

# Basics in Machine Learning

Karel Beneš



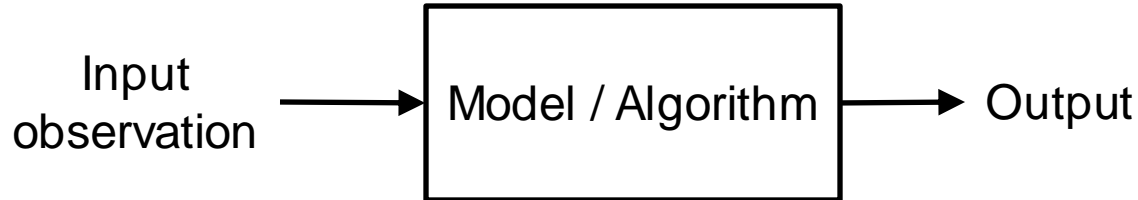
International Summer School in IT  
Brno, July 2021

Slides obtained from Lukáš Burget

# What is Machine Learning?

- Tom M. Mitchell : "A computer program is said to learn from experience  $E$  with respect to some class of tasks  $T$  and performance measure  $P$  if its performance at tasks in  $T$ , as measured by  $P$ , improves with experience  $E$ ."

- Typical tasks:

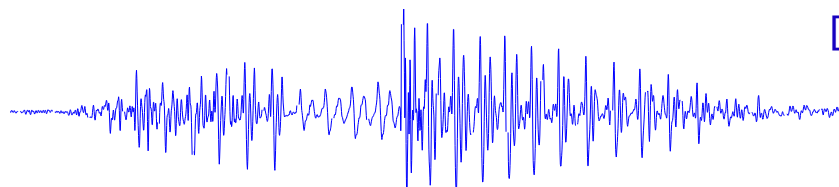


Examples:

- Recognize words in speech recording
  - Recognize person identity from an image of face
  - Translate Czech text to Korean
  - Classify object from its measured size and weight
  - Predict stock prices using companies' past quarterly revenue results
- Typical experience representation: Collection of input and/or corresponding output **training examples**.
  - Typical way of measuring performance: Check how well is the task performed on some new (evaluation) inputs and desired outputs.

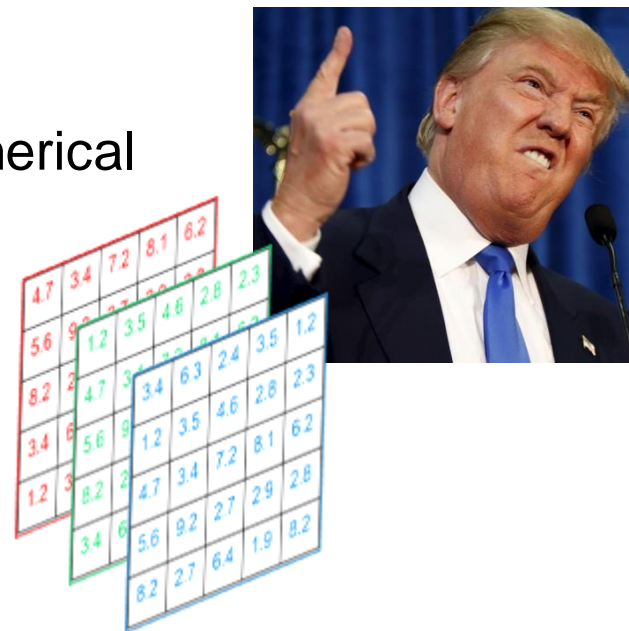
# Examples of input observations I.

- Speech waveform – variable length sequence of numerical values



[ 0.1, 1.5, 5.4, 5.2, 1.1, -2.3, -5.4, ..., 0.8 ]

- 100x100 image of face – 3D matrix of numerical values (one dimension for color channels)



- Sequence of words – variable length sequence of discrete symbols

# Examples of input observations II.

- Input observations can have
  - have continuous or discrete values
  - fixed or variable size (vector or matrix vs. sequence of symbols, vectors, ...)
- In this lecture, we will often consider only  $D$ -dimensional vector of numerical values (or even scalar) as an input observation:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_D \end{bmatrix}$$

- A set of  $N$  input observations (e.g. set of training examples) will be then represented by a matrix:

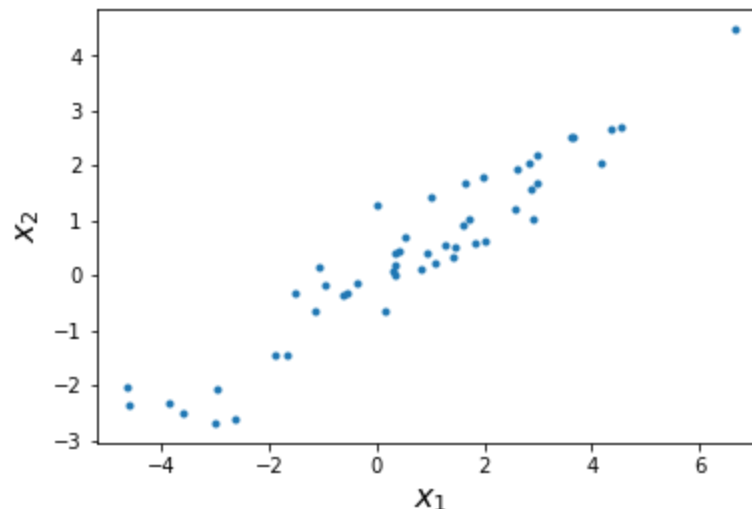
$$\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N] = \begin{bmatrix} x_{11} & x_{21} & \cdots & x_{N1} \\ x_{12} & x_{22} & \cdots & x_{N2} \\ \vdots & \vdots & \ddots & \vdots \\ x_{1D} & x_{2D} & \cdots & x_{ND} \end{bmatrix}$$

# Examples of input observations III.

- For simplicity, we will often consider 2D vectors as input observations:  
Examples:
  - Vector of two values corresponding to the size and weight of an input object
  - Trivia grayscale image with only two pixels

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

- This allows us to visualize a set of input observations as set of points in 2D space:

