MLRF Lecture 05

J. Chazalon, LRDE/EPITA, 2019

Some classifiers – part 1

Lecture 05 part 03

Disclaimer

What follows is a very limited selection.

Only classifiers suitable for image classification as we present it today.

> ***input = feature vector***
> ***output = label***

Many other approaches, in particular for structured and/or symbolical data (graphs, etc.)

What is our goal?

*Given **samples** (described by features) and **true labels**,*
*find a **good** function*
*which will correctly **predict labels***
*given **new data samples***

Problems:

- Which family for our function?
- What is “good”?
- How to train / find such function?

Let us have a look a some classical approaches.

Parametric vs Non Parametric classifiers

Parametric vs non parametric

Often heard.
Very misleading.

***Parametric* examples:** Logistic Regression, Linear Discriminant Analysis, Naive Bayes, Perceptron, Simple Neural Networks…

*"A learning model that **summarizes data** with a **set of parameters of fixed size** (**independent of the number of training examples**) is called a **parametric model**. No matter how much data you throw at a parametric model, it won't change its mind about how many parameters it needs."*

— Russell & Norvig, Artificial Intelligence: A Modern Approach, page 737

Parametric vs non parametric

***Non-parametric* examples:** k-Nearest Neighbors, Decision Trees, SVMs

*"**Non-parametric** models differ from parametric models in that **the model structure is not specified a priori but is instead determined from data.** The term non-parametric is **not meant to imply that such models completely lack parameters** but that **the number and nature of the parameters are flexible** and not fixed in advance."*

— https://en.wikipedia.org/wiki/Nonparametric_statistics

"Nonparametric methods are good when you have a lot of data and no prior knowledge, and when you don't want to worry too much about choosing just the right features."

— Russell & Norvig, Artificial Intelligence: A Modern Approach, page 757

Dummy classifiers

Dummy classifiers and how good they can pretend to be

Say you have a **dataset** with **9** muffins, and **1** chihuahua.

You have a new sample to classify.

Which class should you bet on?

Dummy classifiers and how good they can pretend to be

If your class prior probabilities $P(C_1)$, $P(C_2)$, … are not equal,
then you should bet on the **most frequent class**! ($g(x) = \text{argmax}_y\ p(y)$)

Without such information, you can just pick at random.

What is the expected accuracy (true predictions / total predictions)
if you have N classes and pick one at random?

N=100 *N=10* *N=2*

Dummy classifiers and how good they can pretend to be

Scikit-learn offers a DummyClassifier class which helps testing such strategy.

What's the point?

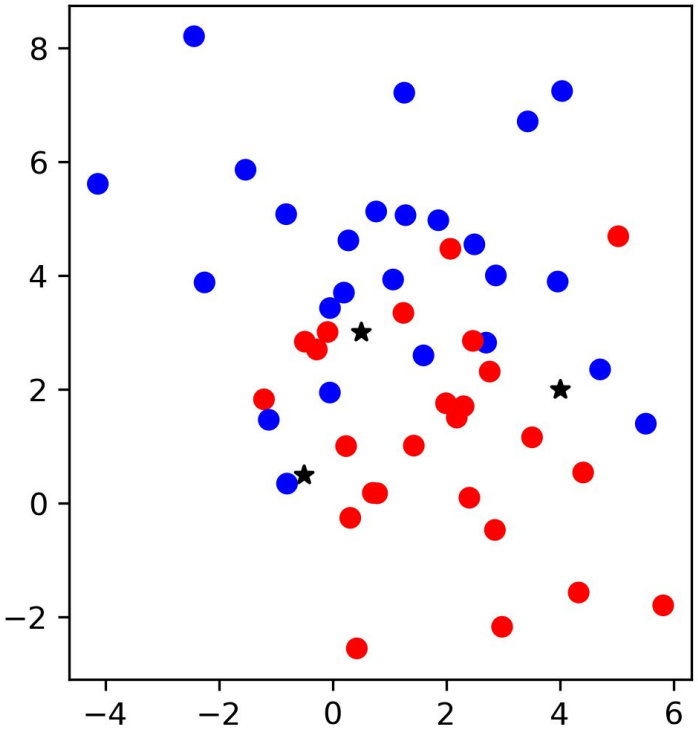
1. Quickly build and test your complete pipeline with a **mockup** classifier
2. Quickly get a **baseline** for the performance
3. (look for obvious bias in the dataset, but you should have cleaned it before!)

K Nearest Neighbors (kNN)

K Nearest Neighbors (kNN)

Keep all training samples

View new samples as queries over the previously learned / indexed samples



from A. Müller

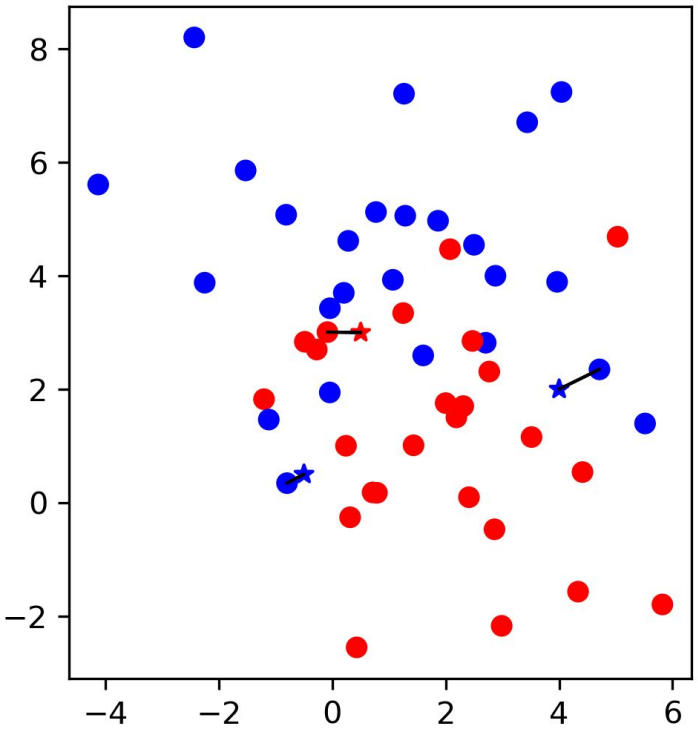
K Nearest Neighbors (kNN)

Keep all training samples

View new samples as queries over the previously learned / indexed samples

Assign the class of the closest(s) samples

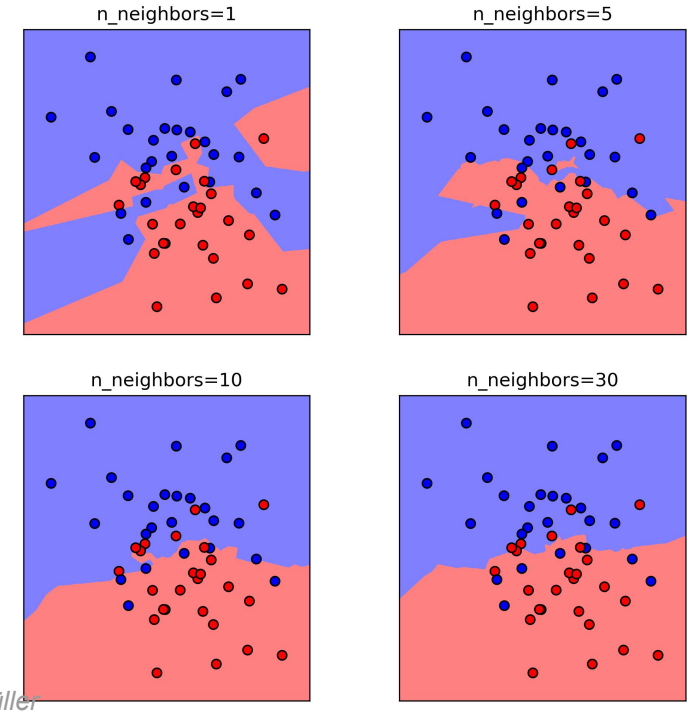
$$f(x) = y_i, i = \text{argmin}_j ||x_j - x||$$

