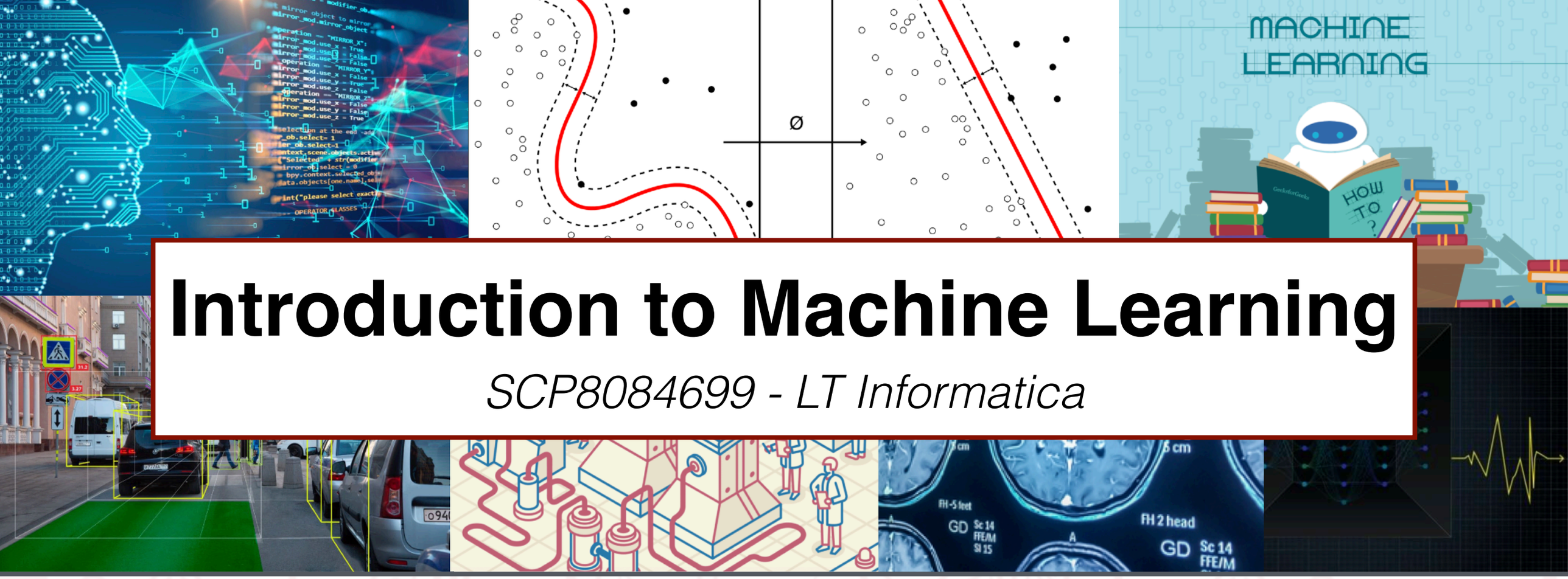


Introduction to Machine Learning

SCP8084699 - LT Informatica

Non-parametric Models, kNN

Prof. Lamberto Ballan

University of Padova, A.Y. 2022/23 May 16, 2023

DIPARTIMENTO MATEMATICA
Department of Mathematics “Tullio Levi-Civita”

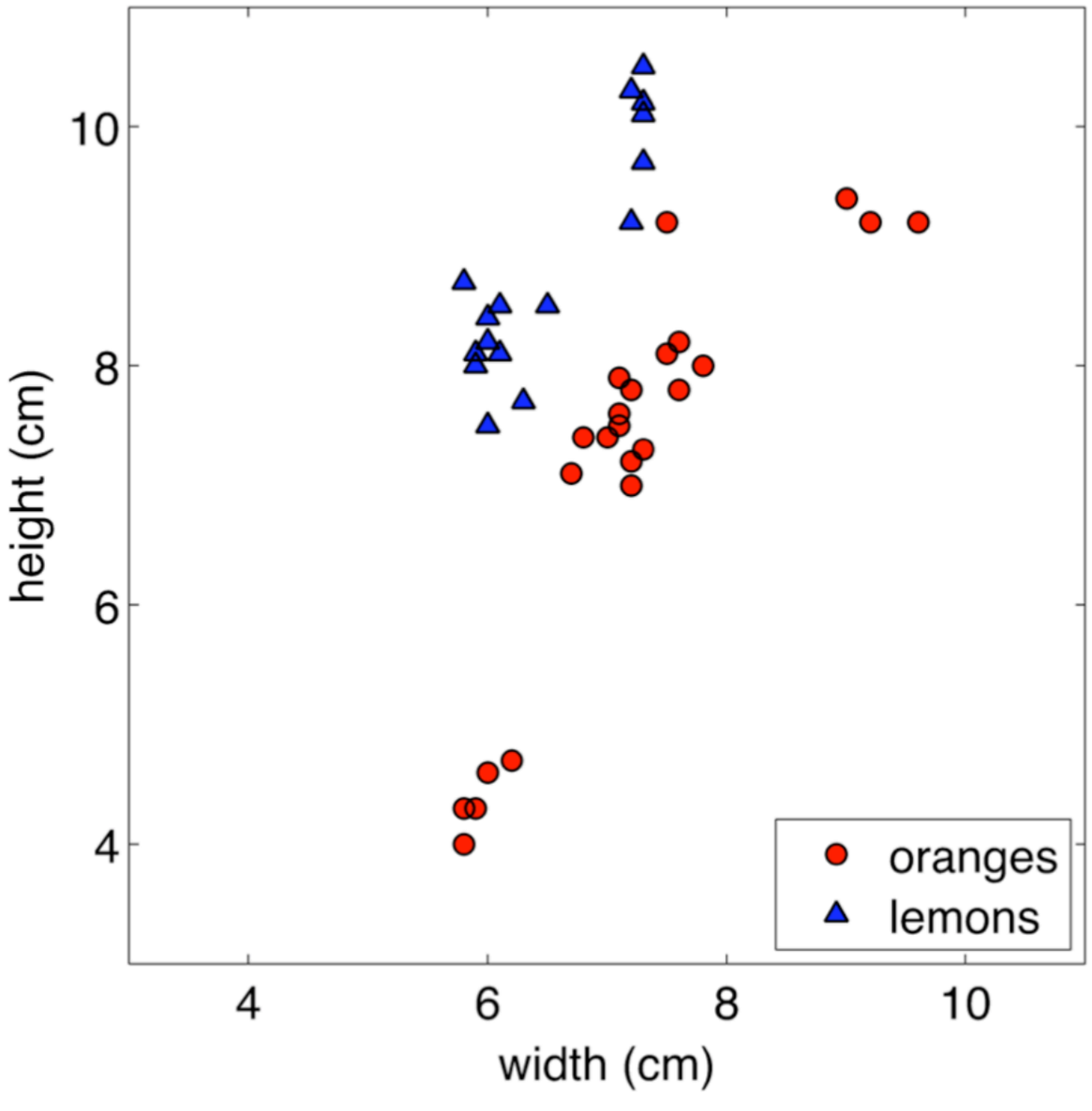
Non-parametric Models

- Here we refer to machine learning methods that don’t require many parameters
  - Mostly by "non-parametric models” we refer to nearest neighbors methods
  - These approaches are often referred as to lazy learning since they do not build any model
  - All the work is done at prediction time

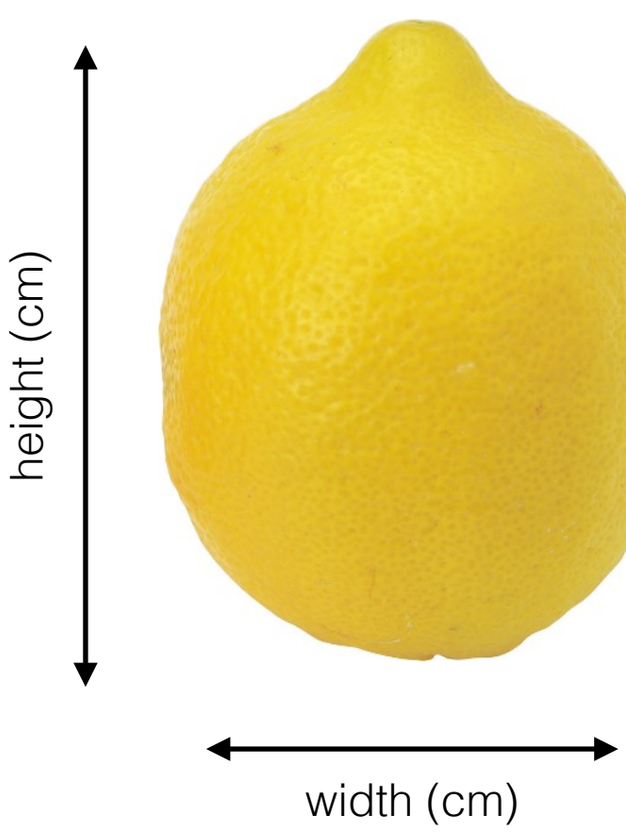
Note: we will mainly focus on classification, but these approaches can also be used for regression

