Industrial Computer Vision

- Object Detector

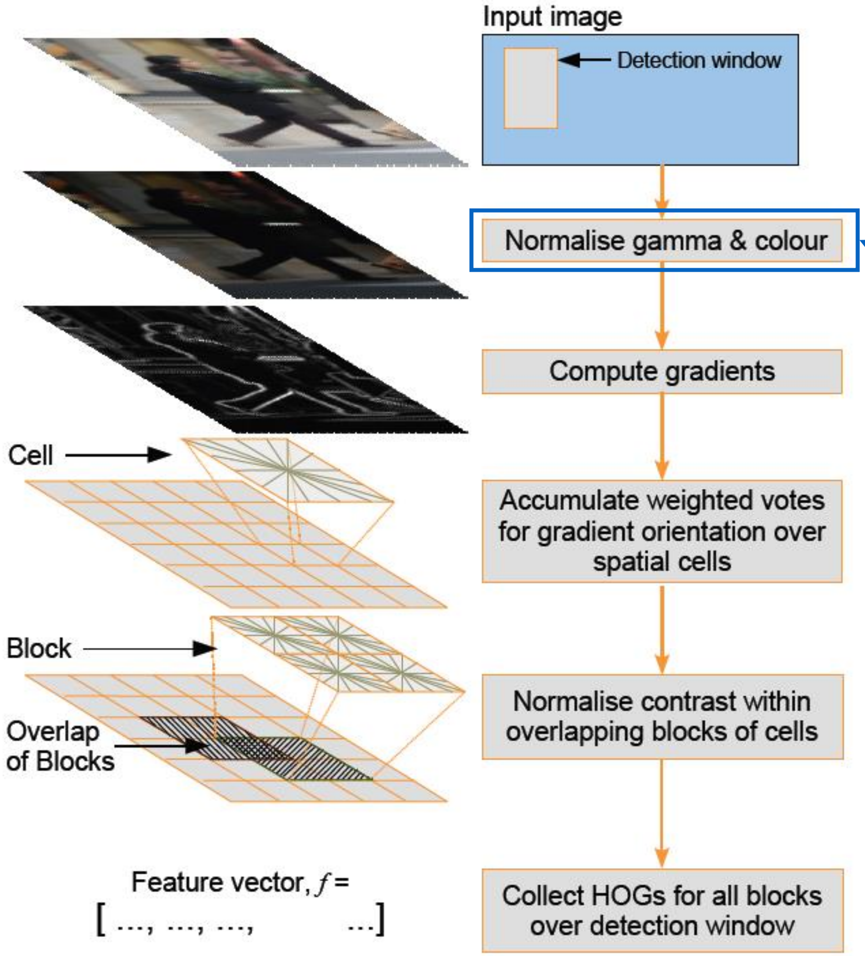
14^{th} lecture, 2022.12.07

Lecturer: Youngbae Hwang

Contents

- HoG (Histogram of Oriented Gradient)
- K-Nearest Neighbor
- SVM (Support Vector Machine)
- Haar-like feature
- Adaboost
- Deep Learning (CNN)

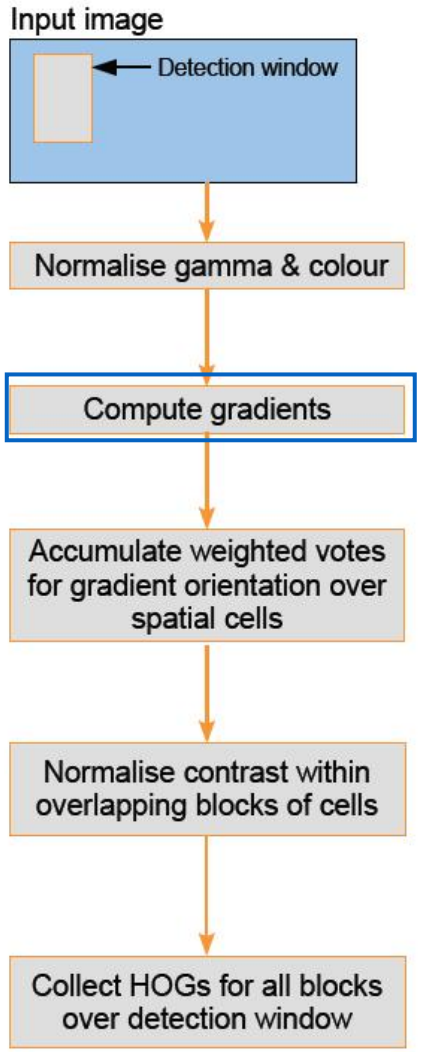
Typical person detection scheme using SVM



In practice, effect is very small (about 1%) while some computational time is required*

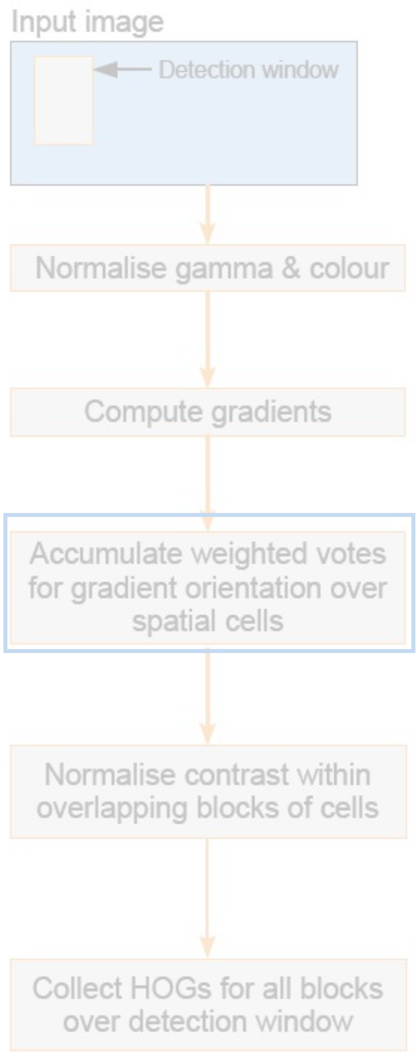
*Navneet Dalal and Bill Triggs. Histograms of Oriented Gradients for Human Detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, SanDiego, USA, June 2005. Vol. II, pp. 886-893.

Computing gradients

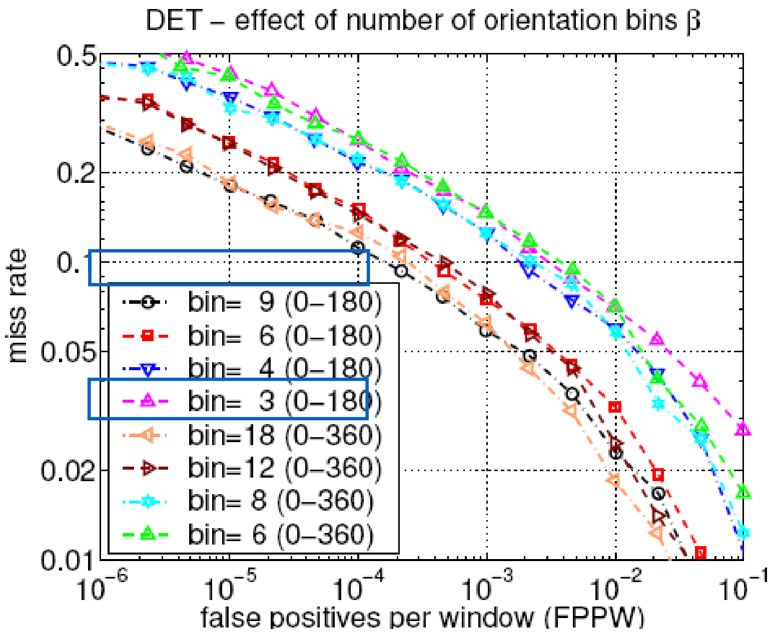


Mask Type	1D centered	1D uncentered	1D cubic-corrected	2x2 diagonal	3x3 Sobel
Operator	[-1, 0, 1]	[-1, 1]	[1, -8, 0, 8, -1]	$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$ $\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$ $\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$
Miss rate at 10^{-4} FPPW	11%	12.5%	12%	12.5%	14%

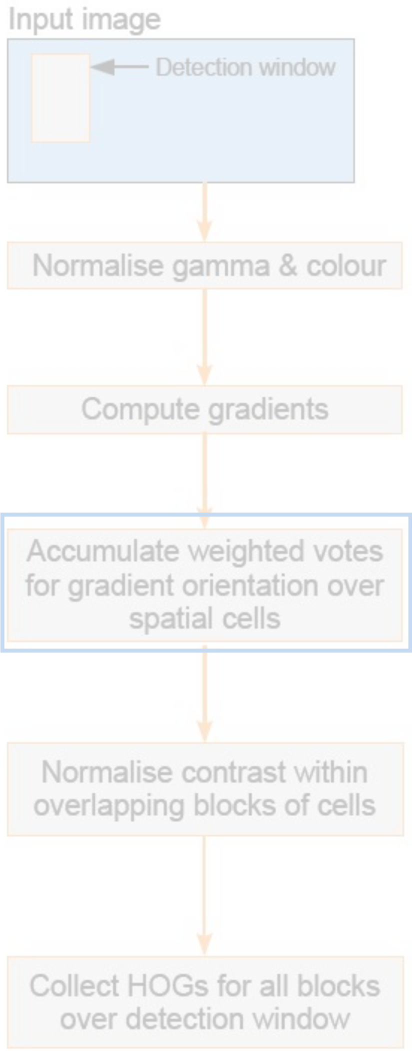
Accumulate weight votes over spatial cells



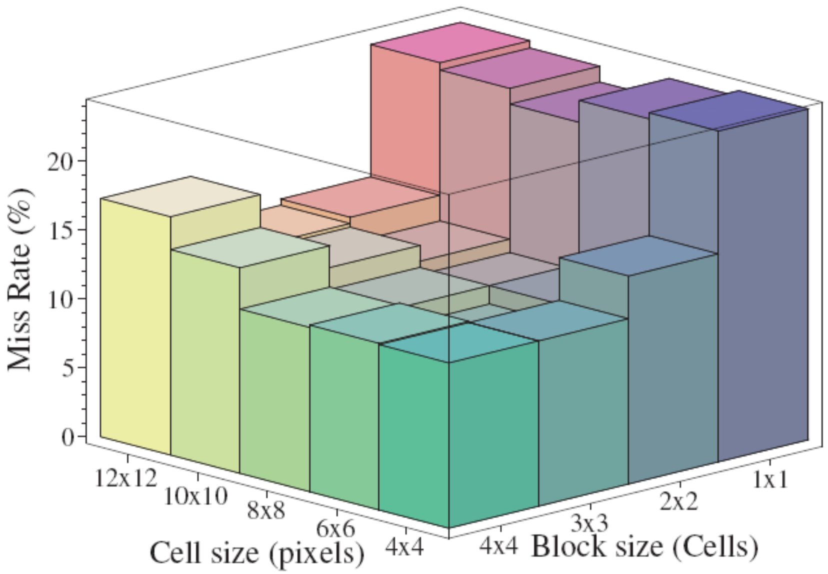
- How many bins should be in histogram?
- Should we use oriented or non-oriented gradients?
- How to select weights?
- Should we use overlapped blocks or not? If yes, then how big should be the overlap?
- What block size should we use?



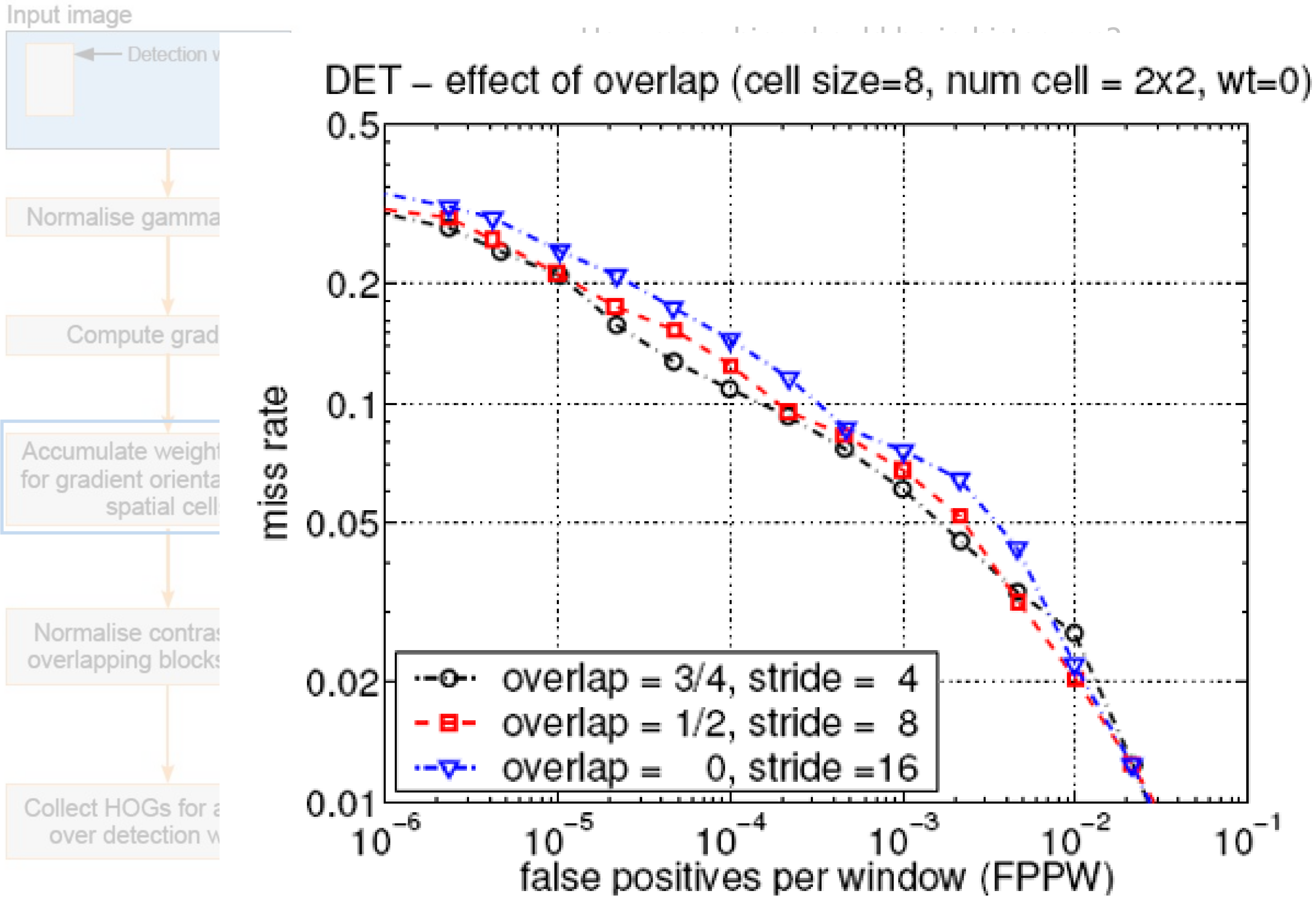
Accumulate weight votes over spatial cells



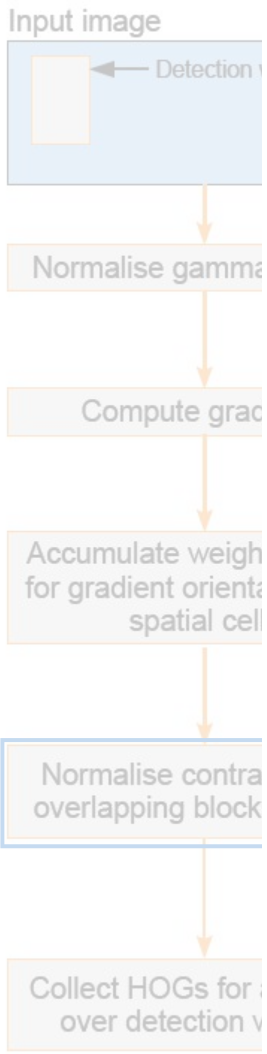
- How many bins should be in histogram?
- Should we use oriented or non-oriented gradients?
- How to select weights?
- Should we use overlapped blocks or not? If yes, then how big should be the overlap?
- What block size should we use?