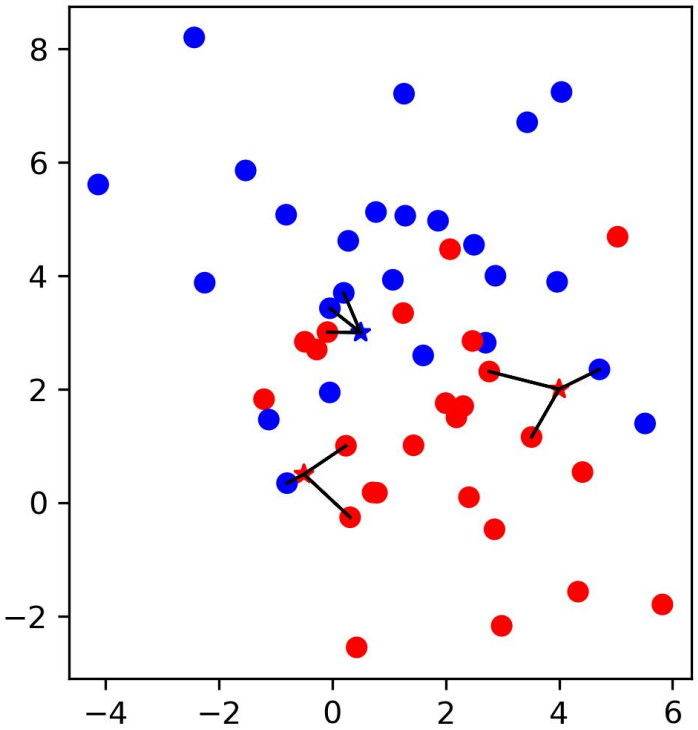


from A. Müller

K Nearest Neighbors (kNN)

We can check more than 1 sample





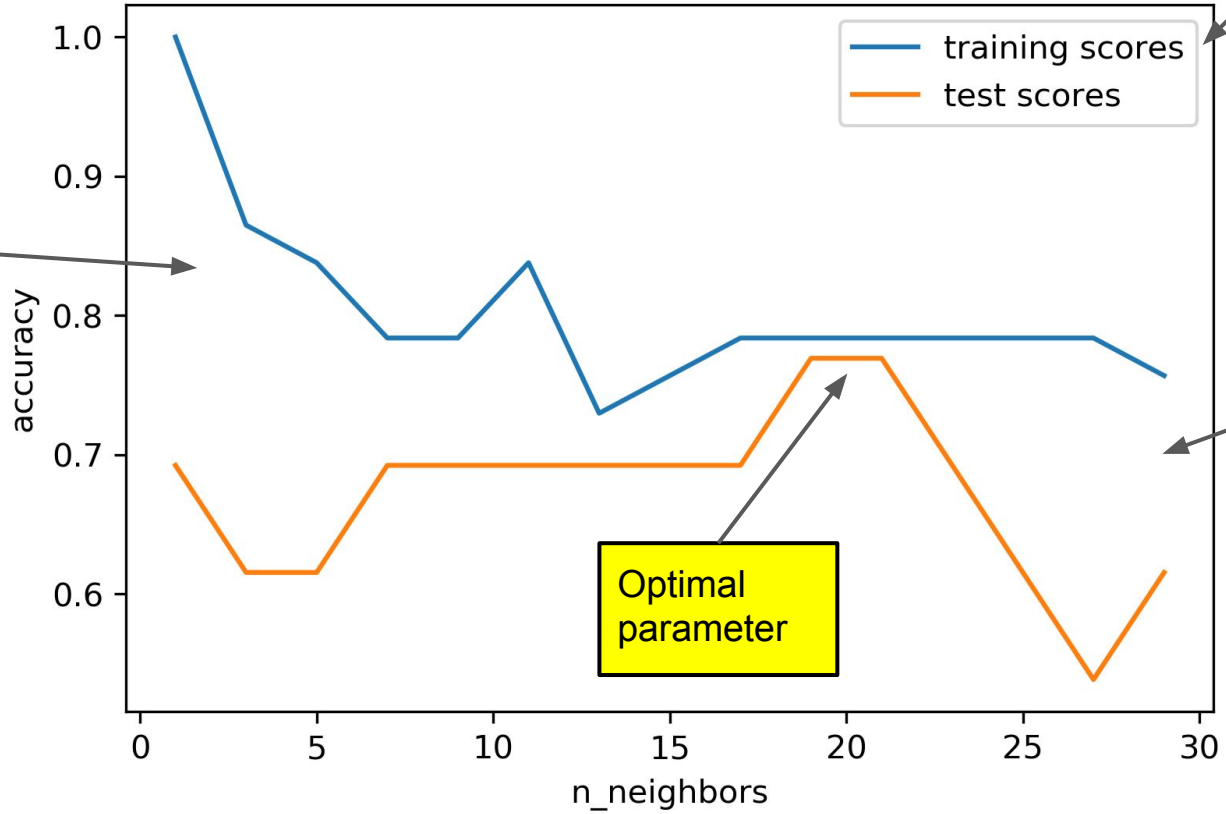
from A. Müller

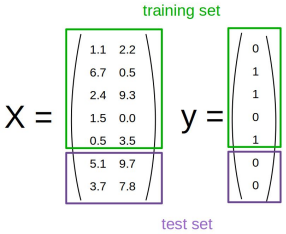
Remember this bias/variance compromise?

Score = accuracy here

High sensibility to noise / variance

Strong smoothing / high bias





from A. Müller

K Nearest Neighbors (kNN)

Pros

Very simple to implement.

Capacity easily controlled with **k.**

Can be tuned to work on large datasets: indexing, data cleaning, etc.

Good baseline.

Non parametric.

Lazy learner.

Cons

In high dimension, all samples tend to be very close (for Euclidean dimension).

Large memory consumption on large datasets.

Requires a large amount of samples and large k to get best performance.

Setting K:

$$K \simeq \sqrt{m/C}$$

m/C: average number of training sample / class

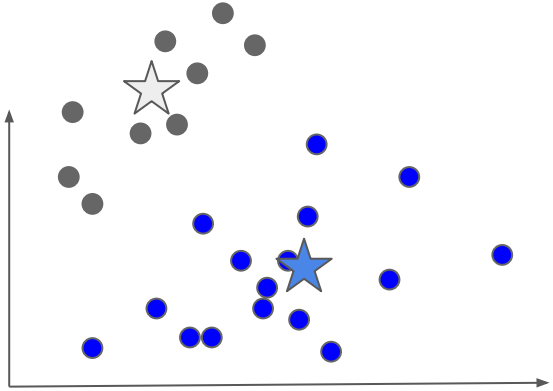
Other distance-based classifiers

Minimal euclidean distance

Very basic classifier

Distance to the mean m_i of the class

It does not take into account differences in variance for each class



$$D_i(x) = (x - m_i)^T (x - m_i)$$

Predicted class for x :

$$\mathbf{g(x) = \mathrm{argmin}_i\, D_i(x)}$$

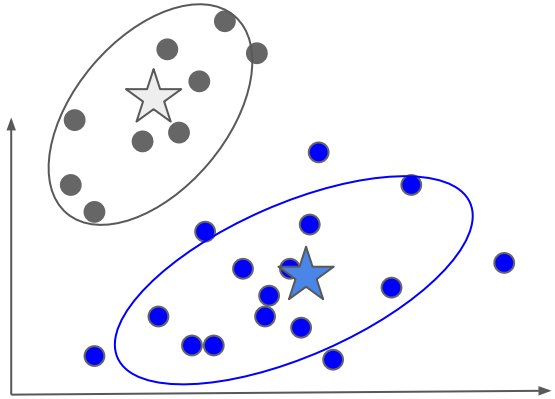
Minimal quadratic distance (Mahalanobis)

For each class i, the mean m_i and covariance matrix S_i are computer from the set of examples

The covariance matrix is taken into account when computing the distance from an image to the class i