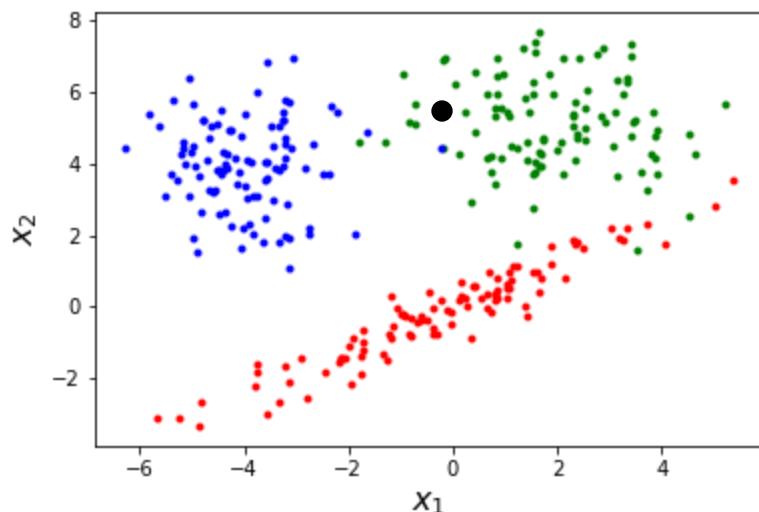


# What is Machine Learning? II.

- Given set of training examples (inputs and/or outputs) learn to map new “unseen” inputs to desired outputs
  - Examples:
    - Given a data set of hundreds of hours of transcribed speech audio recordings learn to automatically transcribe new speech recordings
    - Given a data set of millions of images of human faces labeled by person identity learn recognize person identity in a new images
- Main types of learning algorithm
  - Supervised learning
    - Training examples are pairs of inputs and desired outputs
    - Typical tasks: classification, more general pattern recognition, regression, ...
  - Unsupervised learning
    - Training examples are only unannotated (input) data
    - Typical tasks: Clustering, anomaly detection, density estimation, ...
  - Semi-supervised
    - Some of the training examples are annotated input/output pairs, but we have also many unannotated inputs available for the learning.
  - Reinforcement learning
    - Examples: learning to drive autonomous vehicles, learning to play a computer or board games
    - feedback given in the form of positive or negative reinforcement after making series of decisions/action (e.g at the end of the (un)successful drive/game)

# Supervised learning

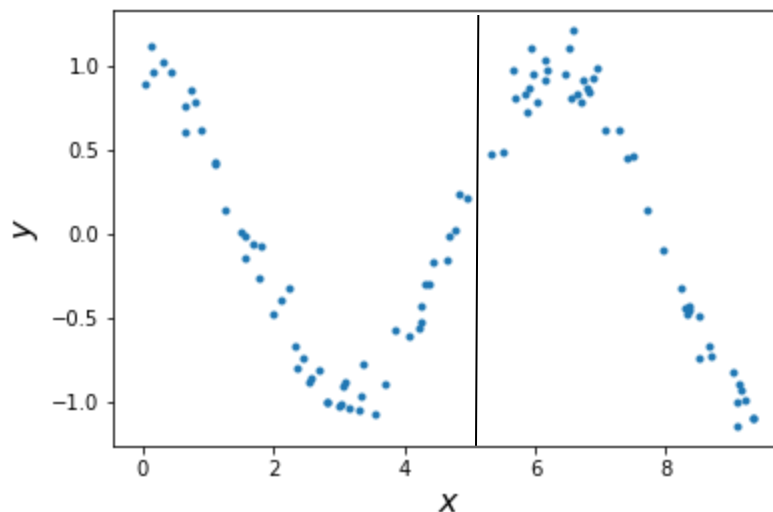
- Training examples are pairs of inputs,  $\mathbf{x}$ , and desired outputs,  $\mathbf{y}$ ,
- Classification:



$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad y = \{\text{red}, \text{green}, \text{blue}\}$$

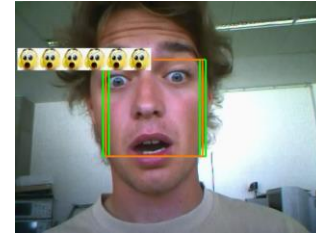
given the training examples (colored dots),  
learn to assign class (color) to new  
“unseen” observation (black dot)

- Regression:  
given training examples, learn to  
predict likely value of  $y$  for new  
“unseen” value  $x$  (horizontal line)  
i.e. learn function  $y = f(x)$



# Supervised learning - examples

- All our previous examples corresponded to supervised learning
- Classification:
  - Recognize person identity from an image of face
  - Classify object from its measured size and weight
  - Classify facial expression for each video frame
- Regression:
  - Predict stock prices using companies' past quarterly revenue results
  - Predict weather (temperature, humidity, chance of rain, ...) from past meteorological measurements
- More general pattern recognition problems
  - Recognize words in speech recording
  - Detect and classify all know (>9k) objects in video <https://youtu.be/MPU2HistivI>
  - Estimate pose of every person in video <https://youtu.be/pW6nZXeWIGM>
- Other supervised learning problem
  - Translate Czech text to Korean (i.e. Machine Translation)
  - Automatically describe image using English text
  - Generate realistic image from its English text description





# Supervised learning - examples

- Automatically describe image using English text

*Based on convolutional and recurrent neural networks*

*Andrej Karpathy, Li Fei-Fei: Deep Visual-Semantic Alignments for Generating Image Descriptions*



man in black shirt is playing guitar.



construction worker in orange safety vest is working on road.



two young girls are playing with lego toy.



boy is doing backflip on wakeboard.