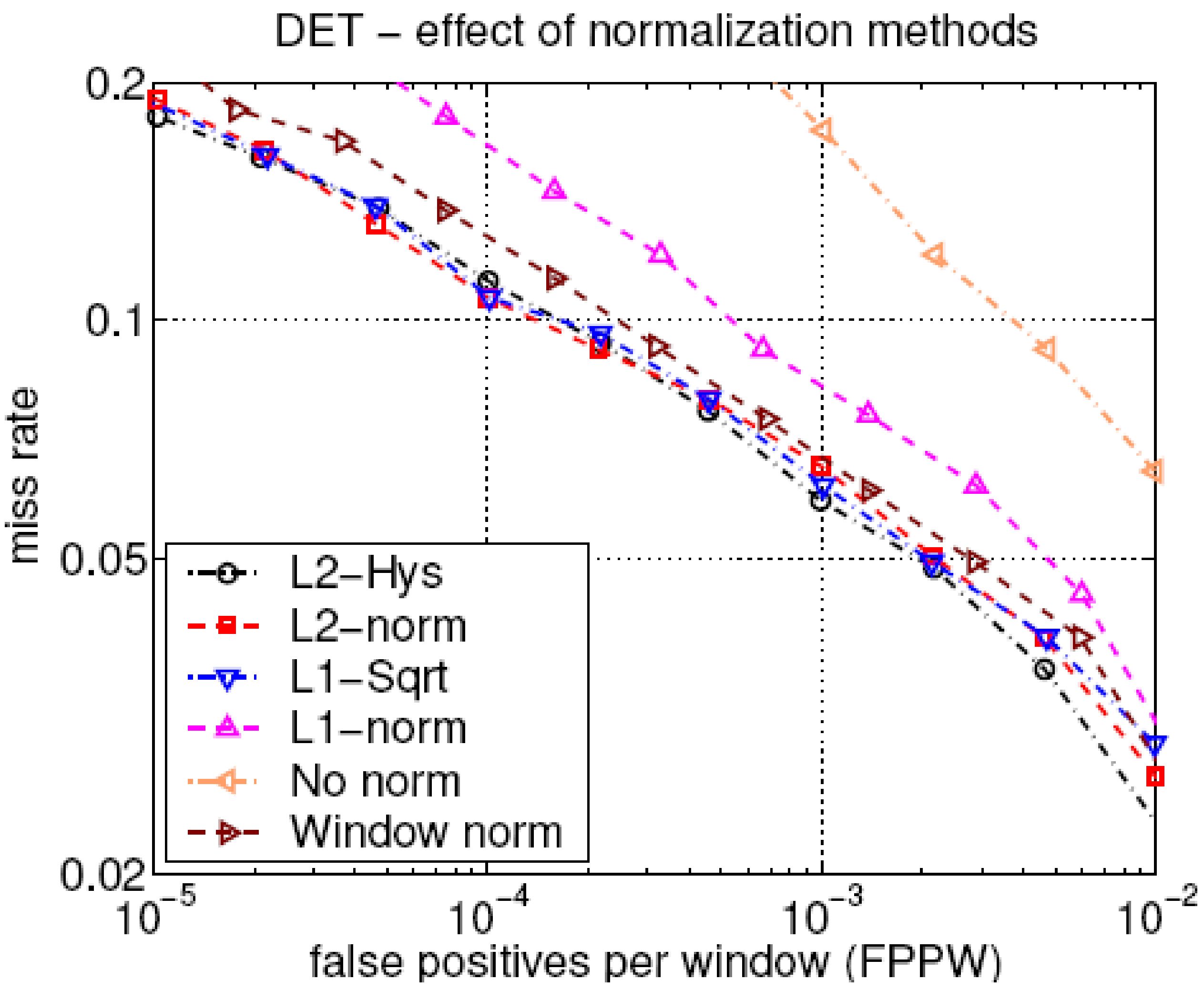


Accumulate weight votes over spatial cells

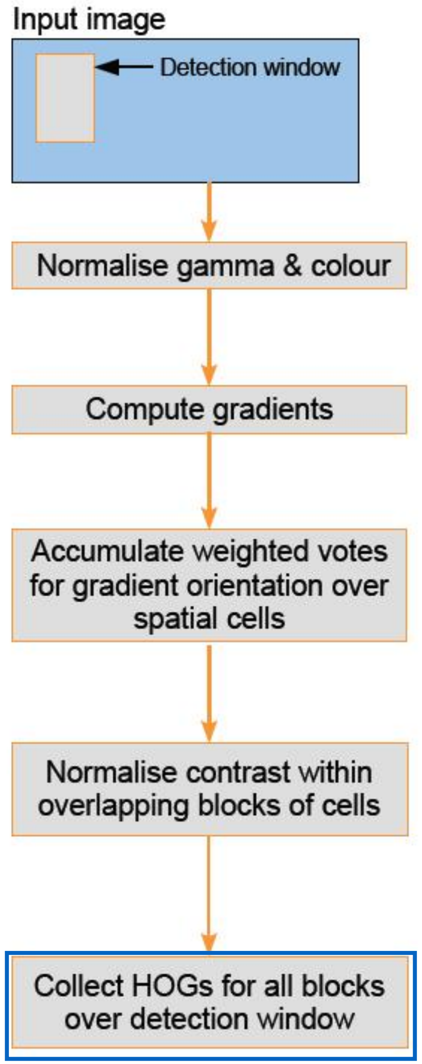


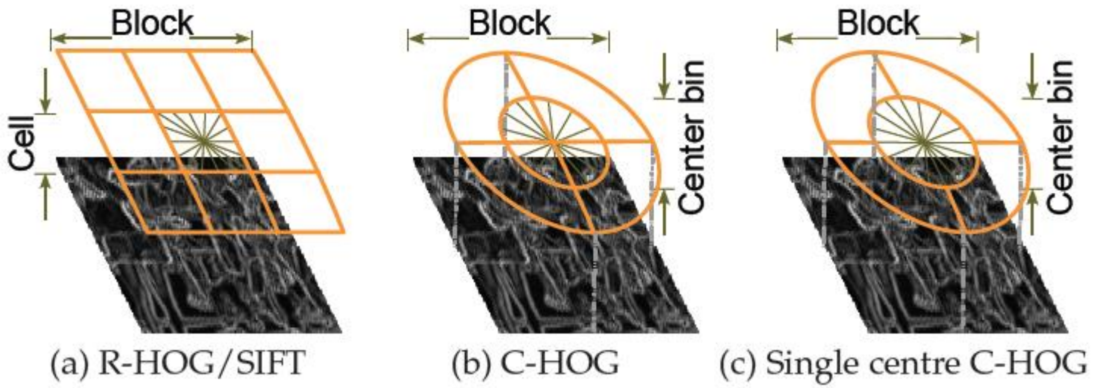
Contrast normalization





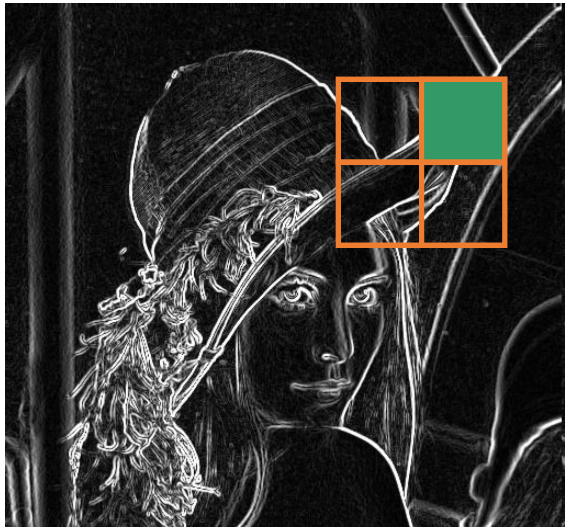
Making feature vector





Variants of HOG descriptors. (a) A rectangular HOG (R-HOG) descriptor with 3 × 3 blocks of cells. (b) Circular HOG (C-HOG) descriptor with the central cell divided into angular sectors as in shape contexts. (c) A C-HOG descriptor with a single central cell.

HOG feature vector for one block



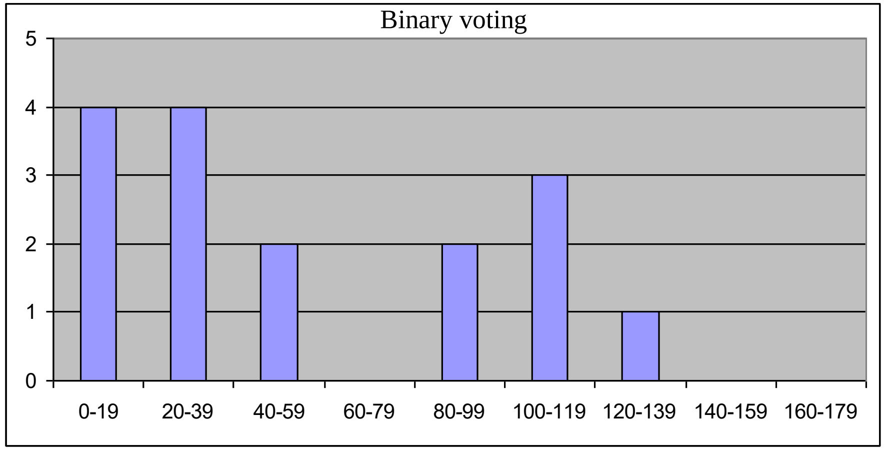
$$f = \left(h_1^1, \ldots, h_9^1,\ \boxed{h_1^2, \ldots, h_9^2},\ h_1^3, \ldots, h_9^3,\ h_1^4, \ldots, h_9^4\right)$$

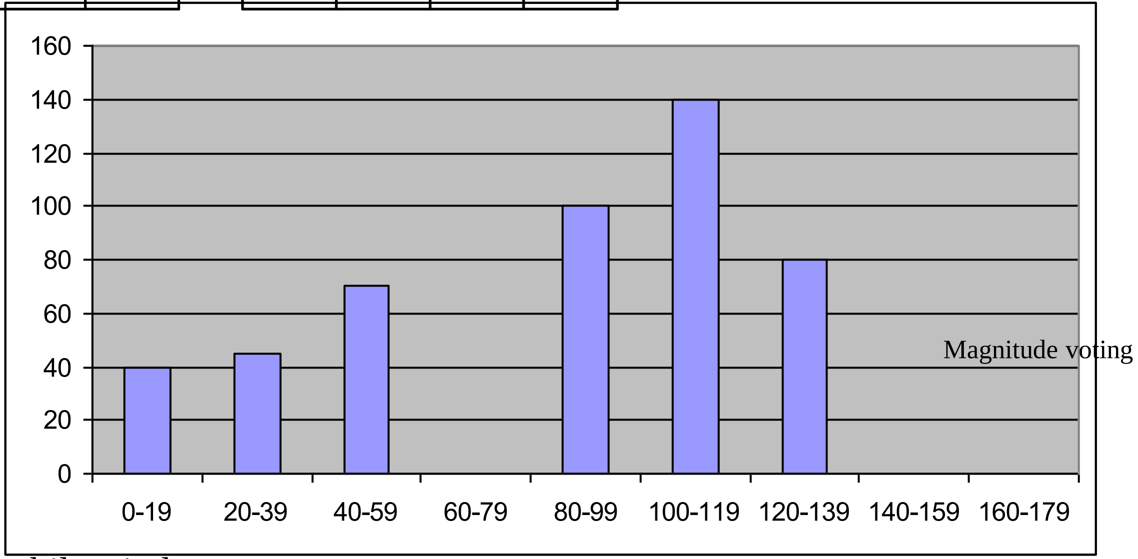
Angle

0	15	25	25
10	15	25	30
45	95	101	110
47	97	101	120

Magnitude

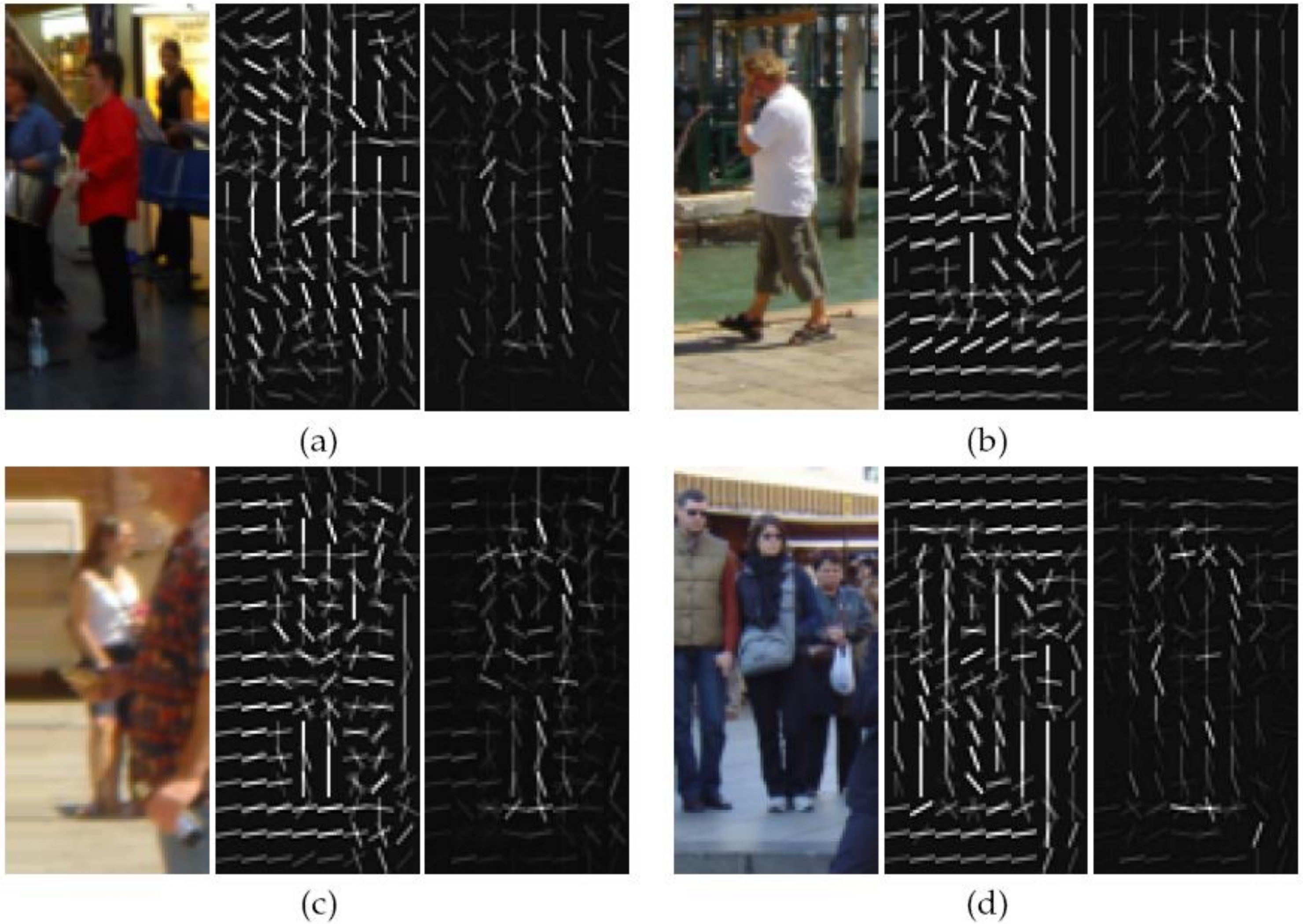
5	20	20	10
5	10	10	5
20	30	30	40
50	70	70	80





Feature vector extends while window moves

HOG example



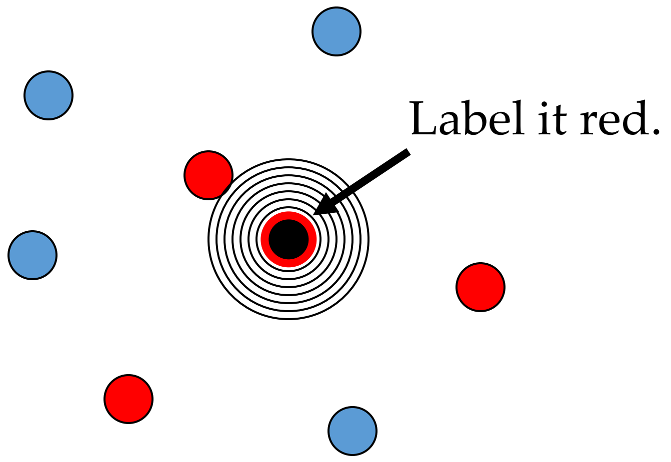
In each triplet: (1) the input image, (2) the corresponding R-HOG feature vector (only the dominant orientation of each cell is shown), (3) the dominant orientations selected by the SVM (obtained by multiplying the feature vector by the corresponding weights from the linear SVM).