The feature vector of the image x is projected over the eigenvectors of the class

$$g(x) = \text{argmin}_i \, D_i(x)$$



$$D_i(x) = (x - m_i)^T S_i^{-1} (x - m_i) = -z^T z$$

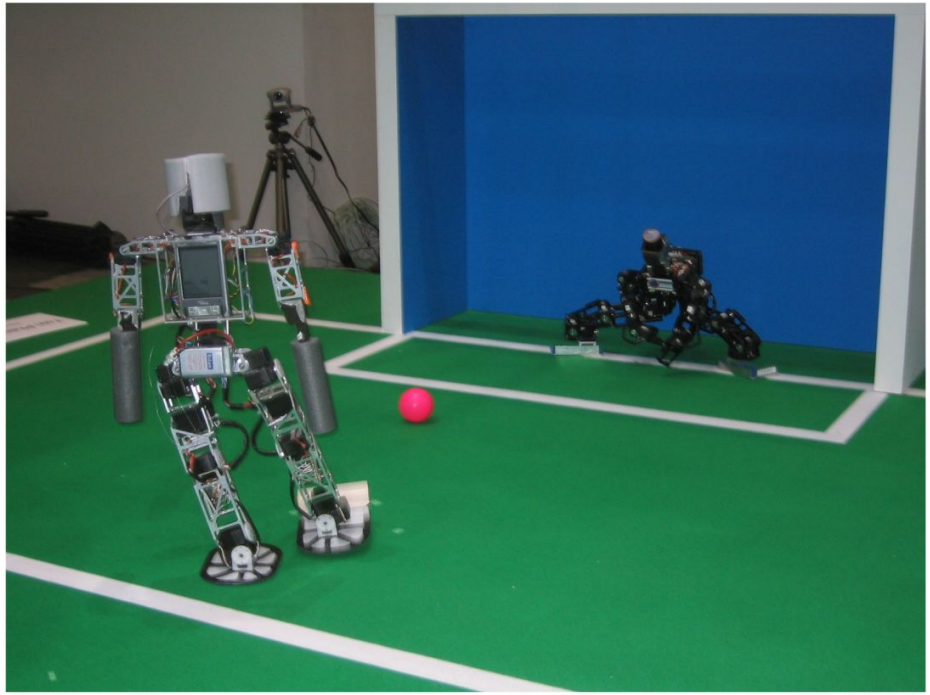
$$z = \Lambda_i^{-1/2} \Psi_i^T (x - m_i)$$

Λ_i : eigenvalues of S_i

Ψ_i : eigenvectors of S_i

A quick introduction to Bayesian Decision Theory

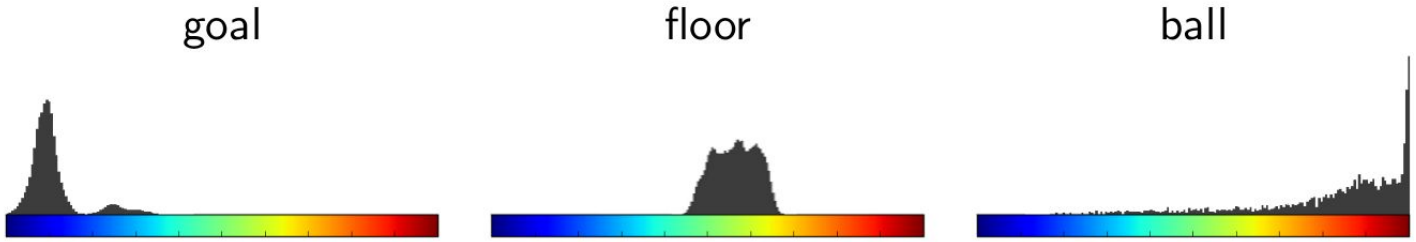
Example – RoboCup



Goal: blue **Floor: green/white** **Ball: red**

From C. Lampert VRML summer school 2013

Example – RoboCup



New object: → ball

New object: → floor

New object: → goal

New object: → floor

From C. Lampert VRML summer school 2013

Example – RoboCup

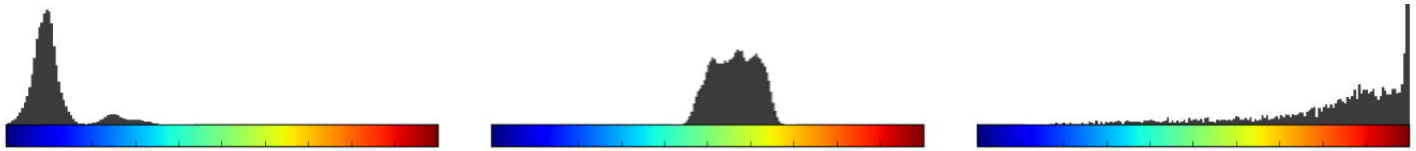
data: $x \in X = R^d$, (here: colors, d = 3)

labels: $y \in Y = \{goal, floor, ball\}$, (here: object classes)

goal: classification rule $g : X \rightarrow Y$.

Histograms: class-conditional probability densities $p(x|y)$.

For any $y \in Y$ $\forall x \in X : p(x|y) \geq 0$, $\Sigma_{x \in X}\, p(x|y) = 1$



p(x | y = goal) p(x | y = floor) p(x | y = ball)

Maximum Likelihood Rule:

$$\mathbf{g(x) = \operatorname{argmax}_{y \in Y}\, p(x|y)}$$

From C. Lampert VRML summer school 2013

General case: maximum a posteriori (MAP)

General case: need to take into consideration p(y) and p(x)

p(x|y): class conditional density (here: histograms)

p(y): class priors, e.g. for indoor RoboCup
p(floor) = 0.6, p(goal) = 0.3, p(ball) = 0.1

p(x): probability of seeing data x

Optimal decision rule (Bayes classifier): maximum a posteriori (MAP):

$$\mathbf{g(x) = \operatorname{argmax}_{y \in Y} p(y|x)}$$

How to compute p(y|x)?

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)} \qquad \text{(Bayes' rule)}$$

If classes are equiprobables and error cost is the same, then, because p(x) is constant, we get the **maximum likelihood estimation**:

$$g(x) = \underbrace{\operatorname{argmax}_{y \in Y} p(y \mid x)}_{\text{MAP}} \approx \underbrace{\operatorname{argmax}_{y \in Y} p(x \mid y)}_{\text{ML}}$$

From C. Lampert VRML summer school 2013

Generative, discriminant, and “direct” classifiers

Given: training data $\{(x_1, y_1), \ldots, (x_n, y_n)\} \subset X \times Y$

Approach 1: **Generative** Probabilistic Models

1. Use training data to obtain an **estimate $p(x|y)$ for any $y \in Y$**
2. Compute $\mathbf{p(y|x) \propto p(x|y)p(y)}$
3. Predict using $\mathbf{g(x) = \operatorname{argmax}_{y \in Y} p(y|x)}$

Can lossy reconstruct data from label.

Reject easier to implement.

Approach 2: **Discriminative** Probabilistic Models

1. Use training data to **estimate $p(y|x)$ directly.**
2. Predict using $\mathbf{g(x) = \operatorname{argmax}_{y \in Y} p(y|x)}$ (same)

Better performance in general.

Approach 3: Loss-minimizing Parameter Estimation

1. Use training data to **search for best $g : X \rightarrow Y$ directly**

Almost only **kNN**.

From C. Lampert VRML summer school 2013

Generative Probabilistic Models

Some classical Generative Probabilistic Models

Training data $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, x_n\}$. $X \times Y \subset \mathcal{X} \times \mathcal{Y}$

For each $y \in \mathcal{Y}$, build model for $p(x|y)$ of $X_y := \{x_i \in X : y_i = y\}$

Histogram: if x can have only few discrete values.

