernels)
- The importance of generalization
  - The bias-variance trade-off (applies to all classifiers)
- Example approach for scene classification

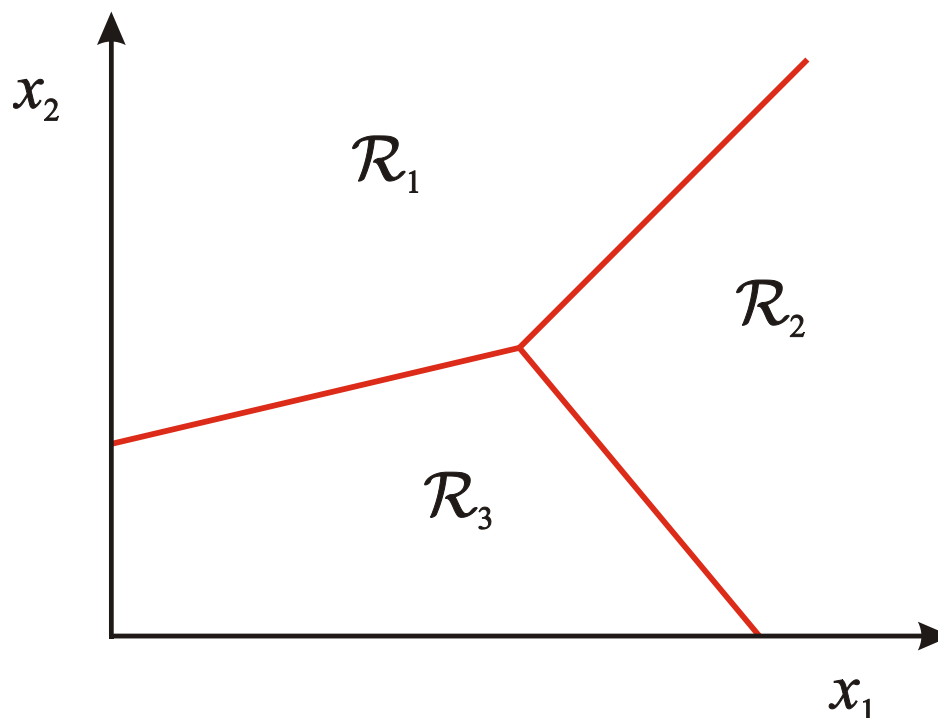
# Classification

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



# Classification

- Assign input vector to one of two or more classes
- Input space divided into *decision regions* separated by *decision boundaries*



# Examples of image classification

- Two-class (binary): Cat vs Dog





# Examples of image classification

- Multi-class (often): Object recognition

ImageNet

www.image-net.org/search?q=car

120%

**Synset:** [racer, race car, racing car](#)  
**Definition:** a fast car that competes in races.  
*Popularity percentile:* 85%  
*Depth in WordNet:* 10

**Synset:** [car mirror](#)  
**Definition:** a mirror that the driver of a car can use.  
*Popularity percentile:* 83%  
*Depth in WordNet:* 8

**Synset:** [passenger car, coach, carriage](#)  
**Definition:** a railcar where passengers ride.  
*Popularity percentile:* 83%  
*Depth in WordNet:* 8

**Synset:** [beach wagon, station wagon, wagon, estate car, beach waggon, station waggon, waggon](#)  
**Definition:** a car that has a long body and rear door with space behind rear seat.  
*Popularity percentile:* 74%  
*Depth in WordNet:* 10

**Synset:** [freight car](#)  
**Definition:** a railway car that carries freight.  
*Popularity percentile:* 64%  
*Depth in WordNet:* 8

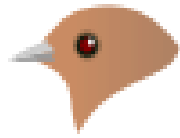
**Synset:** [bumper car, Dodgem](#)  
**Definition:** a small low-powered electrically powered vehicle driven on a special platform where there others to be dodged.  
*Popularity percentile:* 63%  
*Depth in WordNet:* 7

# Examples of image classification

- Fine-grained recognition



Generalist



Insect catching



Grain eating



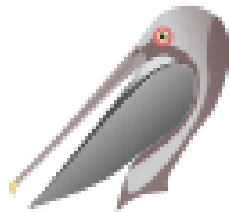
Coniferous-seed eating



Nectar feeding



Chiseling



Dip netting



Surface skimming



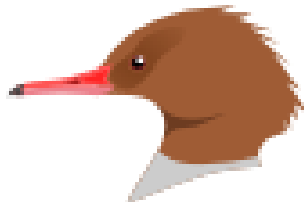
Scything



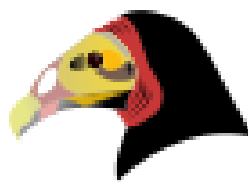
Probing



Aerial fishing



Pursuit fishing



Scavenging



Raptorial



Filter feeding

[Visipedia Project](#)

# Examples of image classification

- Place recognition

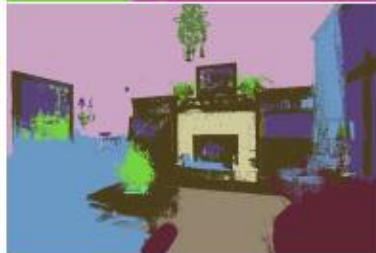
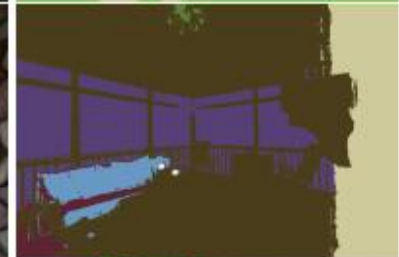


Places Database [[Zhou et al. NIPS 2014](#)]



# Examples of image classification

- Material recognition



[[Bell et al. CVPR 2015](#)]

# Examples of image classification

- Dating historical photos



1940



1953



1966



1977

[Palermo et al. ECCV 2012]

# Examples of image classification

- Image style recognition



HDR



Macro



Baroque



Rococo



Vintage



Noir



Northern Renaissance



Cubism



Minimal



Hazy



Impressionism



Post-Impressionism



Long Exposure



Romantic



Abs. Expressionism



Color Field Painting

Flickr Style: 80K images covering 20 styles.

Wikipaintings: 85K images for 25 art genres.

[[Karayev et al. BMVC 2014](#)]



# Recognition: A machine learning approach



# The machine learning framework

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

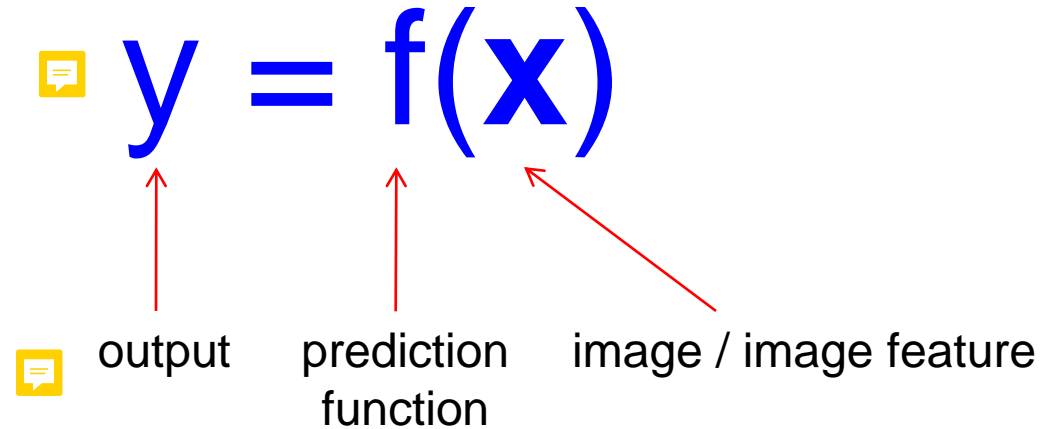
$$f(\text{apple}) = \text{"apple"}$$

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

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




# The machine learning framework



The diagram shows the equation  $y = f(x)$  in blue. A yellow speech bubble icon is to the left of the equation. Three red arrows point from labels below to parts of the equation: one from 'output' to  $y$ , one from 'prediction function' to  $f$ , and one from 'image / image feature' to  $x$ . A second yellow speech bubble icon is to the left of the 'output' label.

$y = f(x)$

output      prediction function      image / 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 
  - Evaluate multiple hypotheses  $f_1, f_2, f_H \dots$  and pick the best one as  $f$
- **Testing:** apply  $f$  to a never before seen *test example*  $\mathbf{x}$  and output the predicted value  $y = f(\mathbf{x})$

# The old-school way

## Training

Training  
Images



Image  
Features

Training  
Labels

Training

Learned  
model

## Testing



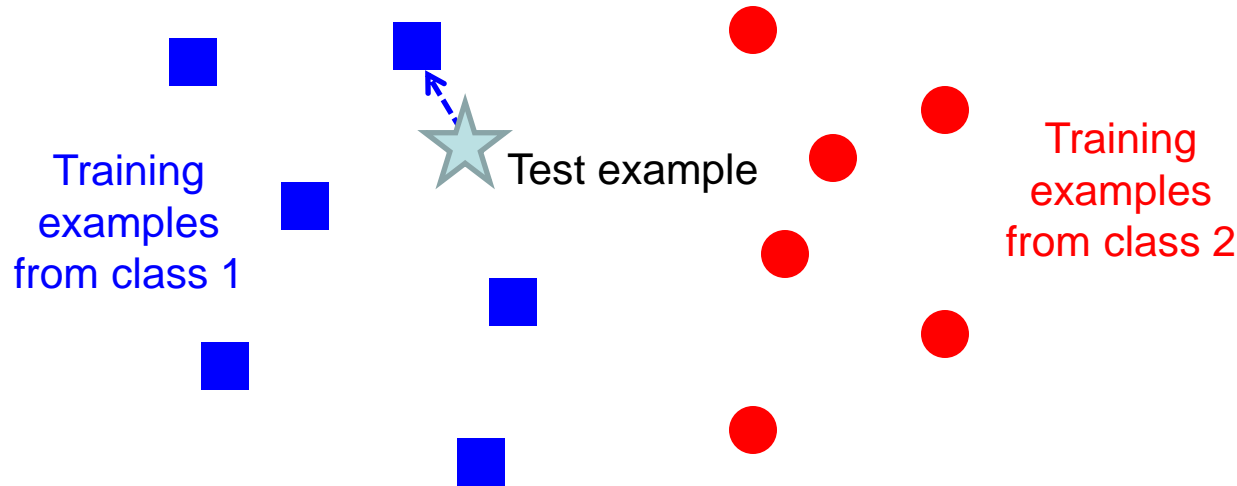
Test Image

Image  
Features

Learned  
model

Prediction

# The simplest classifier

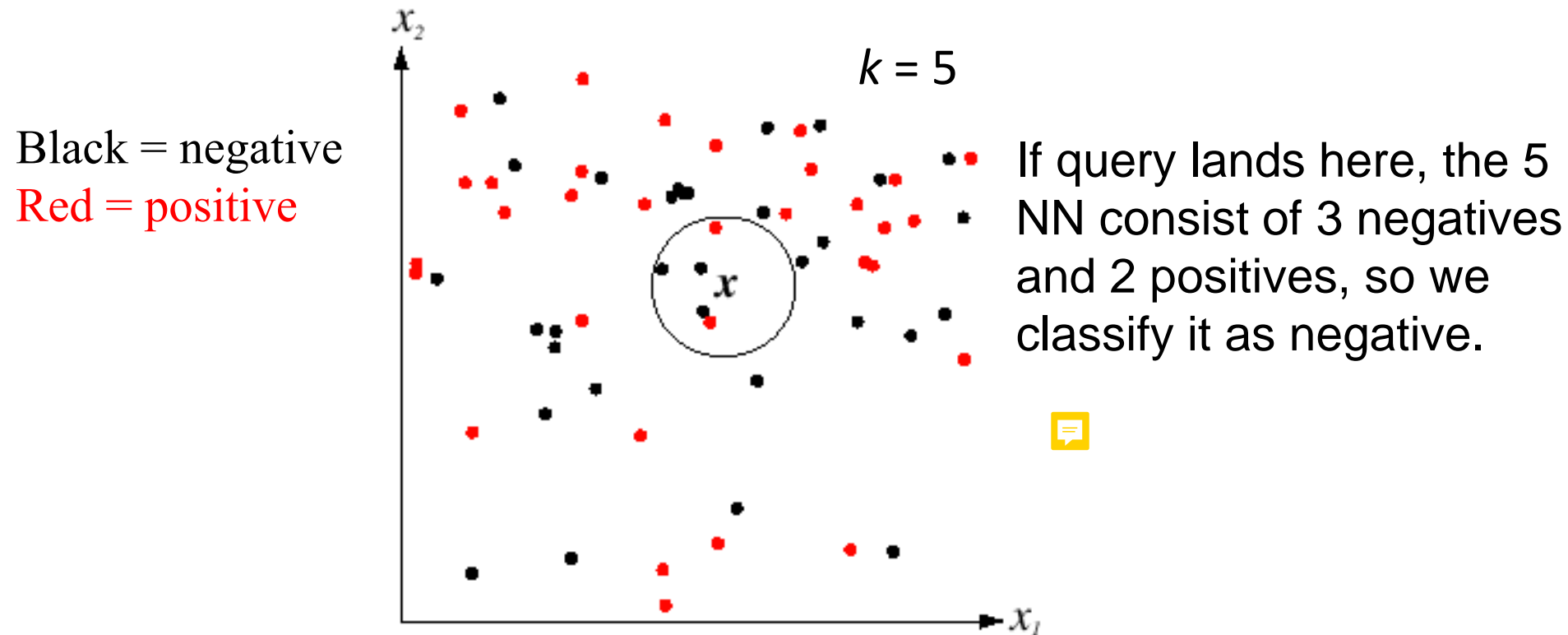


$f(\mathbf{x}) = \text{label of the training example nearest to } \mathbf{x}$

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

# K-Nearest Neighbors classification

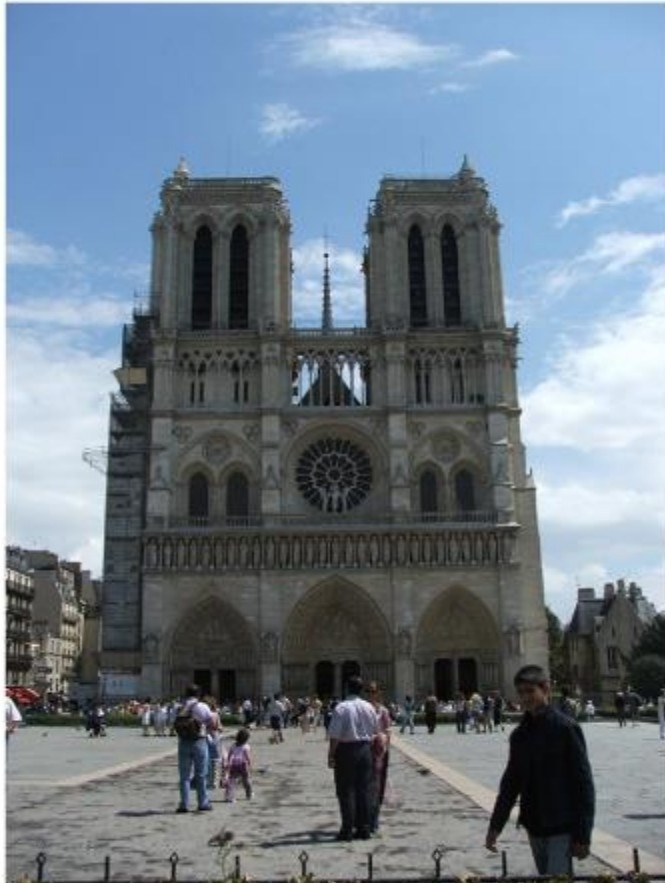
- For a new point, find the  $k$  closest points from training data
- Labels of the  $k$  points “vote” to classify