# Supervised learning - examples

- Generate realistic image from its English text description

*Based on Generative Adversal Neural Networks*

*Han Zhang, et al.: StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks.*



This bird has a yellow belly and tarsus, grey back, wings, and brown throat, nape with a black face



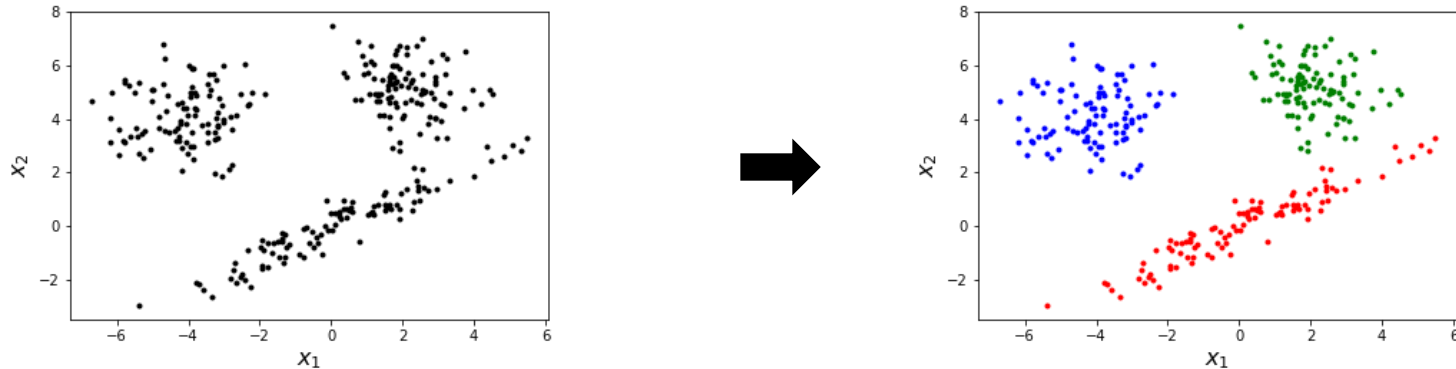
This bird is white with some black on its head and wings, and has a long orange beak



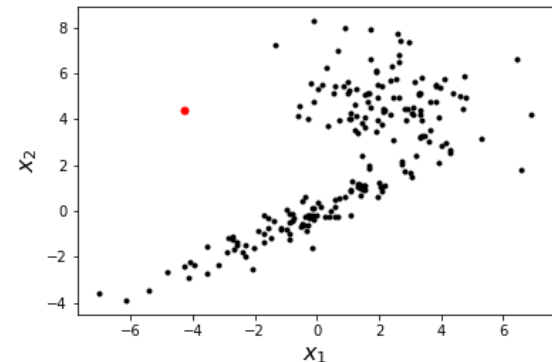
This flower has overlapping pink pointed petals surrounding a ring of short yellow filaments

# Unsupervised learning I.

- Clustering: find clusters of “similar examples” in the input data



- In collection of documents, find similar documents (on the same topic)
- In a recording of a conversation, find and cluster segments spoken by the same speaker (i.e. speaker diarization)
- Anomaly detection: detect unusual inputs (outliers)
  - to refuse them for further processing
  - to point them out as particularly interesting



# Unsupervised learning II.

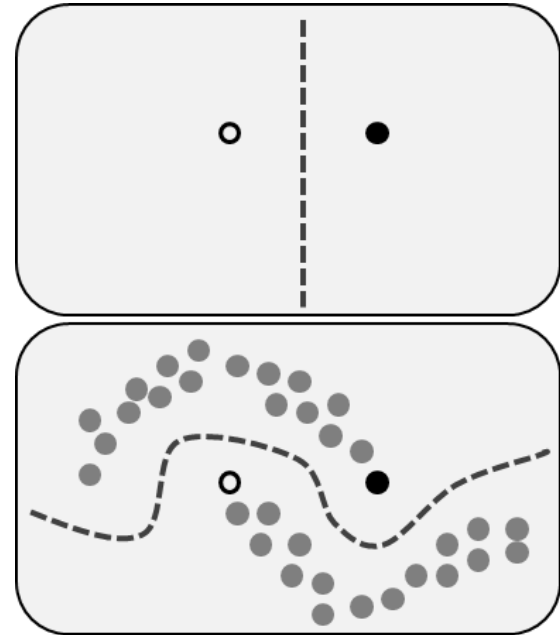
- Density estimation: learn probability density function from data
  - We will see how to estimate simple (Gaussian) distribution from data
  - However, neural networks can be used to learn complex distribution (e.g. distribution of human face images) and generate new samples from it

*Diederik P. Kingma, Prafulla Dhariwal: Glow: Generative Flow with Invertible 1x1 Convolutions*



# Semi-supervised learning

- Unannotated examples can help to find better decision boundary between classes
- There is lots of unannotated data available on the internet
  - Text
  - Photos and other images
  - Speech and other recordings
  - ...