Kernel Density Estimator $p(x|y) \propto \sum_{x_i \in X_y} k(x_i, x)$

Gaussian: $p(x|y) = \mathcal{G}(x; \mu_y, \Sigma_y) \propto \exp(-\frac{1}{2}(x - \mu_y)^\top \Sigma_y^{-1}(x - \mu_y))$

Mixture of Gaussians: $p(x|y) = \sum_{k=1}^{K} \pi_y^k \mathcal{G}(x; \mu_y^k, \Sigma_y^k)$

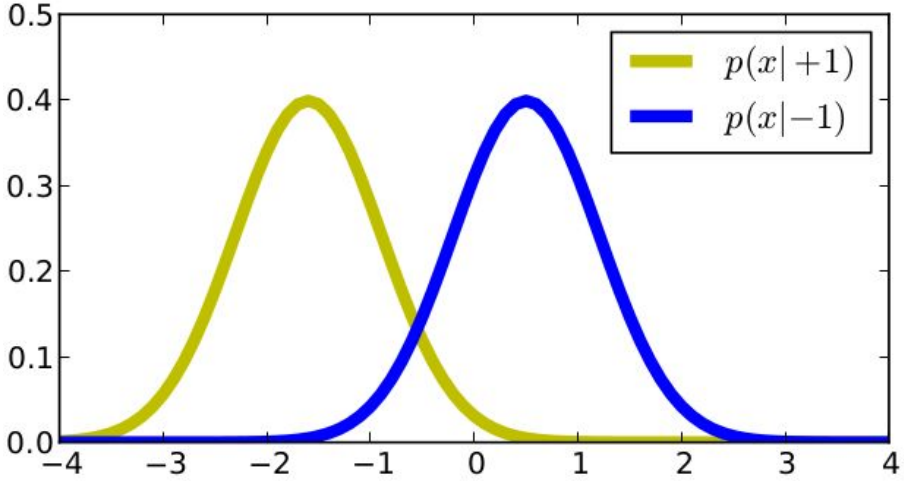
Typically, $\mathcal{Y}$ small (few possible labels), $\mathcal{X}$ low dimensional (RGB colors for ex.)

From C. Lampert VRML summer school 2013

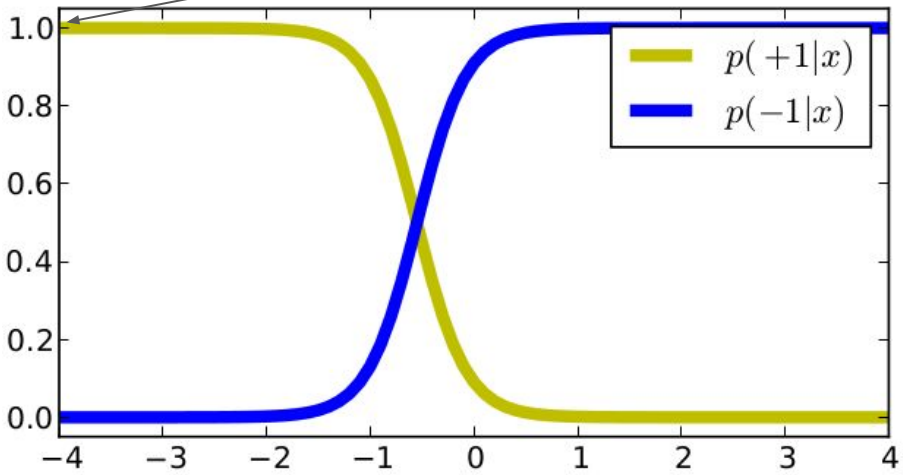
Class conditional densities and posteriors

$p(y_i|x) = p(x|y_i)\, p(y_i) / p(x)$, **$p(x) = \Sigma_i\, p(x|y_i)$**

$p(+1|x) / \Sigma_i\, p(x|yi) \rightarrow 1$



class conditional densities (Gaussian)



class posteriors for $p(+1) = p(-1) = \frac{1}{2}$

Naive Bayes Classifiers

As seen before, **$g(x) = \text{argmax}_{y \in Y}\ p(y \mid x)$**

Use Bayes formula to estimate $p(y \mid x)$.

Hard part: build an estimate of $p(x \mid y)$ — EM algo. with Gaussian mixtures, challenging with non-diagonal covariance matrices.

Solution: make strong independence assumption between variables.

If $X = (x_1, x_2, x_3, \ldots, x_n)$, then $P(y|x_1, ..., x_n) = \frac{P(x_1|y)P(x_2|y)...P(x_n|y)P(y)}{P(x_1)P(x_2)...P(x_n)}$

Or, as $P(x_i)$ are constant: $P(y|x_1, ..., x_n) \propto P(y) \prod_{i=1}^{n} P(x_i|y)$

https://nlp.stanford.edu/IR-book/pdf/13bayes.pdf
https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c

Naive Bayes Classifiers

The previous simplification leads to very simple classifiers, easy to train and fast to run, for which the decision rule is:

$$y = argmax_y P(y) \prod_{i=1}^{n} P(x_i|y)$$

Some actual Naive Bayes Classifiers:

- **Multinomial Naive Bayes**: Widely used for document (spam!) classification. $P(x_i|y)$ = frequency of the words present in the document
- **Gaussian Naive Bayes**: Assume continuous values.

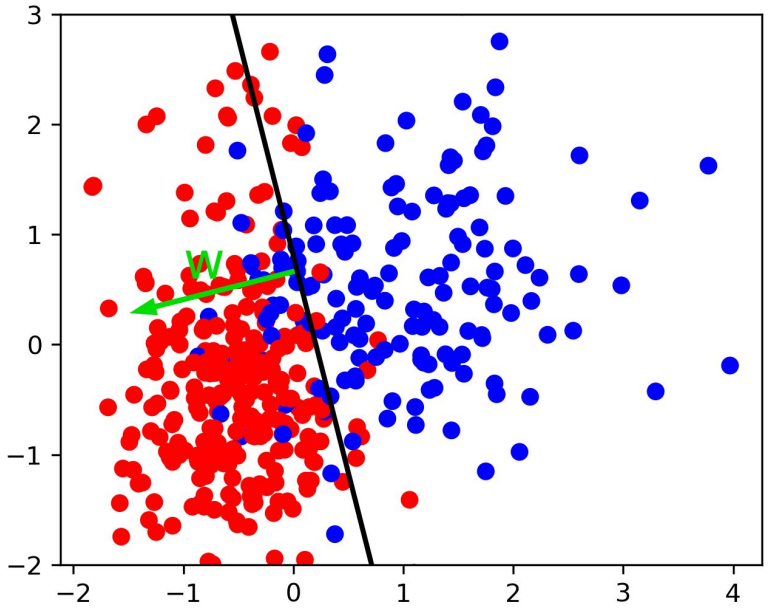
$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} exp\left(-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}\right)$$

Drawback: in real life, features ARE dependent, and this penalizes NB classifiers.

https://nlp.stanford.edu/IR-book/pdf/13bayes.pdf
https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c

Linear discriminant classifiers

General idea for binary classification



w: weights

b: bias term

$$\hat{y} = \text{sign}(\underbrace{w^T \mathbf{x}} + b) = \text{sign}\left(\sum_i w_i x_i + b\right)$$

Scalar product

Otherwise decision must cross (0,0)

Learn w and b

→ you can compute $p(y|x) \approx \hat{y}$

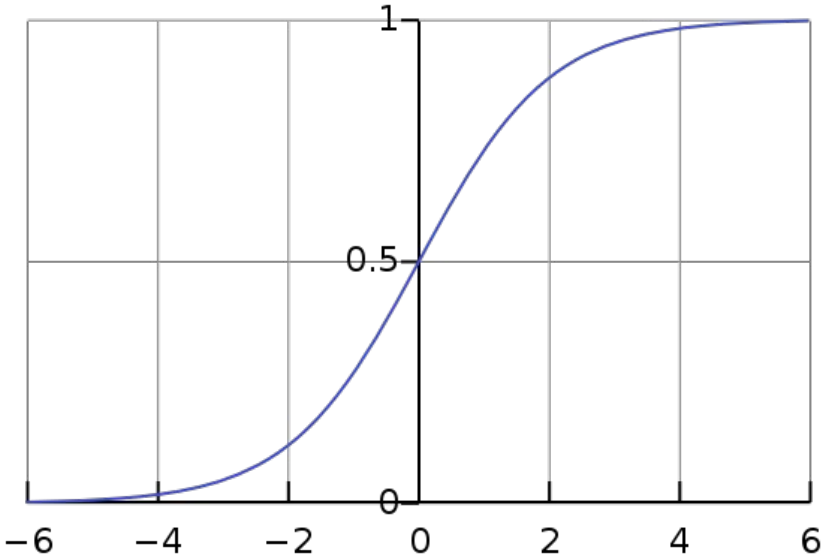
Problem: How to learn w and b?

Logistic Regression which is used for classification, not regression!