# im2gps: Estimating Geographic Information from a Single Image

James Hays and Alexei Efros, CVPR 2008

Where was this image taken?



Paris



Paris



Paris



Paris



Paris



Paris



Paris



Madrid



Rome



Paris



Cuba



Paris



Paris



Poland



Paris

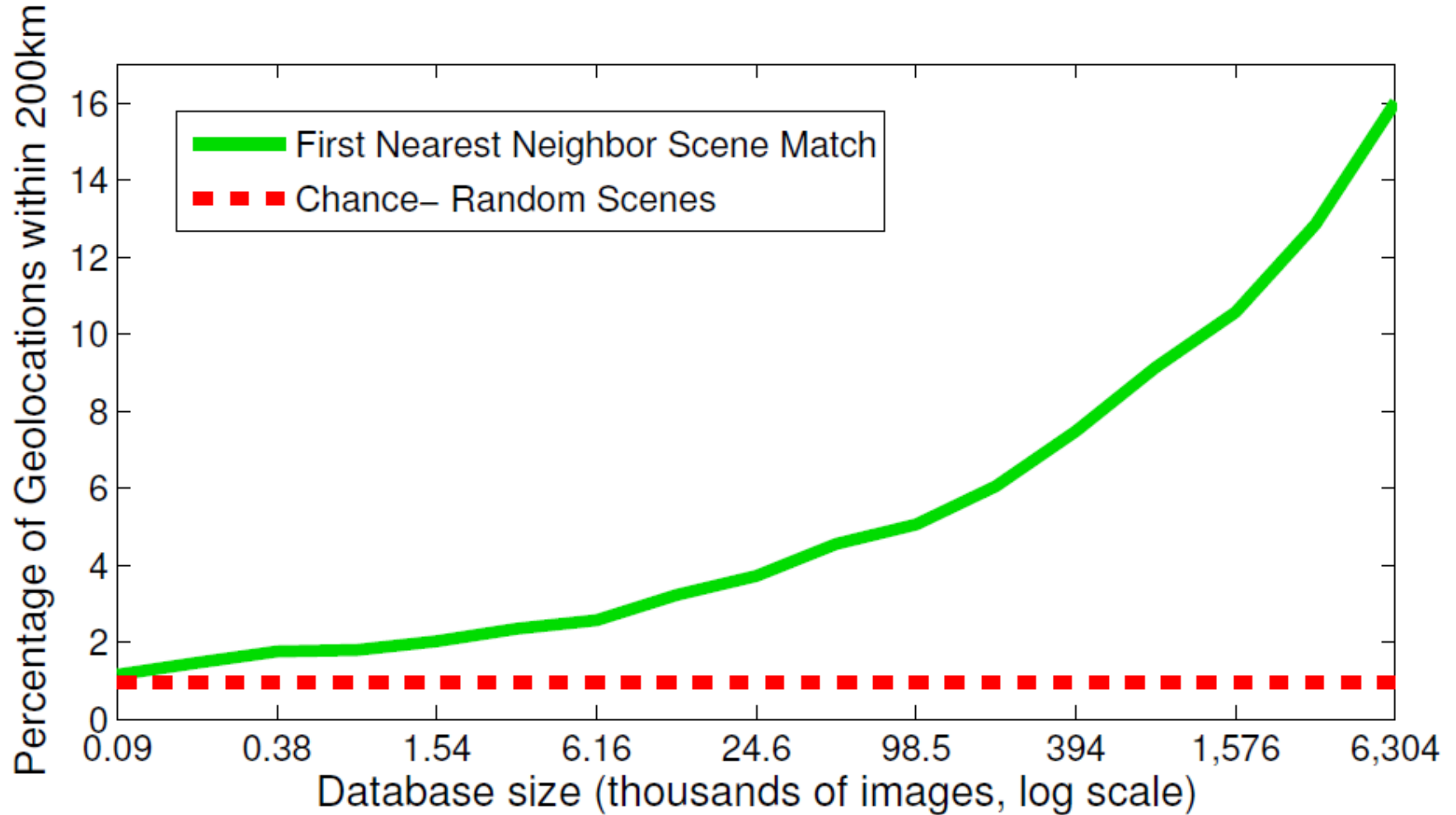


Paris

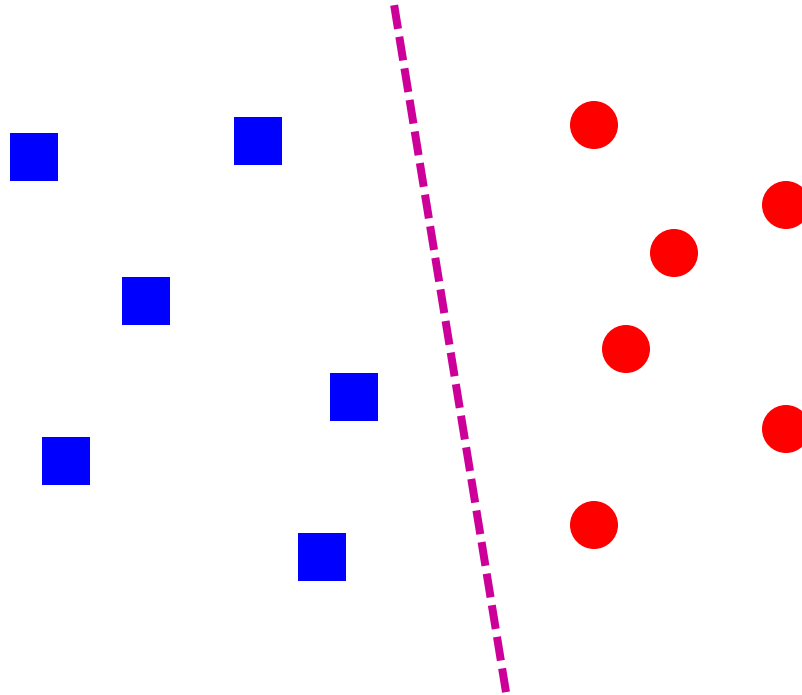
Nearest Neighbors according to bag of SIFT + color histogram + a few others



# The Importance of Data



# Linear classifier



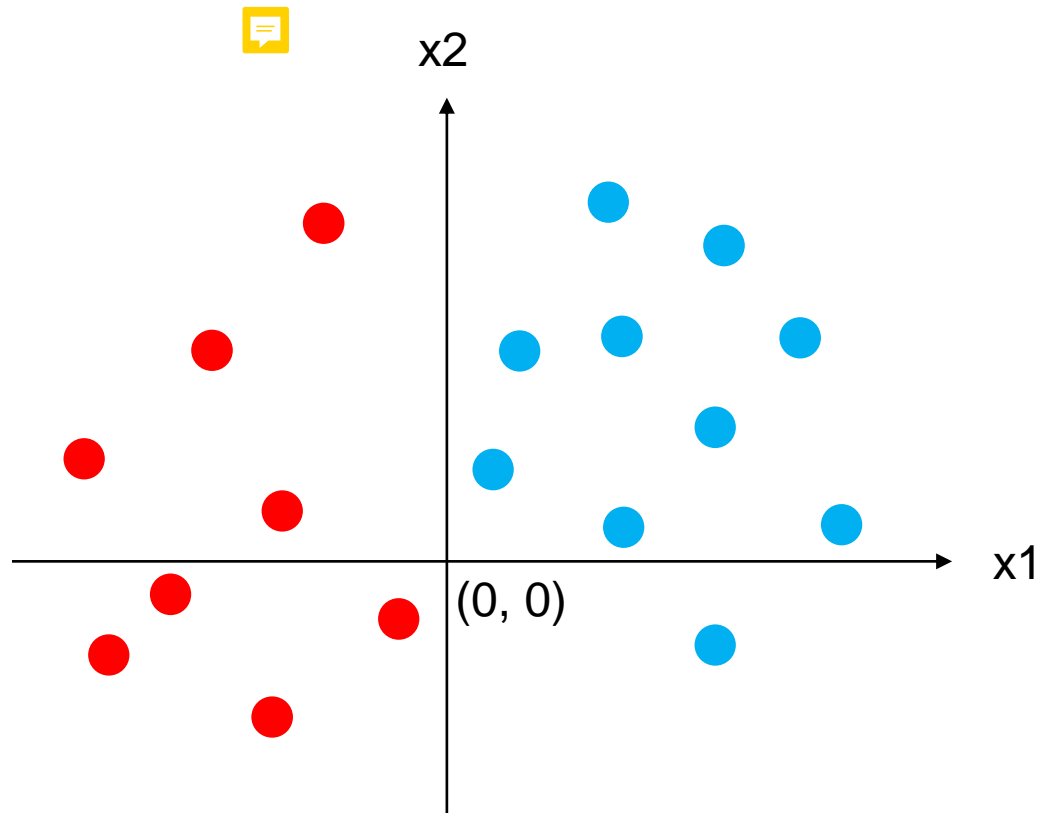
- Find a *linear function* to separate the classes

$$f(\mathbf{x}) = \text{sgn}(w_1x_1 + w_2x_2 + \dots + w_Dx_D) = \text{sgn}(\mathbf{w} \cdot \mathbf{x})$$



# Linear classifier

- Decision =  $\text{sign}(w^T x) = \text{sign}(w_1 * x_1 + w_2 * x_2)$



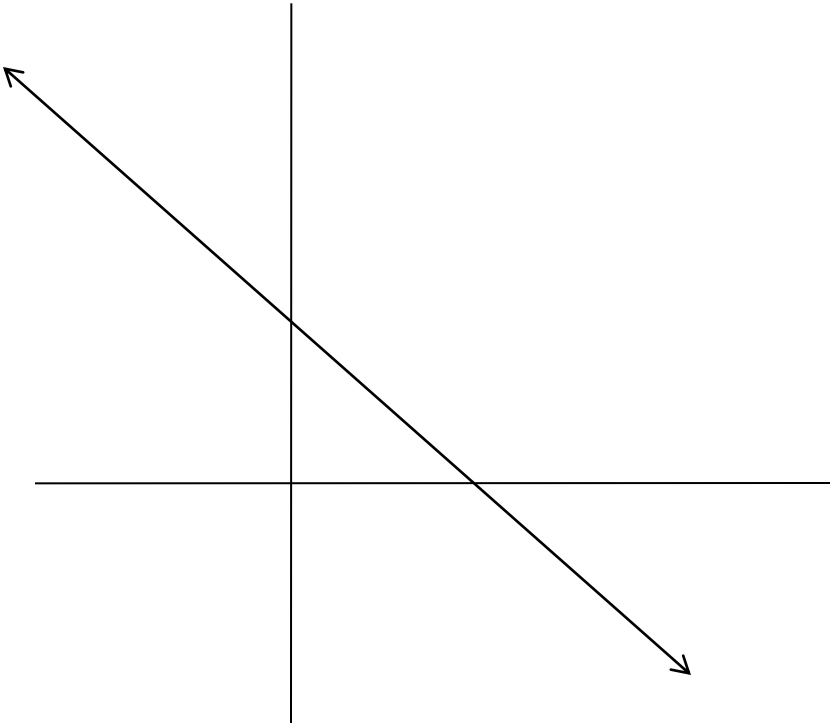
- What should the weights be? 