Instance-Based Learning

- Idea:
  - Similar examples have similar label.
  - Classify new examples like similar training examples.
- Algorithm:
  - Given some new example x for which we need to predict its class y
  - Find most similar training examples
  - Classify *x* "like" these most similar examples
- Questions:
  - How to determine similarity?
  - How many similar training examples to consider?
  - How to resolve inconsistencies among the training examples?

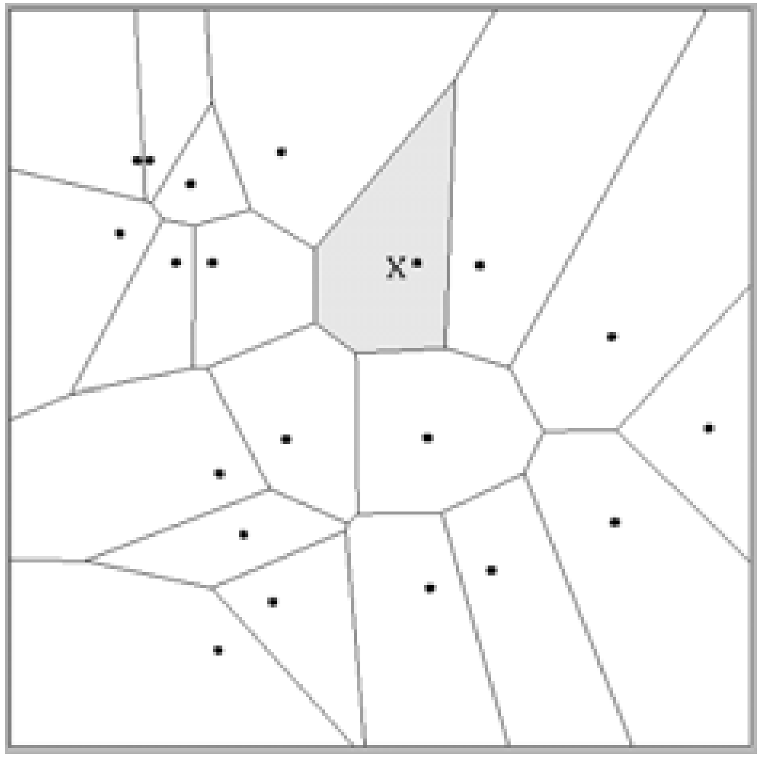
1-Nearest Neighbor

- One of the simplest of all machine learning classifiers
- Simple idea: label a new point the same as the closest known point



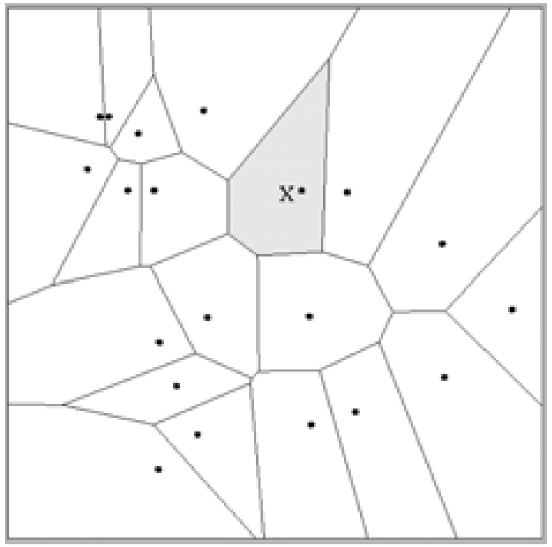
1-Nearest Neighbor

- A type of instance-based learning
  - Also known as “memory-based” learning
- Forms a Voronoi tessellation of the instance space

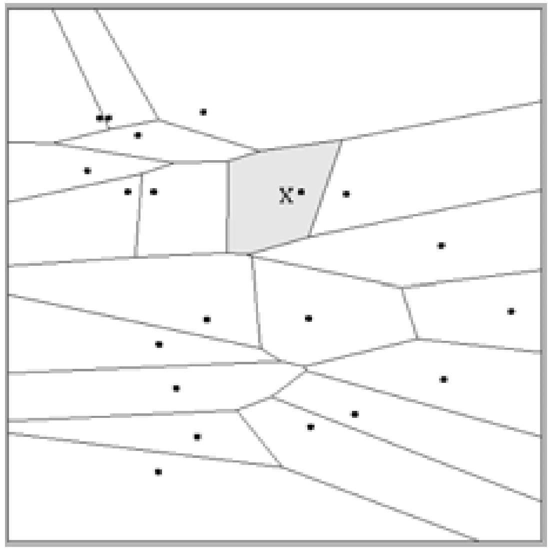


Distance Metrics

- Different metrics can change the decision surface



$\text{Dist}(\mathbf{a},\mathbf{b}) = (a_1 - b_1)^2 + (a_2 - b_2)^2$



$\text{Dist}(\mathbf{a},\mathbf{b}) = (a_1 - b_1)^2 + (3a_2 - 3b_2)^2$

- Standard Euclidean distance metric:
  - Two-dimensional: Dist(a,b) = sqrt((a1 – b1)2 + (a2 – b2)2)
  - Multivariate: Dist(a,b) = sqrt(∑ (ai – bi)2)

1-NN's Aspects as an Instance-Based Learner:

A distance metric

- Euclidean
- When different units are used for each dimension

  → normalize each dimension by standard deviation
- For discrete data, can use hamming distance

  → D(x1,x2) =number of features on which x1 and x2 differ
- Others (e.g., normal, cosine)

How many nearby neighbors to look at?

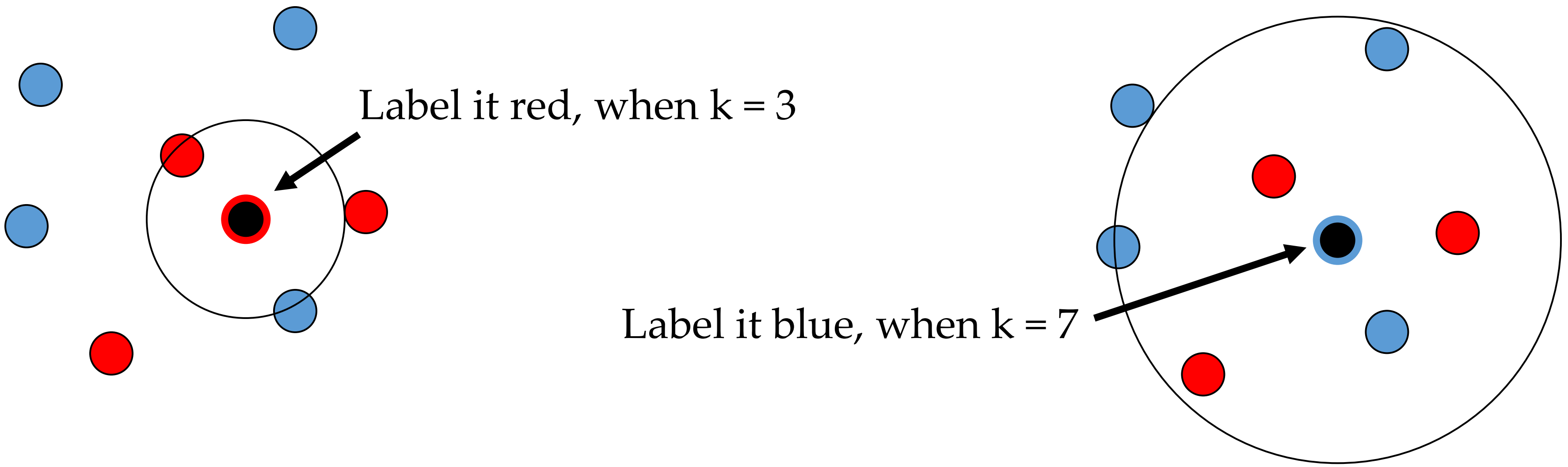
- One

How to fit with the local points?

- Just predict the same output as the nearest neighbor.

k – Nearest Neighbor

- Generalizes 1-NN to smooth away noise in the labels
- A new point is now assigned the most frequent label of its *k* nearest neighbors



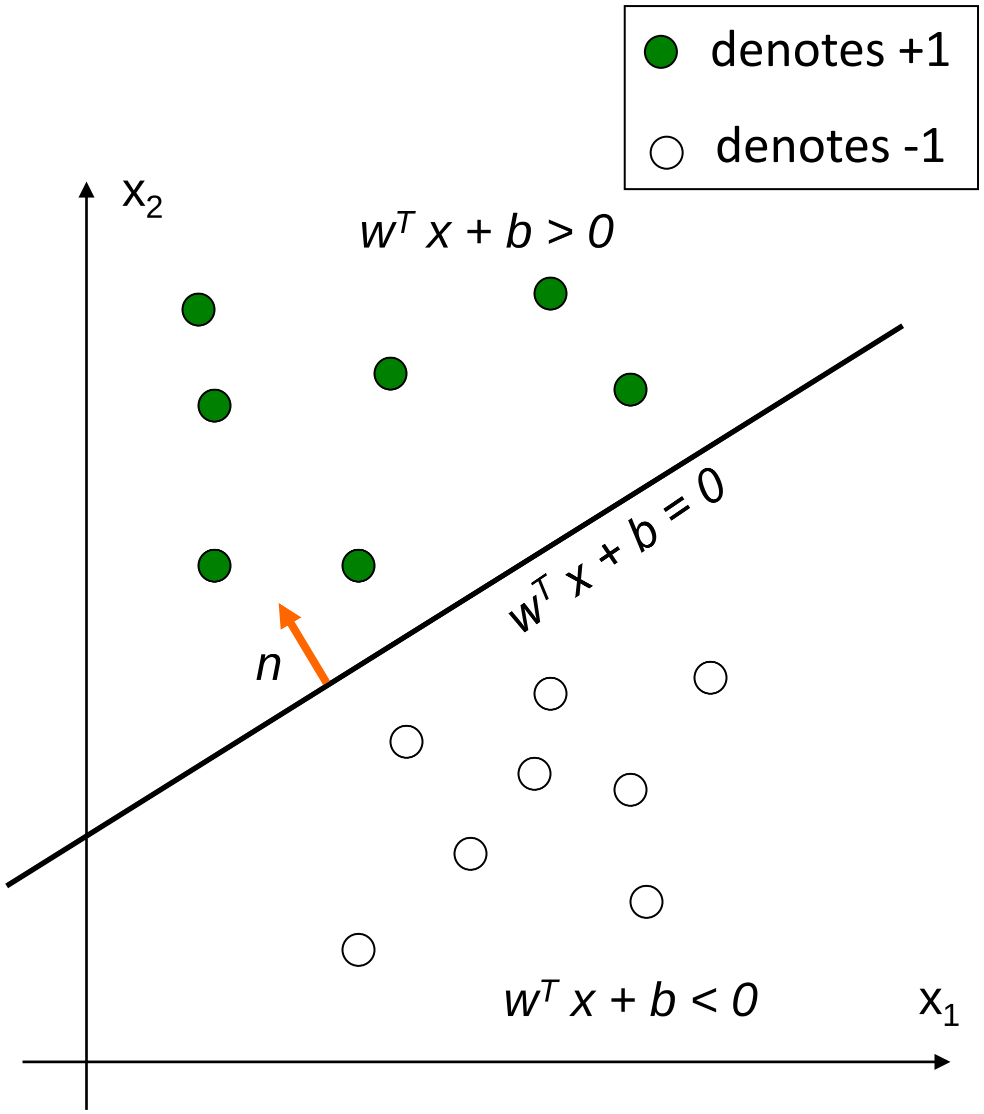
Problem setting for SVM

- A hyper-plane in the feature space

$$\mathrm{w}^{\mathrm{T}}\mathrm{x}+b=0$$

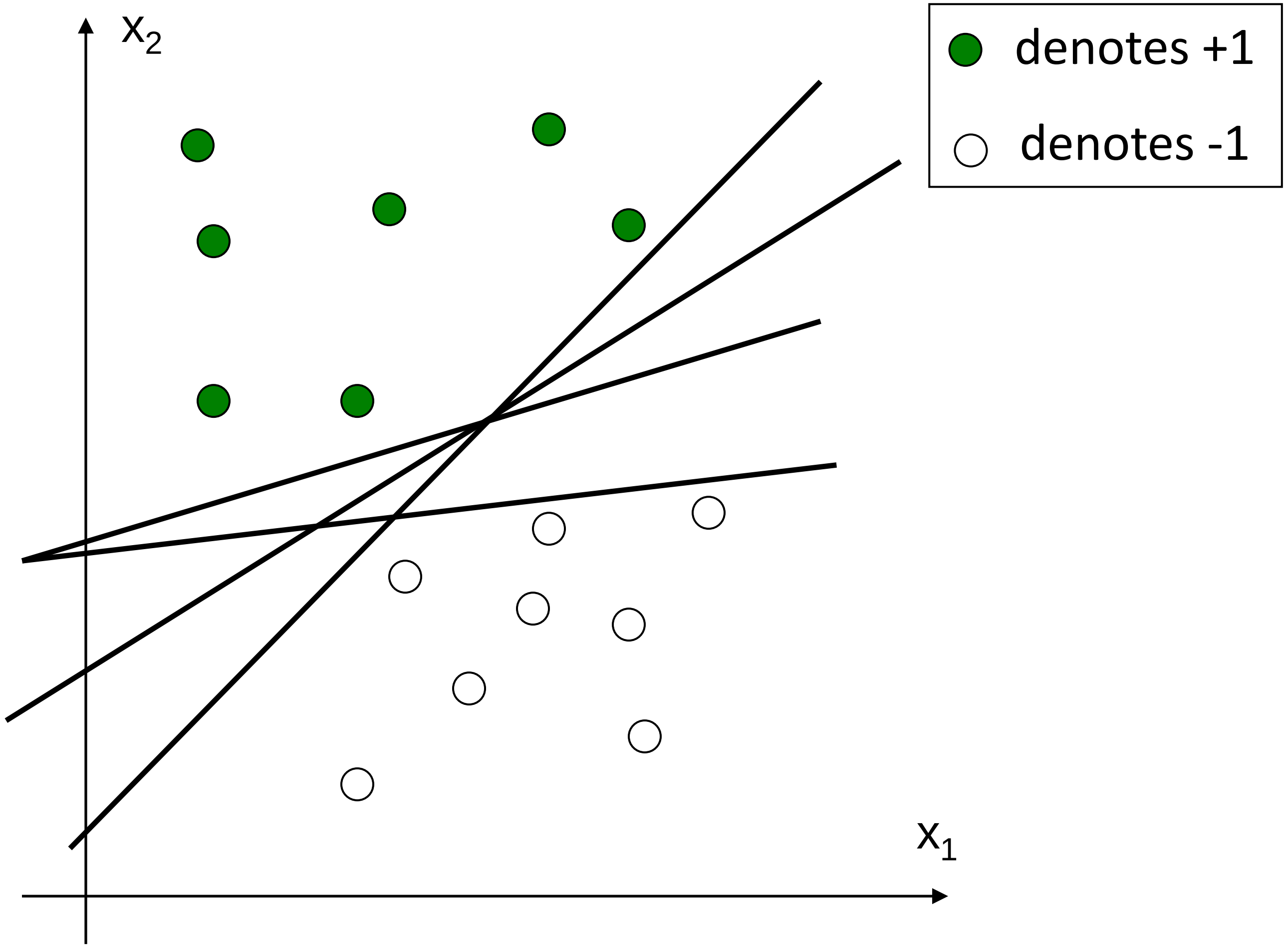
- (Unit-length) normal vector of the hyper-plane:

$$\mathbf{n}=\frac{\mathbf{w}}{\|\mathbf{w}\|}$$



Problem setting for SVM

- How would you classify these points using a linear discriminant function in order to minimize the error rate?