Another look at linear classifiers

- An example: “oranges” vs “lemons”



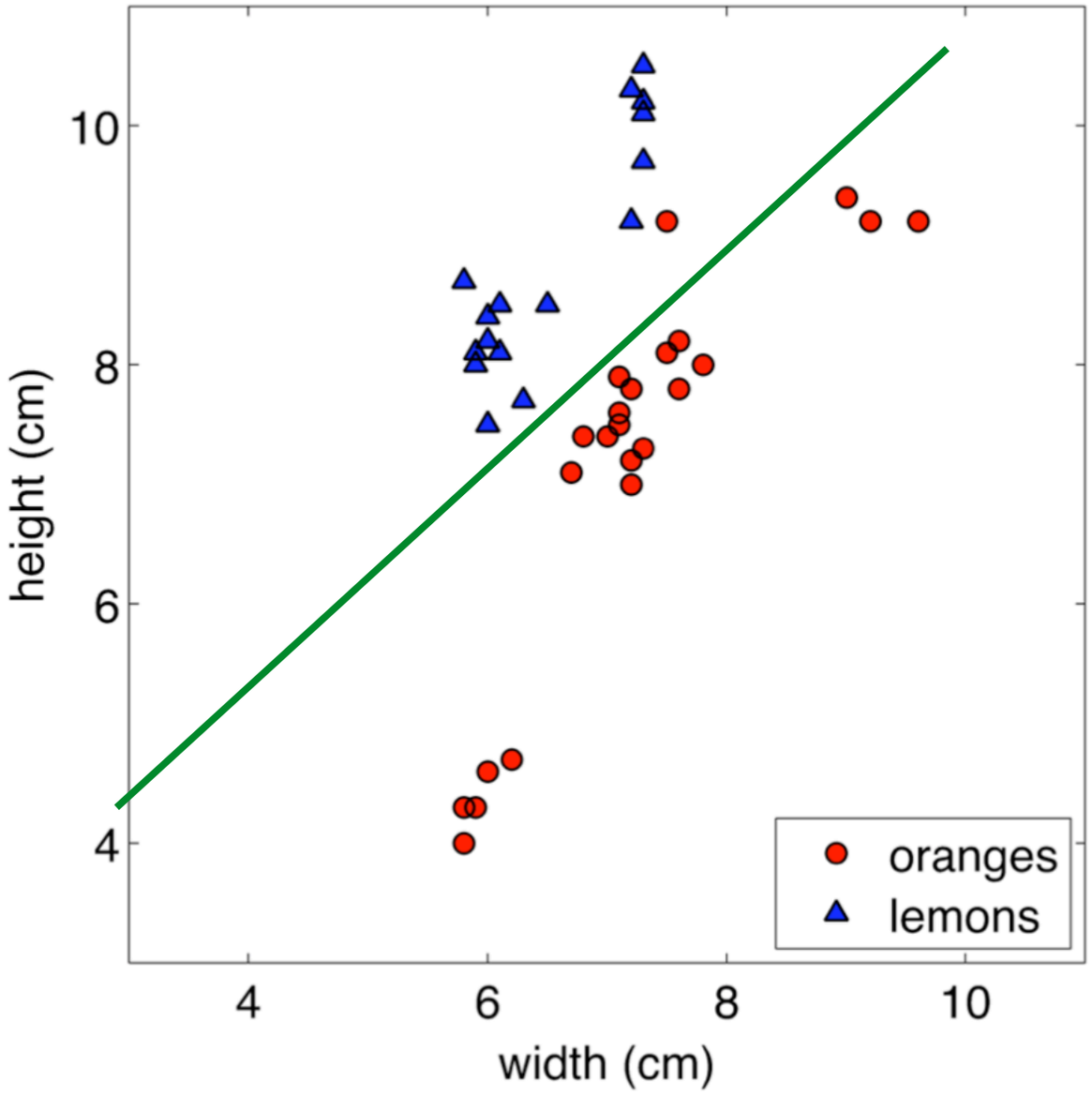
Binary classifier based on two simple features:



Adapted from slide by S. Fidler

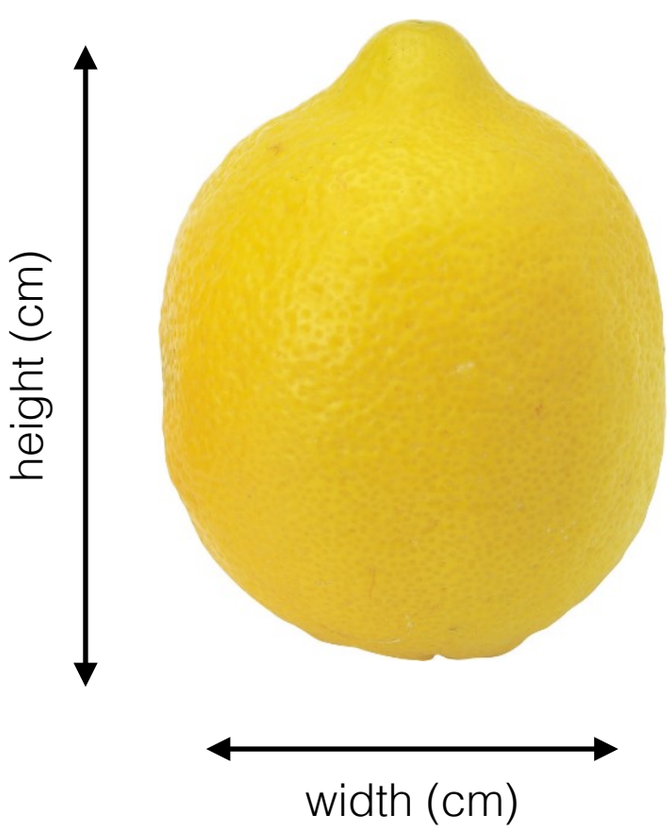
Another look at linear classifiers

- An example: “oranges” vs “lemons”



We can construct simple linear decision boundary:

$$y = \text{sign}(h_\theta(\mathbf{x})), \quad h_\theta(\mathbf{x}) = \boldsymbol{\theta}^{\mathrm{T}}\,\mathbf{x}$$



