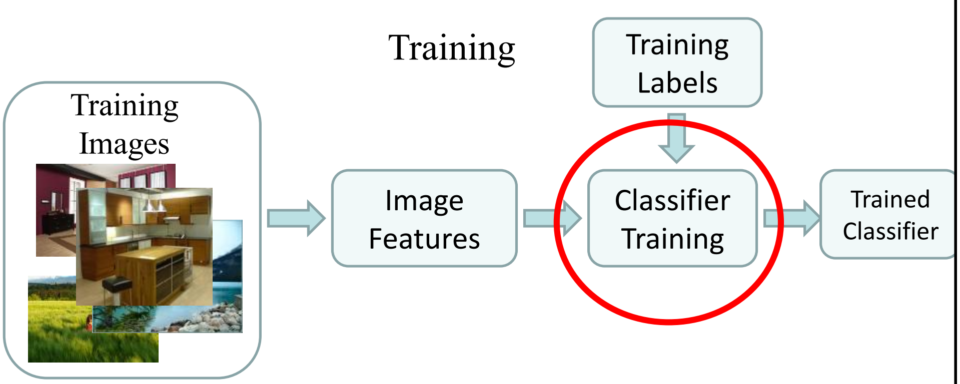
ECS797P - Machine Learning for Visual Data Analysis

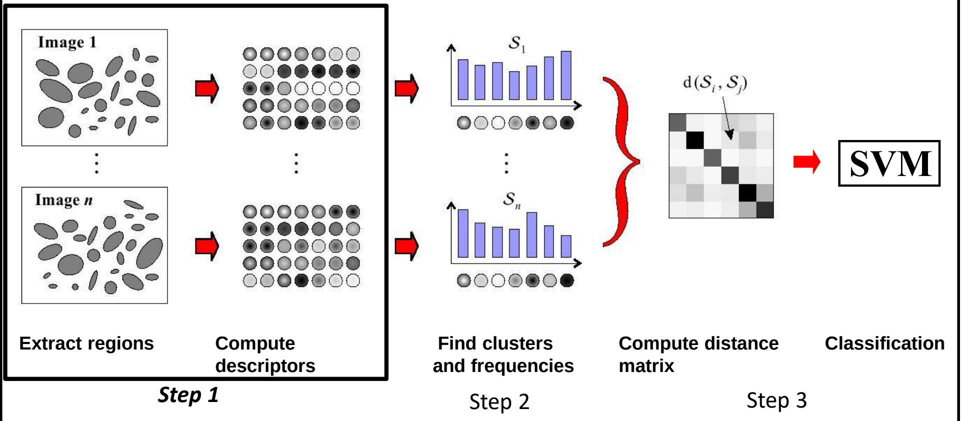
Tao Xiang
t.xiang@qmul.ac.uk

Lecture 2

Part 2: Classifiers



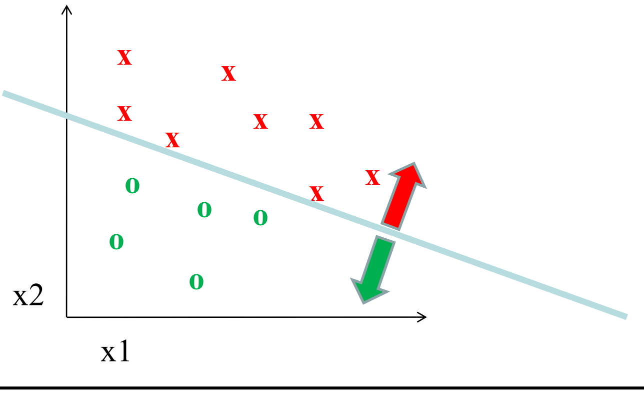
Bag-of-features for image classification

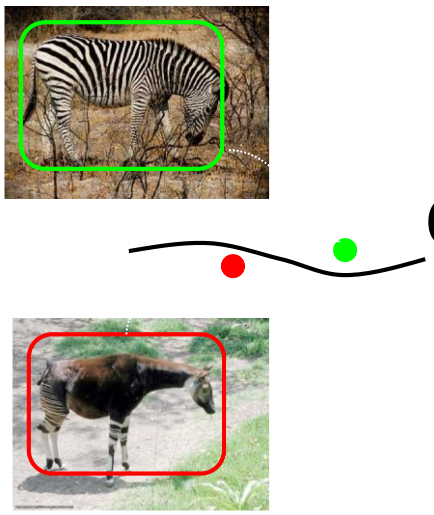


[Nowak,Jurie&Triggs,ECCV'06], [Zhang,Marszalek,Lazebnik&Schmid,IJCV'07]

Learning a classifier

Given some set of features with corresponding labels, learn a function to predict the labels from the features





Classifier-based methods

Many classifiers to choose from

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

Which is the best one?