# How to build a classifier?

**The task:**



**VS.**



apple

**Marmelade  
factory**



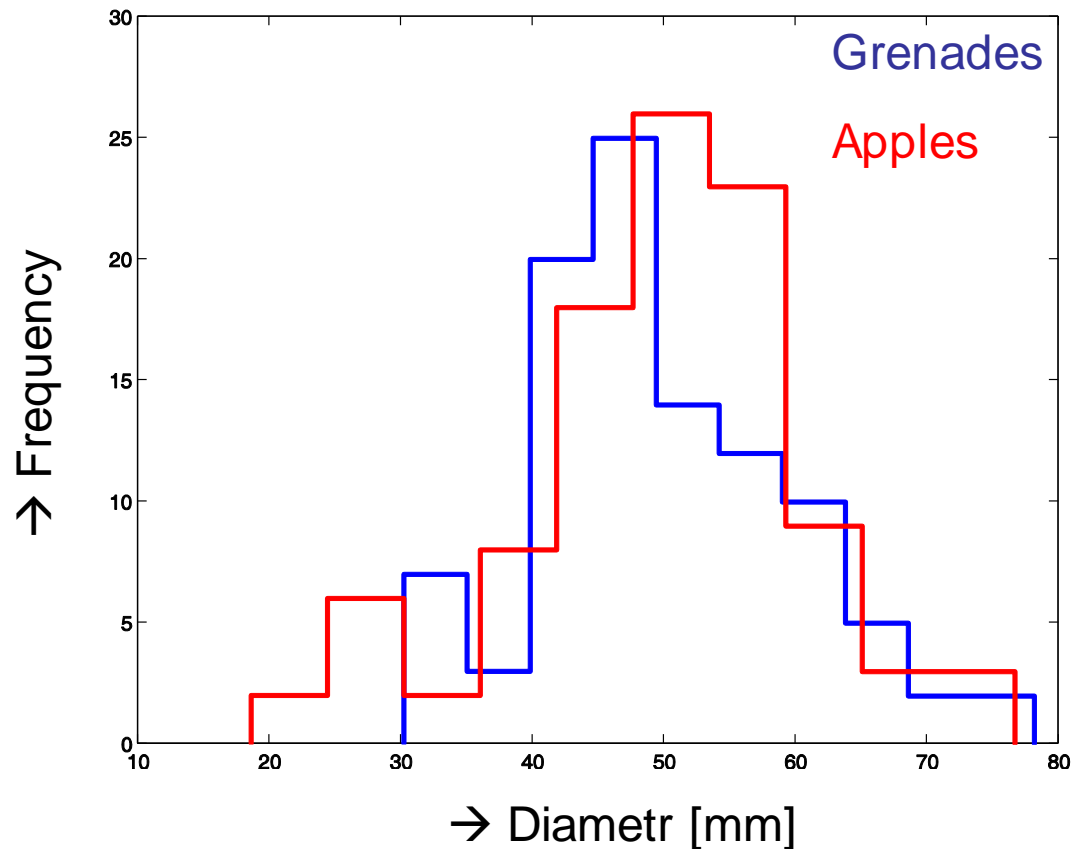
**Pyrotechnician**



grenade

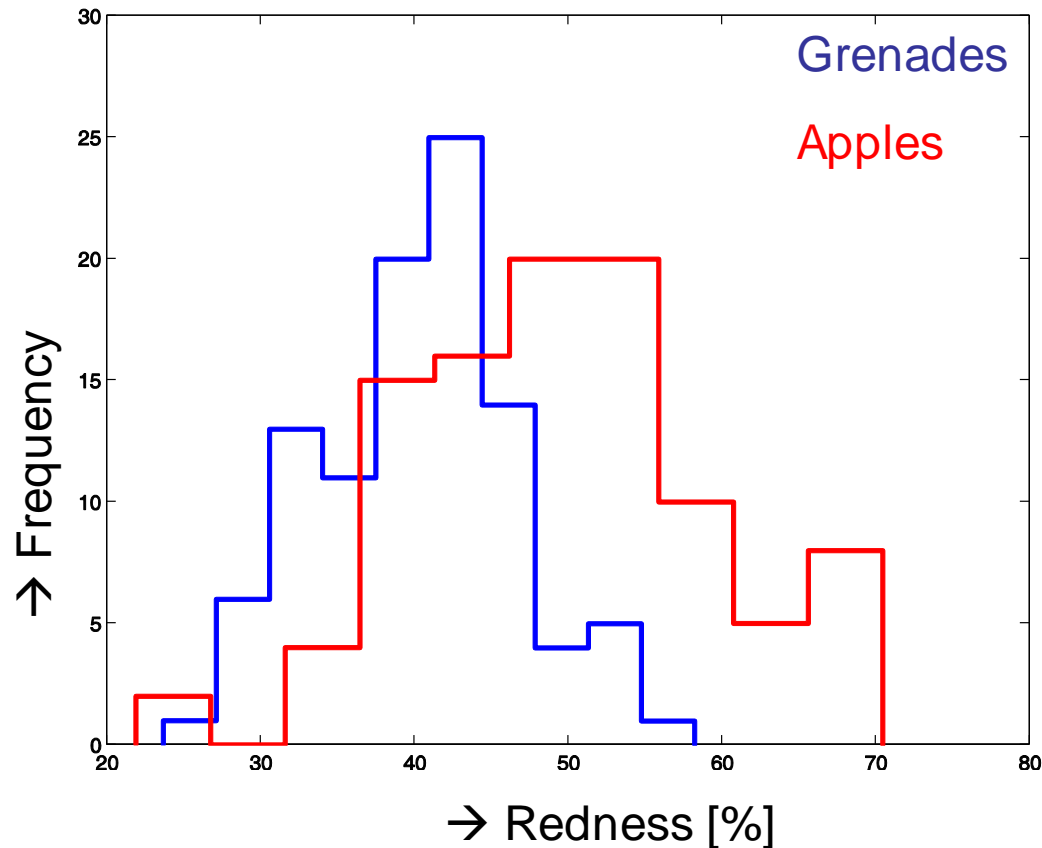
# Finding discriminative features

- Let's have the following histograms estimated using training examples.
- Is diameter of apple/grenade good feature?



# Finding discriminative features II.

Little better, but still not very discriminative

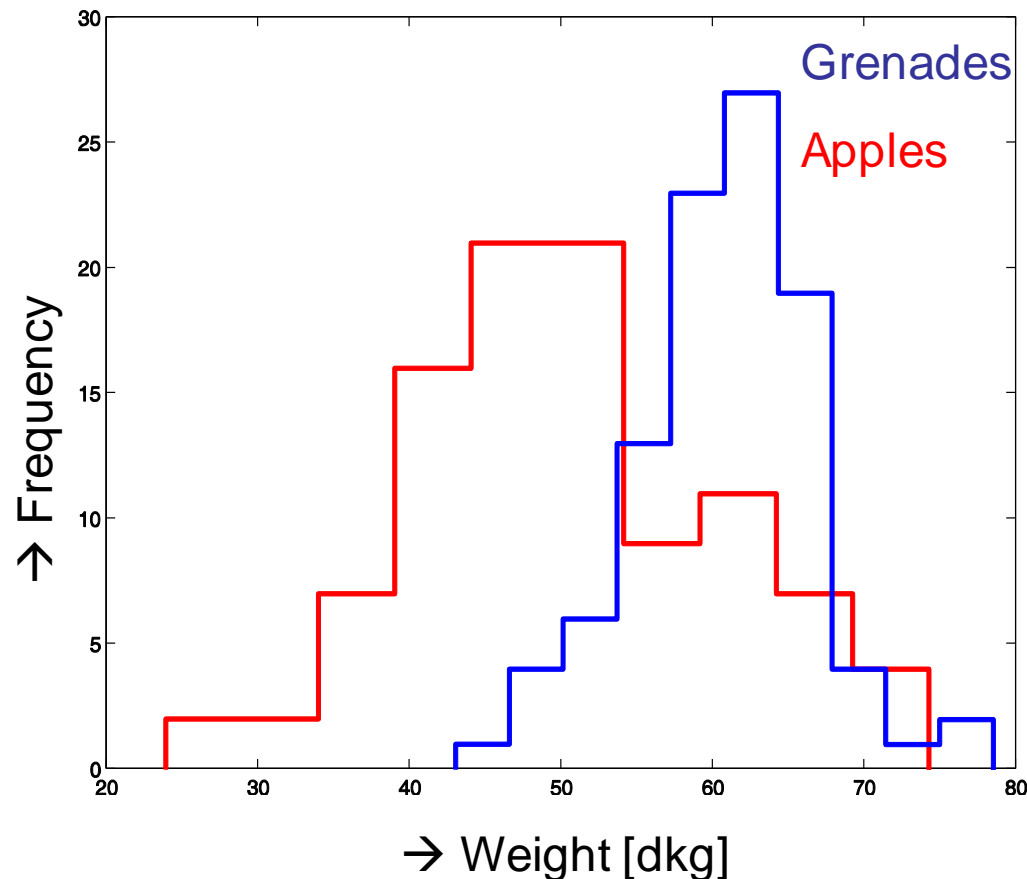


Intuitively, we could start recognizing apples from grenades by setting threshold so that most of the training examples would be classified correctly.