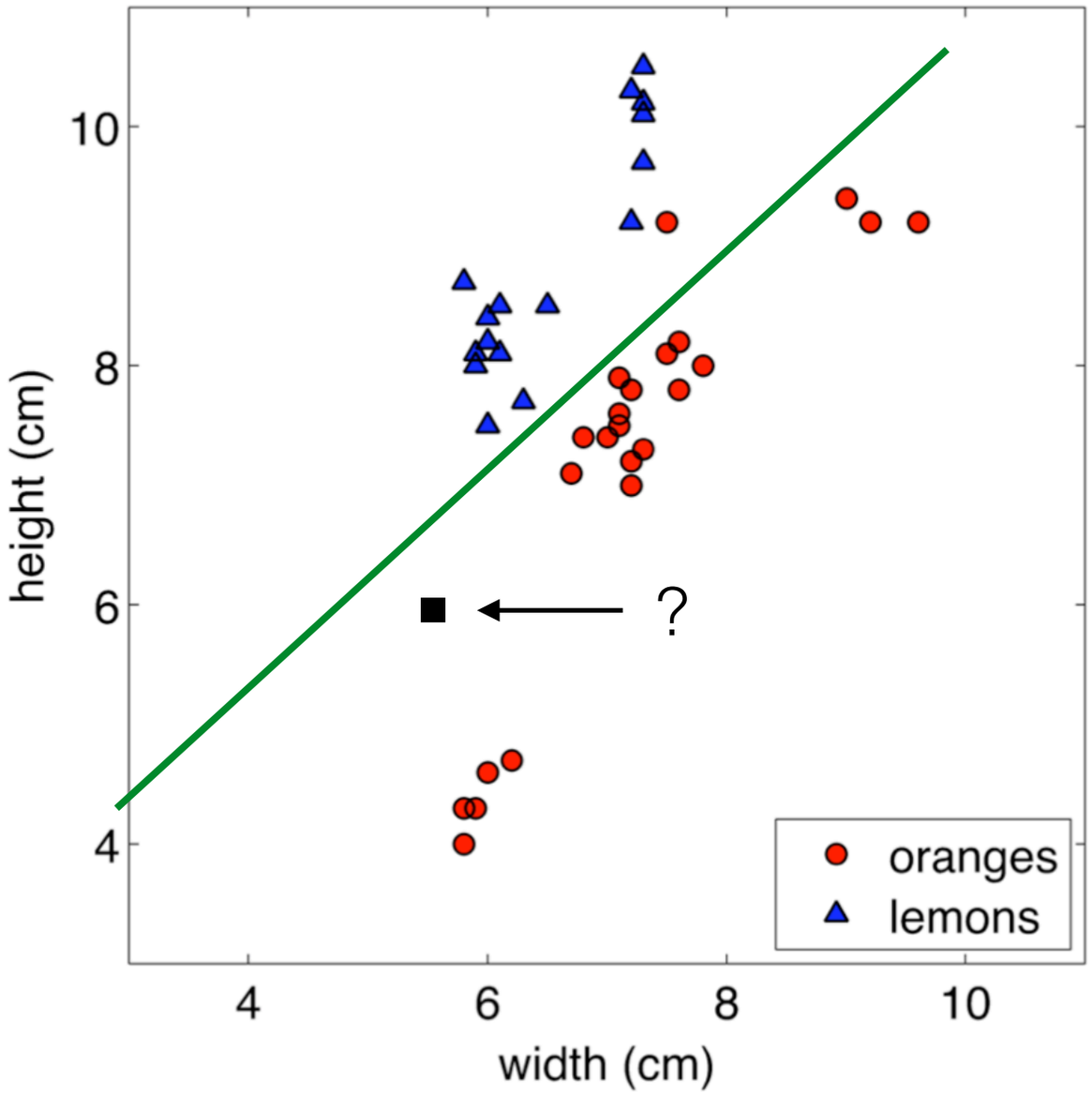
Adapted from slide by S. Fidler

Another look at linear classifiers

- What's the meaning of "linear" classification?
  - Classification is non-linear: it puts non-identical things in the same class, so a difference in the input sometimes causes zero change in the answer
  - Linear classification means that the part that adapts through learning is linear (i.e. the parameters $\boldsymbol{\theta}$)
  - The "adaptive part" is followed by a non-linearity f to make the decision: $y = f(h_\theta(\mathbf{x}))$, $h_\theta(\mathbf{x}) = \boldsymbol{\theta}^{\mathrm{T}}\,\mathbf{x}$
- Classification as induction

Another look at linear classifiers

- An example: “oranges” vs “lemons”



We can construct simple linear decision boundary:

$$y = \text{sign}(h_\theta(\mathbf{x})), \quad h_\theta(\mathbf{x}) = \boldsymbol{\theta}^{\mathrm{T}}\,\mathbf{x}$$

Classification as **induction**

Instance-based Learning

- Alternatively we can use non-parametric models
  - These are simple methods for approximating any target function (either discrete-valued or real-valued)
- Learning amounts to simply *storing* training data
- Test instances are classified using similar training instances
- Underlying assumptions:
  - Output varies smoothly with the input
  - Data occupies sub-space of high-dimensional input space

Nearest Neighbors

- Assume that your training examples corresponds to points in d-dimensional Euclidean space
  - Key idea: the value of the target function for a new sample is estimated from the known (stored) training examples
    - This is done by computing distances between the new sample and all the training samples
  - Decision rule: assign the label of the nearest example

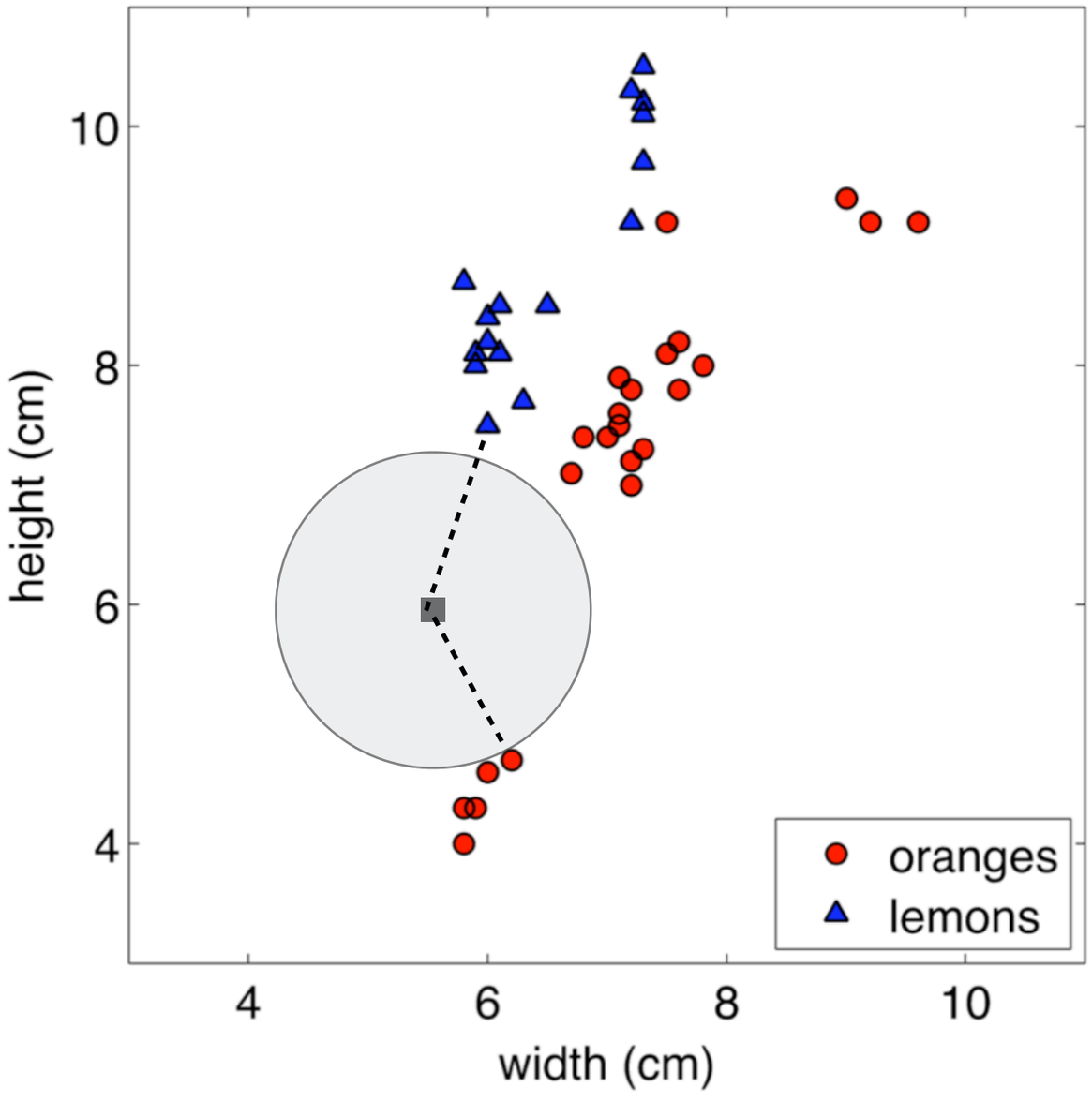
Algorithm:

Find (x^*, y^*) (from the stored training set) closest to the test sample x

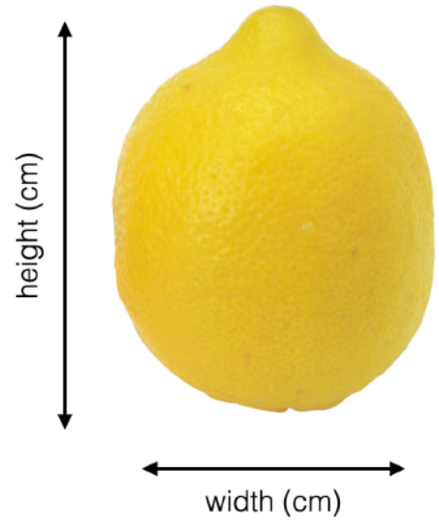
i.e. $x^* = \underset{x^{(i)} \in TrainSet}{\arg\min}\ \text{Distance}(x^{(i)}, x)$. Output: $y = y^*$

Nearest Neighbors

- An example: “oranges” vs “lemons”



Binary classifier based on two simple features:



Which distance?

$$\|x_j^{(a)} - x_j^{(b)}\|_2 = \sqrt{\sum_{j=1}^{d} (x_j^{(a)} - x_j^{(b)})^2}$$