- Infinite number of answers!
- Which one is the best?

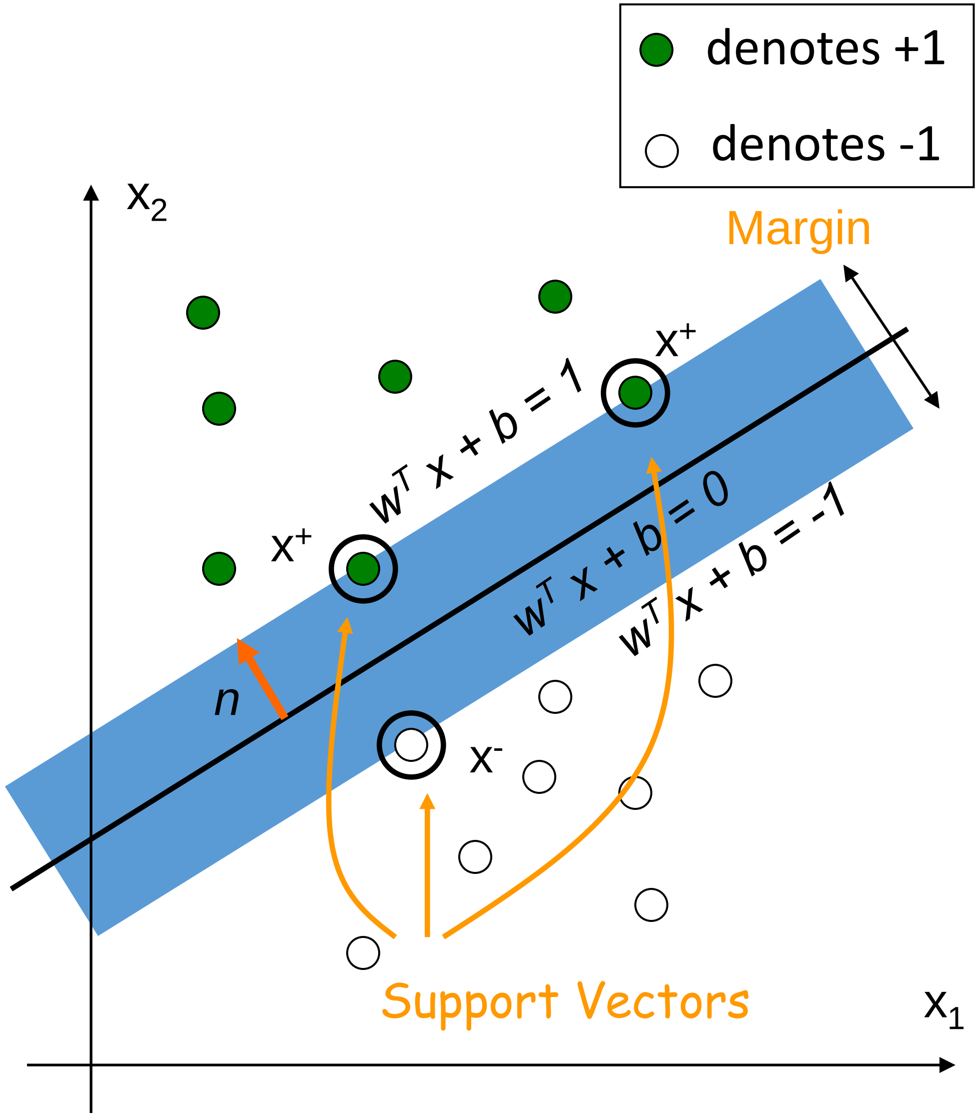
Large Margin Linear Classifier

- We know that

$$\mathbf{w}^T\mathbf{x}^+ + b = 1$$
$$\mathbf{w}^T\mathbf{x}^- + b = -1$$

- The margin width is:

$$M = (\mathbf{x}^+ - \mathbf{x}^-)\cdot\mathbf{n}$$
$$= (\mathbf{x}^+ - \mathbf{x}^-)\cdot\frac{\mathbf{w}}{\|\mathbf{w}\|} = \frac{2}{\|\mathbf{w}\|}$$



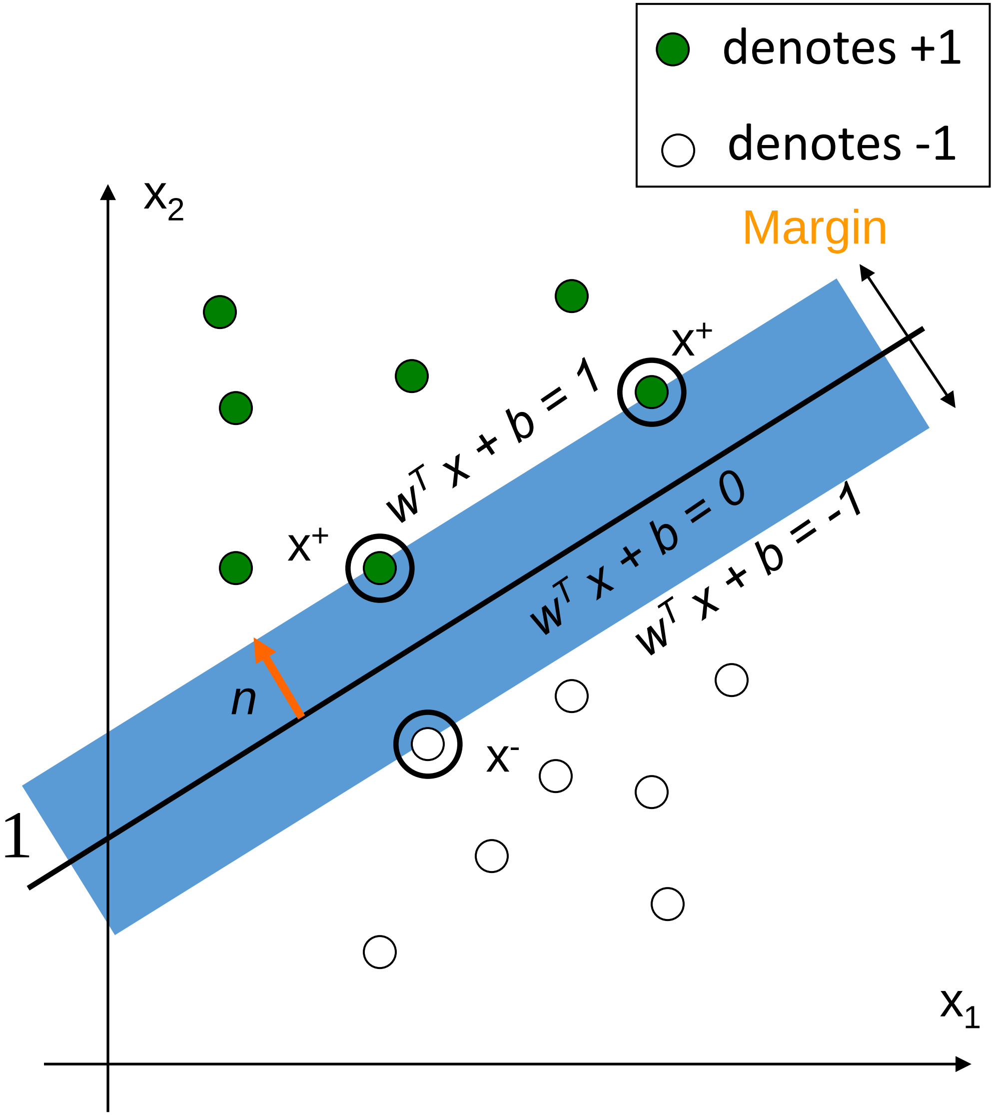
Large Margin Linear Classifier

- Formulation:

$$\text{maximize} \quad \frac{2}{\|\mathbf{w}\|}$$

such that

$$\text{For } y_i = +1, \quad \mathbf{w}^T \mathbf{x}_i + b \geq 1$$
$$\text{For } y_i = -1, \quad \mathbf{w}^T \mathbf{x}_i + b \leq -1$$



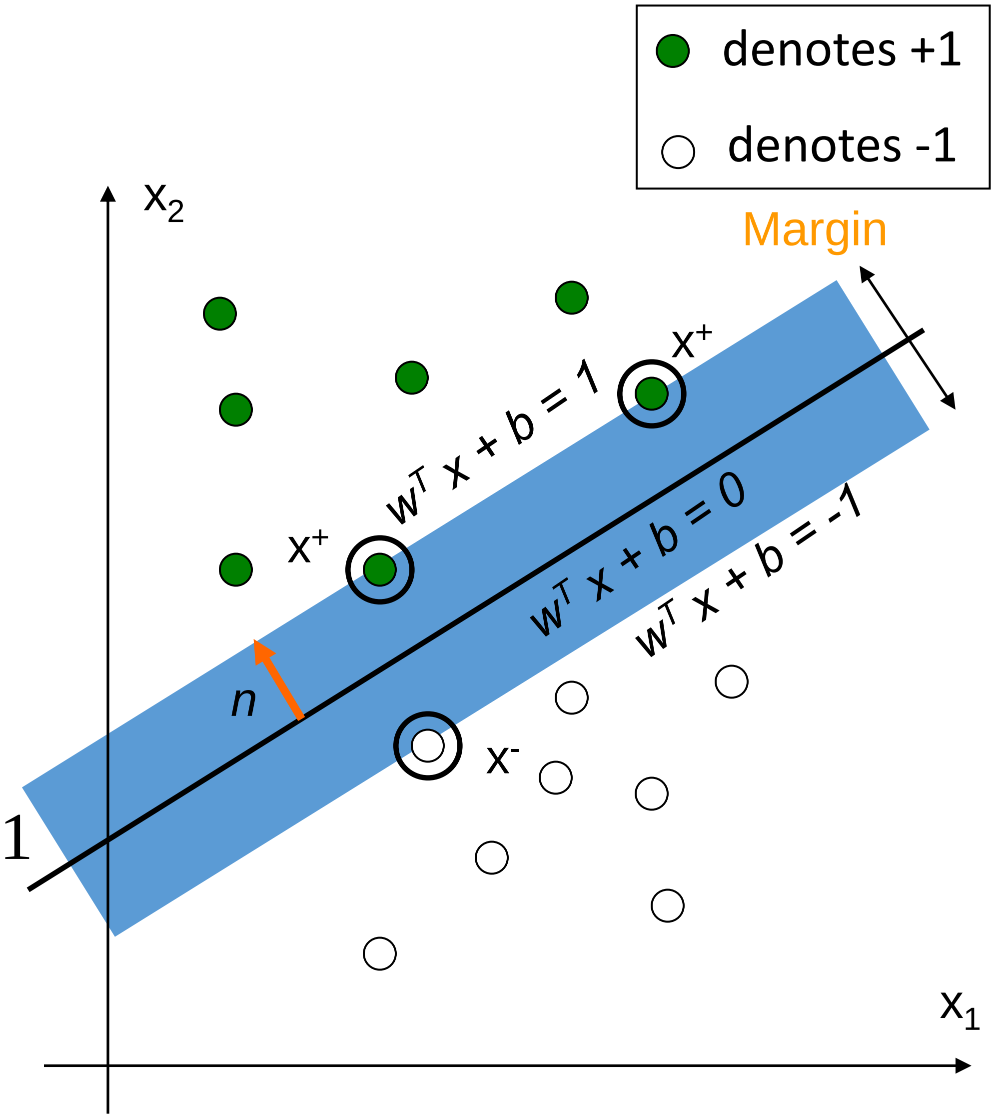
Large Margin Linear Classifier

- Formulation:

$$\text{minimize} \quad \frac{1}{2}\|\mathbf{w}\|^2$$

such that

$$\text{For } y_i = +1, \quad \mathbf{w}^T\mathbf{x}_i + b \geq 1$$
$$\text{For } y_i = -1, \quad \mathbf{w}^T\mathbf{x}_i + b \leq -1$$



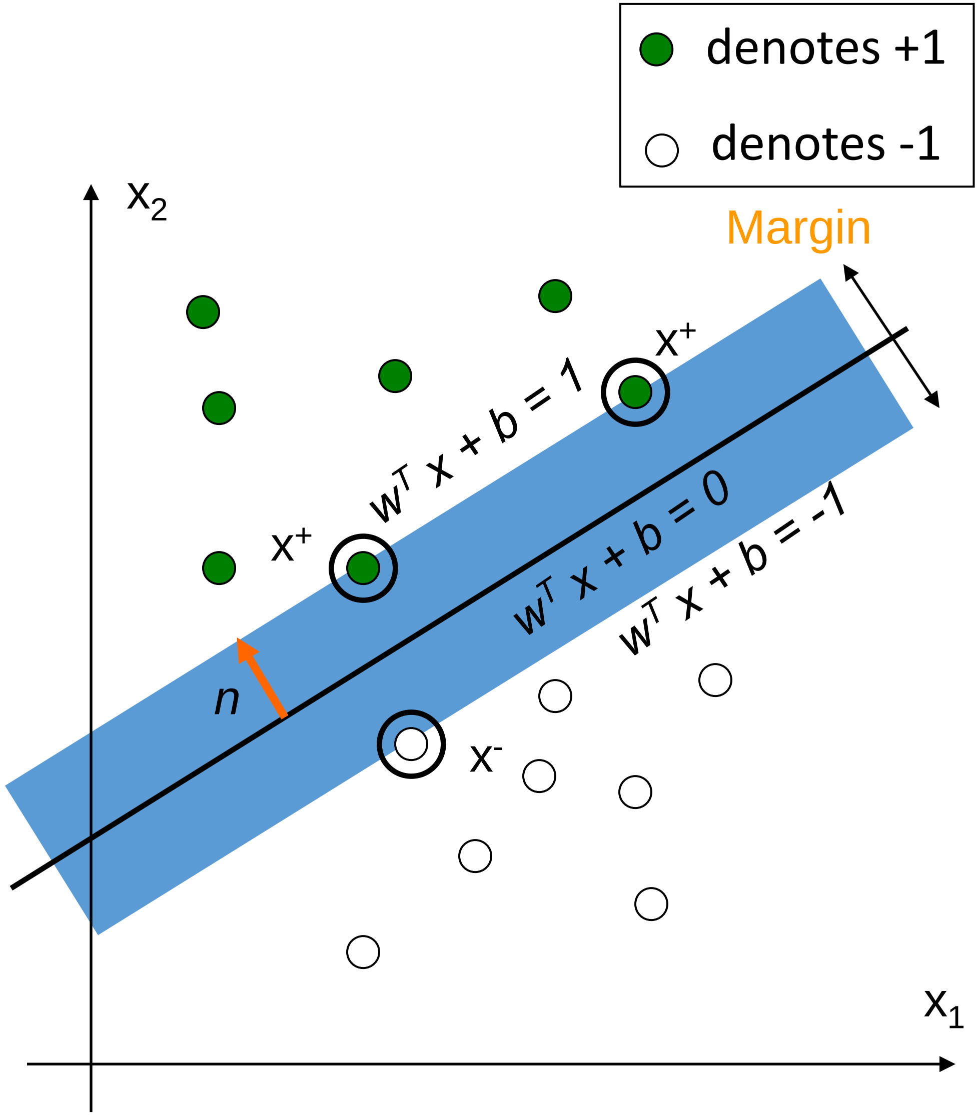
Large Margin Linear Classifier

- Formulation:

$$\text{minimize} \quad \frac{1}{2}\|\mathbf{w}\|^2$$

such that

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$$



Solving the Optimization Problem

Quadratic prog
ramming
with linear con
straints

$$\text{minimize} \quad \frac{1}{2}\|\mathbf{w}\|^2$$

$$\text{s.t.} \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$$

Lagrangian
Function

$$\text{minimize} \;\; L_p(\mathbf{w}, b, \alpha_i) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i \left(y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 \right)$$

$$\text{s.t.} \quad \alpha_i \geq 0$$

Solving the Optimization Problem

$$\text{minimize} \;\; L_p(\mathbf{w}, b, \alpha_i) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i \left(y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 \right)$$

s.t. $\alpha_i \geq 0$

$$\frac{\partial L_p}{\partial \mathrm{w}} = \mathrm{w} - \sum_{\mathrm{i}=1}^{\mathrm{n}} \alpha_i y_i \mathrm{x}_i \quad \Rightarrow \quad \mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$

$$\frac{\partial L_p}{\partial \mathrm{b}} = -\sum_{\mathrm{i}=1}^{\mathrm{n}} \alpha_i y_i \quad \Rightarrow \quad \sum_{i=1}^{n} \alpha_i y_i = 0$$