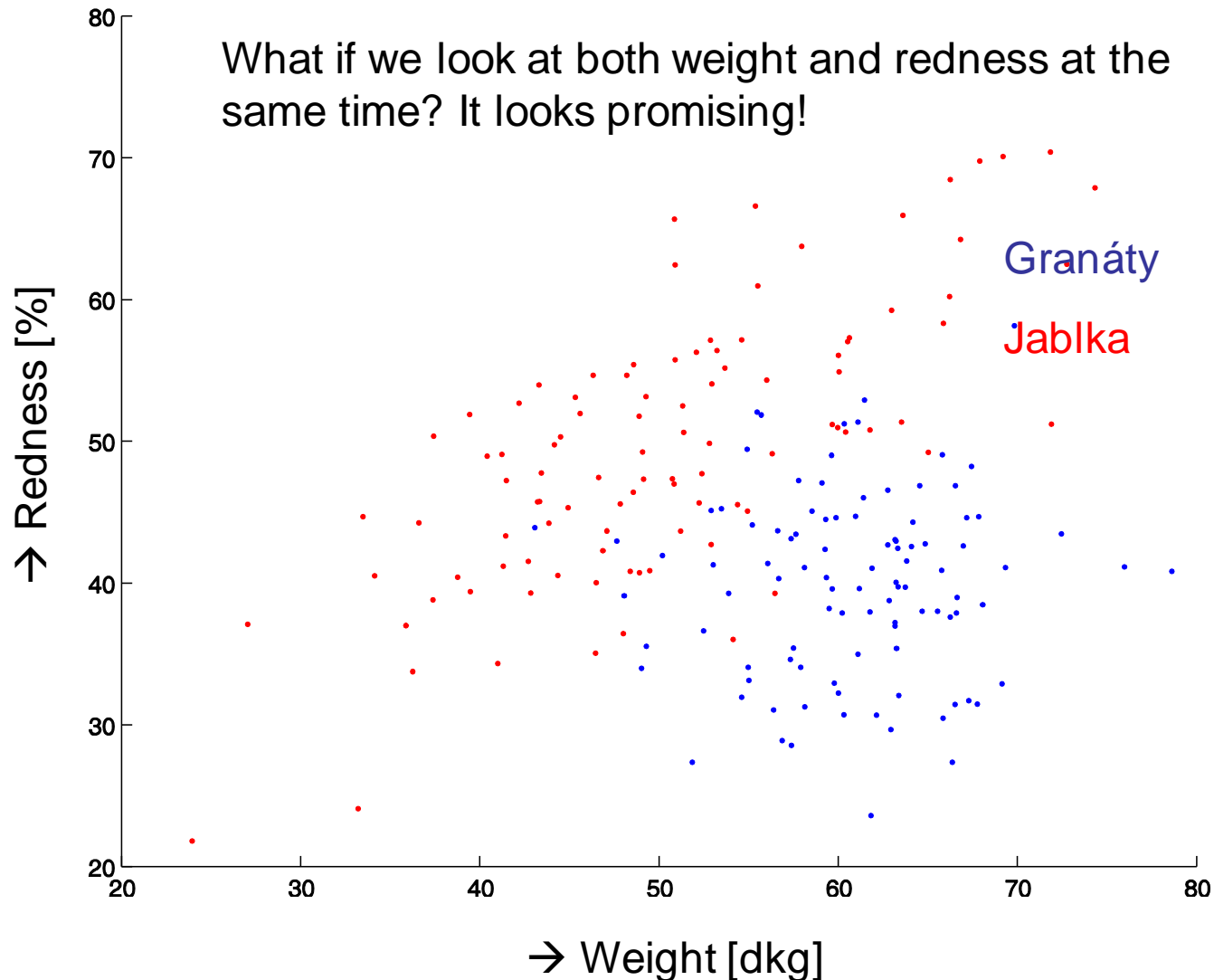


# Finding discriminative features III.

Histogram is an estimate of the probability distribution. So, knowing probability distributions of input features will be perhaps useful for classification.