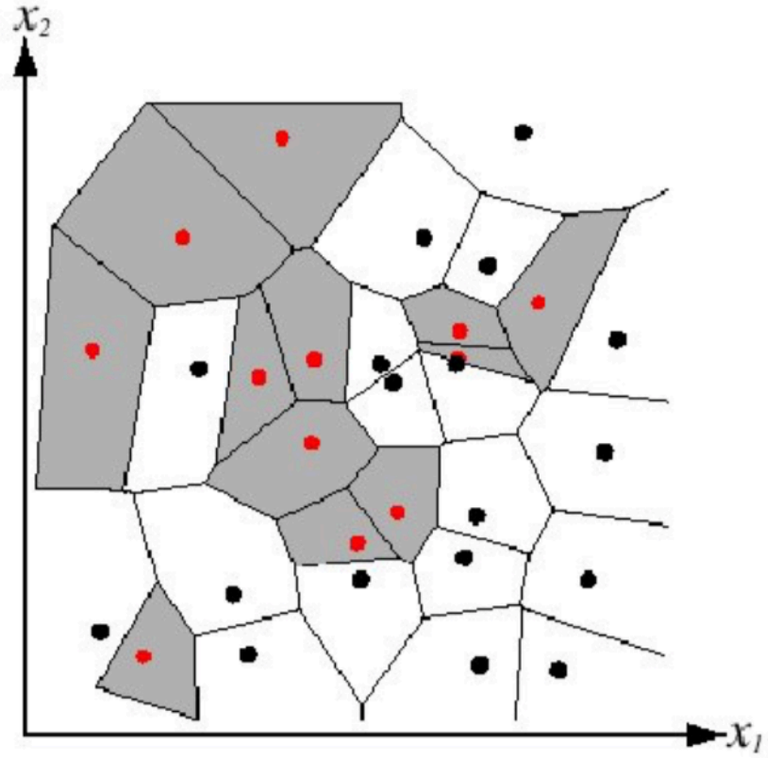
L2 (Euclidean)

$$\|x_j^{(a)} - x_j^{(b)}\|_1 = \sum_{j=1}^{d} |x_j^{(a)} - x_j^{(b)}|$$

L1

Decision boundaries

- Nearest Neighbor does not explicitly compute decision boundaries, but these can be inferred
  - Voronoi diagram visualization
    - It shows how the input space is divided into classes
    - Each line segment is equidistant between two samples of opposite classes

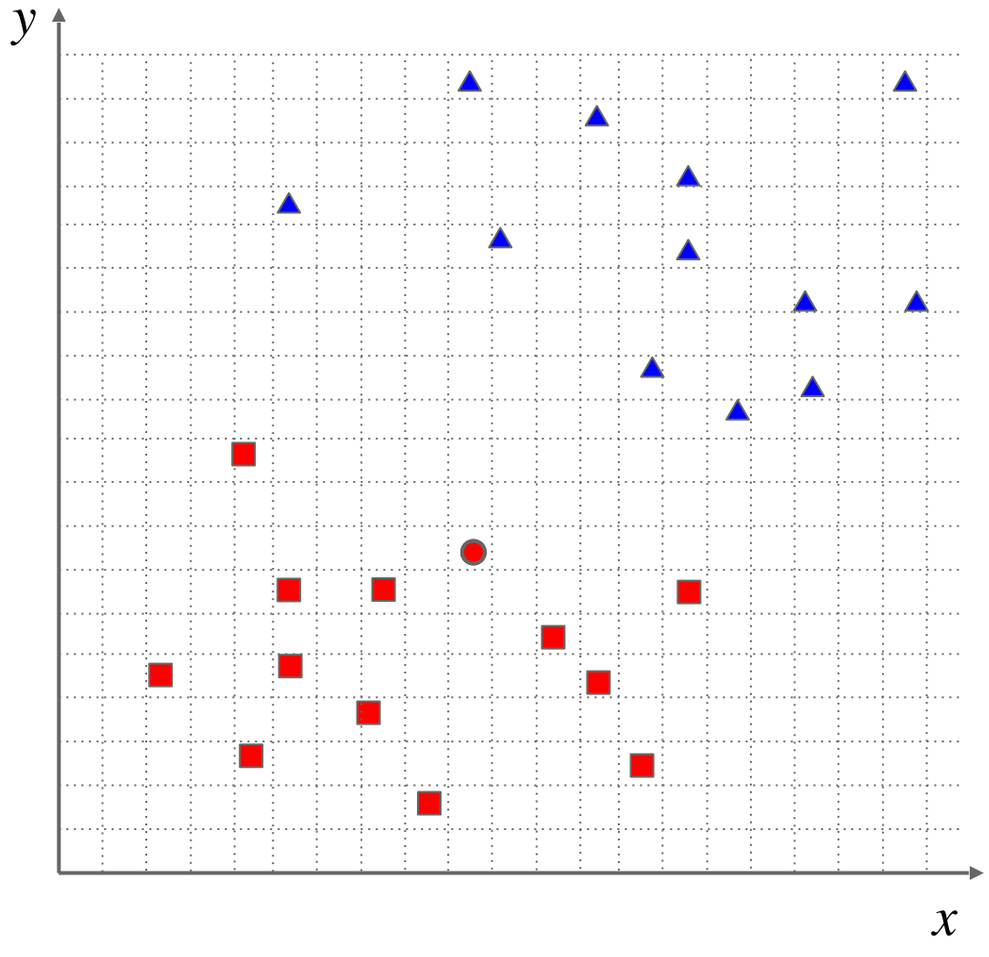


Nearest Neighbor classifier

- Compute distances (Euclidean):

$$d = \text{sqrt}((x_q - x_p)^2 + (y_q - y_p)^2)$$

- Class 1
- Class 2



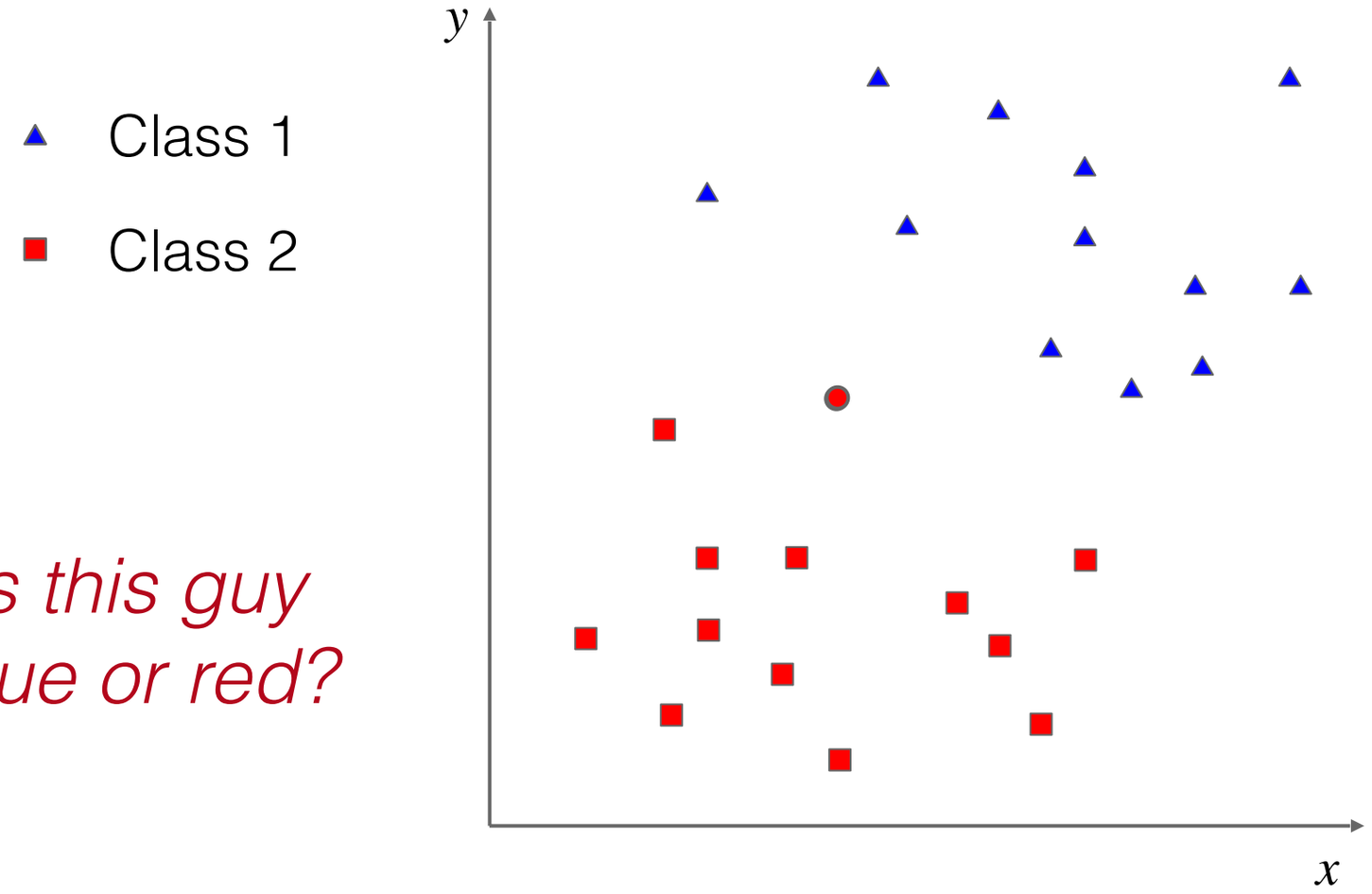
x_q	y_q
10	8

x_q	y_q	d	x_q	y_q	d
9	2	6.1	16	11	6.7
5	3	7.1	14	12	5.7
14	3	6.4	18	12	8.9
8	4	4.5	18	14	10
3	5	7.6	20	14	11.7
6	5	5.0	11	15	7.1
13	5	4.2	15	15	8.6
12	6	2.8	6	16	8.9
6	7	4.1	15	17	10.3
8	7	2.2	13	18	10.4
15	7	5.1	10	19	11
5	10	5.4	20	19	14.9