The perfect classification algorithm

- Objective function: encodes the right loss for the problem
- Parameterization: makes assumptions that fit the problem
- Regularization: right level of regularization for amount of training data
- Training algorithm: can find parameters that maximize objective on training set
- Inference algorithm: can solve for objective function in evaluation

Generative vs. Discriminative Classifiers

Generative

- Training
  - Models the data and the labels
  - Assume (or learn) probability distribution and dependency structure
  - Can impose priors
- Testing
  - P(y=1, x) / P(y=0, x) > t?
- Examples
  - Foreground/background GMM
  - Naïve Bayes classifier
  - Bayesian network

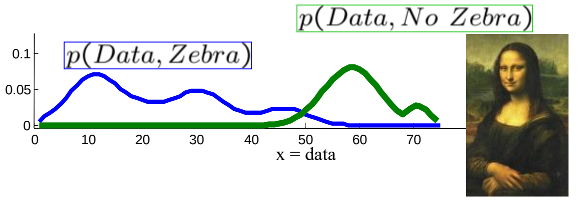
Discriminative

- Training
  - Learn to directly predict the labels from the data
  - Assume form of boundary
  - Margin maximization or parameter regularization
- Testing
  - f(x) > t ; e.g., $w^T x > t$
- Examples
  - Logistic regression
  - SVM
  - Boosted decision trees

Discriminative vs. generative

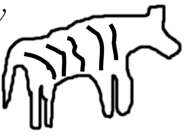
- Generative model

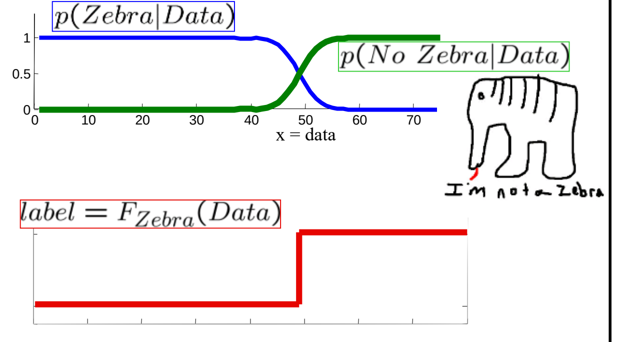
(*The artist*)



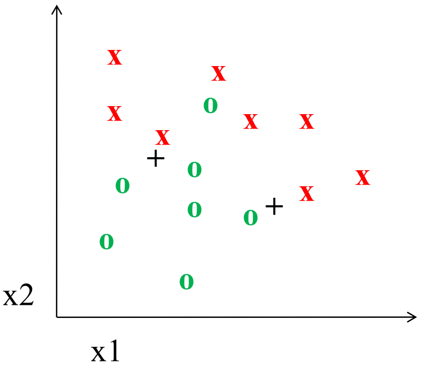
- Discriminative model

(The lousy painter)

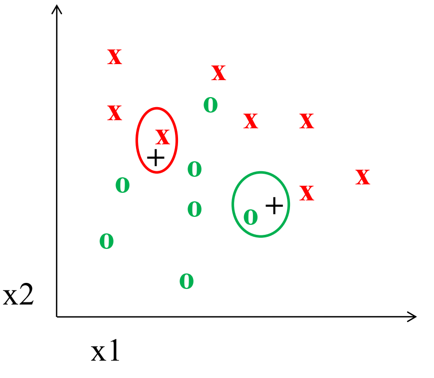




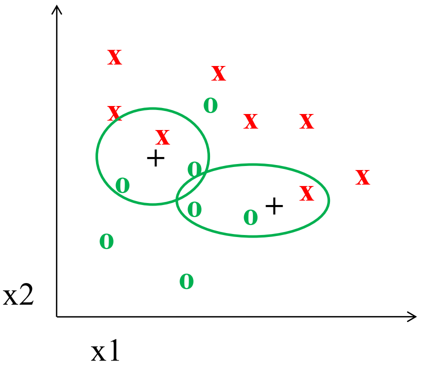
K-nearest neighbor



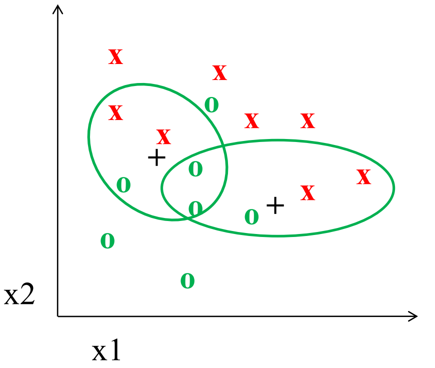
1-nearest neighbor



3-nearest neighbor



5-nearest neighbor



What is the parameterization? The regularization? The training algorithm? The inference?

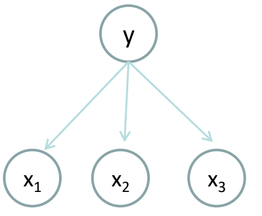
Is K-NN generative or discriminative?