Linear classifier, f is logistic function

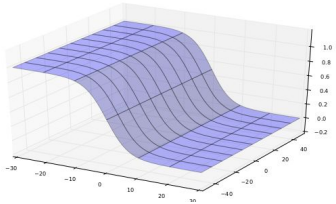
$\sigma(x) = 1/(1 + e^{-x}) = e^{x}/(1 + e^{x})$

Maps all reals → [0,1]



Optimize $\sigma(w^T \cdot x + b)$ to find best w

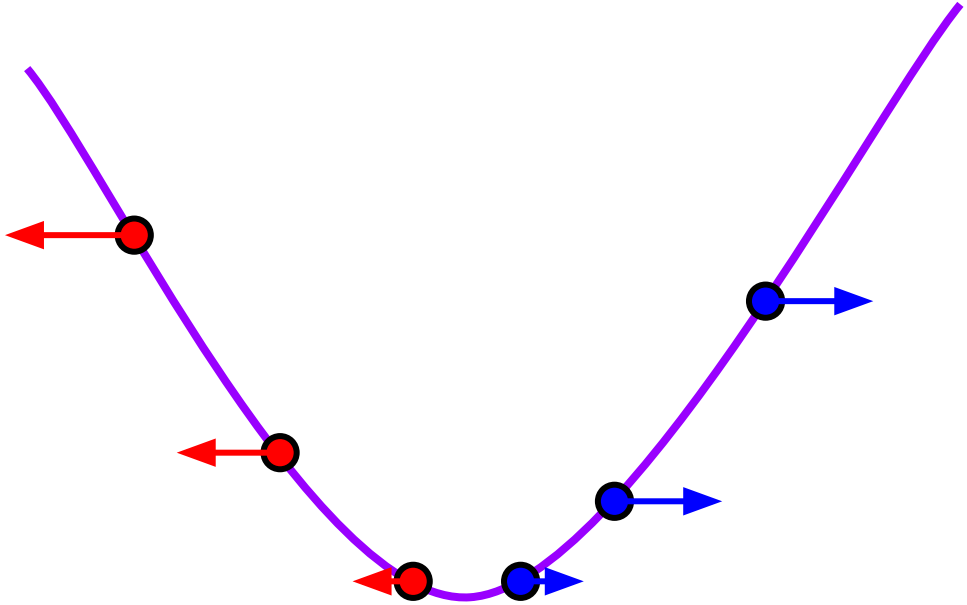
Trained using gradient descent (no closed form solution)



Gradient descent

For some loss function $L_{data}(\mathbf{w})$, gradient $\nabla L_{data}(\mathbf{w})$ points towards in direction of steepest ascent.

In 1d, either points left or right



From JP Reddie

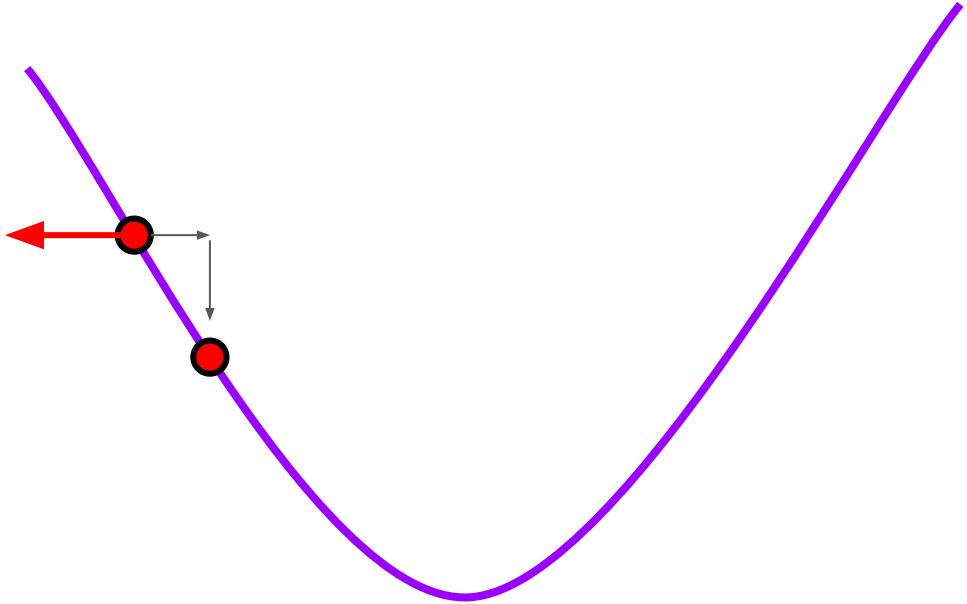
Gradient descent

For some loss function $L_{data}(\mathbf{w})$, gradient $\nabla L_{data}(\mathbf{w})$ points towards in direction of steepest ascent.

In 1d, either points left or right

Algorithm:

Take derivative
Move slightly in other direction
Repeat



From JP Reddie

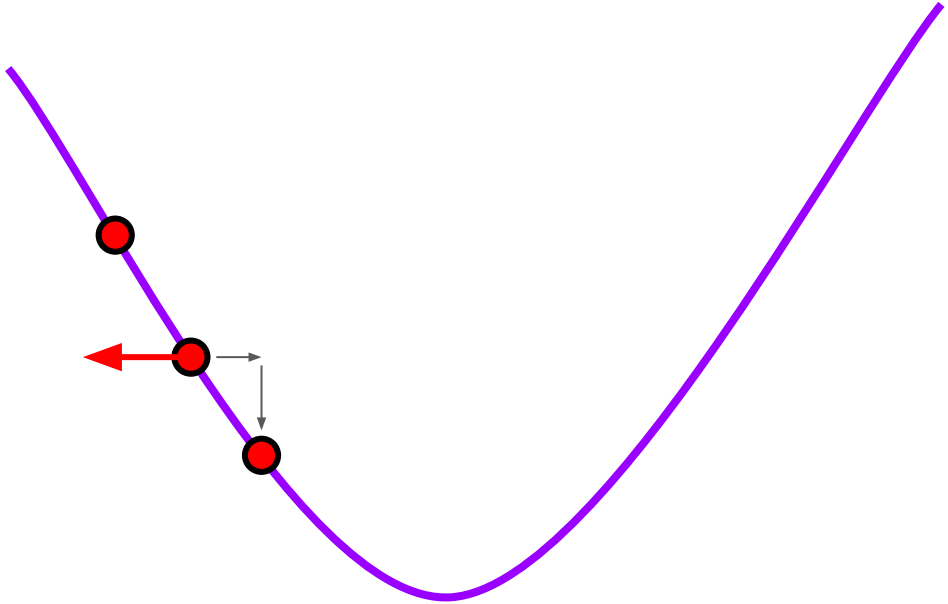
Gradient descent

For some loss function $L_{data}(\mathbf{w})$, gradient $\nabla L_{data}(\mathbf{w})$ points towards in direction of steepest ascent.

In 1d, either points left or right

Algorithm:

Take derivative
Move slightly in other direction
Repeat



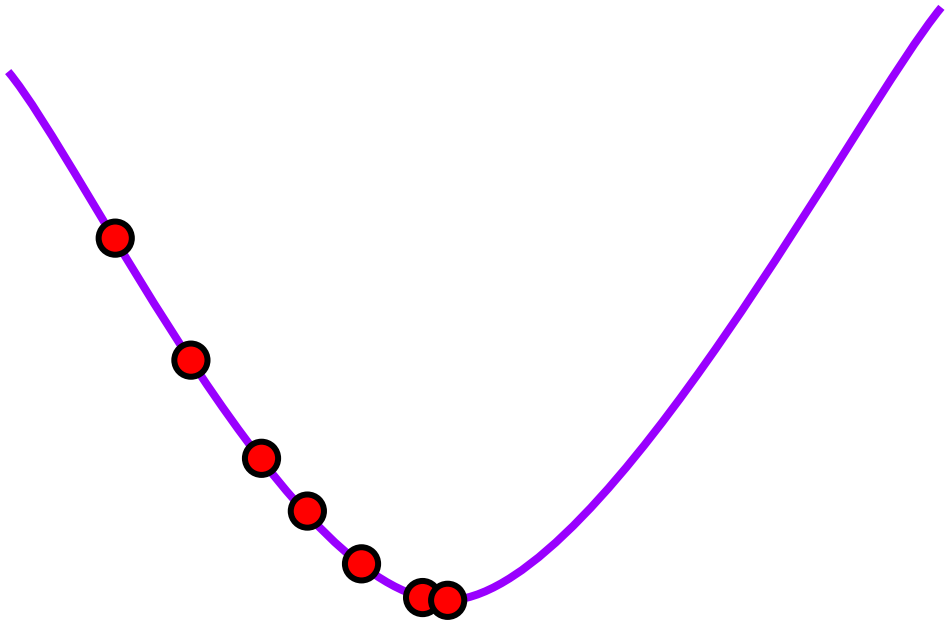
From JP Reddie

Gradient descent

Algorithm:

Take derivative
Move slightly in other direction
Repeat

End up at local optima



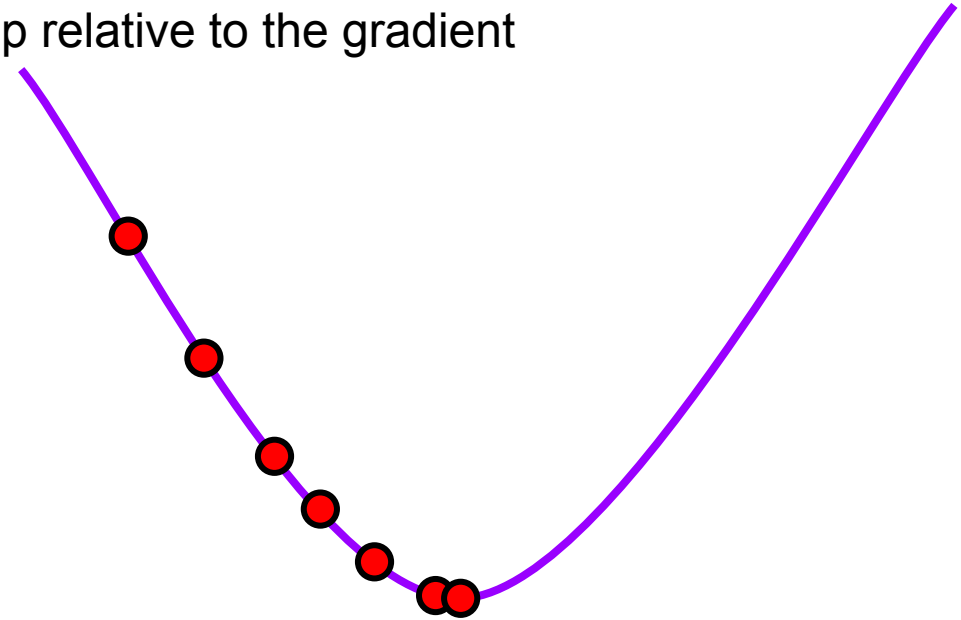
From JP Reddie

Gradient descent

Formally:

$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla L(\mathbf{w})$

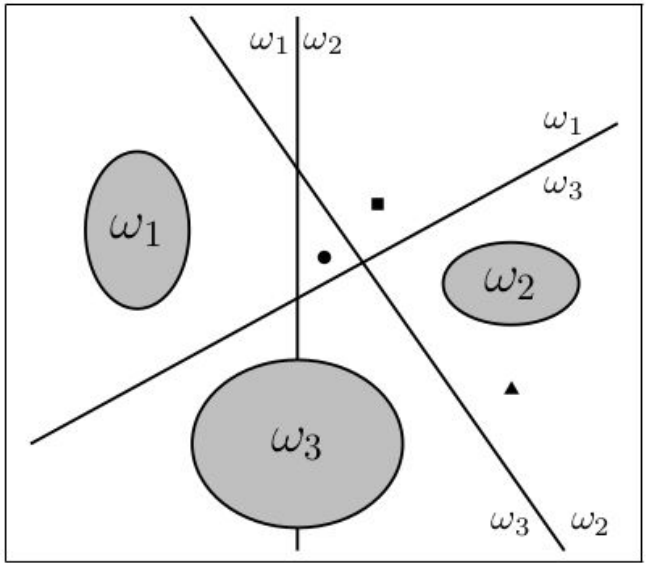
Where η is *step size*, how far to step relative to the gradient