Solving the Optimization Problem

$$\text{minimize} \quad L_p(\mathbf{w}, b, \alpha_i) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i \left(y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1 \right)$$

s.t. $\quad \alpha_i \geq 0$

Lagrangian Dual Problem

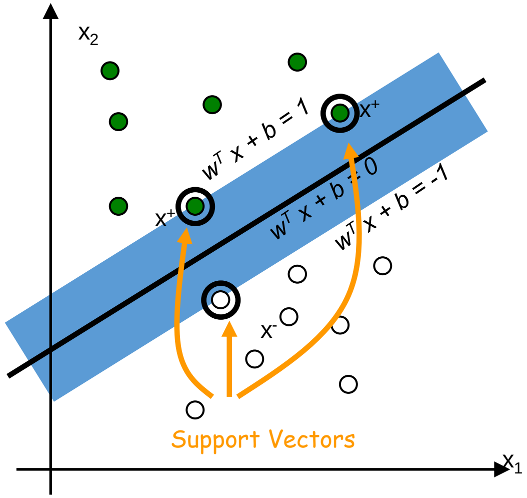
$$\text{maximize} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

s.t. $\quad \alpha_i \geq 0$, and $\quad \sum_{i=1}^{n} \alpha_i y_i = 0$

Solving the Optimization Problem

- From KKT condition, we know:

$$\alpha_i \left(y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1 \right) = 0$$



- Thus, only support vectors have $\alpha_i \neq 0$

- The solution has the form:

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i = \sum_{i \in SV} \alpha_i y_i \mathbf{x}_i$$

get b from $y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1 = 0,$

where $\mathbf{x}_i$ is support vector

Solving the Optimization Problem

- The linear discriminant function is:

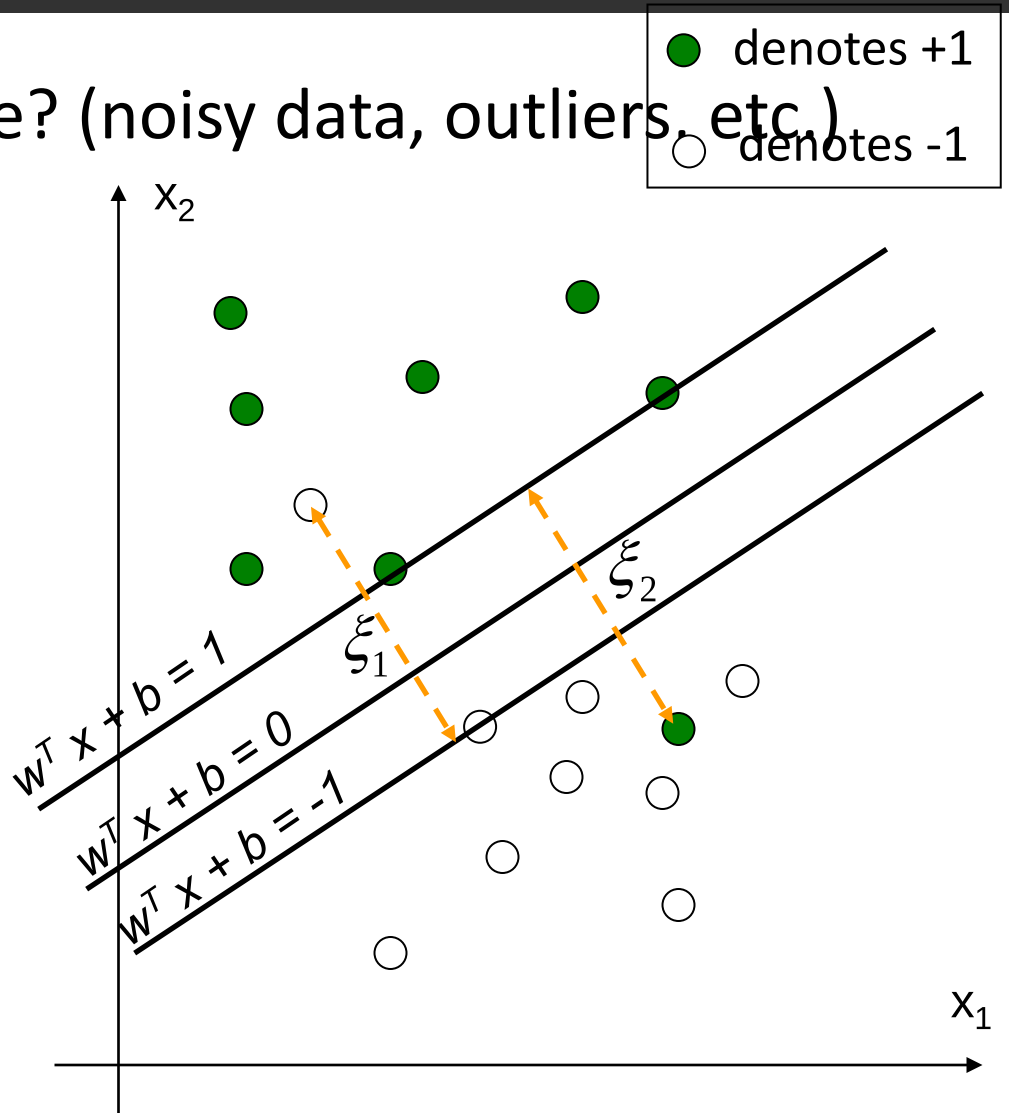
$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = \sum_{i \in \mathrm{SV}} \alpha_i \mathbf{x}_i^T \mathbf{x} + b$$

- Notice it relies on a *dot product* between the test point x and the support vectors x_i

- Also keep in mind that solving the optimization problem involved computing the dot products $x_i^T x_j$ between all pairs of training points

Large Margin Linear Classifier

- What if data is not linear separable? (noisy data, outliers, etc.)



- Slack variables ξ_i can be added to allow miss-classification of difficult or noisy data points

Large Margin Linear Classifier

- Formulation:

$$\text{minimize} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i$$

such that

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i$$

$$\xi_i \geq 0$$

- Parameter *C* can be viewed as a way to control over-fitting.

Large Margin Linear Classifier

- Formulation: (Lagrangian Dual Problem)

$$\text{maximize} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

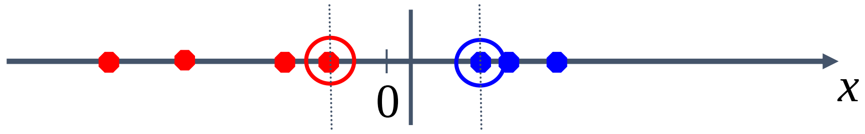
such that

$$0 \le \alpha_i \le C$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

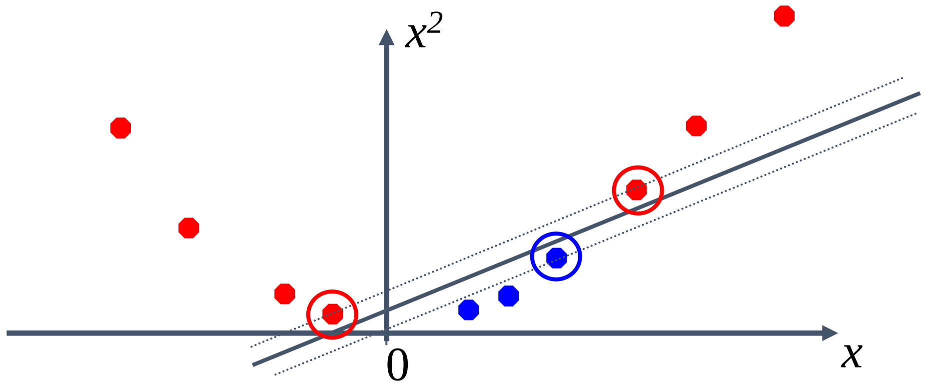
Non-linear SVM

- Datasets that are linearly separable with noise work out great:



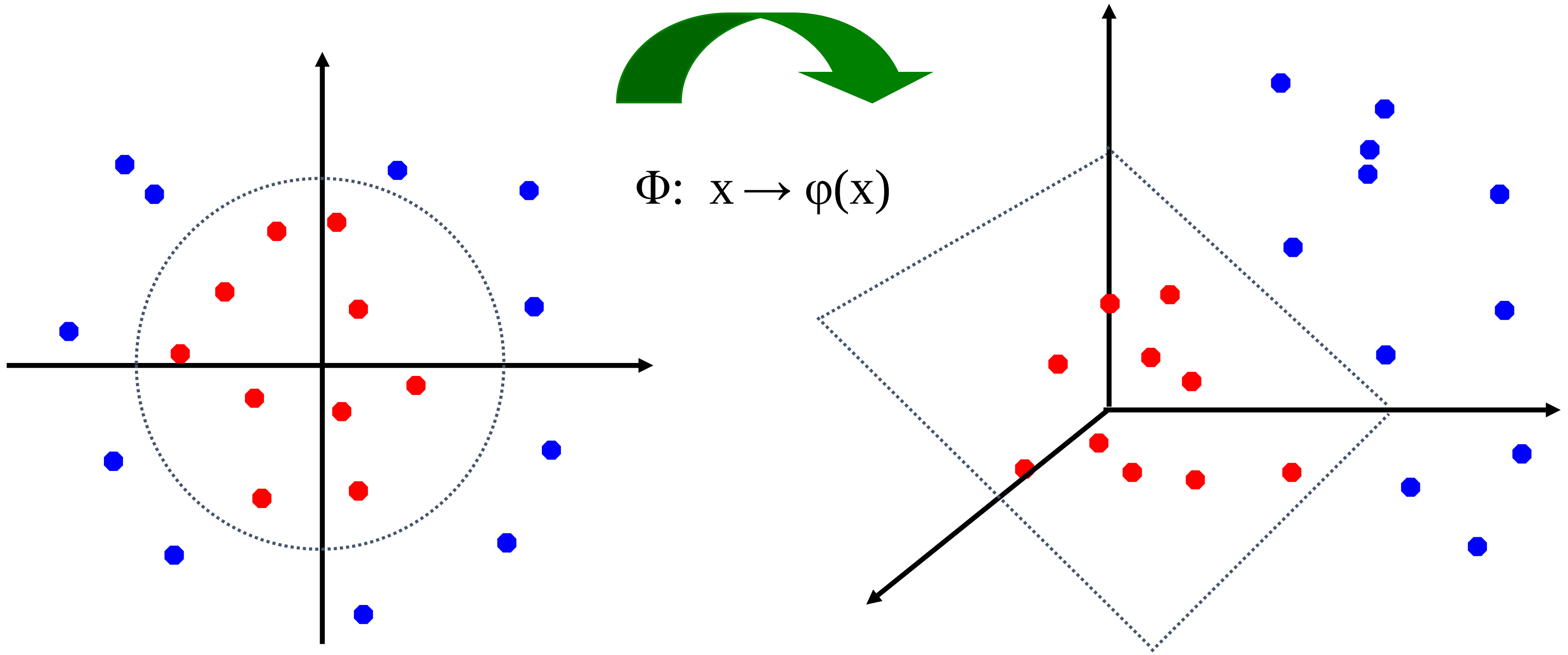
- But what are we going to do if the dataset is just too hard?

- How about… mapping data to a higher-dimensional space:



Non-linear SVMs: Feature Space

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



Non-linear SVMs: The Kernel Trick

- With this mapping, our discriminant function is now:

$$g(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b = \sum_{i \in SV} \alpha_i \boxed{\phi(\mathbf{x}_i)^T \phi(\mathbf{x})} + b$$

- No need to know this mapping explicitly, because we only use the dot product o f feature vectors in both the training and test.

- A *kernel function* is defined as a function that corresponds to a dot product of two feature vectors in some expanded feature space:

$$K(\mathbf{x}_i, \mathbf{x}_j) \equiv \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

Non-linear SVMs: The Kernel Trick

- Examples of commonly-used kernel functions:
  - Linear kernel: $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$
  - Polynomial kernel: $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^p$
  - Gaussian (Radial-Basis Function (RBF)) kernel:

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

Nonlinear SVM: Optimization

- Formulation: (Lagrangian Dual Problem)

$$\text{maximize} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

such that

$$0 \le \alpha_i \le C$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

- The solution of the discriminant function is

$$g(\mathbf{x}) = \sum_{i \in \mathrm{SV}} \alpha_i K(\mathbf{x}_i, \mathbf{x}) + b$$

- The optimization technique is the same.