# Lines in $\mathbb{R}^2$

Slope:  $-a/c$   
Y-intercept:  $-b/c$

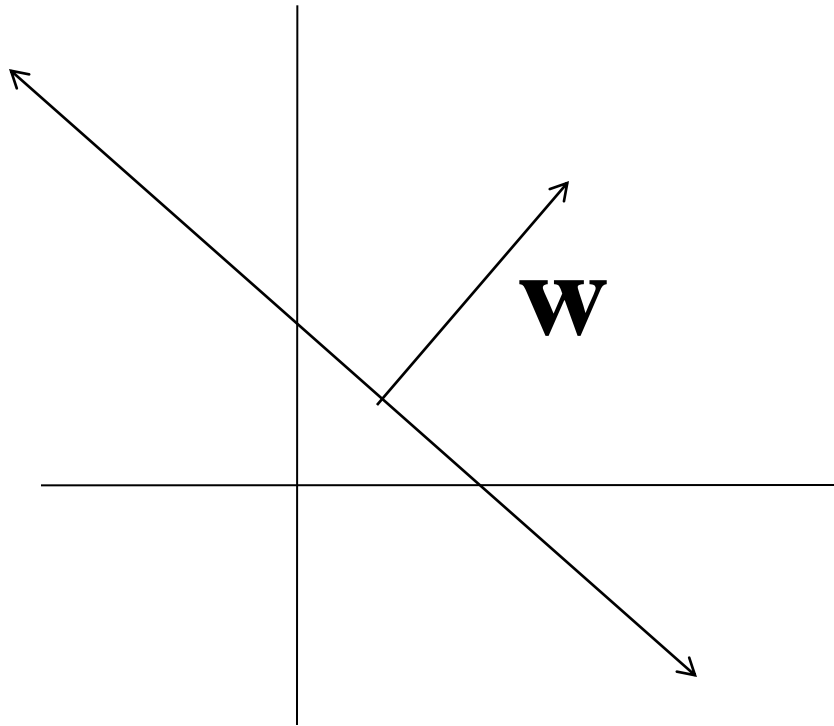
Let  $\mathbf{w} = \begin{bmatrix} a \\ c \end{bmatrix}$   $\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}$

$$ax + cy + b = 0$$



# Lines in $\mathbb{R}^2$

Slope:  $-a/c$   
Y-intercept:  $-b/c$



Let  $\mathbf{w} = \begin{bmatrix} a \\ c \end{bmatrix}$   $\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}$

$$ax + cy + b = 0$$

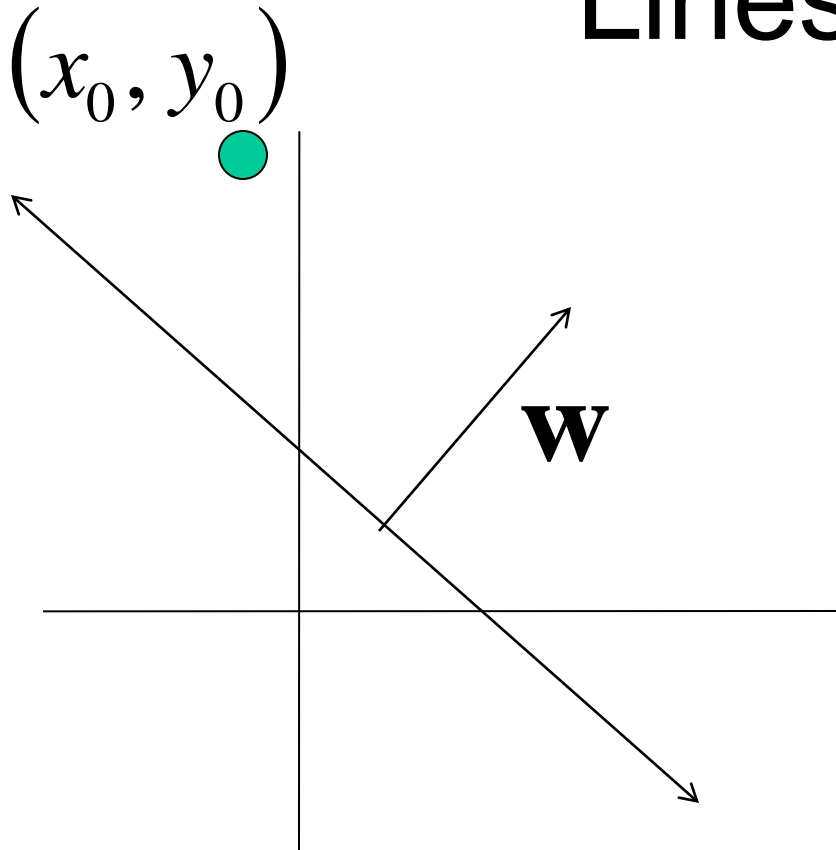


$$\mathbf{w} \cdot \mathbf{x} + b = 0$$



# Lines in $\mathbb{R}^2$

Slope:  $-a/c$   
Y-intercept:  $-b/c$



Let  $\mathbf{w} = \begin{bmatrix} a \\ c \end{bmatrix}$   $\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}$

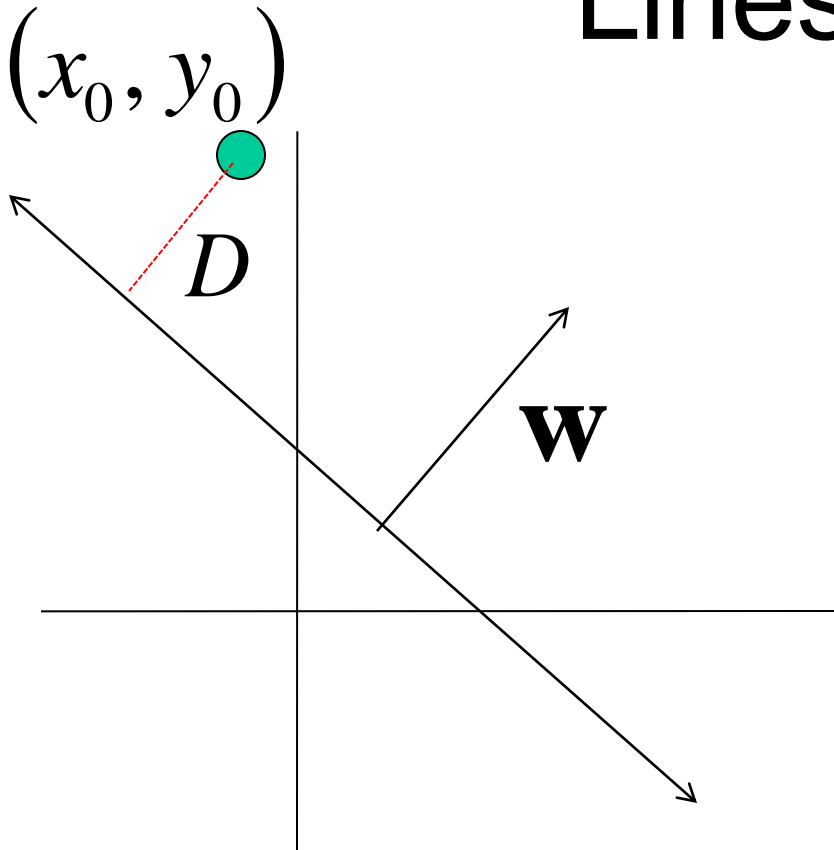
$$ax + cy + b = 0$$



$$\mathbf{w} \cdot \mathbf{x} + b = 0$$

# Lines in $\mathbb{R}^2$

Slope:  $-a/c$   
Y-intercept:  $-b/c$



Let  $\mathbf{w} = \begin{bmatrix} a \\ c \end{bmatrix}$   $\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}$

$$ax + cy + b = 0$$



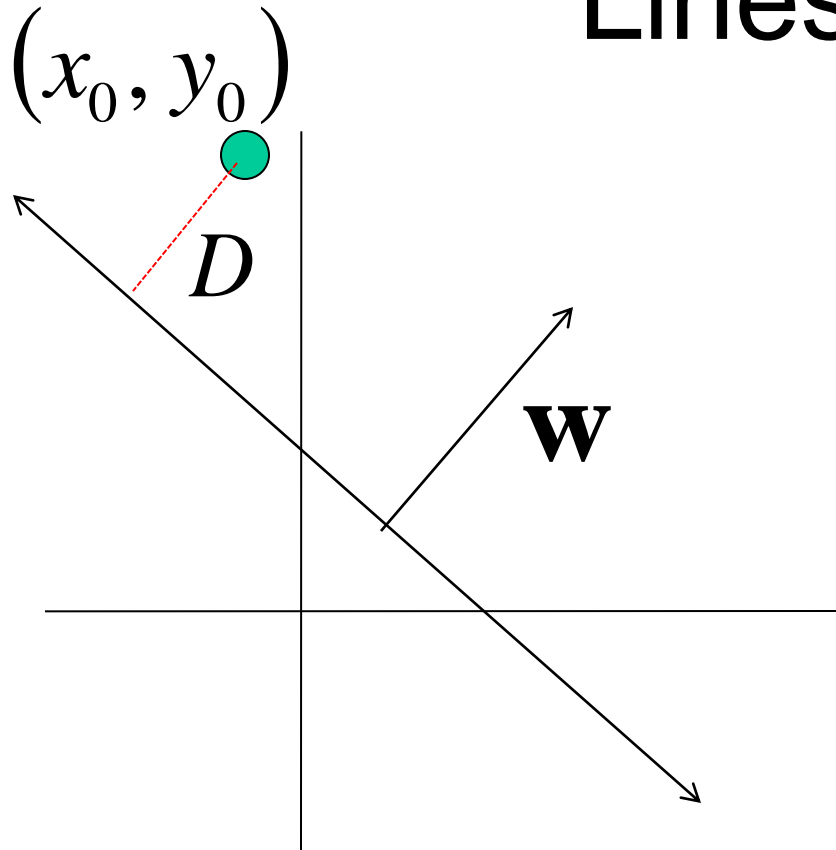
$$\mathbf{w} \cdot \mathbf{x} + b = 0$$

$$D = \frac{|ax_0 + cy_0 + b|}{\sqrt{a^2 + c^2}}$$

} distance from  
point to line

# Lines in $\mathbb{R}^2$

Slope:  $-a/c$   
Y-intercept:  $-b/c$



Let  $\mathbf{w} = \begin{bmatrix} a \\ c \end{bmatrix}$   $\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}$

$$ax + cy + b = 0$$



$$\mathbf{w} \cdot \mathbf{x} + b = 0$$

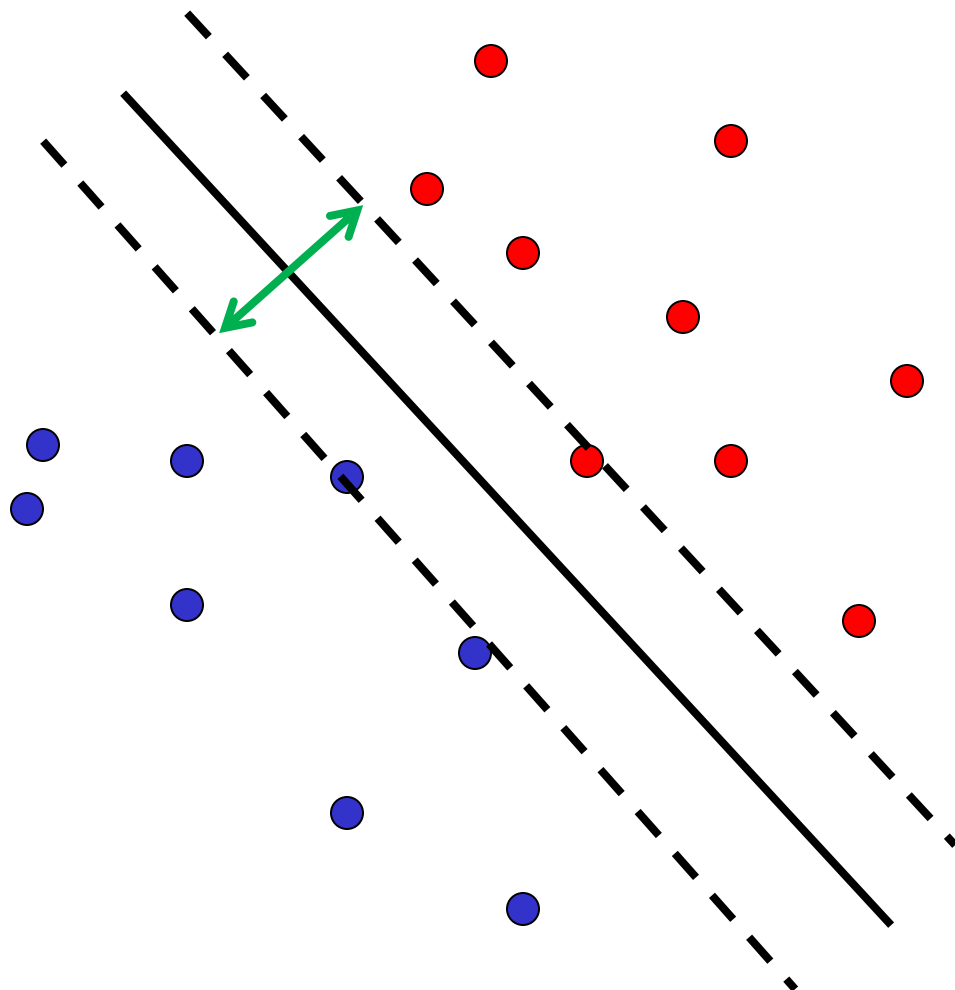
$$D = \frac{|ax_0 + cy_0 + b|}{\sqrt{a^2 + c^2}} = \frac{|\mathbf{w}^T \mathbf{x} + b|}{\|\mathbf{w}\|} \left. \vphantom{\frac{|\mathbf{w}^T \mathbf{x} + b|}{\|\mathbf{w}\|}} \right\} \text{distance from point to line}$$