Nearest Neighbor classifier

- Compute distances (Euclidean):

$$d = \text{sqrt}((x_q - x_p)^2 + (y_q - y_p)^2)$$



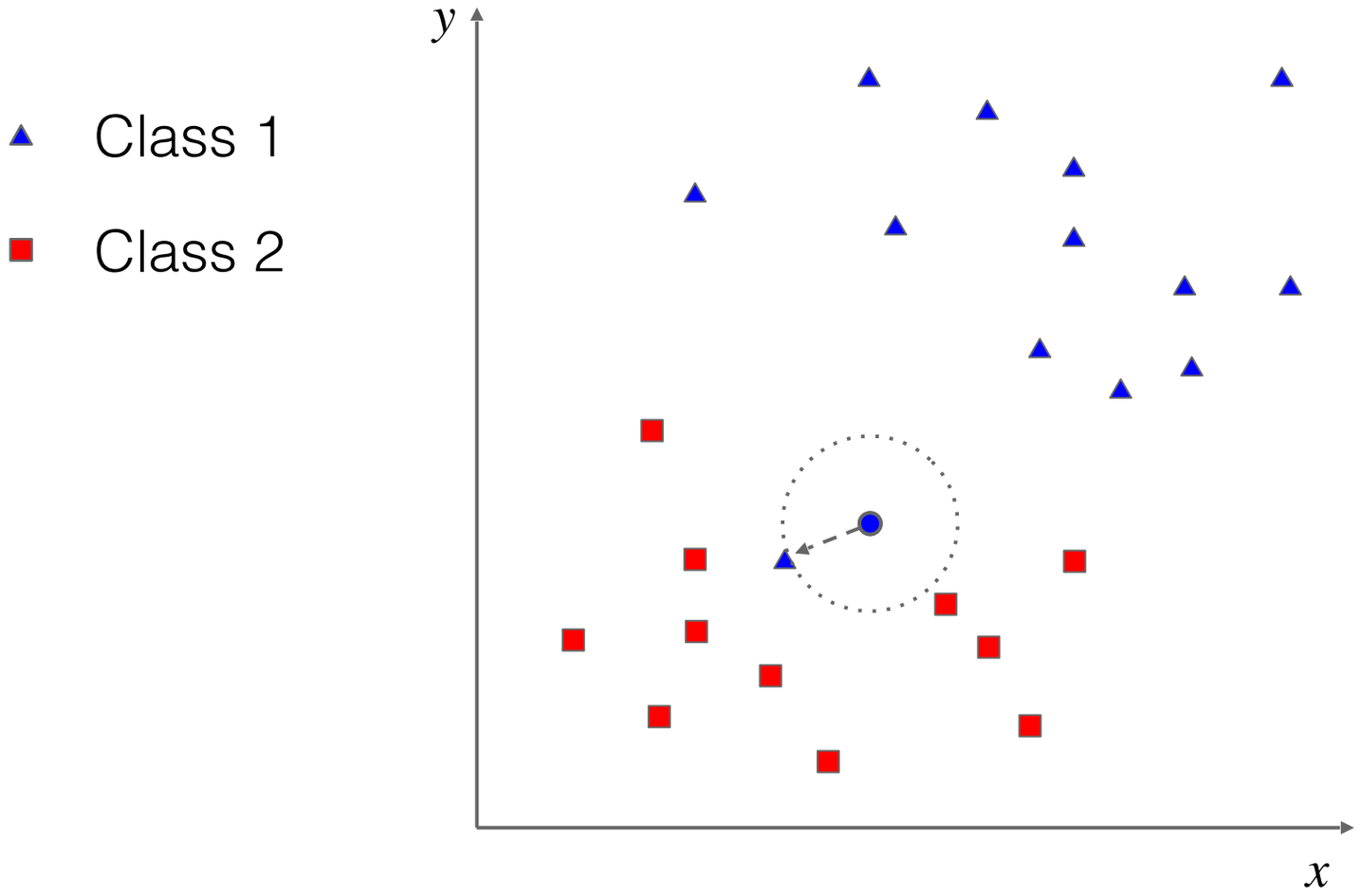
Is this guy blue or red?

x_q	y_q
9	11

x_q	y_q	d	x_q	y_q	d
9	2	9.0	16	11	7.0
5	3	8.9	14	12	5.1
14	3	9.4	18	12	9.1
8	4	7.1	18	14	9.5
3	5	8.5	20	14	11.4
6	5	6.7	11	15	4.5
13	5	7.2	15	15	7.2
12	6	5.8	6	16	5.8
6	7	5.0	15	17	8.5
8	7	4.1	13	18	8.1
15	7	7.2	10	19	8.1
5	10	4.1	20	19	13.6

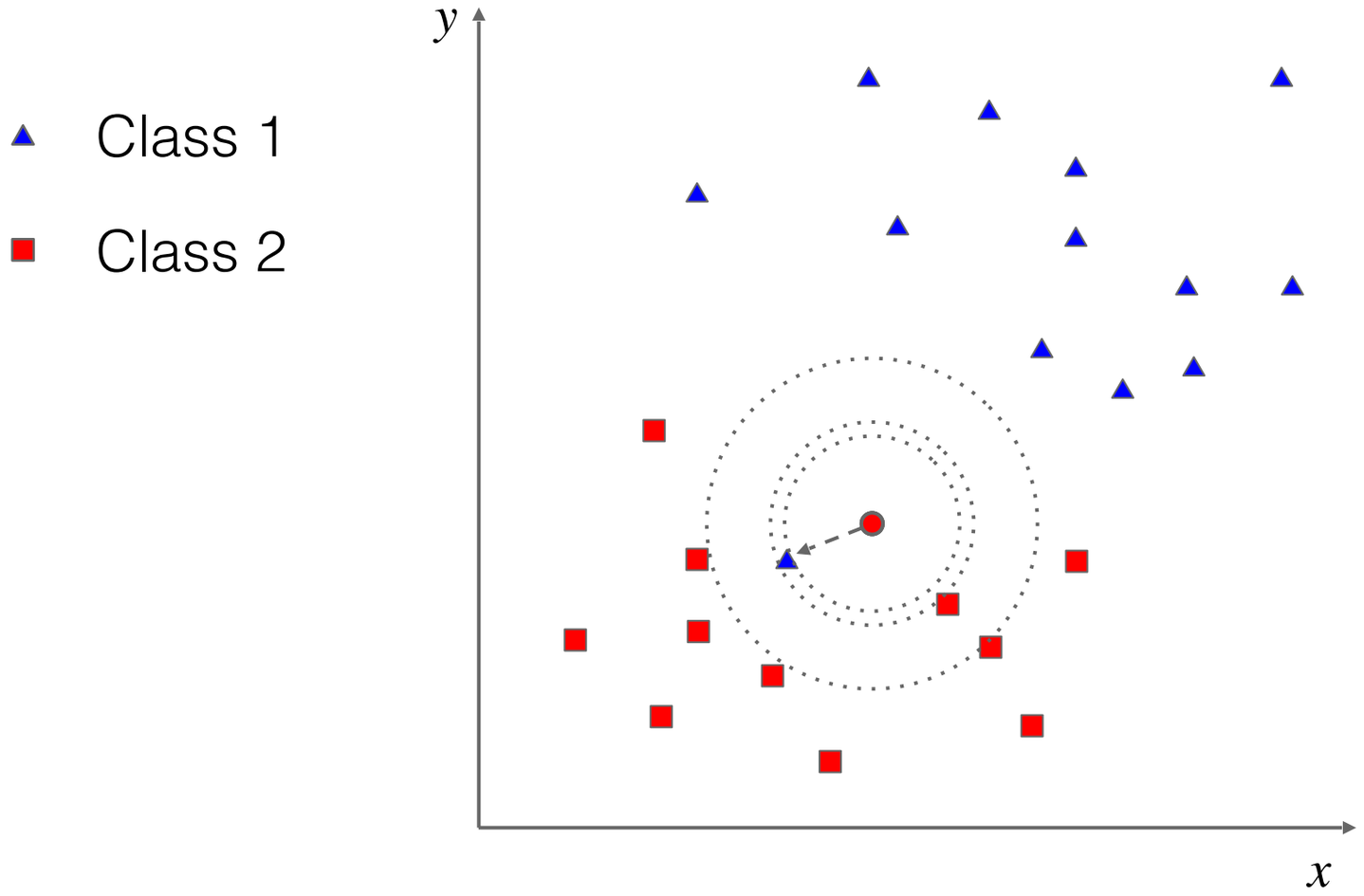
Nearest Neighbor classifier

- Properties: NN is sensitive to outliers



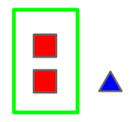
k-Nearest Neighbors classifier

- A “generalization”: from NN to k-NN



k=3
(a.k.a. 3-NN)

majority voting:



k-Nearest Neighbors

- Assume that your training examples corresponds to points in d-dimensional Euclidean space
  - Key idea: the value of the target function for a new sample is estimated from the known (stored) training examples
    - This is done by computing distances between the new sample and all the training samples
  - Decision rule: assign the label of the majority class among the k nearest neighbors