Support Vector Machine: Algorithm

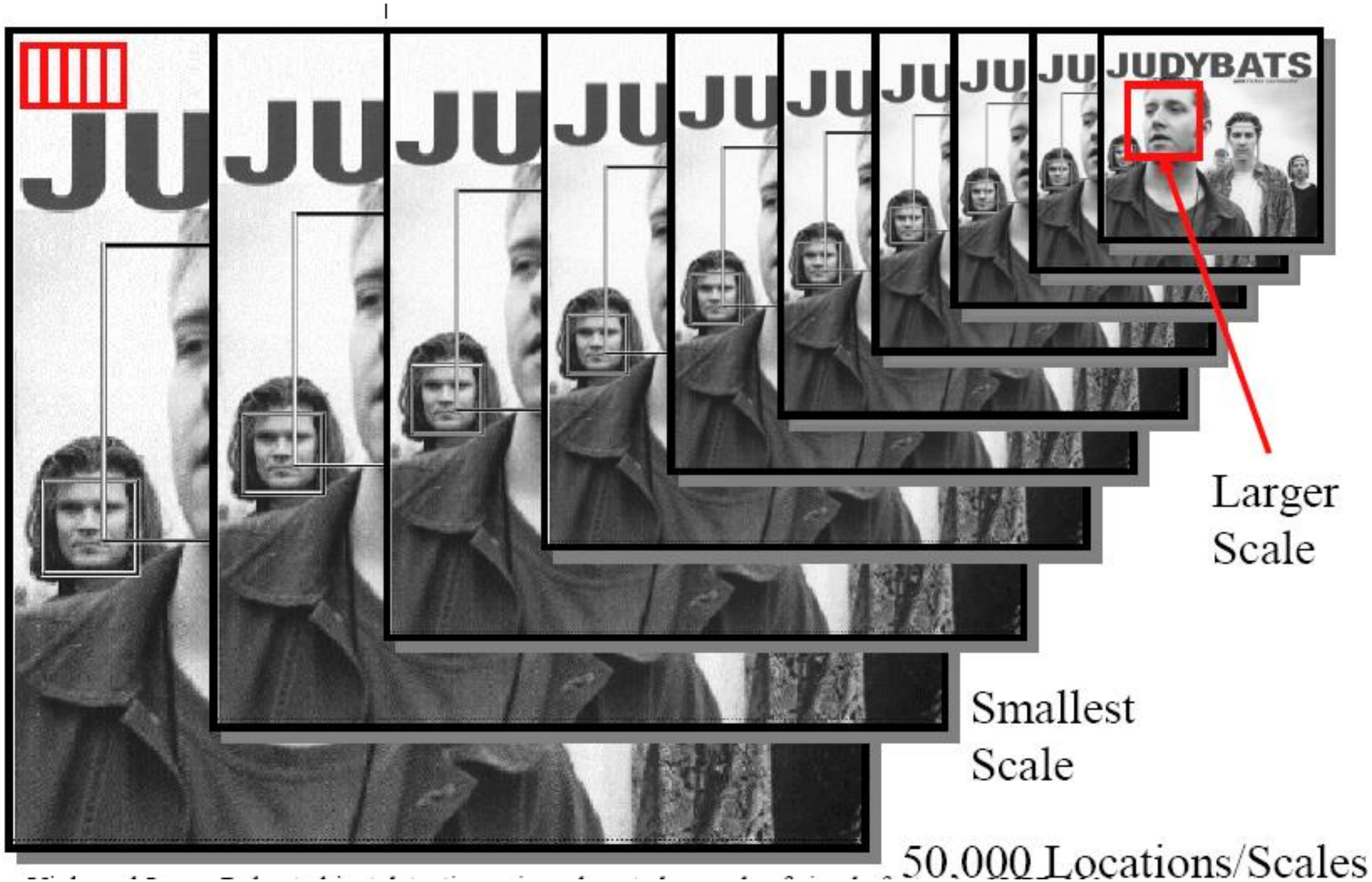
- 1. Choose a kernel function
- 2. Choose a value for C
- 3. Solve the quadratic programming problem (many algorithms and software packages available)
- 4. Construct the discriminant function from the support vectors

Summary: Support Vector Machine

- 1. Large Margin Classifier
  - Better generalization ability & less over-fitting

- 2. The Kernel Trick
  - Map data points to higher dimensional space in order to make them linearly separable.
  - Since only dot product is used, we do not need to represent the mapping explicitly.

Scan classifier over locs. & scales



“Learn” classifier from data

- Training Data
- 5000 faces (frontal)
- 108 non faces
- Faces are normalized
  - Scale, translation
- Many variations
- Across individuals
- Illumination
- Pose (rotation both in plane and out)

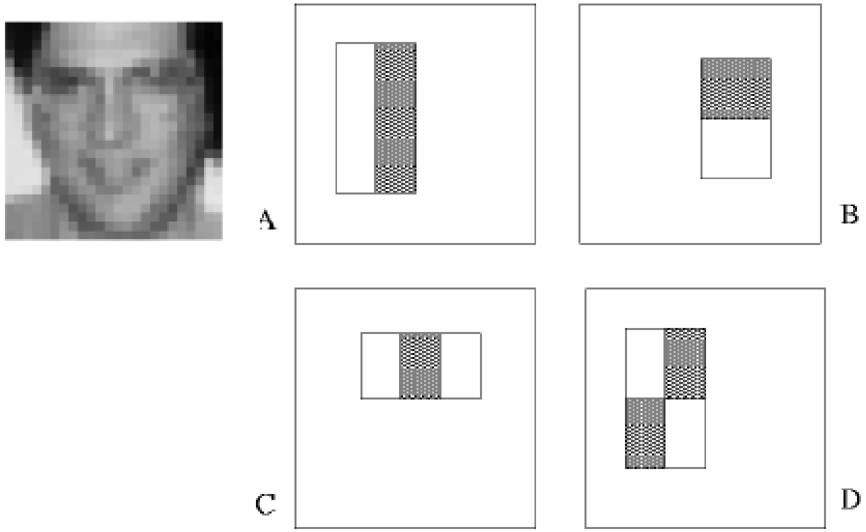
Characteristics of algorithm

- Feature set (…is huge about 16M features)
- Efficient feature selection using AdaBoost
- New image representation: Integral Image
- Cascaded Classifier for rapid detection

- Fastest known face detector for gray scale images

Image features

- "Rectangle filters"
  - Similar to Haar wavelets
- Differences between sums of pixels in adjacent rectangles

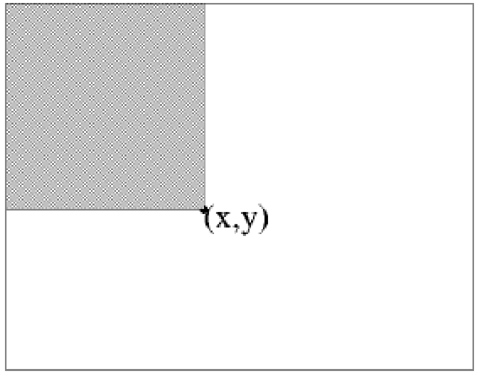


$$h_t(x) = \begin{cases} +1 & \text{if } f_t(x) > \theta_t \\ -1 & \text{otherwise} \end{cases}$$

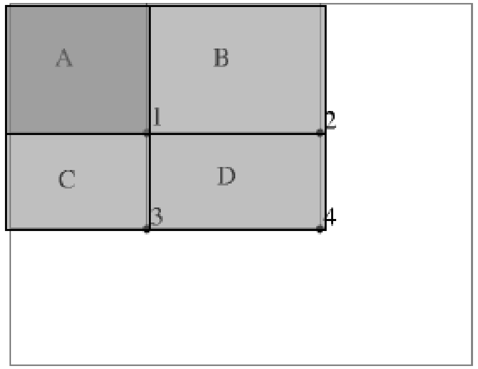
Integral Image

- Partial sum

$$I'(x,y) = \sum_{\substack{x' \le x \\ y' \le y}} I(x',y')$$

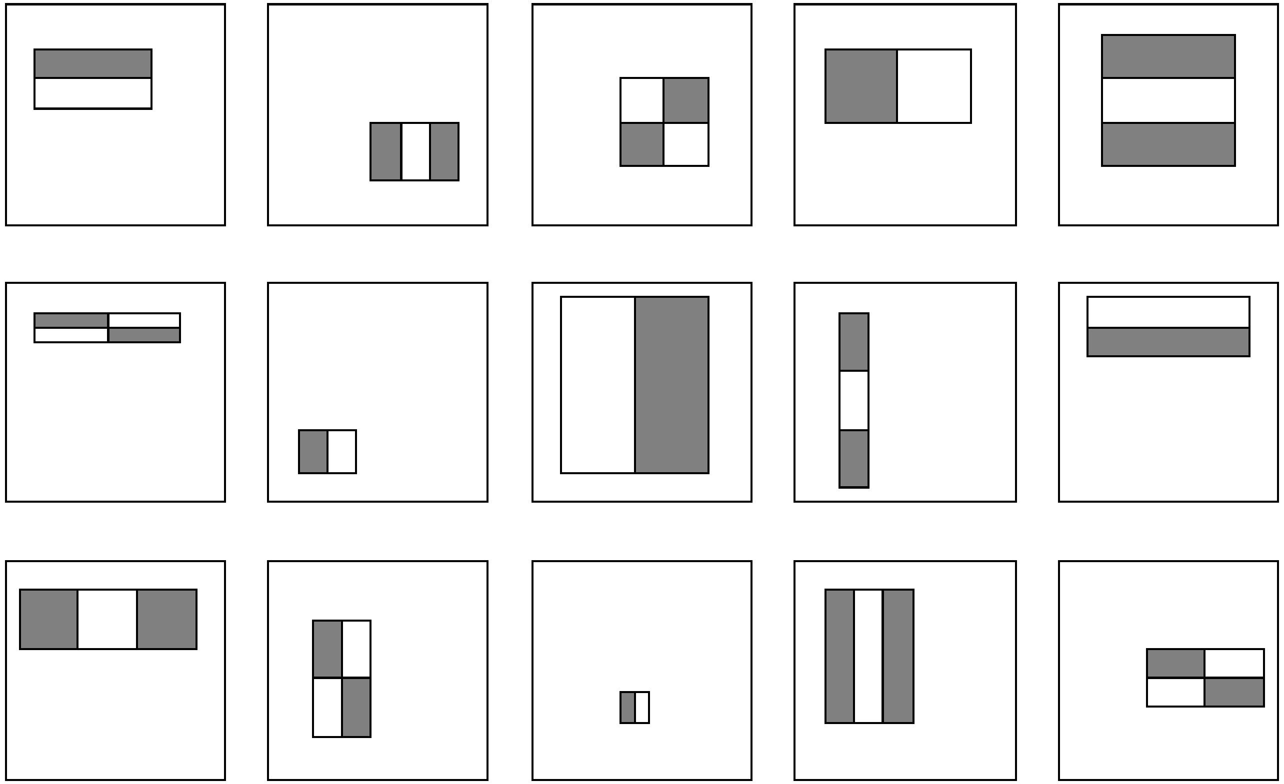


- Any rectangle is
- D = 1+4-(2+3)



- Also known as:
- summed area tables [Crow84]
- boxlets [Simard98]

Huge library of filters

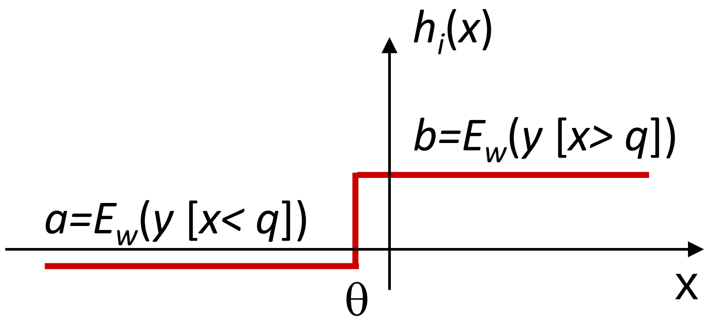


Constructing the classifier

- Perceptron yields a sufficiently powerful classifier

$$C(x) = \theta\left(\sum_i \alpha_i h_i(x) + b\right)$$

- Use AdaBoost to efficiently choose best features
- add a new hi(x) at each round
- each hi(xk) is a “decision stump”



Constructing the classifier

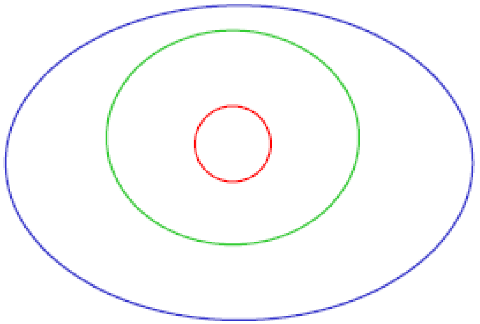
- For each round of boosting:
- Evaluate each rectangle filter on each example
- Sort examples by filter values
- Select best threshold for each filter (min error)
  - Use sorting to quickly scan for optimal threshold
- Select best filter/threshold combination
- Weight is a simple function of error rate
- Reweight examples
  - (There are many tricks to make this more efficient.)

Good reference on boosting

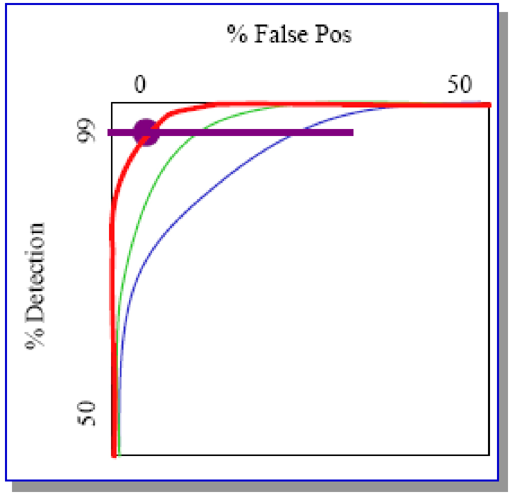
- Friedman, J., Hastie, T. and Tibshirani, R. Additive Logistic Regression: a Statistical View of Boosting
  - http://www-stat.stanford.edu/~hastie/Papers/boost.ps
- “We show that boosting fits an additive logistic regression model by stagewise optimization of a criterion very similar to the log-likelihood, and present likelihood based alternatives. We also propose a multi-logit boosting procedure which appears to have advantages over other methods proposed so far.”

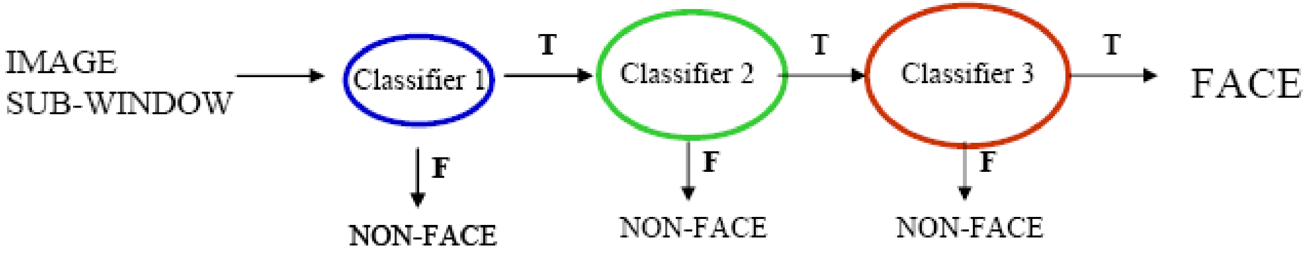
Trading speed for accuracy

- Given a nested set of classifier hypothesis classes



- Computational Risk Minimization

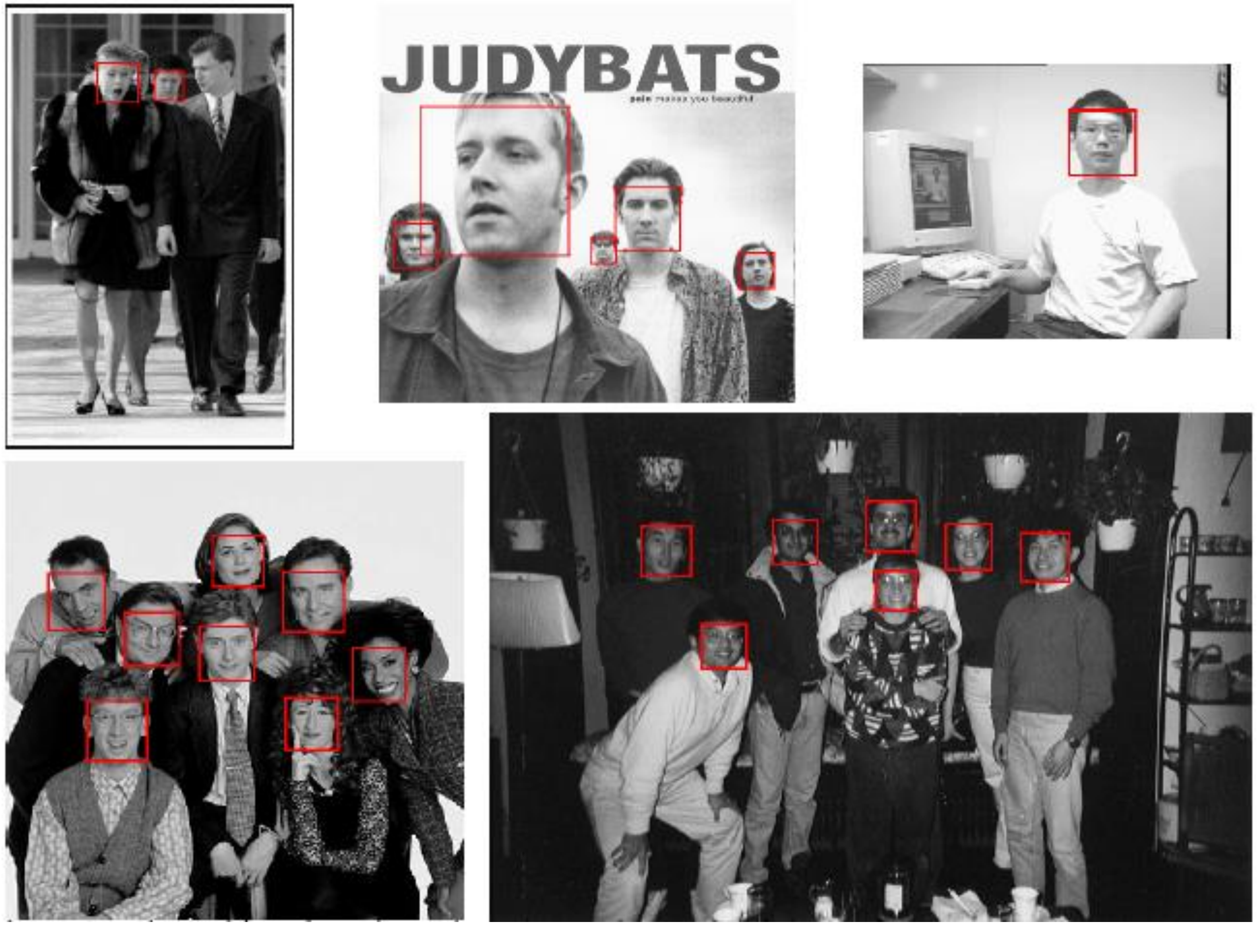




Speed of face detector (2001)