- Find linear function to separate positive and negative examples



# Support vector machines

---

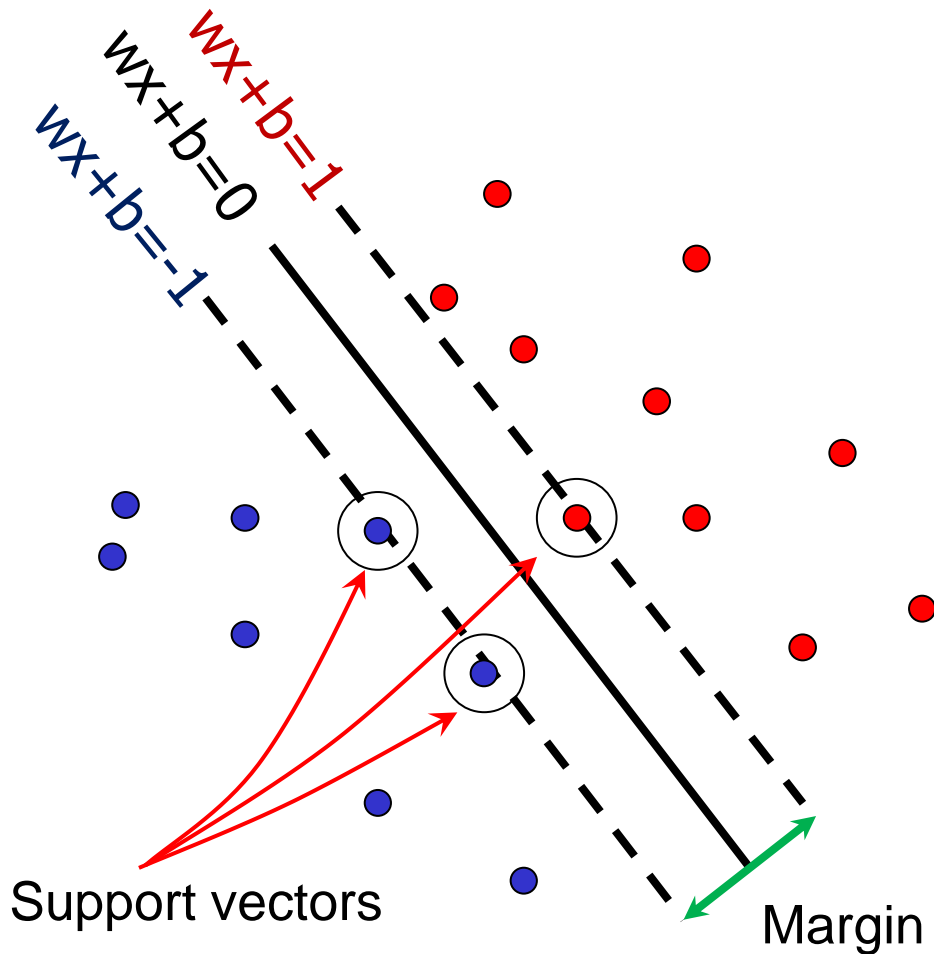


- Discriminative classifier based on *optimal separating line* (for 2d case)
- Maximize the *margin* between the positive and negative training examples 



# Support vector machines

- Want line that maximizes the margin. 



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

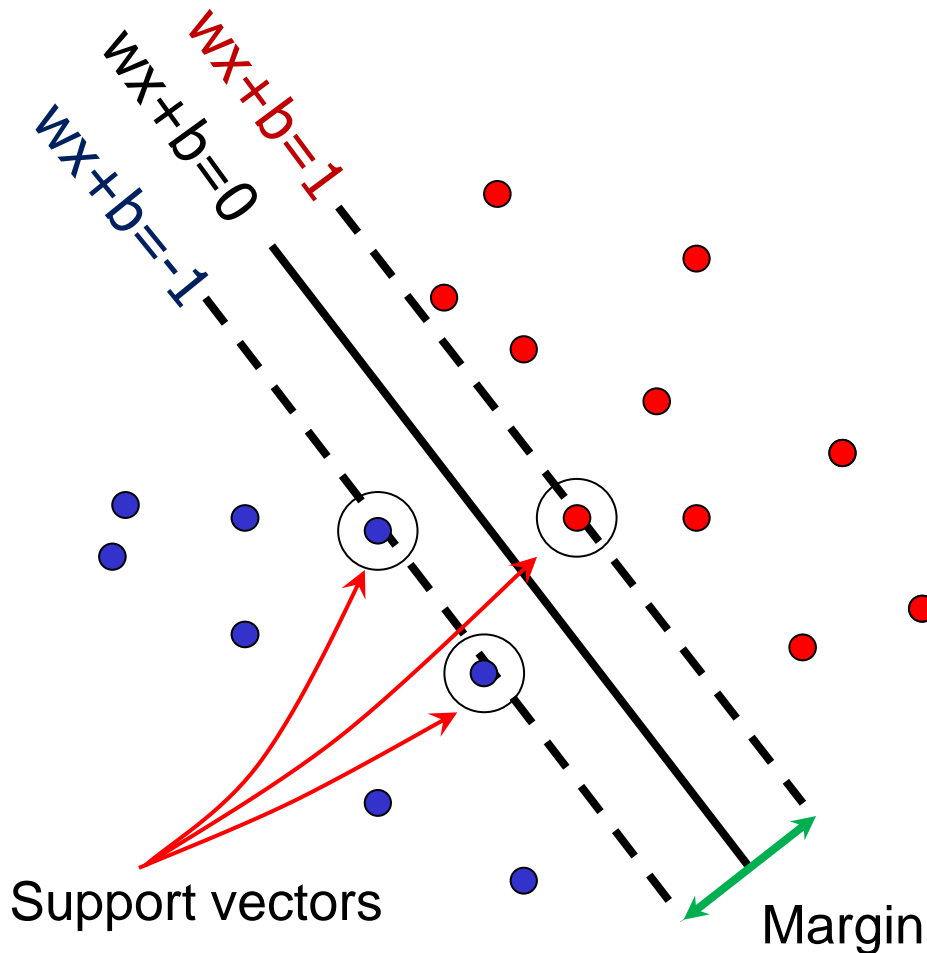
$$\mathbf{x}_i \text{ negative } (y_i = -1): \quad \mathbf{x}_i \cdot \mathbf{w} + b \leq -1$$

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



# Support vector machines

- Want line that maximizes the margin.



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

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

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

$$\text{Distance between point and line: } \frac{|\mathbf{x}_i \cdot \mathbf{w} + b|}{\|\mathbf{w}\|}$$

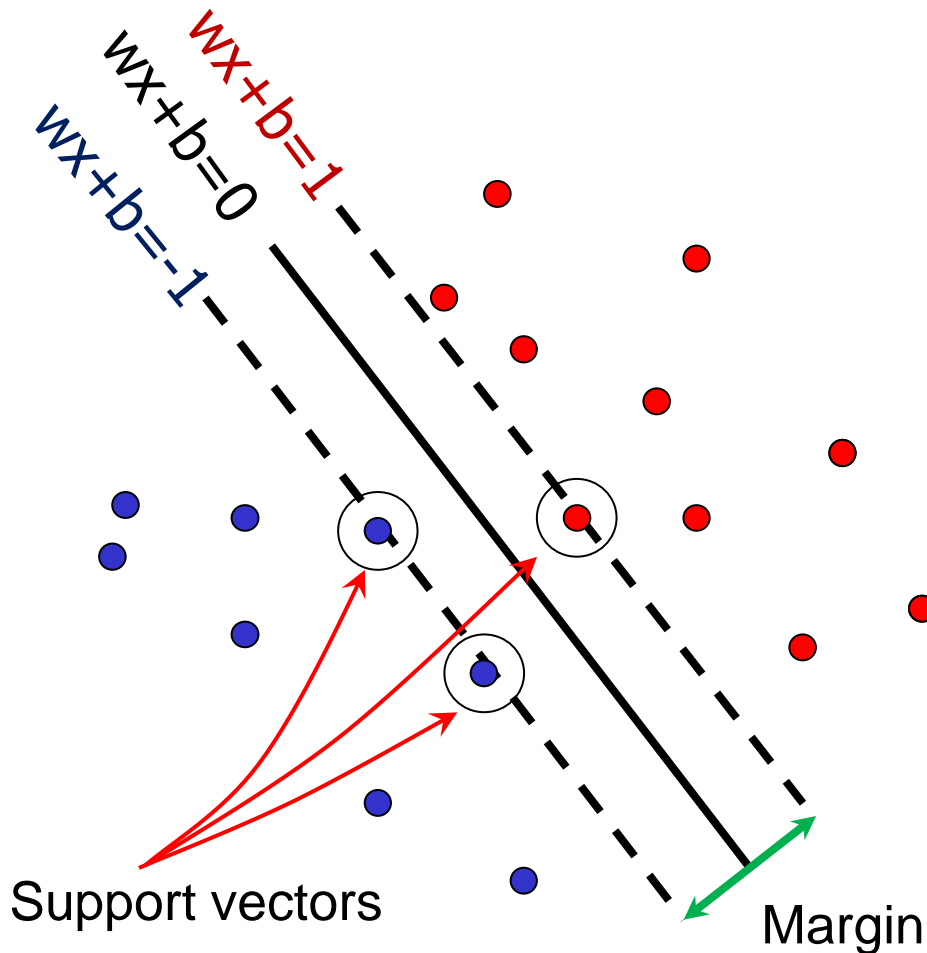
For support vectors:

$$\frac{\mathbf{w}^T \mathbf{x} + b}{\|\mathbf{w}\|} = \frac{\pm 1}{\|\mathbf{w}\|} \quad M = \left| \frac{1}{\|\mathbf{w}\|} - \frac{-1}{\|\mathbf{w}\|} \right| = \frac{2}{\|\mathbf{w}\|}$$



# Support vector machines

- Want line that maximizes the margin.



$\mathbf{x}_i$  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 line:  $\frac{|\mathbf{x}_i \cdot \mathbf{w} + b|}{\|\mathbf{w}\|}$

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

# Finding the maximum margin line


---


1. Maximize margin  $2/\|\mathbf{w}\|$
2. Correctly classify all training data points:


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

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

*Quadratic optimization problem:*



Minimize  $\frac{1}{2} \mathbf{w}^T \mathbf{w}$  

Subject to  $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$  

One constraint per training point.

Note sign trick: 

$$\mathbf{w} \cdot \mathbf{x}_i + b \geq 1 \quad (\text{if } y_i = 1)$$

$$\mathbf{w} \cdot \mathbf{x}_i + b \leq -1 \quad (\text{if } y_i = -1)$$
$$(-1) \mathbf{w} \cdot \mathbf{x}_i + b \geq 1$$

# Finding the maximum margin line

---

- Solution:  $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$


Learned  
weight

Support  
vector



# Finding the maximum margin line


---

- Solution:  $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$    
 $b = y_i - \mathbf{w} \cdot \mathbf{x}_i$  (for any support vector)

- Classification function:

$$\begin{aligned} f(x) &= \text{sign}(\mathbf{w} \cdot \mathbf{x} + b) \quad \text{comment icon} \\ &= \text{sign}\left(\sum_i \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x} + b\right) \end{aligned}$$

*If  $f(x) < 0$ , classify as negative, otherwise classify as positive.*

- Notice that it relies on an *inner product* between the test point  $\mathbf{x}$  and the support vectors  $\mathbf{x}_i$
- (Solving the optimization problem also involves computing the inner products  $\mathbf{x}_i \cdot \mathbf{x}_j$  between all pairs of training points) 


# Inner product

---

- The decision boundary for the SVM and its optimization depend on the inner product of two data points (vectors):

$$\mathbf{x}_i^T \mathbf{x}_j$$

$$\begin{aligned} f(x) &= \text{sign}(\mathbf{w} \cdot \mathbf{x} + b) \\ &= \text{sign}\left(\sum_i \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x} + b\right) \end{aligned}$$

- The inner product is equal 

$$(\mathbf{x}_i^T \mathbf{x}) = \|\mathbf{x}_i\| * \|\mathbf{x}_i\| \cos \theta$$



If the angle in between them is 0 then:

$$(\mathbf{x}_i^T \mathbf{x}) = \|\mathbf{x}_i\| * \|\mathbf{x}_i\|$$



If the angle between them is 90 then:

$$(\mathbf{x}_i^T \mathbf{x}) = 0$$

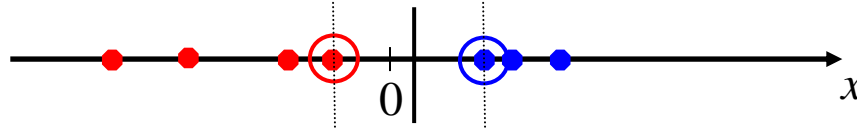
The inner product measures how similar the two vectors are



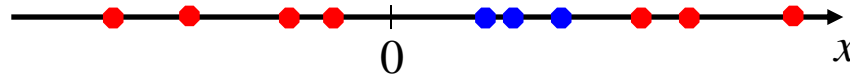
# Nonlinear SVMs

---

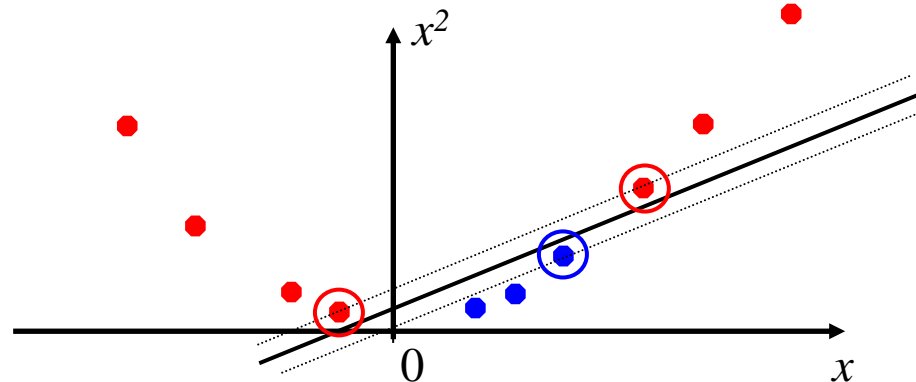
- Datasets that are linearly separable work out great:



- But what if the dataset is just too hard?