Using K-NN

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

Naïve Bayes

- Objective
- Parameterization
- Regularization
- Training
- Inference



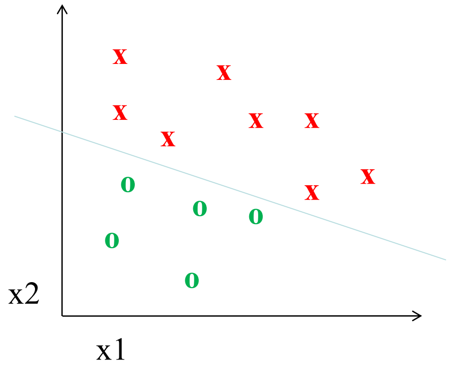
Naïve Bayes Classifier wikipedia link

Using Naïve Bayes

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

Classifiers: Logistic Regression

- Objective
- Parameterization
- Regularization
- Training
- Inference



[Logistic Regression wikipedia link]

[A document on how to use Logistic Regression as a classifier]

Using Logistic Regression

- Quick, simple classifier (try it first)
- Use L2 or L1 regularization
  - L1 does feature selection and is robust to irrelevant features but slower to train

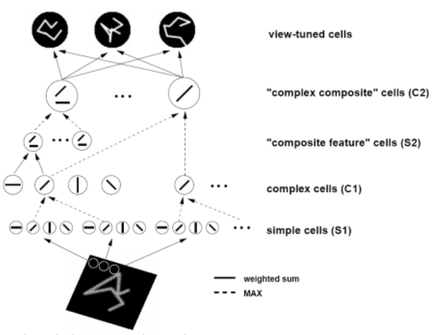
Classifier: Neural Networks

Fukushima's Neocognitron, 1980

Rowley, Baluja, Kanade 1998

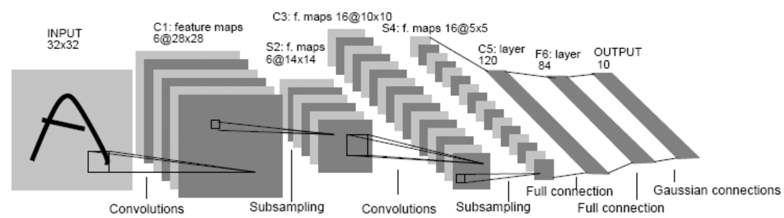
LeCun, Bottou, Bengio, Haffner 1998

Serre et al. 2005

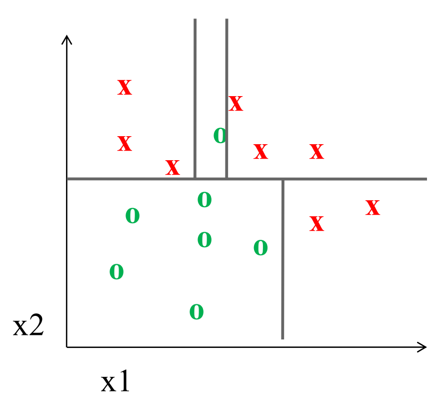


Riesenhuber, M. and Poggio, T. 1999

LeNet convolutional architecture (LeCun 1998)



Classifiers: Decision Trees



[Decision Tree wikipedia link]

Ensemble Methods: Boosting

Discrete AdaBoost(Freund & Schapire 1996b)

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

figure from Friedman et al. 2000

Classifier: Boosting

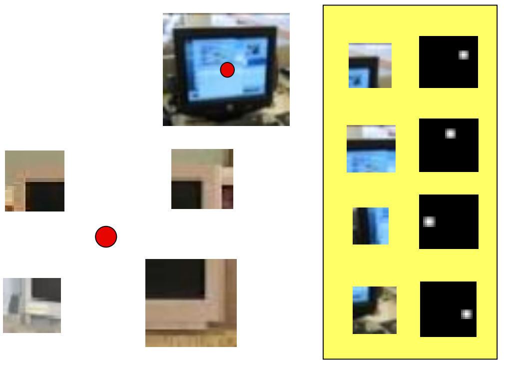
Viola & Jones 2001

- Haar features via Integral Image
- Cascade
- Real-time performance

…….

Torralba et al., 2004

- Part-based Boosting
- Each weak classifier is a part
- Part location modeled by offset mask



Two ways to think about classifiers

1. What is the objective? What are the parameters? How are the parameters learned? How is the learning regularized? How is inference performed?

2. How is the data modeled? How is similarity defined? What is the shape of the boundary?

Comparison

assuming x in {0 1}

	Learning Objective	Training	Inference
Naïve Bayes	$\text{maximize} \sum_i \left[\begin{array}{l} \sum_j \log P(x_{ij} \mid y_i; \theta_j) \\ + \log P(y_i; \theta_0) \end{array} \right]$	$\theta_{kj} = \frac{\sum_i \delta(x_{ij} = 1 \wedge y_i = k) + r}{\sum_i \delta(y_i = k) + Kr}$	$\boldsymbol{\theta}_1^T \mathbf{x} + \boldsymbol{\theta}_0^T (1 - \mathbf{x}) > 0$ where $\theta_{1j} = \log \frac{P(x_j = 1 \mid y = 1)}{P(x_j = 1 \mid y = 0)}$, $\theta_{0j} = \log \frac{P(x_j = 0 \mid y = 1)}{P(x_j = 0 \mid y = 0)}$
Logistic Regression	$\text{maximize} \sum_i \log(P(y_i \mid \mathbf{x}, \boldsymbol{\theta})) + \lambda \lVert \boldsymbol{\theta} \rVert$ where $P(y_i \mid \mathbf{x}, \boldsymbol{\theta}) = 1/(1 + \exp(-y_i \boldsymbol{\theta}^T \mathbf{x}))$	Gradient ascent	$\boldsymbol{\theta}^T \mathbf{x} > 0$
Linear SVM	minimize $\lambda \sum_i \xi_i + \frac{1}{2} \lVert \boldsymbol{\theta} \rVert$ such that $y_i \boldsymbol{\theta}^T \mathbf{x} \geq 1 - \xi_i \quad \forall i$	Linear programming	$\boldsymbol{\theta}^T \mathbf{x} > 0$
Kernelized SVM	complicated to write	Quadratic programming	$\sum_i y_i \alpha_i K(\hat{\mathbf{x}}_i, \mathbf{x}) > 0$
Nearest Neighbor	most similar features → same label	Record data	y_i where $i = \underset{i}{\text{argmin}}\ K(\hat{\mathbf{x}}_i, \mathbf{x})$