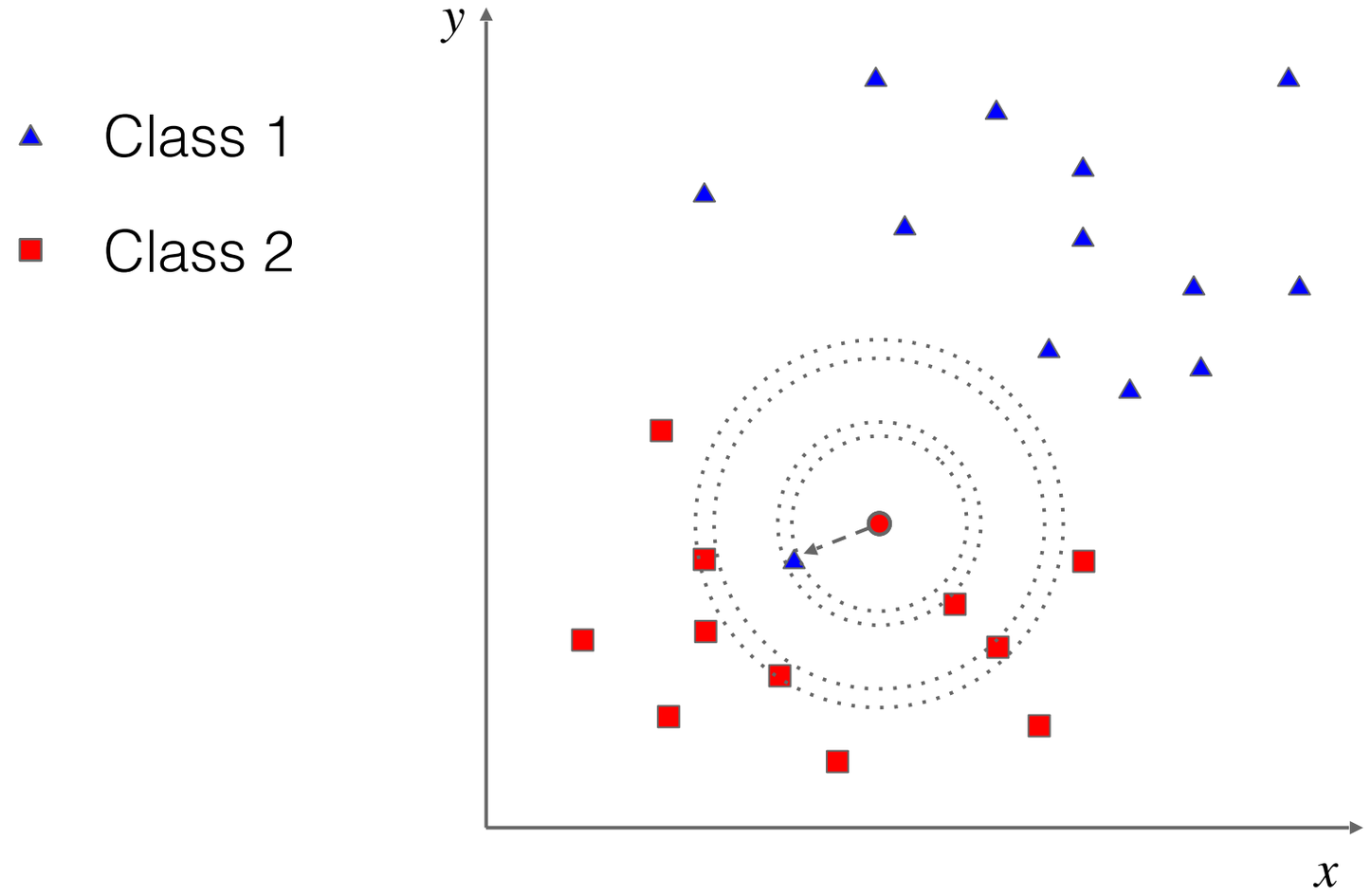
Algorithm:

Find k examples $(x^{(i)}, y^{(i)})$ (from training set) closest to the test sample x

Output: $y = \arg\max_{y^{(z)}} \sum_{j=1}^{k} \delta(y^{(z)}, y^{(j)})$

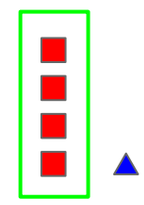
k-Nearest Neighbors classifier

- Parameter k has a very strong effect
  - Small k: sensitive to outliers



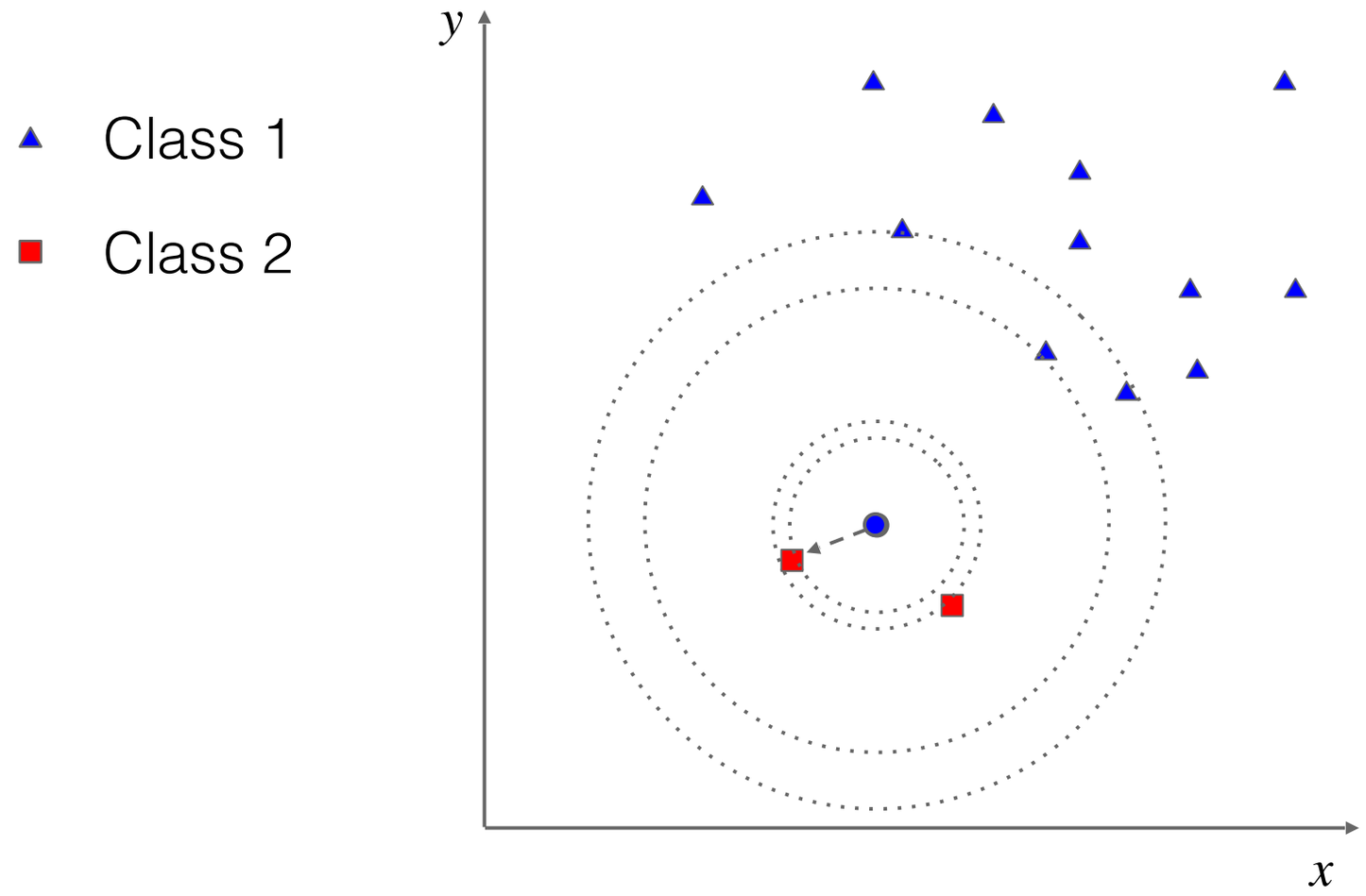
k=5
(a.k.a. 5-NN)

majority voting:



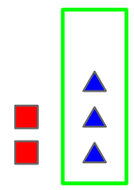
k-Nearest Neighbors classifier

- Parameter k has a very strong effect
  - Large k: everything is classified as the most frequent class