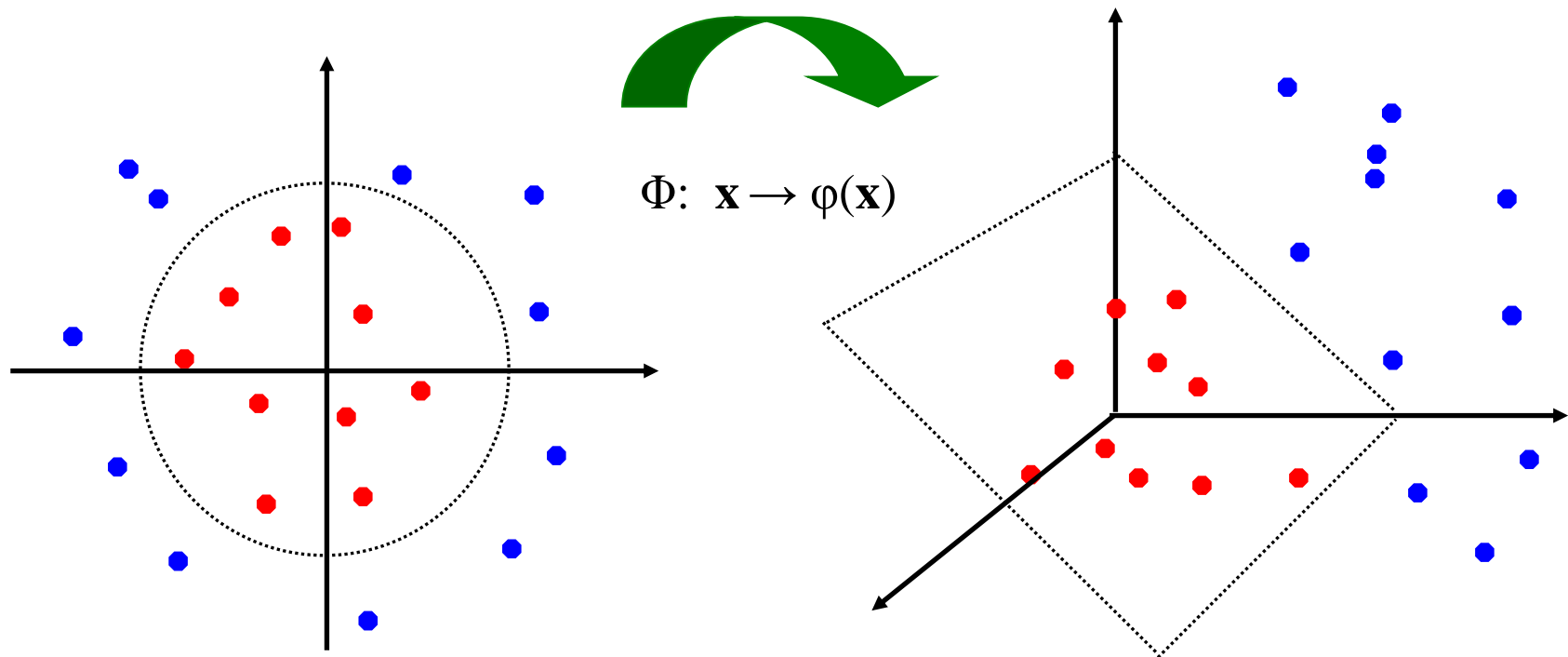
- We can map it to a higher-dimensional space:



# Nonlinear SVMs

---

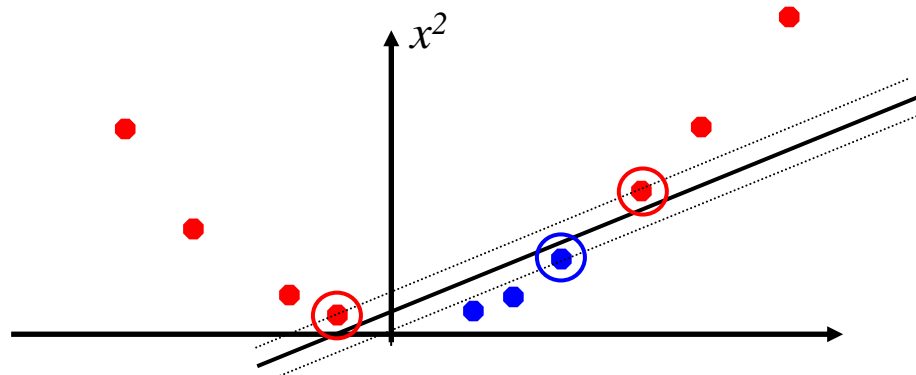
- General idea: the original input space can always be mapped to some higher-dimensional feature space where the training set is separable:



# Nonlinear kernel: Example

---

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



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

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

# The “Kernel Trick”

---

- The linear classifier relies on dot product between vectors  $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$
- If every data point is mapped into high-dimensional space via some transformation  $\Phi: \mathbf{x}_i \rightarrow \phi(\mathbf{x}_i)$ , the dot product becomes:  $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j)$
- A *kernel function* is similarity function that corresponds to an inner product in some expanded feature space
- *The kernel trick*: instead of explicitly computing the lifting transformation  $\phi(\mathbf{x})$ , define a kernel function  $K$  such that:  $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j)$



# Examples of kernel functions

---

- Linear:  $K(x_i, x_j) = x_i^T x_j$

- Polynomials of degree up to  $d$ :

$$K(x_i, x_j) = (x_i^T x_j + 1)^d$$

- Gaussian RBF:

$$K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$$

- Histogram intersection:

$$K(x_i, x_j) = \sum_k \min(x_i(k), x_j(k))$$

# The benefit of the “kernel trick”

---

- Example: Polynomial kernel for 2-dim features

$$\begin{aligned}k(\mathbf{x}, \mathbf{z}) &= (1 + \mathbf{x}^T \mathbf{z})^2 = (1 + x_1 z_1 + x_2 z_2)^2 \\&= 1 + 2x_1 z_1 + 2x_2 z_2 + x_1^2 z_1^2 + 2x_1 z_1 x_2 z_2 + x_2^2 z_2^2 \\&= (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, \sqrt{2}x_1 x_2, x_2^2)(1, \sqrt{2}z_1, \sqrt{2}z_2, z_1^2, \sqrt{2}z_1 z_2, z_2^2)^T \\&= \phi(\mathbf{x})^T \phi(\mathbf{z}).\end{aligned}\tag{7.42}$$

- ... lives in 6 dimensions
- With the kernel trick, we directly compute an inner product in 2-dim space, obtaining a scalar that we add 1 to and exponentiate

# Hard-margin SVMs

---

$$\min_{\mathbf{w}} \quad \underbrace{\frac{1}{2} \|\mathbf{w}\|^2}_{\text{Maximize margin}}$$

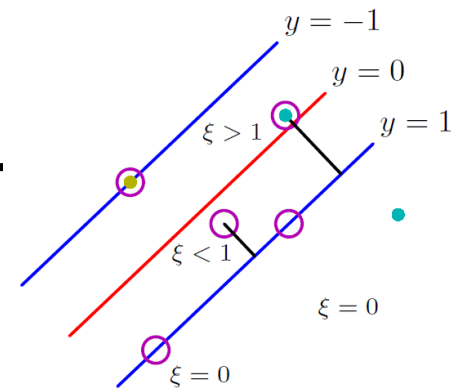
The  $\mathbf{w}$  that minimizes...

Maximize margin

$$\text{subject to} \quad y_i \mathbf{w}^T \mathbf{x}_i \geq 1, \quad \forall i = 1, \dots, N$$



# Soft-margin SVMs



The  $w$  that minimizes...

$$\min_w \underbrace{\frac{1}{2} \|\mathbf{w}\|^2}_{\text{Maximize margin}} + \underbrace{C \sum_{i=1}^N \xi_i}_{\text{Minimize misclassification}}$$

Annotations in the equation:

- $\min_w$ : The  $w$  that minimizes...
- $\frac{1}{2} \|\mathbf{w}\|^2$ : Maximize margin
- $C$ : Misclassification cost
- $N$ : # data samples
- $\xi_i$ : Slack variable
- $\sum_{i=1}^N$ : Minimize misclassification


subject to

$$y_i \mathbf{w}^T \mathbf{x}_i \geq 1 - \xi_i,$$

$$\xi_i \geq 0, \quad \forall i = 1, \dots, N$$

# 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 the test example, and assign it to 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

# Multi-class problems

---

## One-vs-all (a.k.a. one-vs-others)

- Train  $K$  classifiers
- In each, pos = data from class  $i$ , neg = data from classes other than  $i$
- The class with the most confident prediction wins 
- Example:
  - You have 4 classes, train 4 classifiers 
  - 1 vs others: score 3.5
  - 2 vs others: score 6.2
  - 3 vs others: score 1.4
  - 4 vs other: score 5.5
  - Final prediction: class 2 



# Multi-class problems

---

## One-vs-one (a.k.a. all-vs-all)

- Train  $K(K-1)/2$  binary classifiers (all pairs of classes)
- They all vote for the label
- Example:
  - You have 4 classes, then train 6 classifiers
  - 1 vs 2, 1 vs 3, 1 vs 4, 2 vs 3, 2 vs 4, 3 vs 4
  - Votes: 1, 1, 4, 2, 4, 4
  - Final prediction is class 4



# Using SVMs

---




1. Select a kernel function.
2. Compute pairwise kernel values between labeled examples.
3. Use this “kernel matrix” to solve for SVM support vectors & alpha weights.
4. To classify a new example: compute kernel values between new input and support vectors, apply alpha weights, check sign of output.

# Some SVM packages

---

- LIBSVM <http://www.csie.ntu.edu.tw/~cjlin/libsvm/>
- LIBLINEAR  
<https://www.csie.ntu.edu.tw/~cjlin/liblinear/>
- SVM Light <http://svmlight.joachims.org/>

# Linear classifiers vs nearest neighbors

- Linear pros:  
  - + Low-dimensional *parametric* representation
  - + Very fast at test time
- Linear cons:
  - Can be tricky to select best kernel function for a problem
  - Learning can take a very long time for large-scale problem
- NN pros:
  - + Works for any number of classes
  - + Decision boundaries not necessarily linear
  - + *Nonparametric* method 
  - + Simple to implement
- NN cons:
  - Slow at test time (large search problem to find neighbors)
  - Storage of data
  - Especially need good distance function (but true for all classifiers)



# Training vs Testing

- What do we want?
  - High accuracy on training data?
  - No, high accuracy on *unseen/new/test data*!
  - Why is this tricky?
- Training data
  - Features (x) and labels (y) used to learn mapping f
- Test data
  - Features (x) used to make a prediction
  - Labels (y) only used to see how well we've learned f!!!
- Validation data
  - Held-out set of the *training data*
  - Can use both features (x) and labels (y) to tune parameters of the model we're learning

# Generalization





Training set (labels known)



Test set (labels unknown)

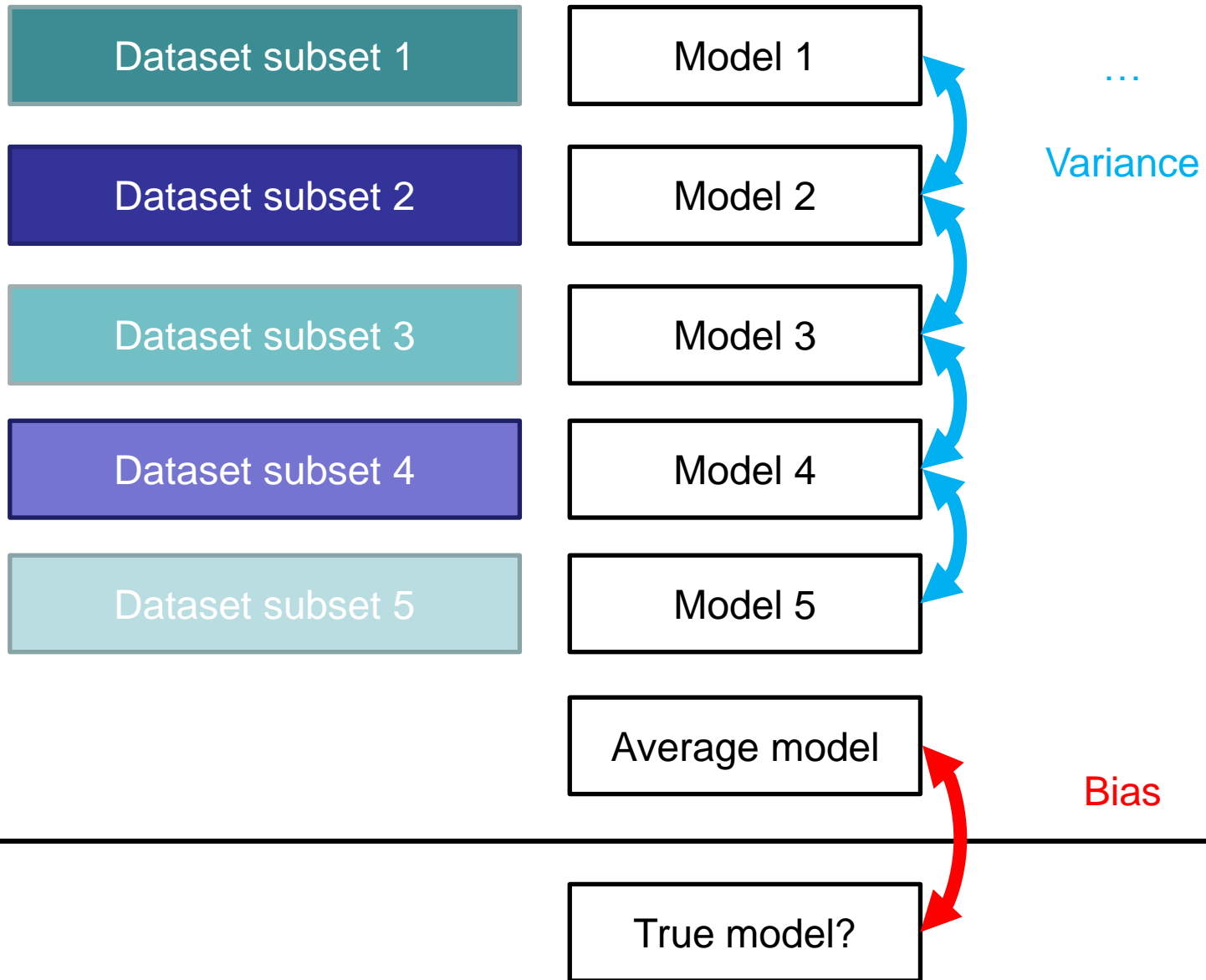
- How well does a learned model generalize from the data it was trained on to a new test set?