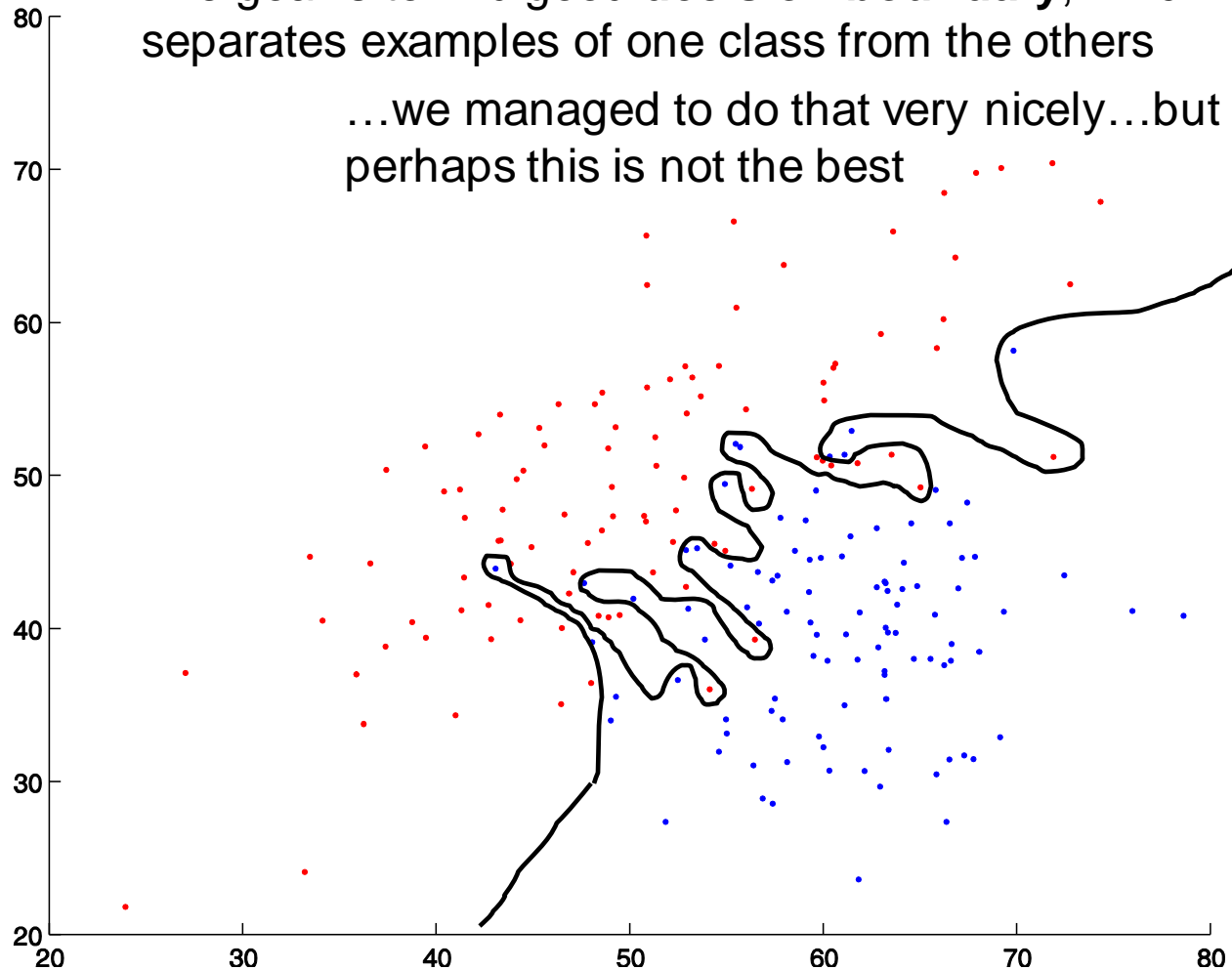


# Multivariate features (input patterns)



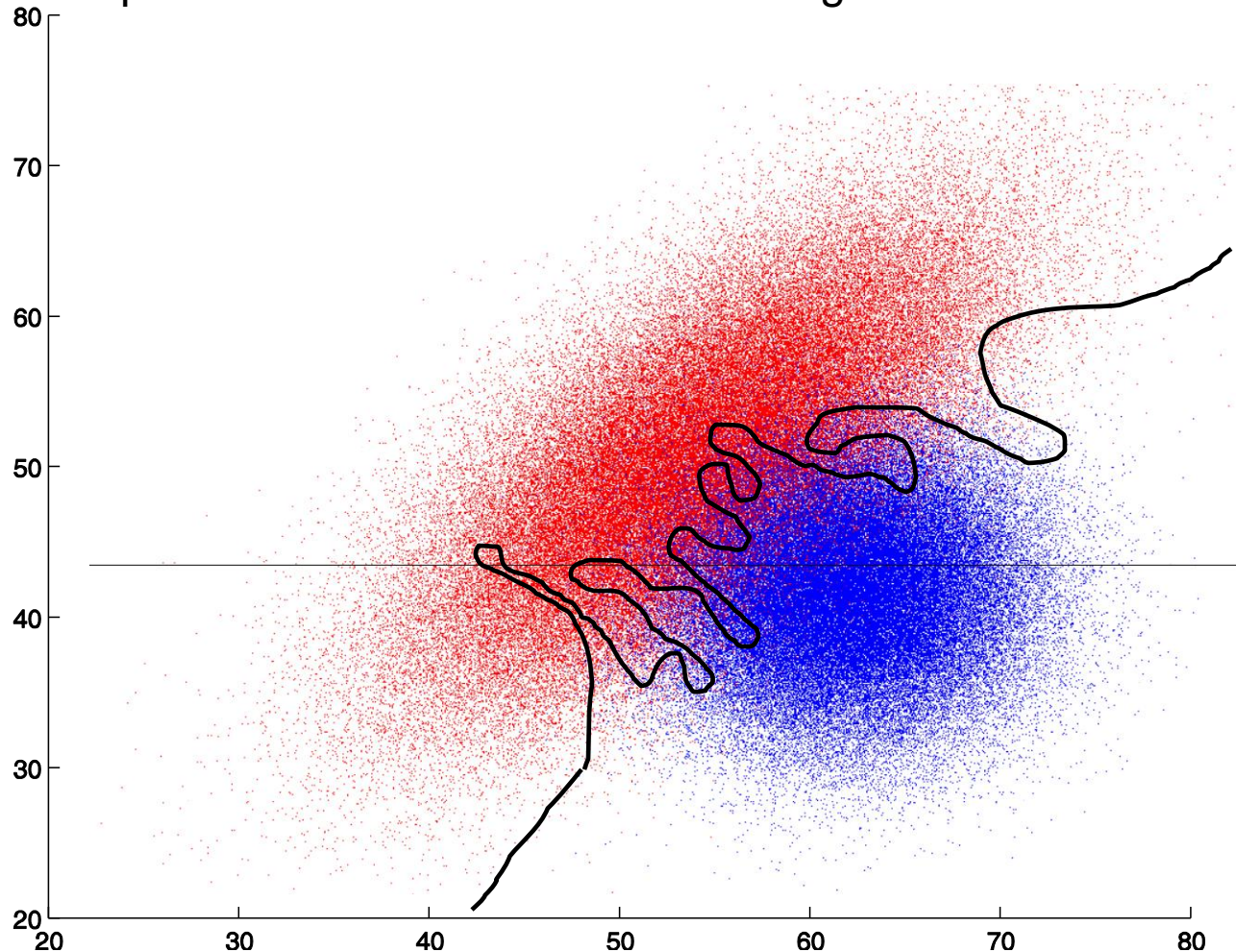
# Classification

The goal is to find good **decision boundary**, which separates examples of one class from the others  
...we managed to do that very nicely...but perhaps this is not the best