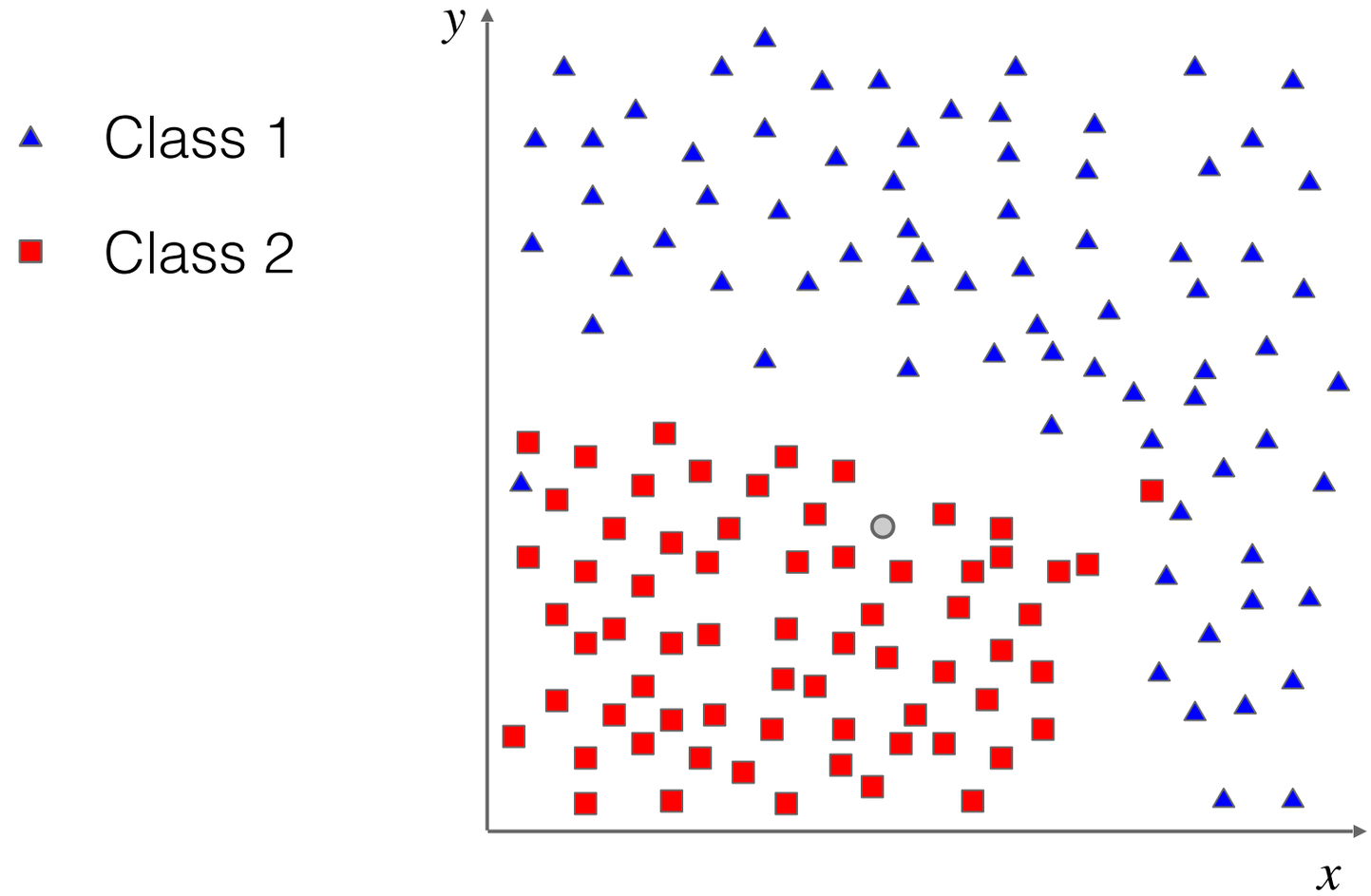
k=5
(a.k.a. 5-NN)

majority voting:



k-Nearest Neighbors classifier

- k-NN: large data is good!



k-Nearest Neighbors classifier

- k-NN recipe: how do we choose k?
  - Large k may lead to better performance (if the training set is sufficiently large)
  - If we pick k too large we may end up looking at examples that are not "real" neighbors (are far away the test sample)
  - We can use cross-validation to find the right k
  - Rule of thumb: choose $k < \mathrm{sqrt}(\mathrm{m})$, where m is the number of training examples

Adapted from slide by O. Veksler

Image classification with k-NN

- What about images? How to represent them?
  - The simplest representation is provided by the raw pixels

- A simple image classification pipeline:
  - Resize images to the same scale/resolution
  - Represent images as a 2D or 1D array of raw pixels
  - Define a distance (e.g. L1 or L2) to compare images
  - Use k-NN for classification

Image classification with k-NN

- Example dataset: CIFAR10

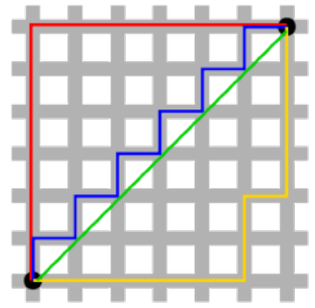
10 classes
50,000 training images
10,000 testing images

test images and nearest neighbors

Slide credit: Fei-Fei, Johnson, Yeung

Image classification with k-NN

- Distance metric to compare images:
  - L1 distance *(Manhattan)*: $d_{L1}(I_1, I_2) = \sum_p |I_1^p - I_2^p|$
  - Example:



test image					training image				pixel-wise absolute value differences				
56	32	10	18	−	10	20	24	17	= 46	12	14	1	add → 456
90	23	128	133		8	10	89	100	82	13	39	33	
24	26	178	200		12	16	178	170	12	10	0	30	
2	0	255	220		4	32	233	112	2	32	22	108	

Slide credit: Fei-Fei, Johnson, Yeung

Image classification with k-NN

- Example (Python implementation):

```python
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in xrange(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

"Memorize" training data

For each test image:

- Find closest train image
- Predict label of nearest train image

Slide credit: Fei-Fei, Johnson, Yeung

Image classification with k-NN

- Example (Python implementation):

```python
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

def predict(self, X):
  """ X is N x D where each row is an example we wish to predict label for """
  num_test = X.shape[0]
  # lets make sure that the output type matches the input type
  Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

  # loop over all test rows
  for i in xrange(num_test):
    # find the nearest training image to the i'th test image
    # using the L1 distance (sum of absolute value differences)
    distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
    min_index = np.argmin(distances) # get the index with smallest distance
    Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