# Generalization

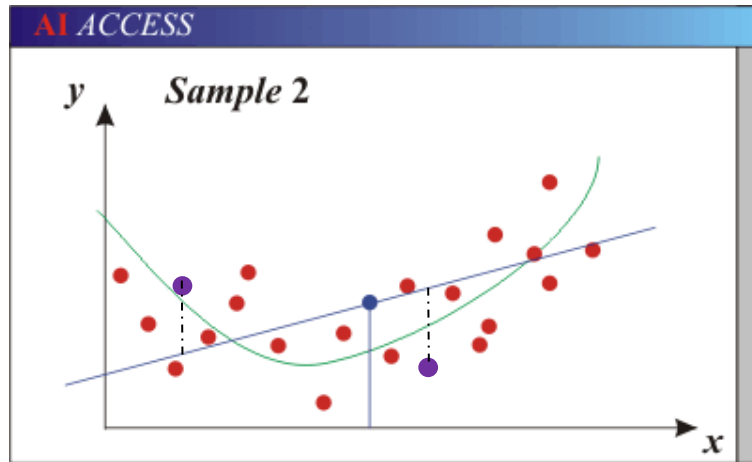
- Components of generalization error
  - **Noise** in our observations: unavoidable
  - **Bias**: due to inaccurate assumptions/simplifications by model 
  - **Variance**: models estimated from different training sets differ greatly from each other
- **Underfitting**: model is too “simple” to represent 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



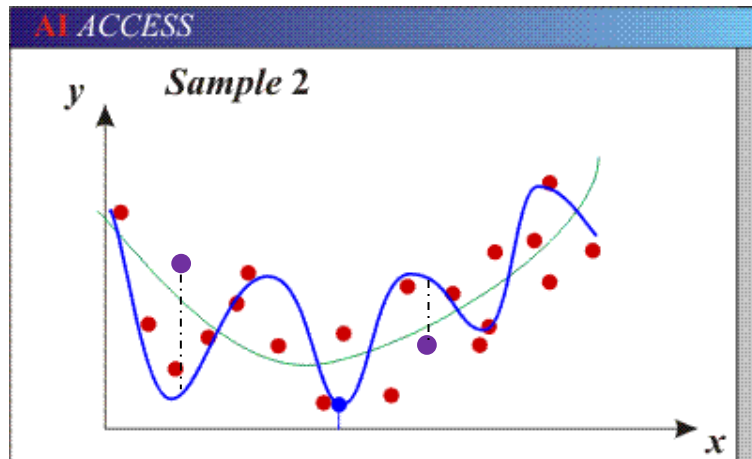
# Generalization



# Generalization



- 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).

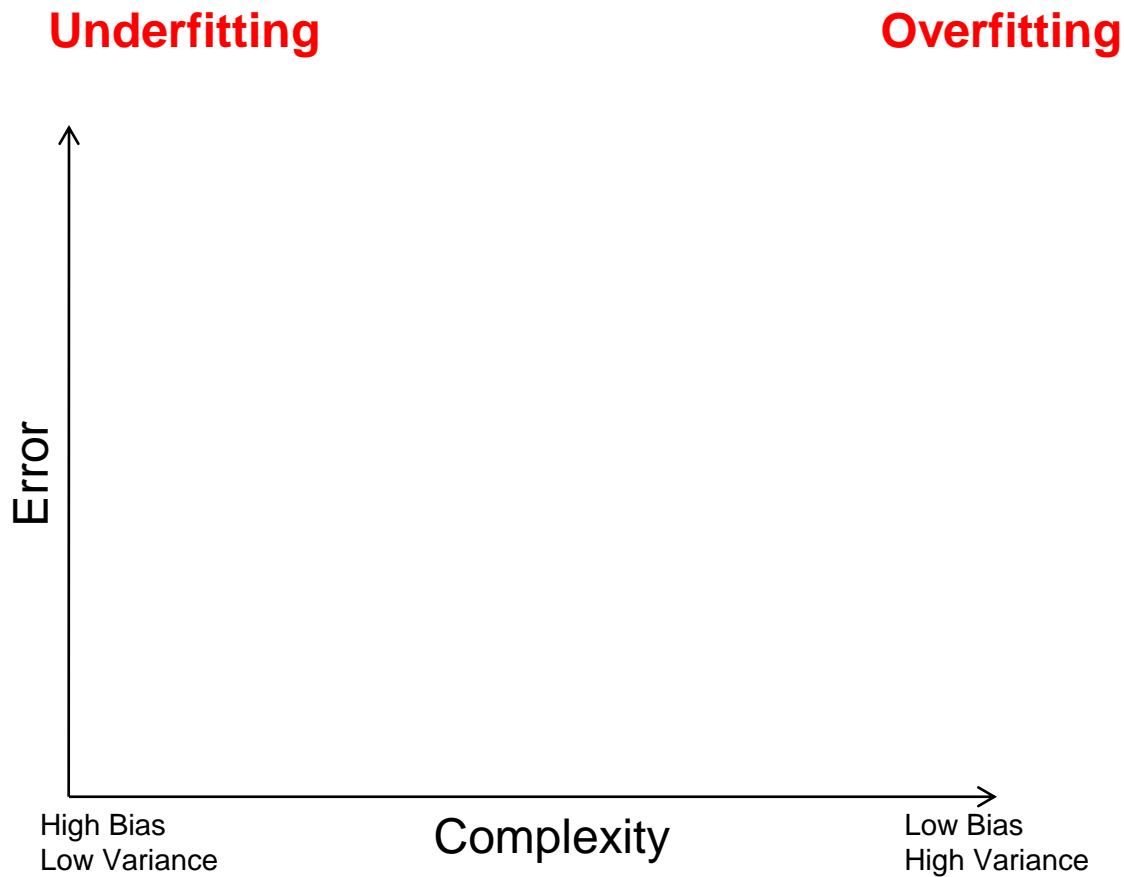
Purple dots = possible test points

Red dots = training data (all that we see before we ship off our model!)

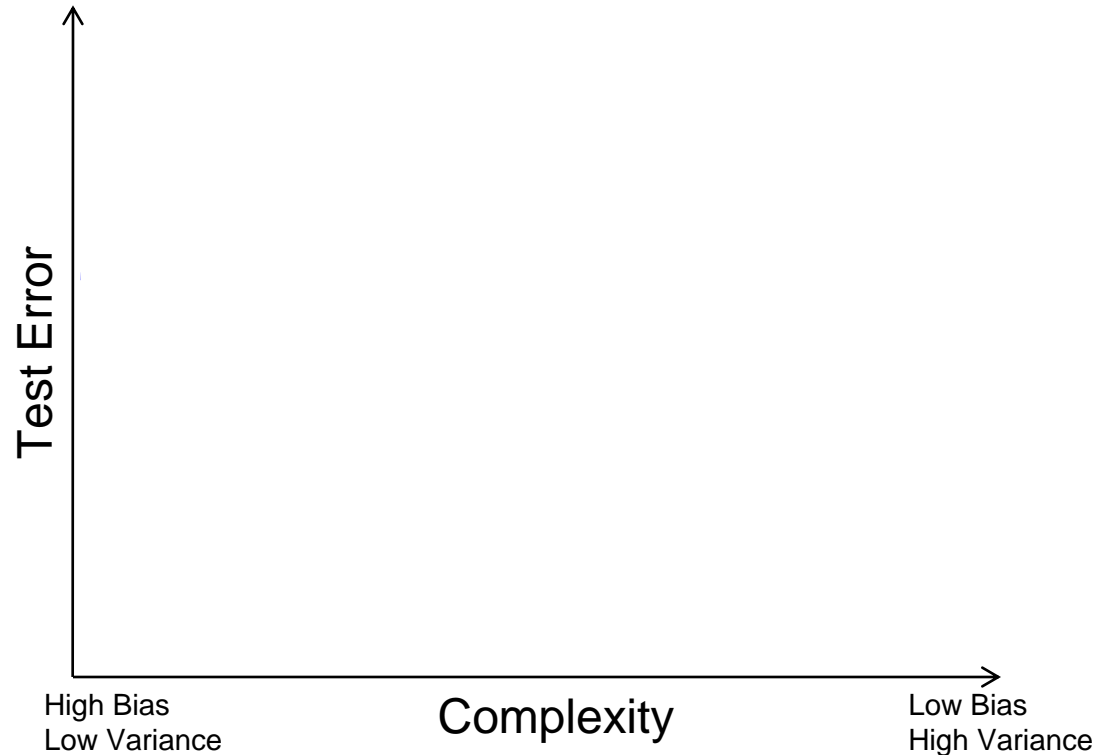
Green curve = true underlying model

Blue curve = our predicted model/fit

# Training vs test error



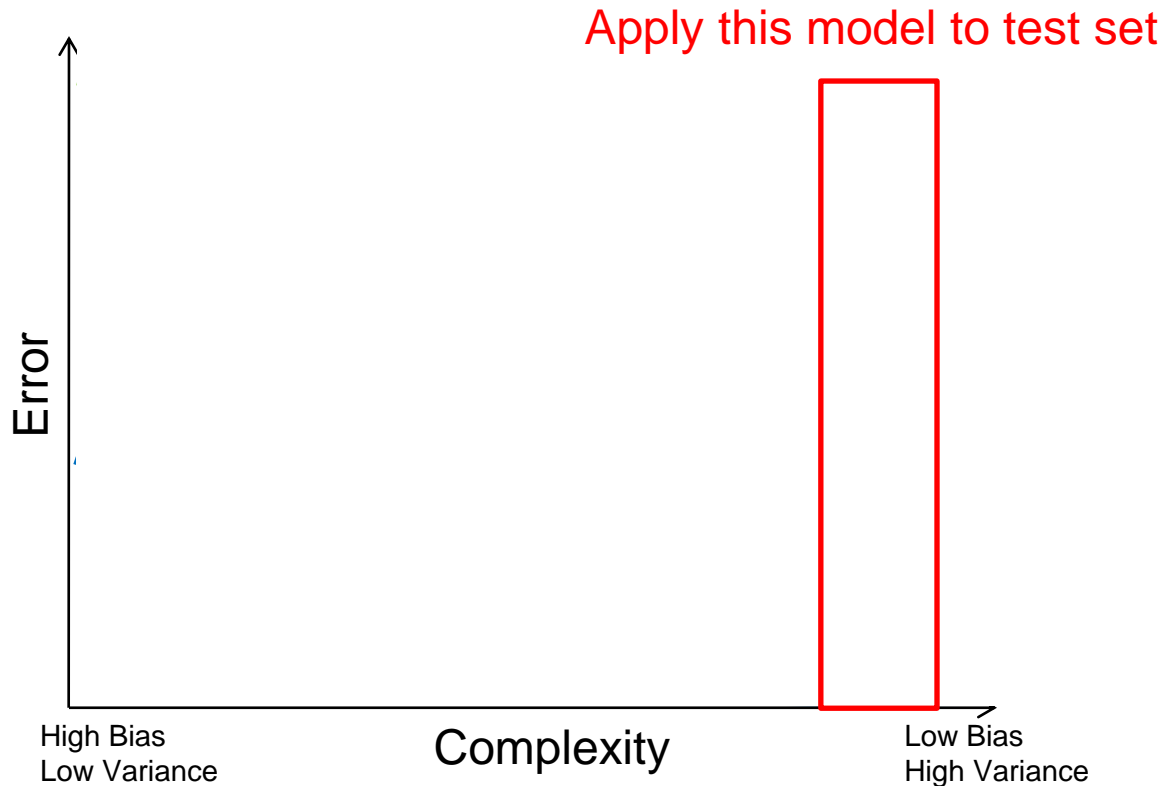
# The effect of training set size





# Choosing the trade-off between bias and variance

- Need validation set (separate from the test set)



# Generalization tips

- 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
- As an additional technique for reducing variance, try *regularizing* the parameters (penalize high magnitude weights)



# Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories

CVPR 2006

Winner of 2016  
Longuet-Higgins Prize

Svetlana Lazebnik (slazebni@uiuc.edu)

Beckman Institute, University of Illinois at Urbana-Champaign

Cordelia Schmid (cordelia.schmid@inrialpes.fr)

INRIA Rhône-Alpes, France

Jean Ponce (ponce@di.ens.fr)

Ecole Normale Supérieure, France

# Scene category dataset

Fei-Fei & Perona (2005), Oliva & Torralba (2001)