What to remember about classifiers

- No free lunch: machine learning algorithms are tools, not dogmas
- Try simple classifiers first
- Better to have smart features and simple classifiers than simple features and smart classifiers
- Use increasingly powerful classifiers with more training data (bias-variance tradeoff)

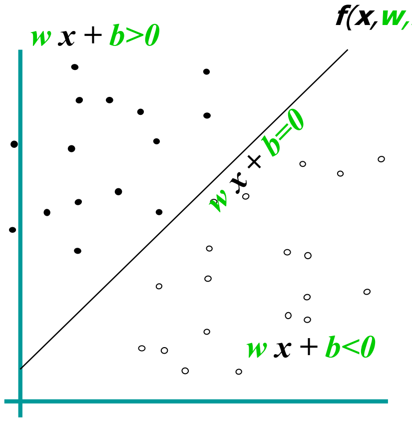
SUPPORT VECTOR MACHINES

Linear Classifiers

$\boldsymbol{x} \longrightarrow \boxed{f} \longrightarrow y^{est}$

$f(\boldsymbol{x},\boldsymbol{w},b) = sign(\boldsymbol{w}\ \boldsymbol{x} + b)$

- • denotes +1
- ° denotes -1



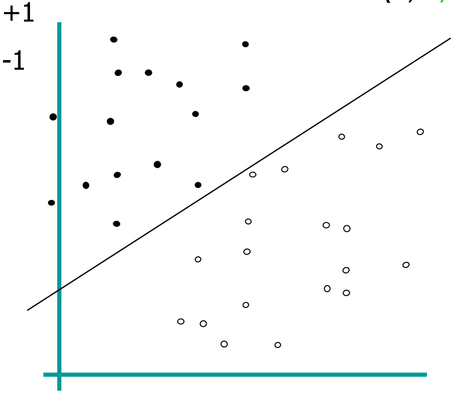
How would you classify this data?

Linear Classifiers

$\boldsymbol{x} \longrightarrow \boxed{f} \longrightarrow y^{est}$

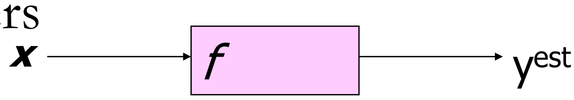
$f(\boldsymbol{x},\boldsymbol{w},b) = sign(\boldsymbol{w}\ \boldsymbol{x} + b)$

- • denotes +1
- ° denotes -1

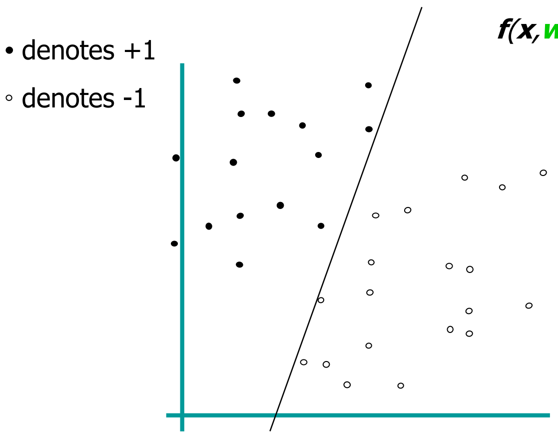


How would you classify this data?

Linear Classifiers

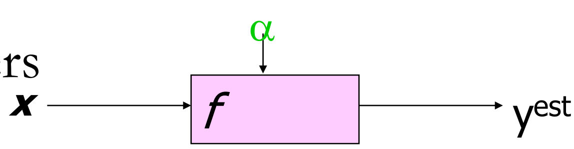


$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}\ \mathbf{x} + b)$

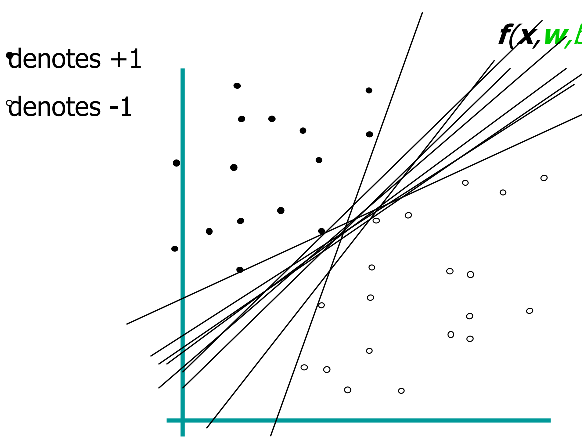


How would you classify this data?