# Generalization

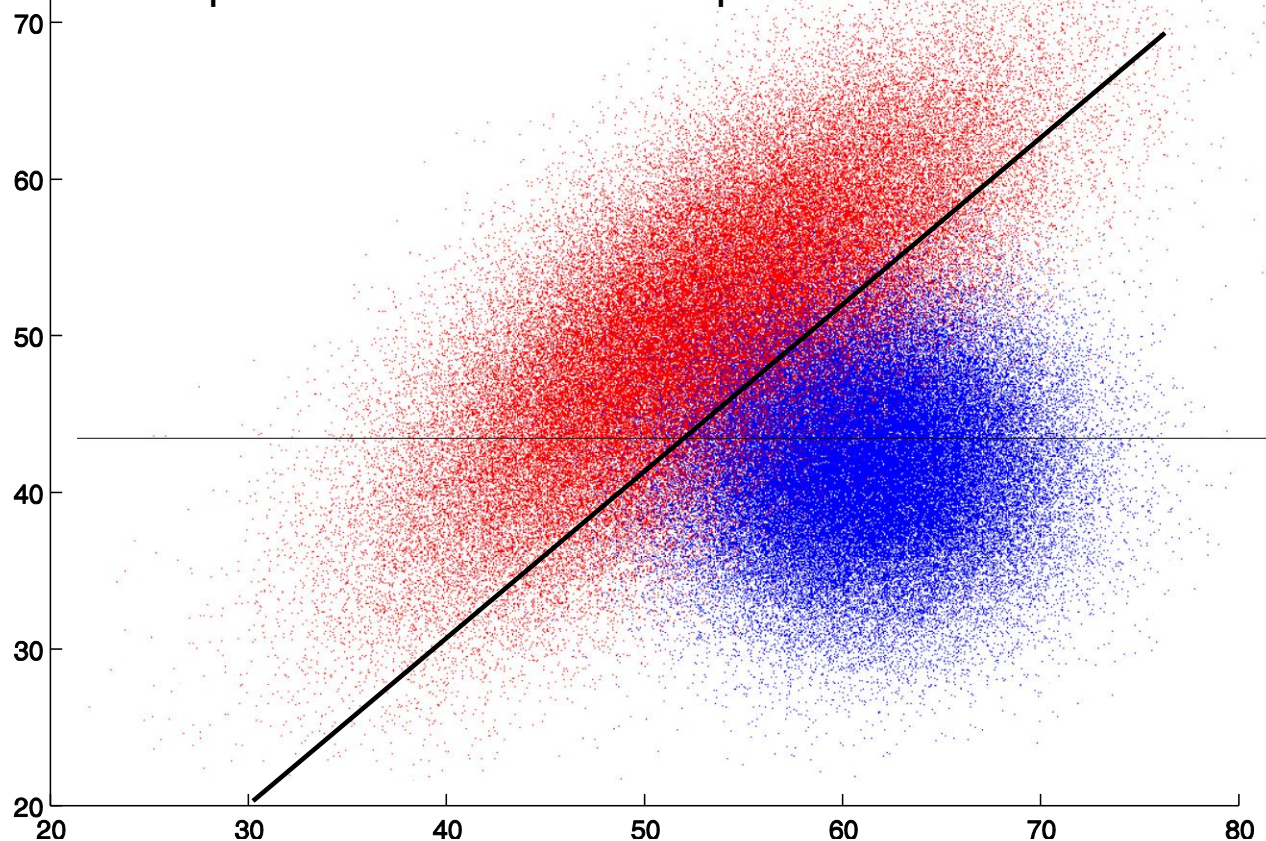
In our example, the data for each class were randomly generated from a Multi-variate Gaussian distribution. If we “collect” more of such data, the result will not be that optimistic. The classifier does NOT generalize to new data.



# Linear classifier

We can improve the generalization by constraining the decision boundary to be a line. We will later see how to determine such line.

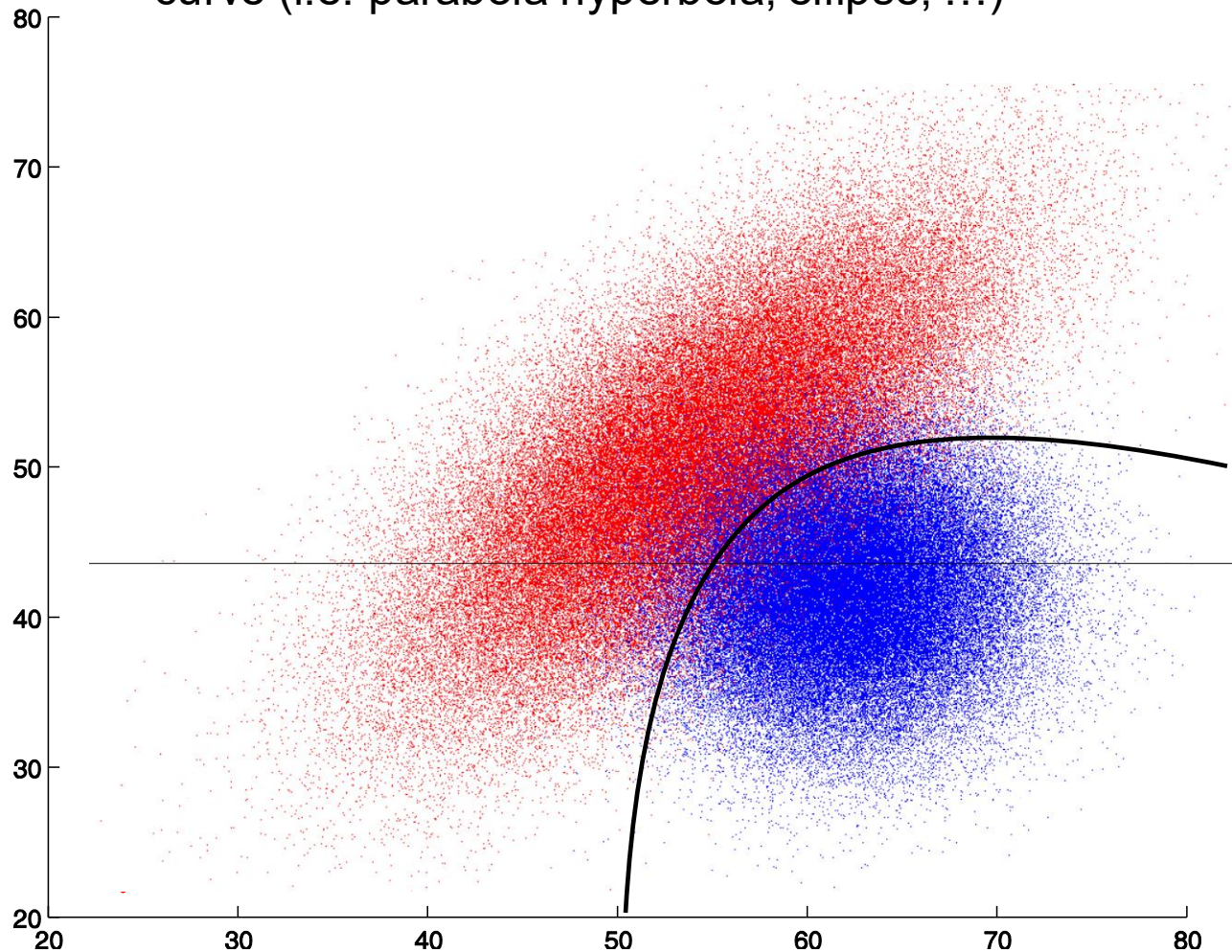
Unfortunately, classification based on these selected features will never be without errors as the distributions for the two classes overlap. We will at least attempt to minimize the error.