[http://www-cvr.ai.uiuc.edu/ponce\\_grp/data](http://www-cvr.ai.uiuc.edu/ponce_grp/data)



office



kitchen



living room



bedroom



store



industrial



tall building



inside city



street



highway



coast



open country



mountain



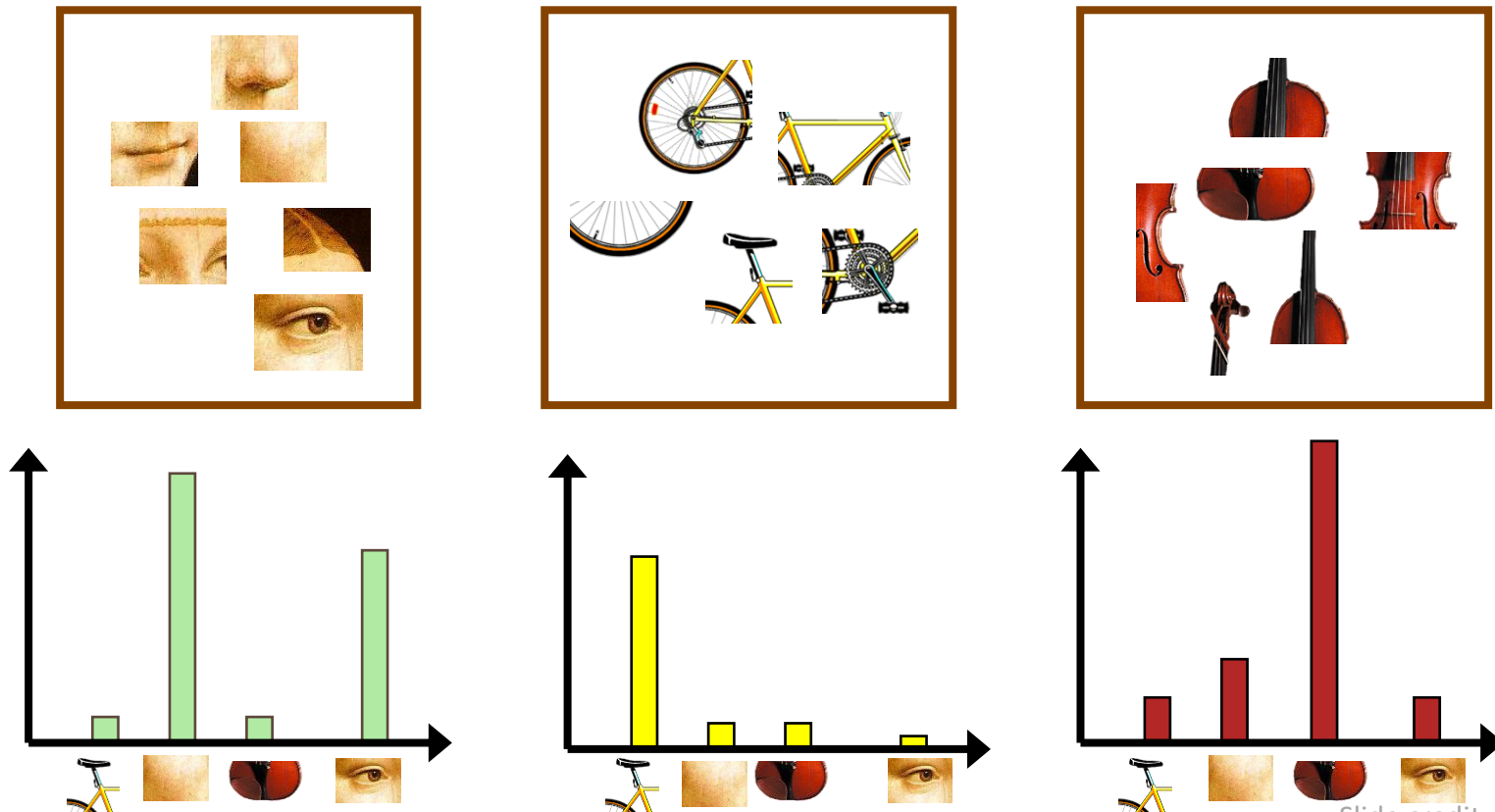
forest



suburb

# Bag-of-words representation

1. Extract local features
2. Learn “visual vocabulary” using clustering
3. Quantize local features using visual vocabulary
4. Represent images by frequencies of “visual words”



# Image categorization with bag of words

## Training

1. Compute bag-of-words representation for training images
2. Train classifier on labeled examples using histogram values as features
3. Labels are the scene types (e.g. mountain vs field)

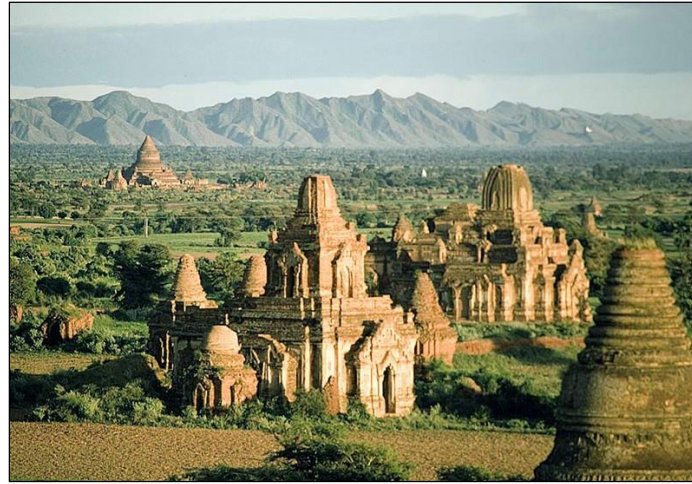
## Testing

1. Extract keypoints/descriptors for test images
2. Quantize into visual words **using the clusters computed at training time**
3. Compute visual word histogram for test images
4. Compute labels on test images using classifier obtained at training time
5. **Evaluation only, do only once:** Measure accuracy of test predictions by comparing them to ground-truth test labels (obtained from humans)

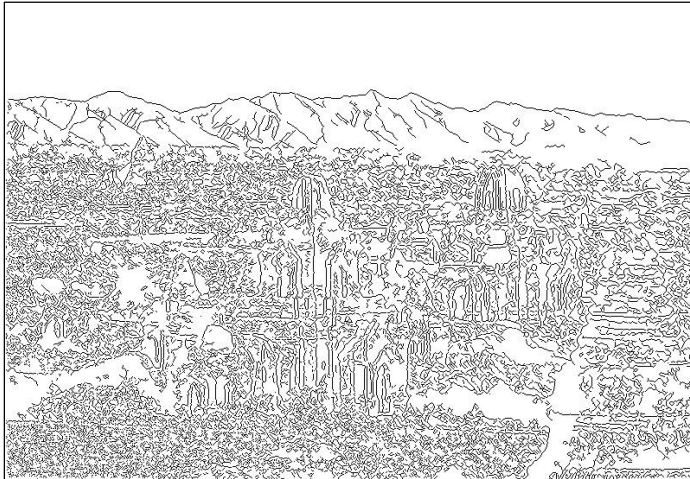




# Feature extraction (on which BOW is based)

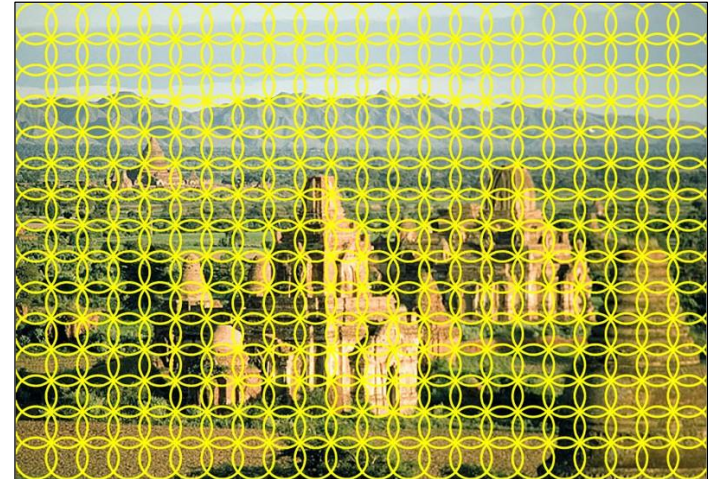


Weak features



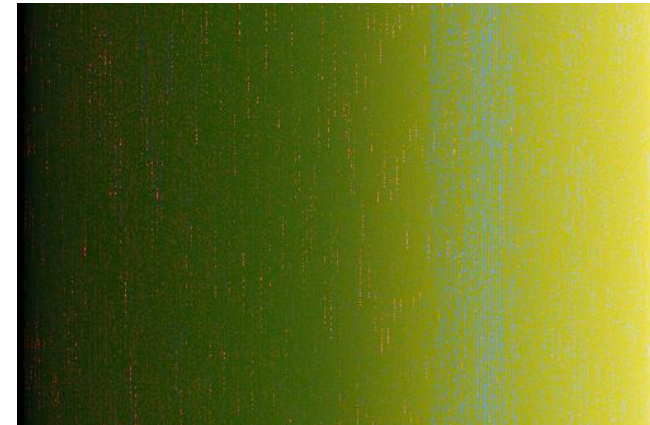
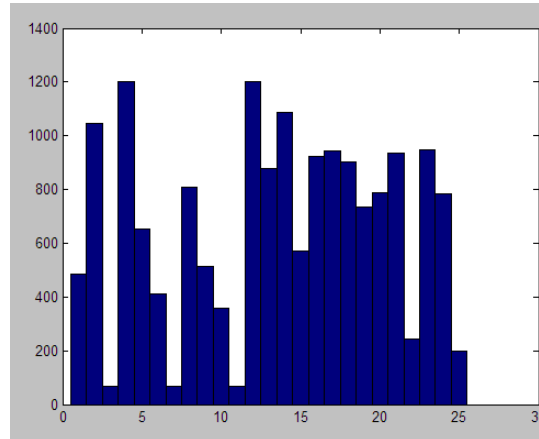
Edge points at 2 scales and 8 orientations  
(vocabulary size 16)

Strong features



SIFT descriptors of 16x16 patches sampled  
on a regular grid, quantized to form visual  
vocabulary (size 200, 400)

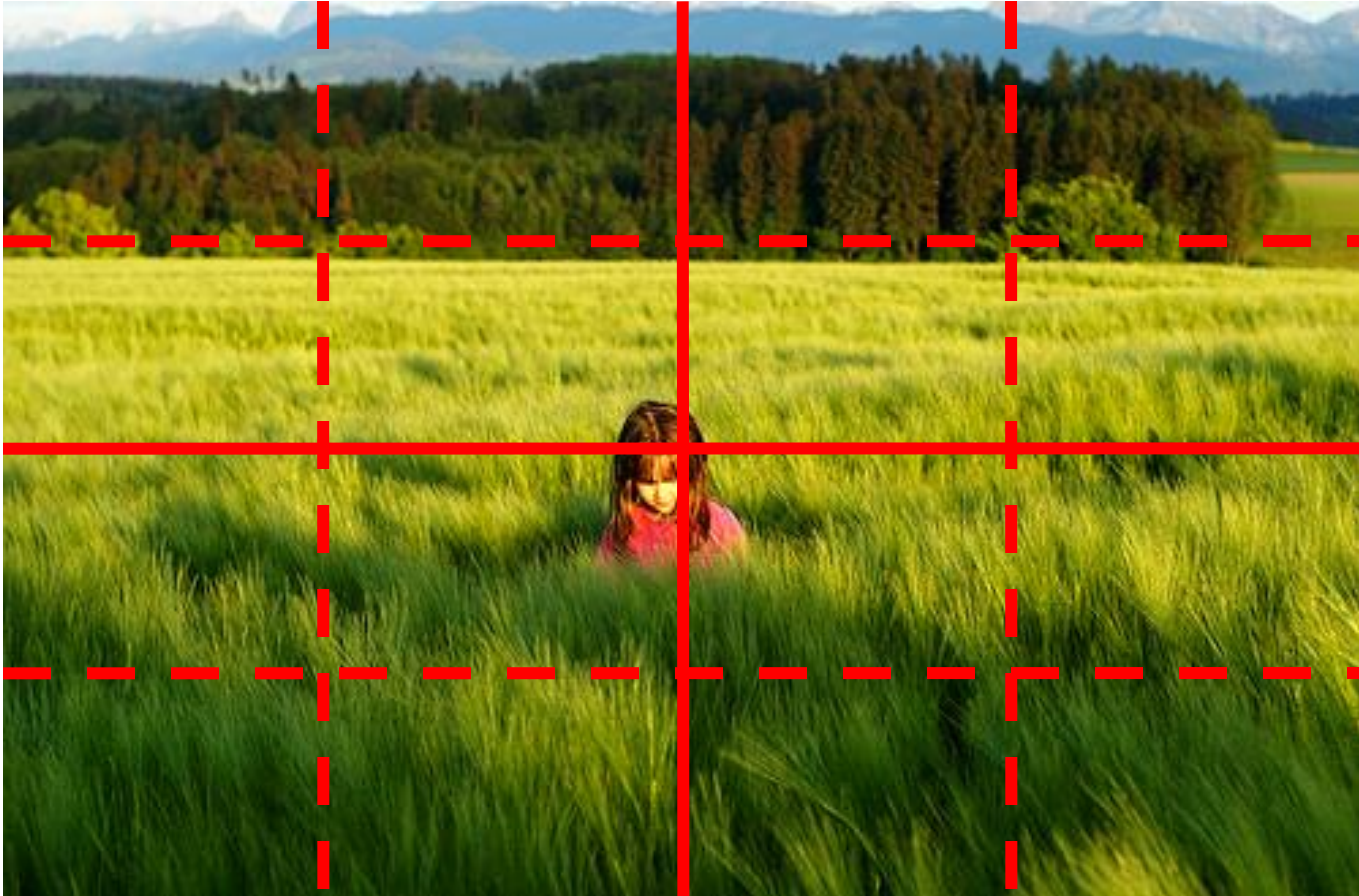
# What about spatial layout?