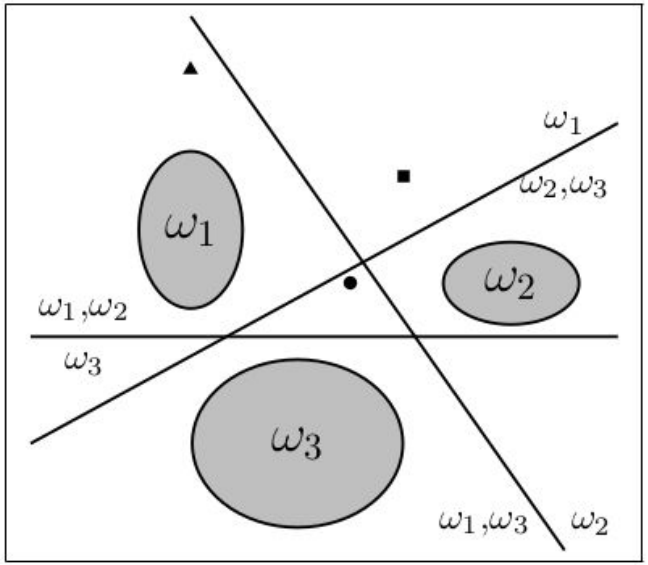
From JP Reddie

From 2 classes to C classes: two strategies

1 vs 1

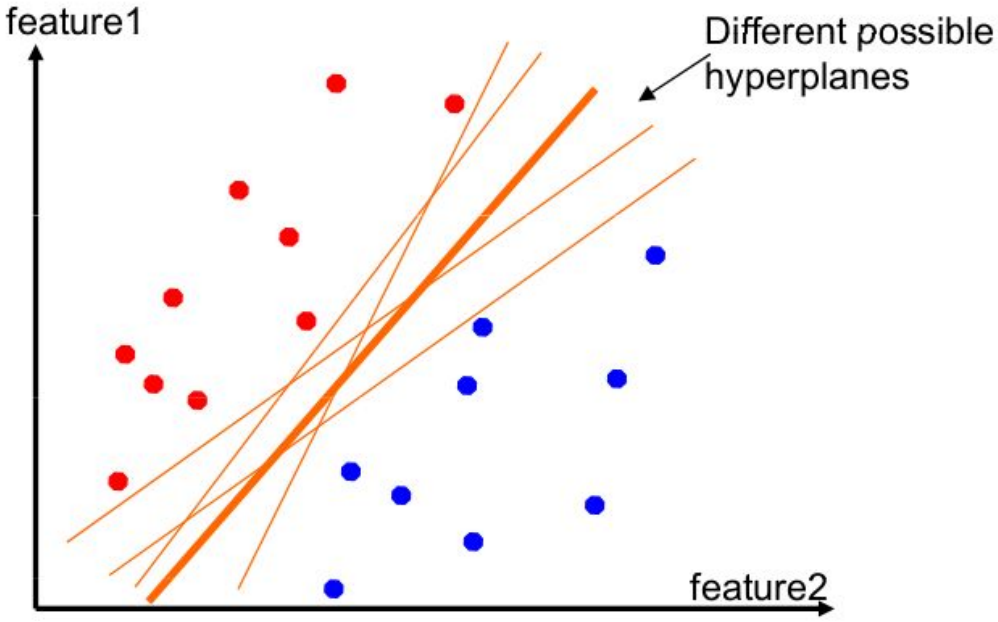


1 vs all



$$\hat{y} = \arg\max_{i \in Y} \mathbf{w}_i \mathbf{x}$$

Maximum Margin Classification



Maximum Margin Classification

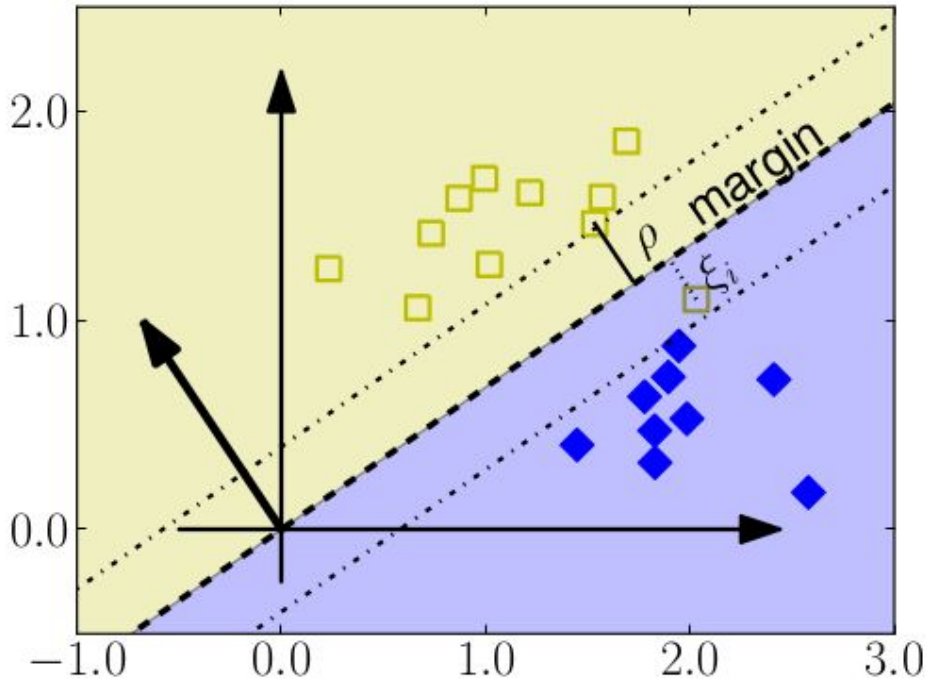
What is the best w for this dataset?

Trade-off:

large margin

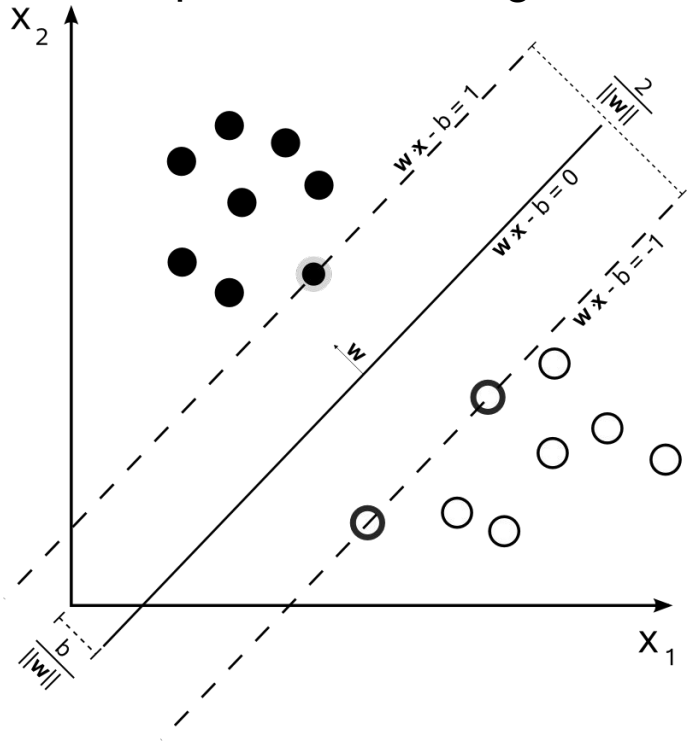
vs.

few mistakes on training set



Support Vector Machine (SVM)

Find max-margin classifier. Examples on the margin are supporting data points, support vectors.



Logistic Regression vs SVM

Optimization problems:

$$\min_{w \in \mathbb{R}^p} C \sum_{i=1}^{n} \log(\exp(-y_i w^T \mathbf{x}_i) + 1) + ||w||_2^2$$

$$\min_{w \in \mathbb{R}^p} C \sum_{i=1}^{n} \max(0, 1 - y_i w^T \mathbf{x}_i) + ||w||_2^2$$

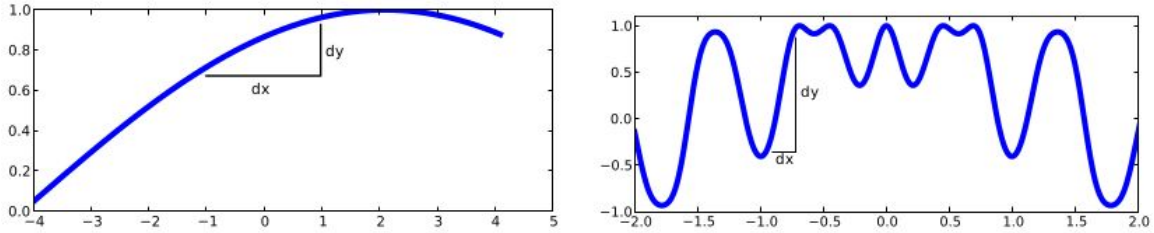
Cost / weighting of classification error

Classification errors on training set

Regularization term
- forces w to remain small
- avoids instability

About the regularizer

Ad-hoc definition: a function f : $R^d \rightarrow R$ is simple, if it not very sensitive to the exact input



sensitivity is measured by slope: f'

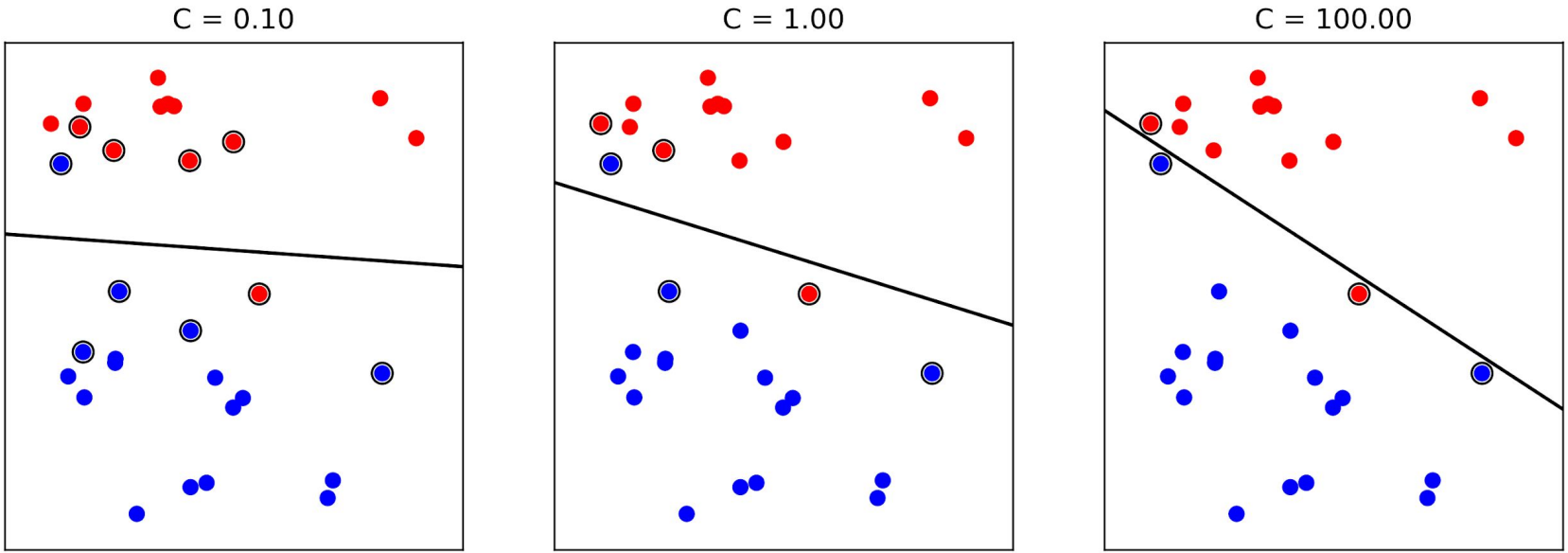
For linear $f(x) = \langle w, x \rangle$, slope is $\|\nabla_x f\| = \|w\|$

Minimizing $||w||^2$ encourages "simple" functions

From C. Lampert VRML summer school 2013

Effect of cost parameter C (regularization, again)