Linear Classifiers



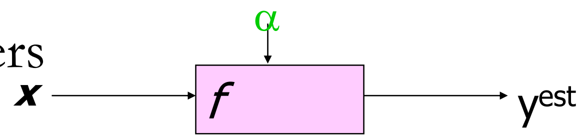
$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}\ \mathbf{x} + b)$



Any of these would be fine..

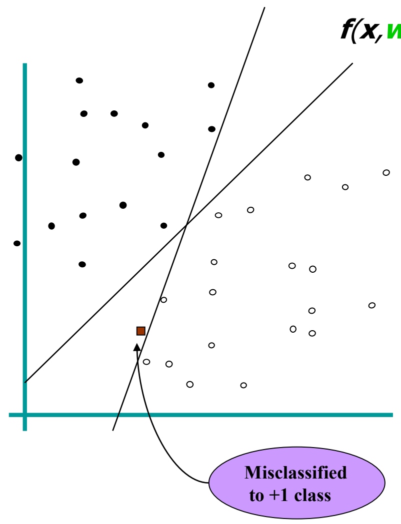
..but which is best?

Linear Classifiers



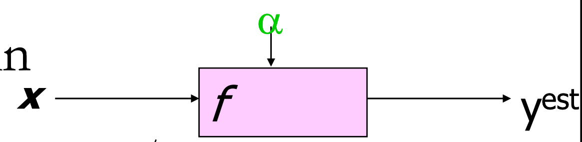
f(***x***,***w***,*b*) = *sign*(***w*** ***x*** + *b*)

- • denotes +1
- ◦ denotes -1



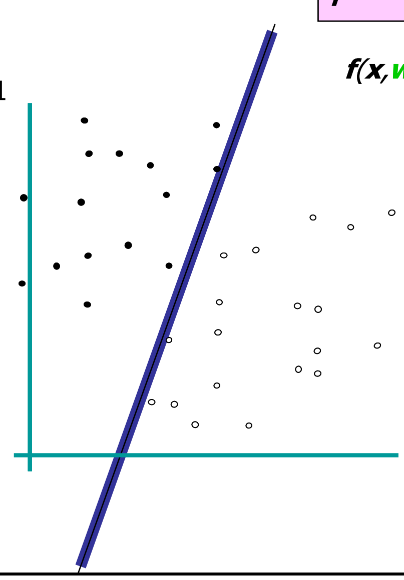
How would you classify this data?

Classifier Margin



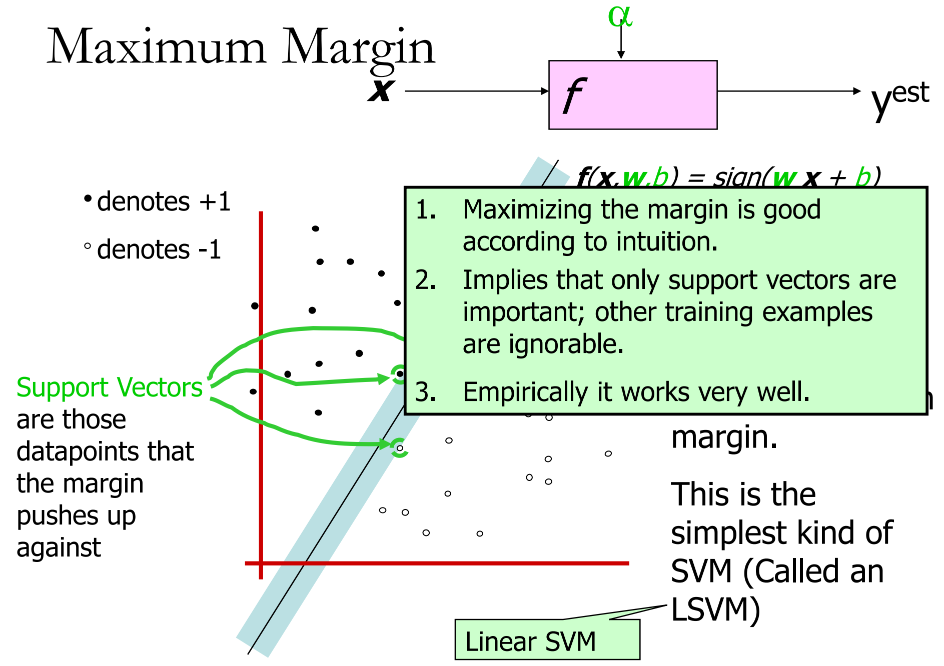
f(***x***,***w***,*b*) = *sign*(***w*** ***x*** + *b*)

- • denotes +1
- ◦ denotes -1



Define the margin of a linear classifier as the width that the boundary could be increased by before hitting a datapoint.