All of these images have the same color histogram



# Spatial pyramid

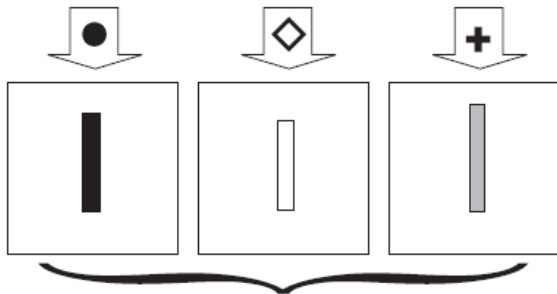
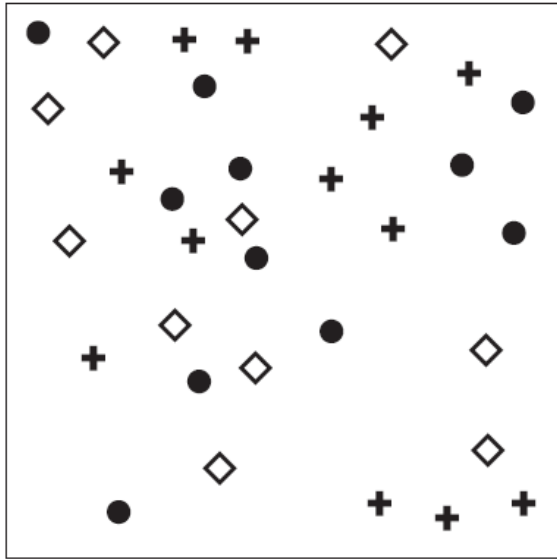


Compute histogram in each spatial bin

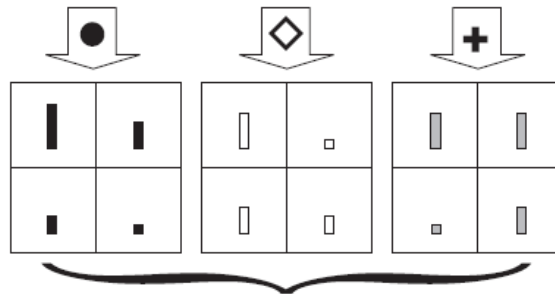
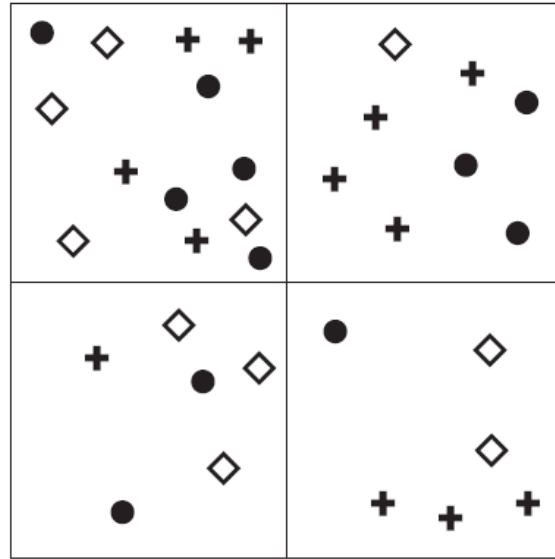
# Spatial pyramid



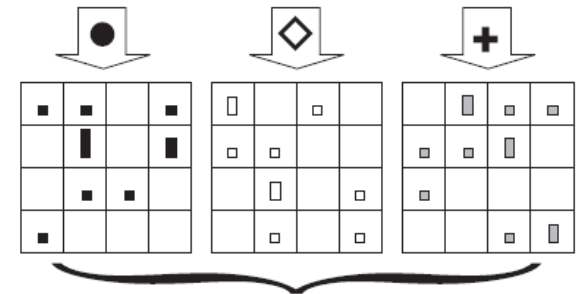
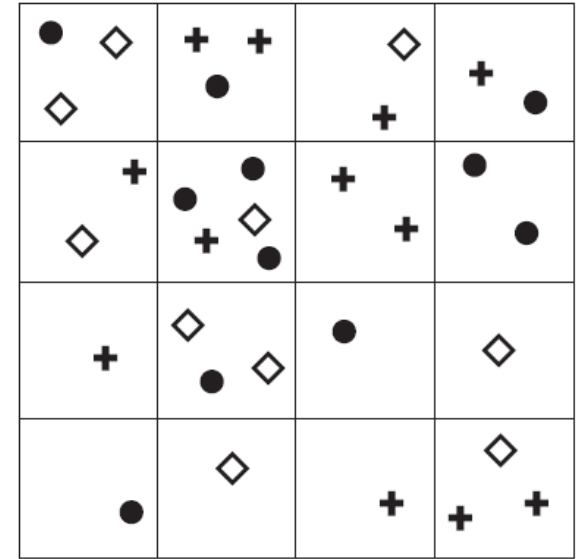
level 0



level 1



level 2



[[Lazebnik et al. CVPR 2006](#)]

# Pyramid matching

Indyk & Thaper (2003), Grauman & Darrell (2005)

Matching using pyramid and histogram intersection for some particular visual word:

Original images



$x_i$



$x_j$

Feature histograms:

Level 3

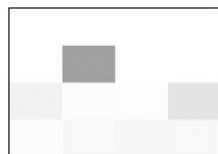


$\cap$

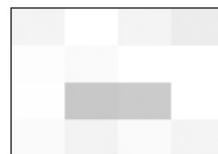


$= \mathcal{I}_3$

Level 2



$\cap$



$= \mathcal{I}_2$

Level 1



$\cap$



$= \mathcal{I}_1$



Level 0



$\cap$



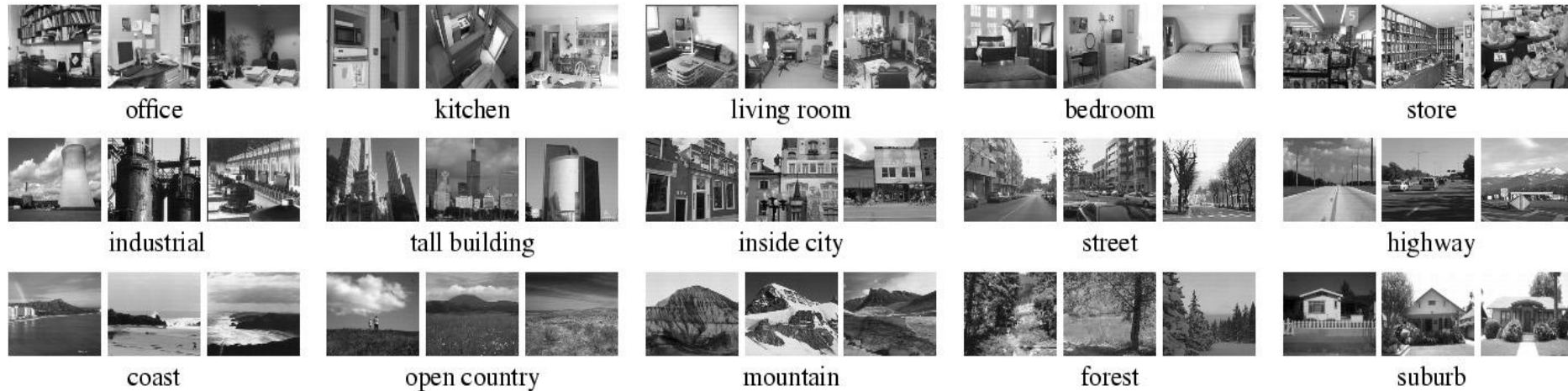
$= \mathcal{I}_0$

$$K(x_i, x_j) \text{ (value of pyramid match kernel): } \mathcal{I}_3 + \frac{1}{2}(\mathcal{I}_2 - \mathcal{I}_3) + \frac{1}{4}(\mathcal{I}_1 - \mathcal{I}_2) + \frac{1}{8}(\mathcal{I}_0 - \mathcal{I}_1)$$

# Scene category dataset

Fei-Fei & Perona (2005), Oliva & Torralba (2001)

[http://www-cvr.ai.uiuc.edu/ponce\\_grp/data](http://www-cvr.ai.uiuc.edu/ponce_grp/data)

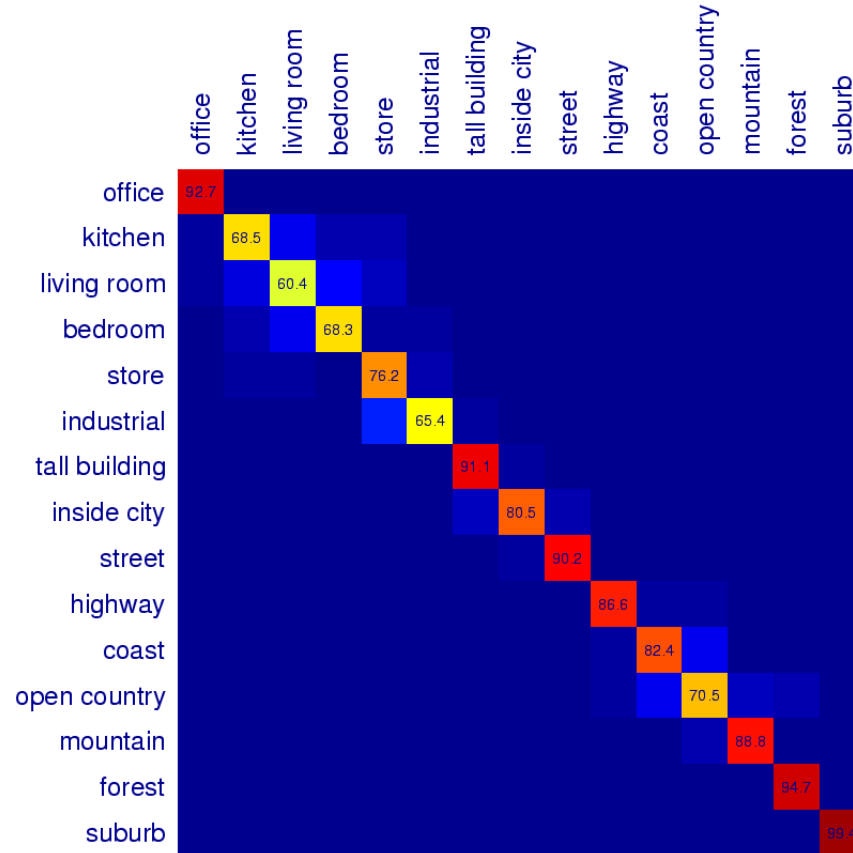


## Multi-class classification results (100 training images per class)

	Weak features (vocabulary size: 16)		Strong features (vocabulary size: 200)	
Level	Single-level	Pyramid	Single-level	Pyramid
0 ( $1 \times 1$ )	45.3 $\pm$ 0.5		72.2 $\pm$ 0.6	
1 ( $2 \times 2$ )	53.6 $\pm$ 0.3	56.2 $\pm$ 0.6	77.9 $\pm$ 0.6	79.0 $\pm$ 0.5
2 ( $4 \times 4$ )	61.7 $\pm$ 0.6	64.7 $\pm$ 0.7	79.4 $\pm$ 0.3	<b>81.1 <math>\pm</math>0.3</b>
3 ( $8 \times 8$ )	63.3 $\pm$ 0.8	<b>66.8 <math>\pm</math>0.6</b>	77.2 $\pm$ 0.4	80.7 $\pm$ 0.3

Fei-Fei & Perona: 65.2%

# Scene category confusions



## Difficult indoor images



kitchen



living room



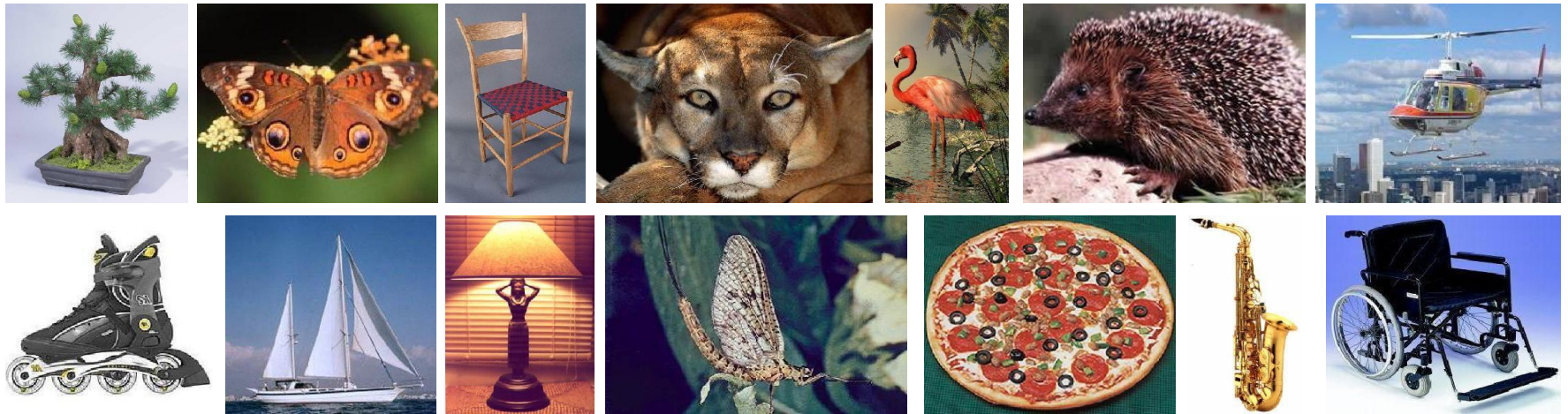
bedroom



# Caltech101 dataset

Fei-Fei et al. (2004)

[http://www.vision.caltech.edu/Image\\_Datasets/Caltech101/Caltech101.html](http://www.vision.caltech.edu/Image_Datasets/Caltech101/Caltech101.html)



## Multi-class classification results (30 training images per class)

	Weak features (16)		Strong features (200)	
Level	Single-level	Pyramid	Single-level	Pyramid
0	15.5 $\pm$ 0.9		41.2 $\pm$ 1.2	
1	31.4 $\pm$ 1.2	32.8 $\pm$ 1.3	55.9 $\pm$ 0.9	57.0 $\pm$ 0.8
2	47.2 $\pm$ 1.1	49.3 $\pm$ 1.4	63.6 $\pm$ 0.9	<b>64.6</b> $\pm$ 0.8
3	52.2 $\pm$ 0.8	<b>54.0</b> $\pm$ 1.1	60.3 $\pm$ 0.9	64.6 $\pm$ 0.7