  return Ypred
```

Q: With *N* examples, how fast are NN training and prediction?

A: Train O(1), Test O(N)

Q: Is this good or bad?

A: This is bad! We want classifiers that are fast at prediction (slow for training is ~ok)

Slide credit: Fei-Fei, Johnson, Yeung

k-NN: issues and remedies

- Some attributes (features) have larger ranges, so are treated as more important
  - You should apply feature scaling, such as:
    - Linearly scale the range of each feature to be in [0,1]
    - Linearly scale each dim. to have 0 mean and variance 1 (i.e. compute μ and σ^2 for a feature x_j and scale: $(x_j-\mu)/\sigma$)
- Irrelevant but correlated features add noise to distance measure computation:
  - You could apply feature selection
  - Pick better distance measure or apply Metric Learning

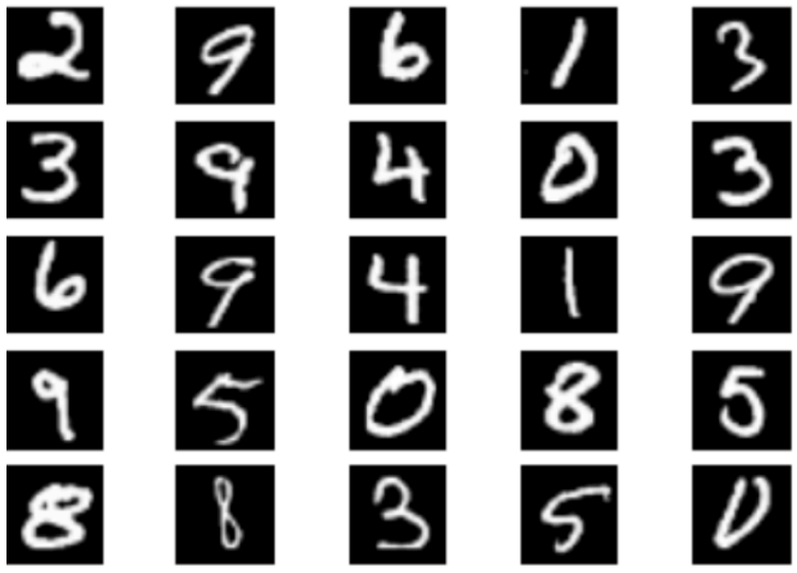
k-NN: issues and remedies

- Expensive at test time: for each test sample we must compute the distance to all $\mathbf{m}$ training examples
  - You can compute approximate distance (e.g. LSH)
  - Use pre-sorting / fast data structures (e.g. kd-trees)
- Storage requirements: we must store all training data
  - Remove redundant data (condensing)
  - Pre-sorting often increases the storage requirements
- High-dimensional data ("curse of dimensionality")
  - Required training data increases exponentially with dim.; consider using dimensionality reduction (e.g. PCA)

Applications

- You can get pretty good results on several tasks:

MNIST digit classification



	Test Error Rate (%)
Linear classifier (1-layer NN)	12.0
K-nearest-neighbors, Euclidean	5.0
K-nearest-neighbors, Euclidean, deskewed	2.4
K-NN, Tangent Distance, 16x16	1.1
K-NN, shape context matching	0.67
1000 RBF + linear classifier	3.6
SVM deg 4 polynomial	1.1
2-layer NN, 300 hidden units	4.7
2-layer NN, 300 HU, [deskewing]	1.6
LeNet-5, [distortions]	0.8
Boosted LeNet-4, [distortions]	0.7

- Handwritten digits
- 28x28 pixels (784d)
- 60,000 training images
- 10,000 test images

Applications

- You can get pretty good results on several tasks:

Image geolocalization

J.Hays, A. Efros, “IM2GPS: estimating geographic information from a single image”, CVPR 2008

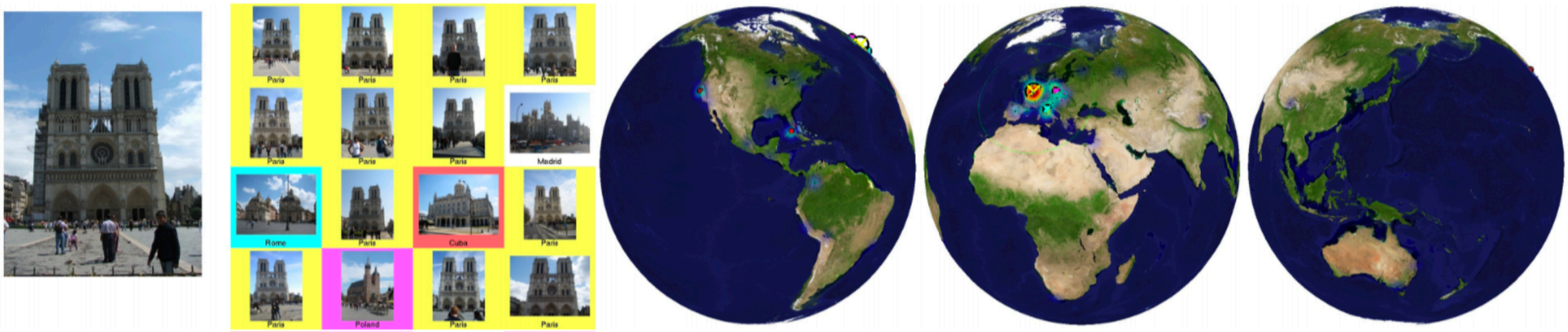
+ “Revisiting IM2GPS in the Deep Learning Era”, ICCV 2017

http://graphics.cs.cmu.edu/projects/im2gps/

Applications

- You can get pretty good results on several tasks:

Image geolocalization: where (GPS) was this picture taken?



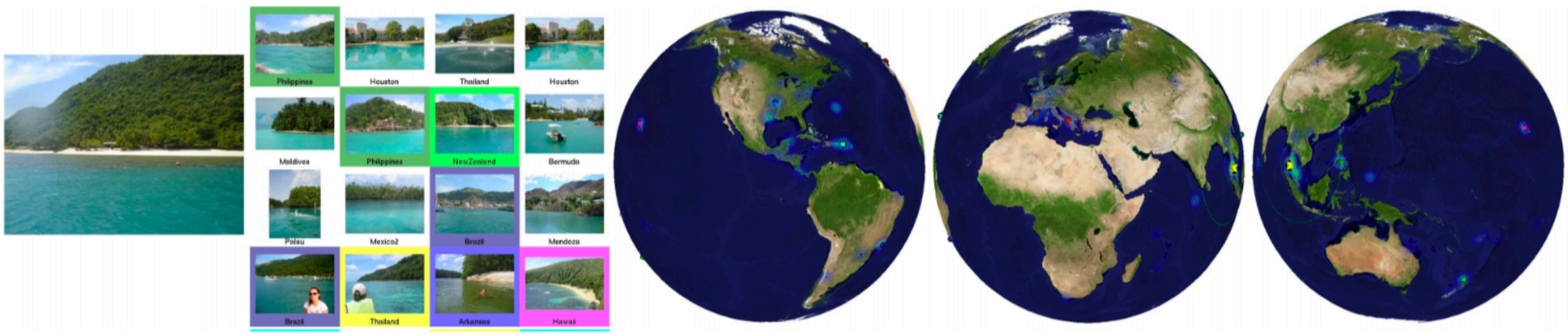
- 6M images from Flickr
- Dense sampling around the world
- Represent each image with meaningful features
- Do k-NN! (using a large k e.g. k=120)

http://graphics.cs.cmu.edu/projects/im2gps/

Applications

- You can get pretty good results on several tasks:

Image geolocalization: where (GPS) was this picture taken?



- 6M images from Flickr
- Dense sampling around the world
- Represent each image with meaningful features
- Do k-NN! (using a large k e.g. k=120)

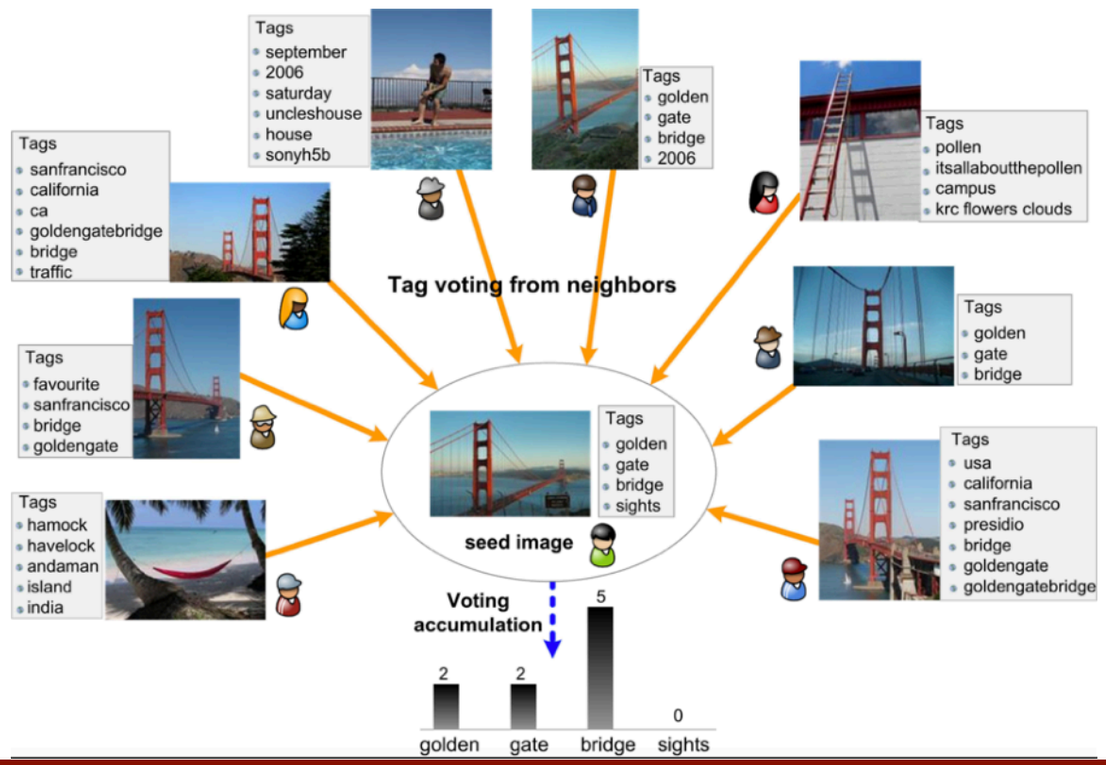
http://graphics.cs.cmu.edu/projects/im2gps/

Applications

Scene completion using millions of pictures

J. Hays
A. Efros
SIGGRAPH'07

Image annotation and learning tag relevance by neighbor voting



A. Makadia
V. Pavlovic
S. Kumar
ECCV'08

X. Li
C. Snoek
M. Worring
IEEE-TMM'09

M. Guillaumin
T. Mensink
J. Verbeek
C. Schmid
ICCV'09

L. Ballan
T. Uricchio
L. Seidenari
A. Del Bimbo
ACM-ICMR'14

k-NN Summary

- k-NN naturally forms complex decision boundaries; it adapts to data density
- If we have lots of samples, k-NN typically works well
- Main limitations/problems:
  - Sensitive to class noise and scales of features (attributes)
  - Distances are less meaningful in high dimensions
  - Scales linearly with number of training examples: i.e. it is extremely expensive at test time
- Inductive bias: assumes that samples that are close in the features space share the same class

Contact

- **Office:** Torre Archimede, room 6CD3
- **Office hours** (ricevimento)**:** Friday 8:30-10:30

✉ lamberto.ballan@unipd.it

🏠 http://www.lambertoballan.net

🏠 http://vimp.math.unipd.it

@ twitter.com/lambertoballan