Maximum Margin

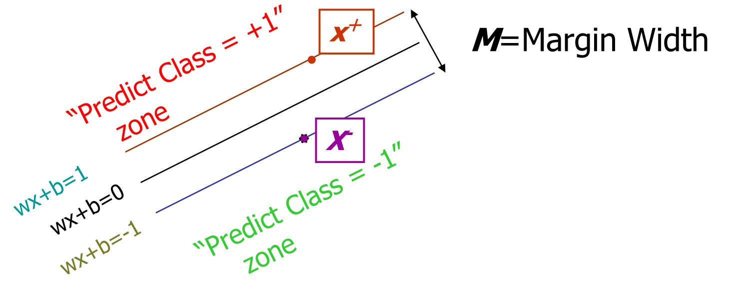


1. Maximizing the margin is good according to intuition.
2. Implies that only support vectors are important; other training examples are ignorable.
3. Empirically it works very well.

margin.

This is the simplest kind of SVM (Called an LSVM)

Linear SVM Mathematically



What we know:

- $\boldsymbol{w} \cdot \boldsymbol{x}^+ + b = +1$
- $\boldsymbol{w} \cdot \boldsymbol{x}^- + b = -1$
- $\boldsymbol{w} \cdot (\boldsymbol{x}^+ - \boldsymbol{x}^-) = 2$

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

Linear SVM Mathematically

- Goal: **1) Correctly classify all training data**

$$wx_i + b \geq 1 \;\; \text{if } y_i = +1$$
$$wx_i + b \leq -1 \;\; \text{if } y_i = -1$$
$$y_i(wx_i + b) \geq 1 \;\; \text{for all i}$$

2) Maximize the Margin $M = \frac{2}{|w|}$

same as minimize $\frac{1}{2} w^t w$

- **We can formulate a Quadratic Optimization Problem and solve for w and b**

- Minimize $\Phi(w) = \frac{1}{2} w^t w$

subject to $y_i(wx_i + b) \geq 1 \qquad \forall i$

Solving the Optimization Problem

Find **w** and b such that
$\mathbf{\Phi(w)} = ½\, \mathbf{w}^T\mathbf{w}$ is minimized;
and for all $\{(\mathbf{x_i}, y_i)\}$: $y_i(\mathbf{w}^T\mathbf{x_i} + b) \geq 1$

- **Need to optimize a *quadratic* function subject to *linear* constraints.**
- **Quadratic optimization problems are a well-known class of mathematical programming problems, and many (rather intricate) algorithms exist for solving them.**
- **The solution involves constructing a *dual problem* where a *Lagrange multiplier* α_i is associated with every constraint in the primary problem:**

Find $\alpha_1 \ldots \alpha_N$ such that
$\mathbf{Q}(\boldsymbol{\alpha}) = \Sigma\alpha_i - ½\Sigma\Sigma\alpha_i\alpha_j y_i y_j \mathbf{x_i}^T\mathbf{x_j}$ is maximized and
(1) $\Sigma\alpha_i y_i = 0$
(2) $\alpha_i \geq 0$ for all α_i

The Optimization Problem Solution

- The solution has the form:

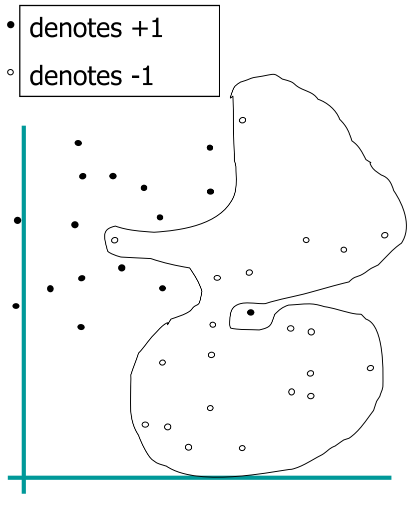
$$\boxed{\mathbf{w} = \Sigma \alpha_i y_i \mathbf{x_i} \qquad b = y_k - \mathbf{w}^\mathbf{T}\mathbf{x_k} \text{ for any } \mathbf{x_k} \text{ such that } \alpha_k \neq 0}$$

- Each non-zero α_i indicates that corresponding $\mathbf{x_i}$ is a support vector.
- Then the classifying function will have the form:

$$\boxed{f(\mathbf{x}) = \Sigma \alpha_i y_i \mathbf{x_i}^\mathbf{T}\mathbf{x} + b}$$

- Notice that it relies on an *inner product* between the test point $\mathbf{x}$ and the support vectors $\mathbf{x_i}$ – we will return to this later.
- Also keep in mind that solving the optimization problem involved computing the inner products $\mathbf{x_i}^\mathbf{T}\mathbf{x_j}$ between all pairs of training points.

Dataset with noise



- **Hard Margin: So far we require all data points be classified correctly**

  - No training error

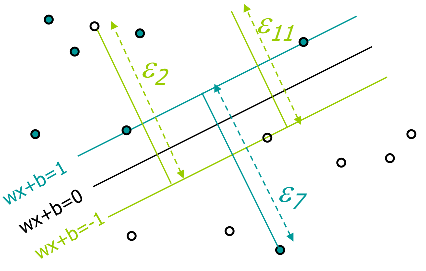
- **What if the training set is noisy?**

  - Solution 1: use very powerful kernels

Overfitting?

Soft Margin Classification

***Slack variables ξi* can be added to allow misclassification of difficult or noisy examples.**



What should our quadratic optimization criterion be?