- Speed is proportional to the average number of features computed per sub-window.
- On the MIT+CMU test set, an average of 9 features (/ 6061) are computed per sub-window.
- On a 700 Mhz Pentium III, a 384x288 pixel image takes about 0.067 seconds to process (15 fps).
- Roughly 15 times faster than Rowley-Baluja-Kanade and 600 times faster than Schneiderman-Kanade.

Sample results



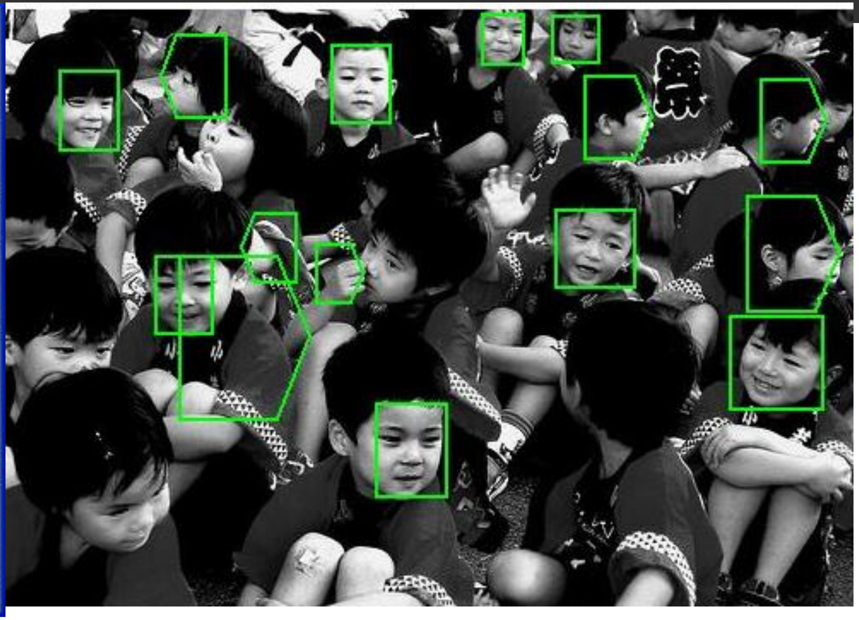
Summary (Viola-Jones)

- Fastest known face detector for gray images
- Three contributions with broad applicability:
  - Cascaded classifier yields rapid classification
  - AdaBoost as an extremely efficient feature selector
  - Rectangle Features + Integral Image can be used for rapid image analysis

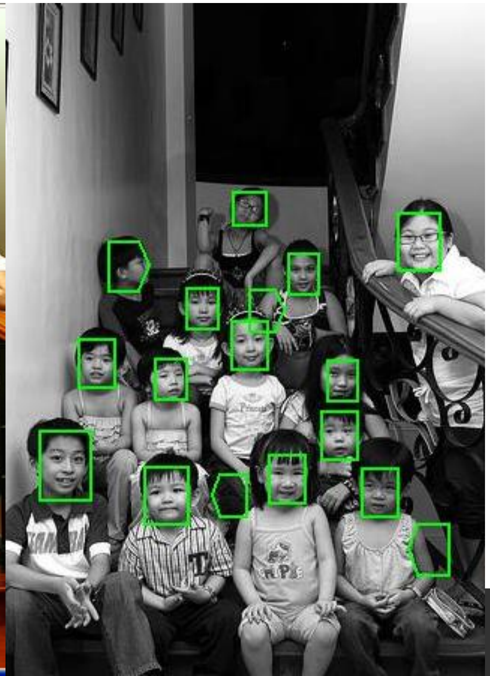
Face detector comparison

- Informal study by Andrew Gallagher, CMU, for Learning-Based Methods in Vision, Spring 2007
  - The Viola Jones algorithm OpenCV implementation was used. (<2 sec per image).
  - For Schneiderman and Kanade, Object Detection Using the Statistics of Parts [IJCV'04], the www.pittpatt.com demo was used. (~10-15 seconds per image, including web transmission).

Face detector comparison



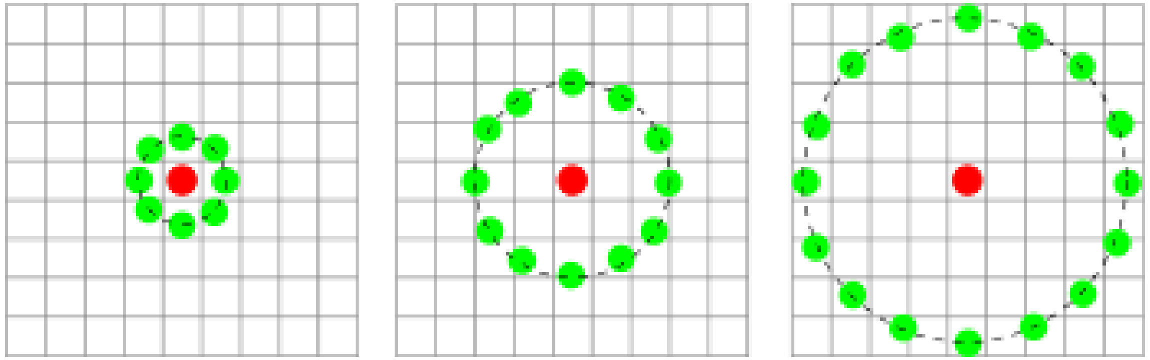
Viola
Jones



Schneiderman
Kanade

Local Binary Pattern

- Divide the examined window to cells (e.g. 16x16 pixels for each cell).
- For each pixel in a cell, compare the pixel to each of its 8 neighbors (on its left-top, left-middle, left-bottom, right-top, etc.). Follow the pixels along a circle, i.e. clockwise or counter-clockwise.



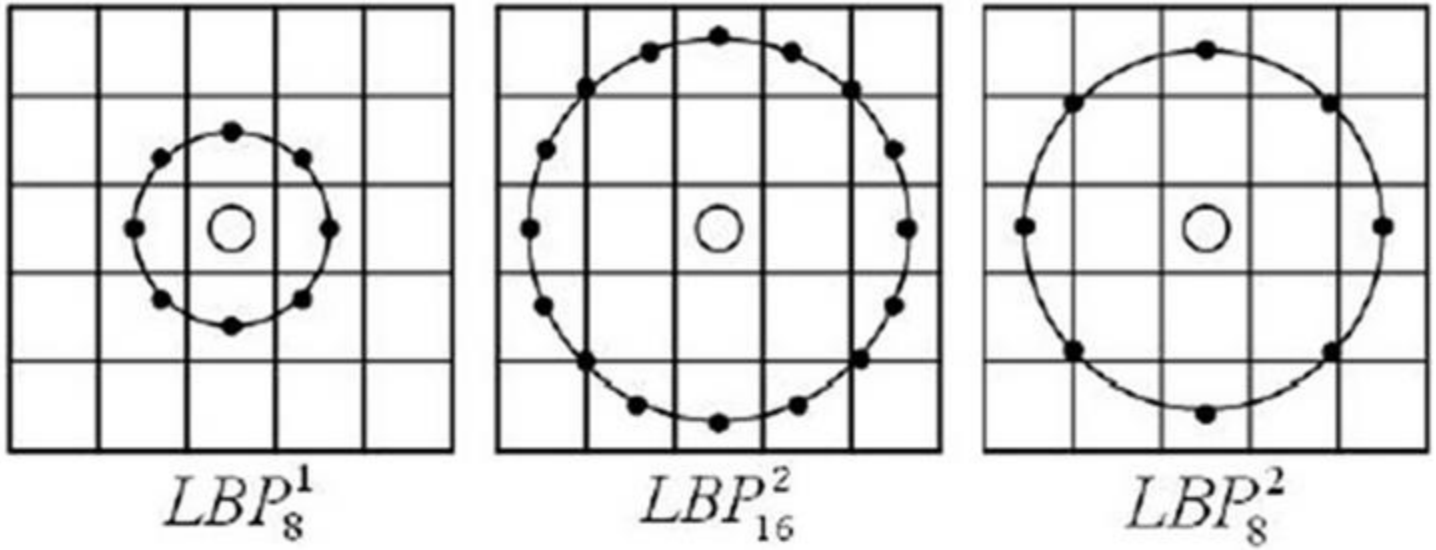
Local Binary Pattern

- Where the center pixel's value is greater than the neighbor, write "1". Other wise, write "0". This gives an 8-digit binary number (which is usually convert ed to decimal for convenience).
- Compute the histogram, over the cell, of the frequency of each "number" oc curring (i.e., each combination of which pixels are smaller and which are gre ater than the center).

Local Binary Pattern

- Optionally normalize the histogram.
- Concatenate normalized histograms of all cells. This gives the feature vector for the window.

- LBP Operations

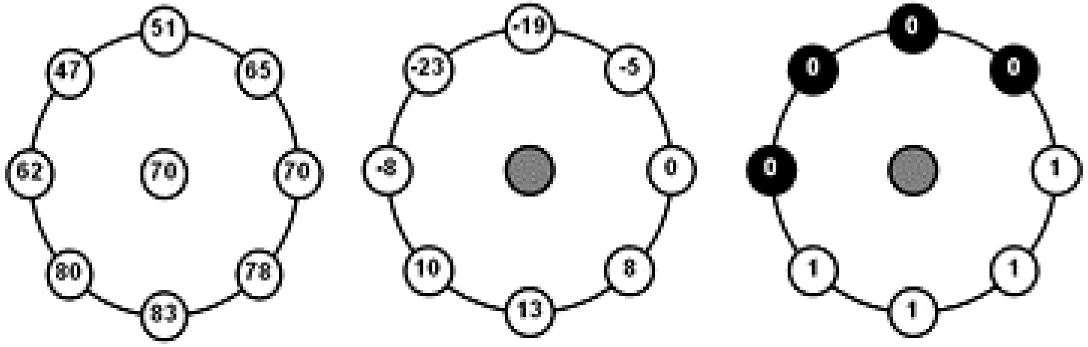


Concept

- Illustration

The value of the LBP code of a pixel (x_c, y_c) is given by:

$$LBP_{P,R} = \sum_{p=0}^{P-1} s(g_p - g_c)2^p \qquad s(x) = \begin{cases} 1, \ if \ x \geq 0; \\ 0, \ otherwise. \end{cases}$$



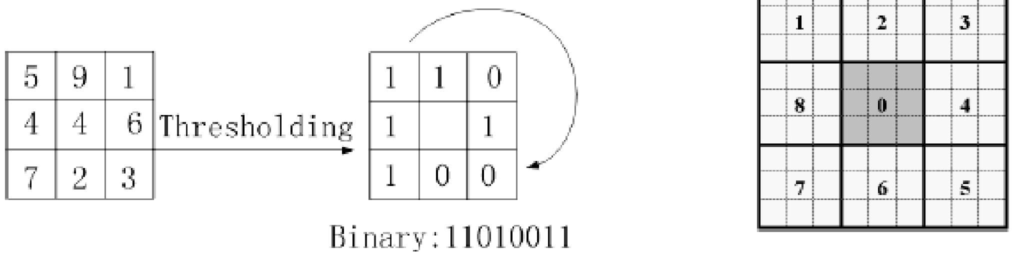
1. Sample 2. Difference 3. Threshold

1*1 + 1*2 + 1*4 + 1*8 + 0*16 + 0*32 + 0*64 + 0*128 = 15

4. Multiply by powers of two and sum

Multi-scale Block LBP

- Shengcai Liao et al.



(a) (b)

Fig. 1. (a) The basic LBP operator. (b) The 9×9 MB-LBP operator. In each sub-region, average sum of image intensity is computed. These average sums are then thresholded by that of the center block. MB-LBP is then obtained.

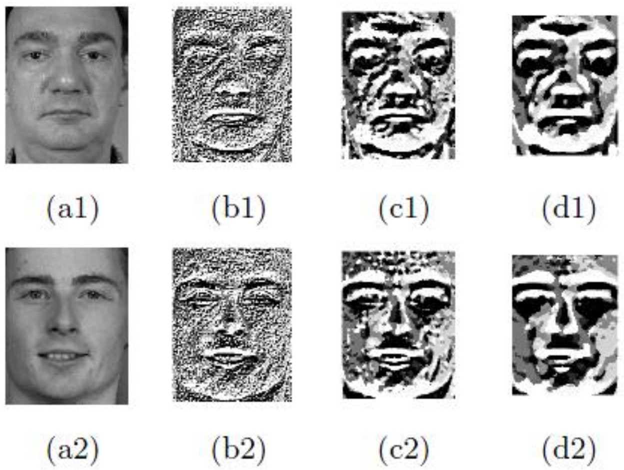
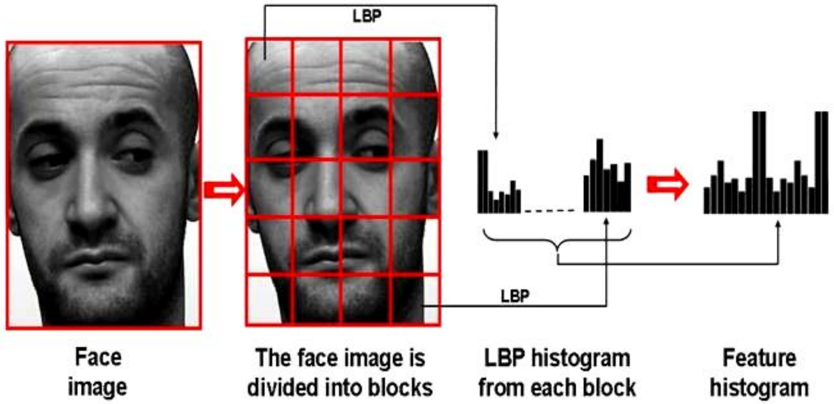


Fig. 2. MB-LBP filtered images of two different faces. (a) original images; (b) filtered by 3 × 3 MB-LBP (c) filtered by 9 × 9 MB-LBP; (d) filtered by 15 × 15 MB-LBP.