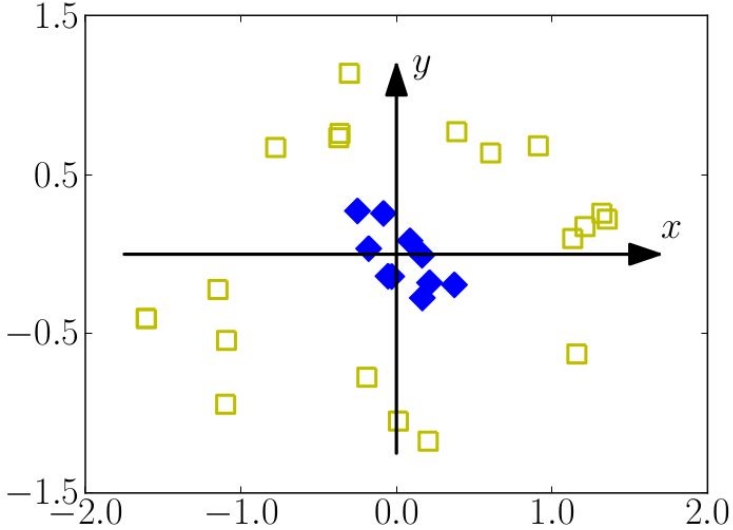
Small C (cost of indiv. errors — a lot of regularization) limits the influence of individual points. Adjust according to the amount of noise in your data.



Non-linear discriminant classifiers

Non-linear classification

What is the best linear classifier for this dataset?



None. We need something nonlinear!

Non-linear classification

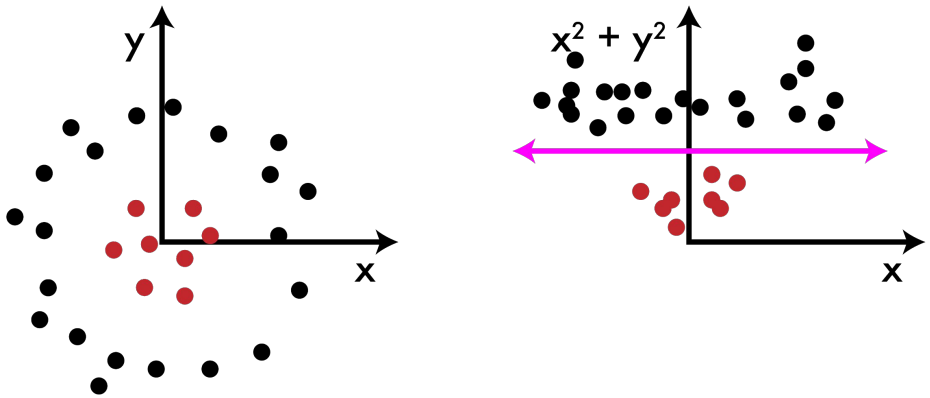
2 solutions:

1. Preprocess the data (explicit embedding, kernel trick…)
2. Combine multiple linear classifiers into nonlinear classifier (boosting, neural networks…)

Non-linear classification using linear classifiers with data preprocessing

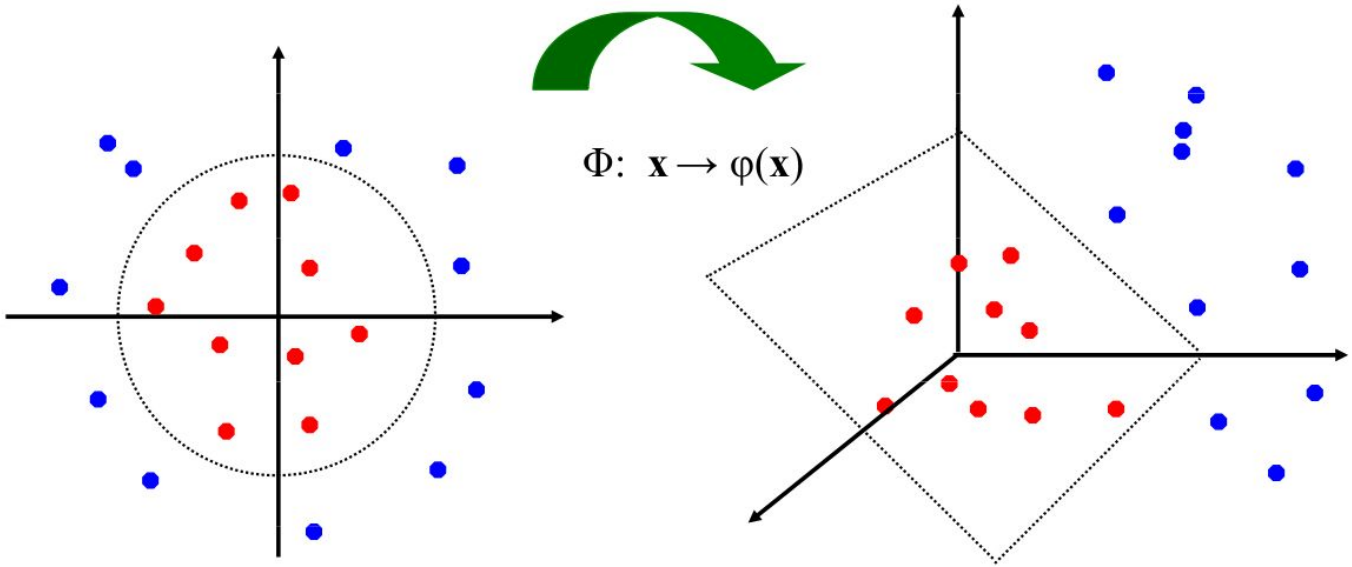
Data preprocessing idea

Transform the dataset to enable linear separability.



Linear separation is always possible

The original input space can always be mapped to some higher-dimensional feature space where the training set is separable.



Explicit embedding

Compute $\varphi(x)$ for all x in the dataset.

Then train a linear classifier just like before.

Used to be avoided because of computation issues, but it is a hot topic again.

Kernel trick

Linear classification requires to compute only dot products $\varphi(x_i)\varphi(x_j)$.

The function $\varphi(x)$ does not need to be explicit, we can use a kernel function

$$\mathbf{k(x, z) = \varphi(x)\varphi(z)}$$

which represents a dot product in a “hidden” feature space.

This gives a non-linear boundary in the original feature space.

Popular kernel functions in Computer Vision

"**Linear kernel**": identical solution as linear SVM

$$k(x, x') = x^\top x' = \sum_{i=1}^{d} x_i x'_i$$

"**Hellinger kernel**": less sensitive to extreme value in feature vector

$$k(x, x') = \sum_{i=1}^{d} \sqrt{x_i \, x'_i} \qquad \text{for } x = (x_1, \ldots, x_d) \in \mathbb{R}_+^d$$

"**Histogram intersection kernel**": very robust

$$k(x, x') = \sum_{i=1}^{d} \min(x_i, x'_i) \qquad \text{for } x \in \mathbb{R}_+^d$$

From C. Lampert VRML summer school 2013

Popular kernel functions in Computer Vision

"**χ^2-distance kernel**": good empirical results

$$k(x, x') = -\chi^2(x, x') = -\sum_{i=1}^{d} \frac{(x_i - x'_i)^2}{x_i + x'_i} \text{ for } x \in \mathbb{R}_+^d$$

"**Gaussian kernel**": overall most popular kernel in Machine Learning

$$k(x, x') = \exp(-\lambda \|x - x'\|^2)$$

… plus others….

From C. Lampert VRML summer school 2013

Explicit embedding for the Hellinger kernel

$$k(x, x') = \sum_{i=1}^{d} \sqrt{x_i \, x'_i} \qquad \text{for } x = (x_1, \ldots, x_d) \in \mathbb{R}_+^d$$

Using simple square root properties, we have:

k(x,x') = φ(x)φ(x') = sqrt(x) sqrt(x')

Tricks for next practice session: given a BoVW vector,

1. L1 normalize it (neutralizes effect of number of descriptors)
2. Take its square root (**explicit** Hellinger embedding)
3. L2 normalize it (more linear-classifier friendly)

You are encouraged to experiment with and without each step.

next lecture: more classifiers

non linear discriminant classifiers