Minimize

$$\frac{1}{2}\mathbf{w}.\mathbf{w} + C\sum_{k=1}^{R}\varepsilon_k$$

Hard Margin v.s. Soft Margin

- **The old formulation:**

> Find **w** and *b* such that
> $\mathbf{\Phi}(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w}$ is minimized and for all $\{(\mathbf{x_i}, y_i)\}$
> $y_i(\mathbf{w}^T\mathbf{x_i} + \mathrm{b}) \geq 1$

- **The new formulation incorporating slack variables:**

> Find **w** and *b* such that
> $\mathbf{\Phi}(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\Sigma\xi_i$ is minimized and for all $\{(\mathbf{x_i}, y_i)\}$
> $y_i(\mathbf{w}^T\mathbf{x_i} + b) \geq 1 - \xi_i$ and $\xi_i \geq 0$ for all i

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

Linear SVMs: Overview

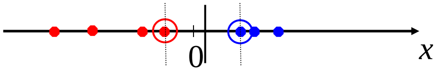
- **The classifier is a *separating hyperplane.***
- **Most "important" training points are support vectors; they define the hyperplane.**
- **Quadratic optimization algorithms can identify which training points $\mathbf{x}_i$ are support vectors with non-zero Lagrangian multipliers α_i.**
- **Both in the dual formulation of the problem and in the solution training points appear only inside dot products:**

Find $\alpha_1 \ldots \alpha_N$ such that
$Q(\alpha) = \Sigma\alpha_i - \frac{1}{2}\Sigma\Sigma\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j$ is maximized and
(1) $\Sigma\alpha_i y_i = 0$
(2) $0 \leq \alpha_i \leq C$ for all α_i

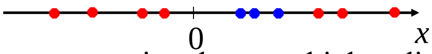
$f(\mathbf{x}) = \Sigma\alpha_i y_i \mathbf{x}_i^T\mathbf{x} + b$

Non-linear SVMs

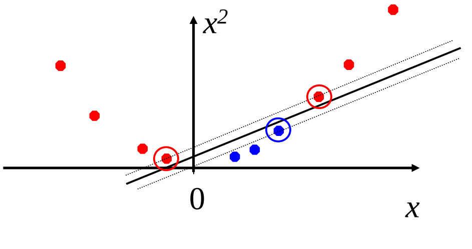
- Datasets that are linearly separable with some noise work out great:



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

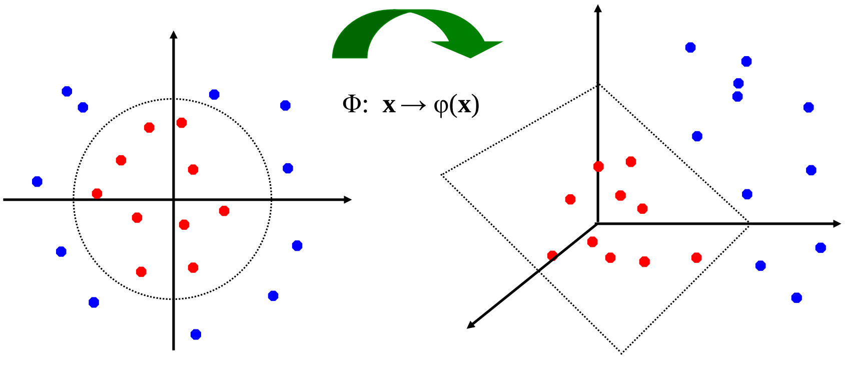


- How about… mapping data to a higher-dimensional space:



Non-linear SVMs: Feature spaces

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



The "Kernel Trick"

- The linear classifier relies on dot product between vectors $K(\mathbf{x}_i,\mathbf{x}_j)=\mathbf{x}_i^T\mathbf{x}_j$
- If every data point is mapped into high-dimensional space via some transformation

Φ: $\mathbf{x} \rightarrow \varphi(\mathbf{x})$, the dot product becomes:

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

- A *kernel function* is some function that corresponds to an inner product in some expanded feature space.

What Functions are Kernels?

- **For some functions $K(x_i,x_j)$ checking that**

  $K(x_i,x_j)= \varphi(x_i)^T\varphi(x_j)$ can be cumbersome.
- **Mercer's theorem:**

Every semi-positive definite symmetric function is a kernel

- **Semi-positive definite symmetric functions correspond to a semi-positive definite symmetric Gram matrix:**

K=

$K(\mathbf{x}_1,\mathbf{x}_1)$	$K(\mathbf{x}_1,\mathbf{x}_2)$	$K(\mathbf{x}_1,\mathbf{x}_3)$	…	$K(\mathbf{x}_1,\mathbf{x}_N)$
$K(\mathbf{x}_2,\mathbf{x}_1)$	$K(\mathbf{x}_2,\mathbf{x}_2)$	$K(\mathbf{x}_2,\mathbf{x}_3)$		$K(\mathbf{x}_2,\mathbf{x}_N)$
…	…	…	…	…
$K(\mathbf{x}_N,\mathbf{x}_1)$	$K(\mathbf{x}_N,\mathbf{x}_2)$	$K(\mathbf{x}_N,\mathbf{x}_3)$	…	$K(\mathbf{x}_N,\mathbf{x}_N)$