Face Description

- The basic methodology for LBP based face description proposed by Ahonen et al. (2006) is as follows:
  - The facial image is divided into local regions and LBP texture descriptors are extracted from each region independently. The descriptors are then concatenated to form a global description of the face

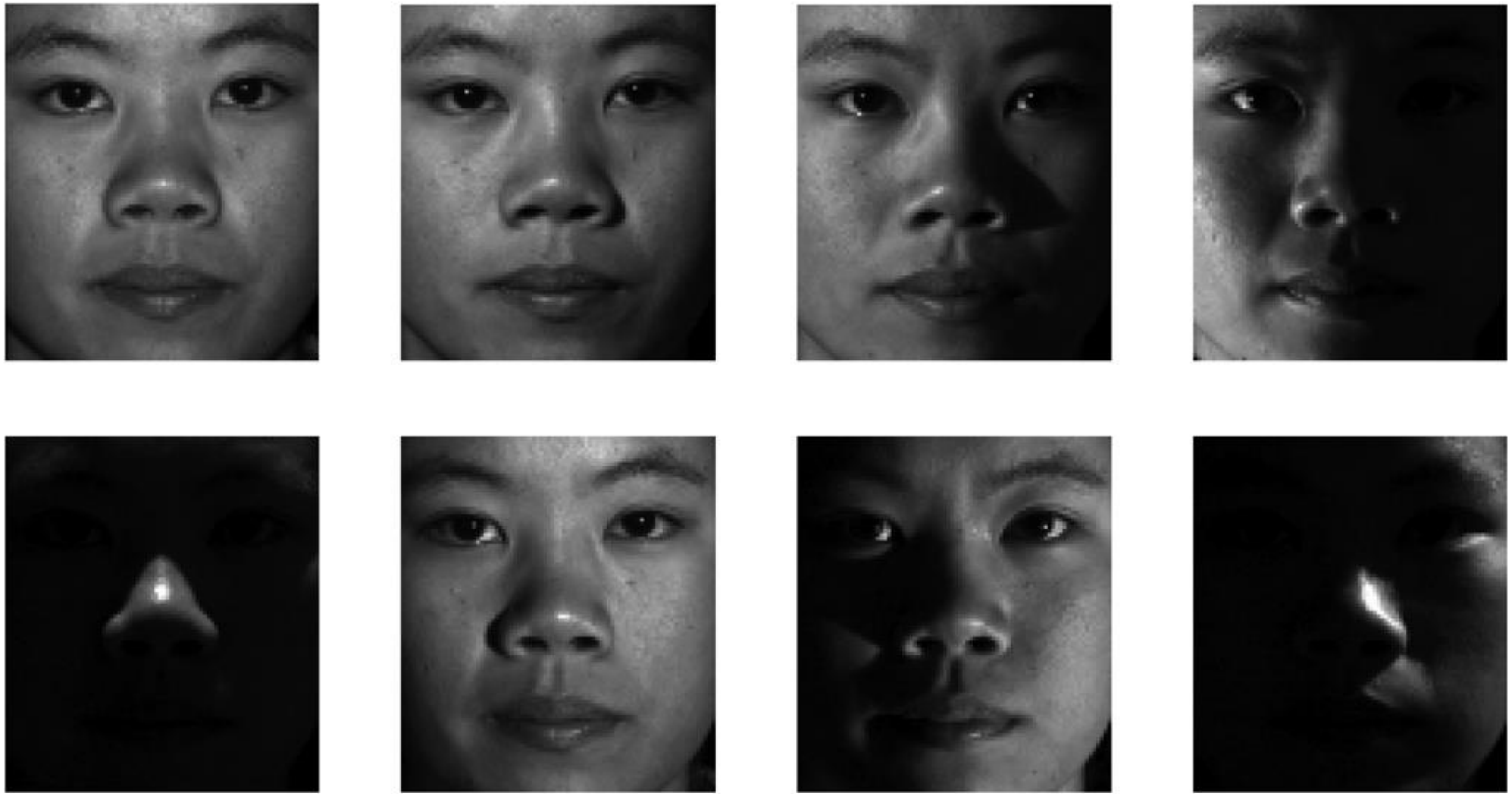


Face Recognition with LBP

- Steps
  - Build Gallery LBP Histograms
  - Build the Probe LBP Histogram
  - Histogram
    - The recognition is performed using a nearest neighbor classifier
    - in the computed feature space with Chi square as a dissimilarity measure.

Face Recognition with LBP

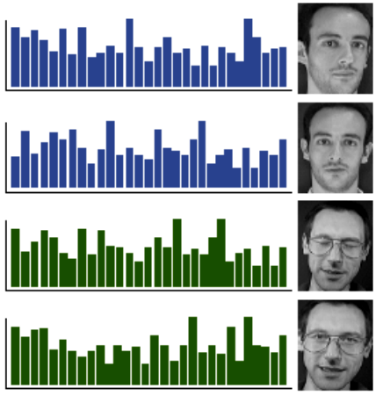
- Gallery and Probe Pictures



Face Recognition with LBP

- Gallery and Probe Pictures

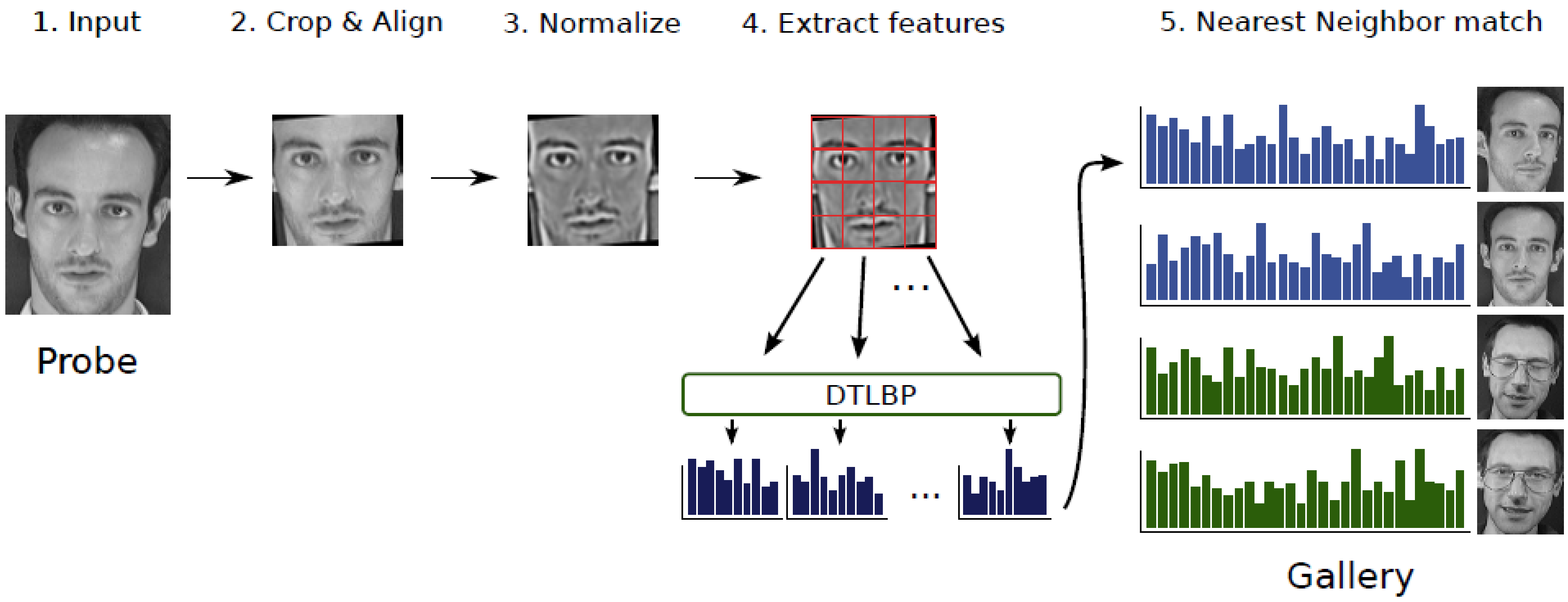
Probe



Gallery

Face Recognition with LBP

- Face Recognition with Decision Tree-based Local Binary Patterns



Histogram Matching

- Many similarity measures for histogram matching have been proposed
  - histogram intersection is used to measure the similarity between two histograms

$$S_{HI}(\mathbf{H}, \mathbf{S}) = \sum_{i=1}^{B} \min(H_i, S_i) \tag{14}$$

where $S_{HI}(\mathbf{H}, \mathbf{S})$ is the histogram intersection statistic with $\mathbf{H} = (H_1, H_2, \cdots, H_B)^T$ and $\mathbf{S} = (S_1, S_2, \cdots, S_B)^T$. Equation (14) is used to calculate the similarity for the nearest neighbor classifier. This measure has an intuitive motivation in that it calculates the common parts of two histograms. Its computational complexity is very low as it requires only simple operations. It should be noted that it is also possible to use other measures such as the chi-square distance [17].

Histogram Matching

- Chi square as a dissimilarity measure

$$\chi^2(\mathbf{S},\mathbf{M}) = \sum_i \frac{(S_i - M_i)^2}{S_i + M_i}$$

- Log-likelihood statistic

$$L(\mathbf{S},\mathbf{M}) = -\sum_i S_i \log M_i$$

- Weighted X 2

$$\chi^2_w(\mathbf{S},\mathbf{M}) = \sum_{i,j} w_j \frac{(S_{i,j} - M_{i,j})^2}{S_{i,j} + M_{i,j}}$$

Pedestrian detection using HoG-SVM

```
import cv2
import matplotlib.pyplot as plt

image = cv2.imread('../data/people.jpg')

hog = cv2.HOGDescriptor()
hog.setSVMDetector(cv2.HOGDescriptor.getDefaultPeopleDetector())

locations, weights = hog.detectMultiScale(image)

dbg_image = image.copy()
for loc in locations:
    cv2.rectangle(dbg_image, (loc[0], loc[1]),
                  (loc[0]+loc[2], loc[1]+loc[3]), (0, 255, 0), 2)

plt.figure(figsize=(12,6))
plt.subplot(121)
plt.title('original')
plt.axis('off')
plt.imshow(image[:,:,[2,1,0]])
plt.subplot(122)
plt.title('detections')
plt.axis('off')
plt.imshow(dbg_image[:,:,[2,1,0]])
plt.tight_layout()
plt.show()
```

Optical character recognition using KNN and SVM

```python
import cv2
import numpy as np

CELL_SIZE = 20
NCLASSES = 10
TRAIN_RATIO = 0.8

digits_img = cv2.imread('../data/digits.png', 0)
digits = [np.hsplit(r, digits_img.shape[1] // CELL_SIZE)
          for r in np.vsplit(digits_img, digits_img.shape[0] // CELL_SIZE)]
digits = np.array(digits).reshape(-1, CELL_SIZE, CELL_SIZE)
nsamples = digits.shape[0]
labels = np.repeat(np.arange(NCLASSES), nsamples // NCLASSES)

for i in range(nsamples):
    m = cv2.moments(digits[i])
    if m['mu02'] > 1e-3:
        s = m['mu11'] / m['mu02']
        M = np.float32([[1, -s, 0.5*CELL_SIZE*s],
                        [0, 1, 0]])
        digits[i] = cv2.warpAffine(digits[i], M, (CELL_SIZE, CELL_SIZE))

perm = np.random.permutation(nsamples)
digits = digits[perm]
labels = labels[perm]

ntrain = int(TRAIN_RATIO * nsamples)
ntest = nsamples - ntrain

def calc_hog(digits):
    win_size = (20, 20)
    block_size = (10, 10)
    block_stride = (10, 10)
    cell_size = (10, 10)
    nbins = 9
    hog = cv2.HOGDescriptor(win_size, block_size, block_stride, cell_size, nbins)
    samples = []
    for d in digits: samples.append(hog.compute(d))
    return np.array(samples, np.float32)
```

```python
fea_hog_train = calc_hog(digits[:ntrain])
fea_hog_test = calc_hog(digits[ntrain:])
labels_train, labels_test = labels[:ntrain], labels[ntrain:]

K = 3
knn_model = cv2.ml.KNearest_create()
knn_model.train(fea_hog_train, cv2.ml.ROW_SAMPLE, labels_train)

svm_model = cv2.ml.SVM_create()
svm_model.setGamma(2)
svm_model.setC(1)
svm_model.setKernel(cv2.ml.SVM_RBF)
svm_model.setType(cv2.ml.SVM_C_SVC)
svm_model.train(fea_hog_train, cv2.ml.ROW_SAMPLE, labels_train)

def eval_model(fea, labels, fpred):
    pred = fpred(fea).astype(np.int32)
    acc = (pred.T == labels).mean()*100

    conf_mat = np.zeros((NCLASSES, NCLASSES), np.int32)
    for c_gt, c_pred in zip(labels, pred):
        conf_mat[c_gt, c_pred] += 1

    return acc, conf_mat

knn_acc, knn_conf_mat = eval_model(fea_hog_test, labels_test,
                                   lambda fea: knn_model.findNearest(fea, K)[1])
print('KNN accuracy (%):', knn_acc)
print('KNN confusion matrix:')
print(knn_conf_mat)

svm_acc, svm_conf_mat = eval_model(fea_hog_test, labels_test,
                                   lambda fea: svm_model.predict(fea)[1])
print('SVM accuracy (%):', svm_acc)
print('SVM confusion matrix:')
print(svm_conf_mat)
```

Face detection using Haar and LBP features

```
import cv2
import numpy as np

def detect_faces(video_file, detector, win_title):
    cap = cv2.VideoCapture(video_file)

    while True:
        status_cap, frame = cap.read()
        if not status_cap:
            break

        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        faces = detector.detectMultiScale(gray, 1.3, 5)

        for x, y, w, h in faces:
            cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 3)
            text_size, _ = cv2.getTextSize('Face', cv2.FONT_HERSHEY_SIMPLEX, 1, 2)
            cv2.rectangle(frame, (x, y - text_size[1]), (x + text_size[0], y), (255, 255, 255), cv2.FILLED)
            cv2.putText(frame, 'Face', (x, y), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 0), 2)
        cv2.imshow(win_title, frame)

        if cv2.waitKey(1) == 27:
            break

    cap.release()
    cv2.destroyAllWindows()

haar_face_cascade = cv2.CascadeClassifier('../data/haarcascade_frontalface_default.xml')
detect_faces('../data/faces.mp4', haar_face_cascade, 'Haar cascade face detector')

lbp_face_cascade = cv2.CascadeClassifier()
lbp_face_cascade.load('../data/lbpcascade_frontalface.xml')

detect_faces(0, lbp_face_cascade, 'LBP cascade face detector')
```

Aruco pattern detector

```python
import cv2
import cv2.aruco as aruco
import numpy as np

aruco_dict = aruco.getPredefinedDictionary(aruco.DICT_6X6_250)

img = np.full((700, 700), 255, np.uint8)

img[100:300, 100:300] = aruco.drawMarker(aruco_dict, 2, 200)
img[100:300, 400:600] = aruco.drawMarker(aruco_dict, 76, 200)
img[400:600, 100:300] = aruco.drawMarker(aruco_dict, 42, 200)
img[400:600, 400:600] = aruco.drawMarker(aruco_dict, 123, 200)

img = cv2.GaussianBlur(img, (11, 11), 0)

cv2.imshow('Created AruCo markers', img)
cv2.waitKey(0)
cv2.destroyAllWindows()

aruco_dict = aruco.getPredefinedDictionary(aruco.DICT_6X6_250)

corners, ids, _ = aruco.detectMarkers(img, aruco_dict)

img_color = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
aruco.drawDetectedMarkers(img_color, corners, ids)

cv2.imshow('Detected AruCo markers', img_color)
cv2.waitKey(0)
cv2.destroyAllWindows()
```