Examples of Kernel Functions

- Linear: $K(\mathbf{x}_i,\mathbf{x}_j)= \mathbf{x}_i^{\mathbf{T}}\mathbf{x}_j$

- Polynomial of power p: $K(\mathbf{x}_i,\mathbf{x}_j)= (1+ \mathbf{x}_i^{\mathbf{T}}\mathbf{x}_j)^p$

- Gaussian (radial-basis function network):

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

- Sigmoid: $K(\mathbf{x}_i,\mathbf{x}_j)= \tanh(\beta_0\mathbf{x}_i^{\mathbf{T}}\mathbf{x}_j + \beta_1)$

Non-linear SVMs Mathematically

- **Dual problem formulation:**

Find $\alpha_1 \ldots \alpha_N$ such that
$Q(\alpha) = \Sigma\alpha_i - \frac{1}{2}\Sigma\Sigma\alpha_i\alpha_j y_i y_j K(x_i, x_j)$ is maximized and
(1) $\Sigma\alpha_i y_i = 0$
(2) $\alpha_i \geq 0$ for all α_i

- **The solution is:**

$$f(x) = \Sigma\alpha_i y_i K(x_i, x) + b$$

- **Optimization techniques for finding α_i's remain the same!**

Nonlinear SVM - Overview

- SVM locates a separating hyperplane in the feature space and classify points in that space
- It does not need to represent the space explicitly, simply by defining a kernel function
- The kernel function plays the role of the dot product in the feature space.

Properties of SVM

- **Flexibility in choosing a similarity function**
- **Sparseness of solution when dealing with large data sets**
  - only support vectors are used to specify the separating hyperplane
- **Ability to handle large feature spaces**
  - complexity does not depend on the dimensionality of the feature space
- **Overfitting can be controlled by soft margin approach**
- **Nice math property: a simple convex optimization problem which is guaranteed to converge to a single global solution**

Some Issues

- **Choice of kernel**
  - Gaussian or polynomial kernel is default
  - if ineffective, more elaborate kernels are needed
  - domain experts can give assistance in formulating appropriate similarity measures

- **Choice of kernel parameters**
  - e.g. **σ** in Gaussian kernel
  - **σ** is the distance between closest points with different classifications
  - In the absence of reliable criteria, applications rely on the use of a validation set or cross-validation to set such parameters.

References

- **An excellent tutorial on VC-dimension and Support Vector Machines:**
  C.J.C. Burges. A tutorial on support vector machines for pattern recognition. Data Mining and Knowledge Discovery, 2(2):955-974, 1998.

- **The VC/SRM/SVM Bible:**
  Statistical Learning Theory by Vladimir Vapnik, Wiley-Interscience; 1998

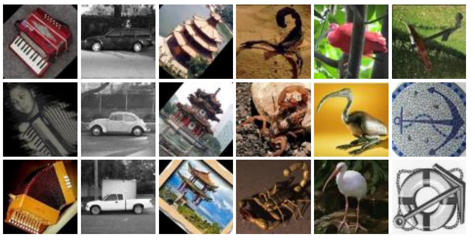
- **The most widely used code library:**
  http://www.csie.ntu.edu.tw/~cjlin/libsvm/

- A good link:
  http://www.kernel-machines.org/

Summary: SVMs for image classification

1. Pick an image representation (in our case, bag of features)
2. Pick a kernel function for that representation
3. Compute the matrix of kernel values between every pair of training examples
4. Feed the kernel matrix into your favorite SVM solver to obtain support vectors and weights
5. At test time: compute kernel values for your test example and each support vector, and combine them with the learned weights to get the value of the decision function

Category classification – CalTech101

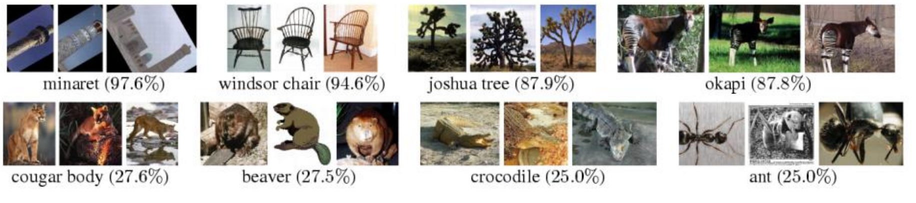


L	Single-level	Pyramid
0(1x1)	41.2±1.2	
1(2x2)	55.9±0.9	57.0 ±0.8
2(4x4)	63.6±0.9	*64.6 ±0.8*
3(8x8)	60.3±0.9	64.6 ±0.7

Bag-of-features approach by Zhang et al.'07: 54 %

CalTech101

Easiest and hardest classes



- Sources of difficulty:
  - Lack of texture
  - Camouflage
  - Thin, articulated limbs
  - Highly deformable shape

Evaluation of image classification

- Averaged classification accuracy
- Confusion matrix

		Predicted class		
		Cat	Dog	Rabbit
Actual class	Cat	5	3	0
	Dog	2	3	1
	Rabbit	0	2	11