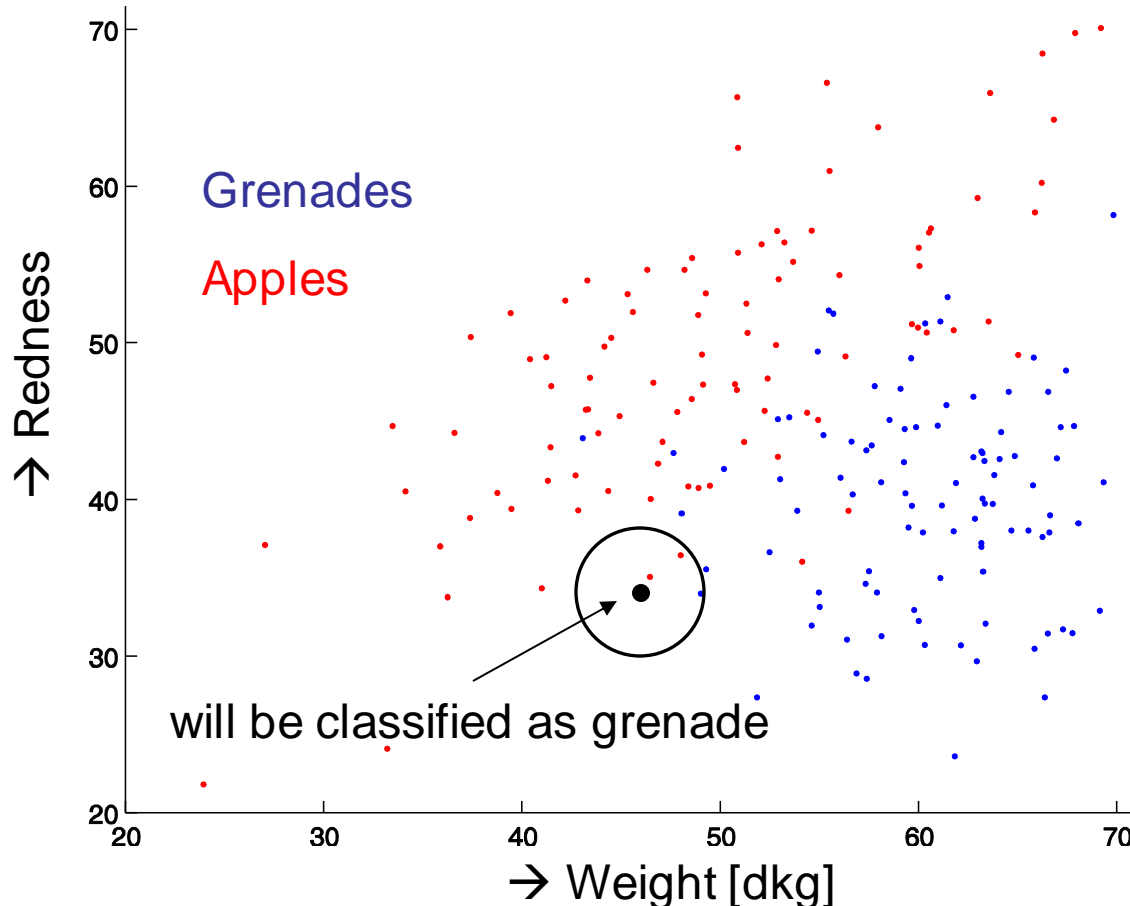
# Quadratic classifier

We will later see that, if the classes are Gaussian distributed, the optimal decision boundary will be conic curve (i.e. parabola hyperbola, ellipse, ...)

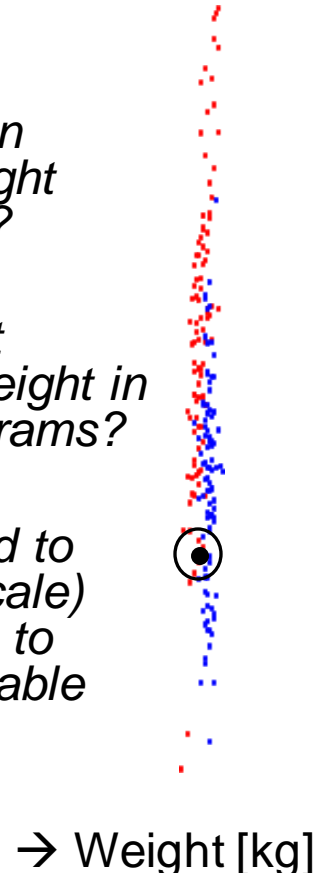


# K-nearest neighbors classifier

- *Nonparametric classifier* – no parameters to train or estimate
- Needs to remember all training examples
- To assign class to a new pattern (black dot), find K nearest examples in the training data and choose the more represented class.

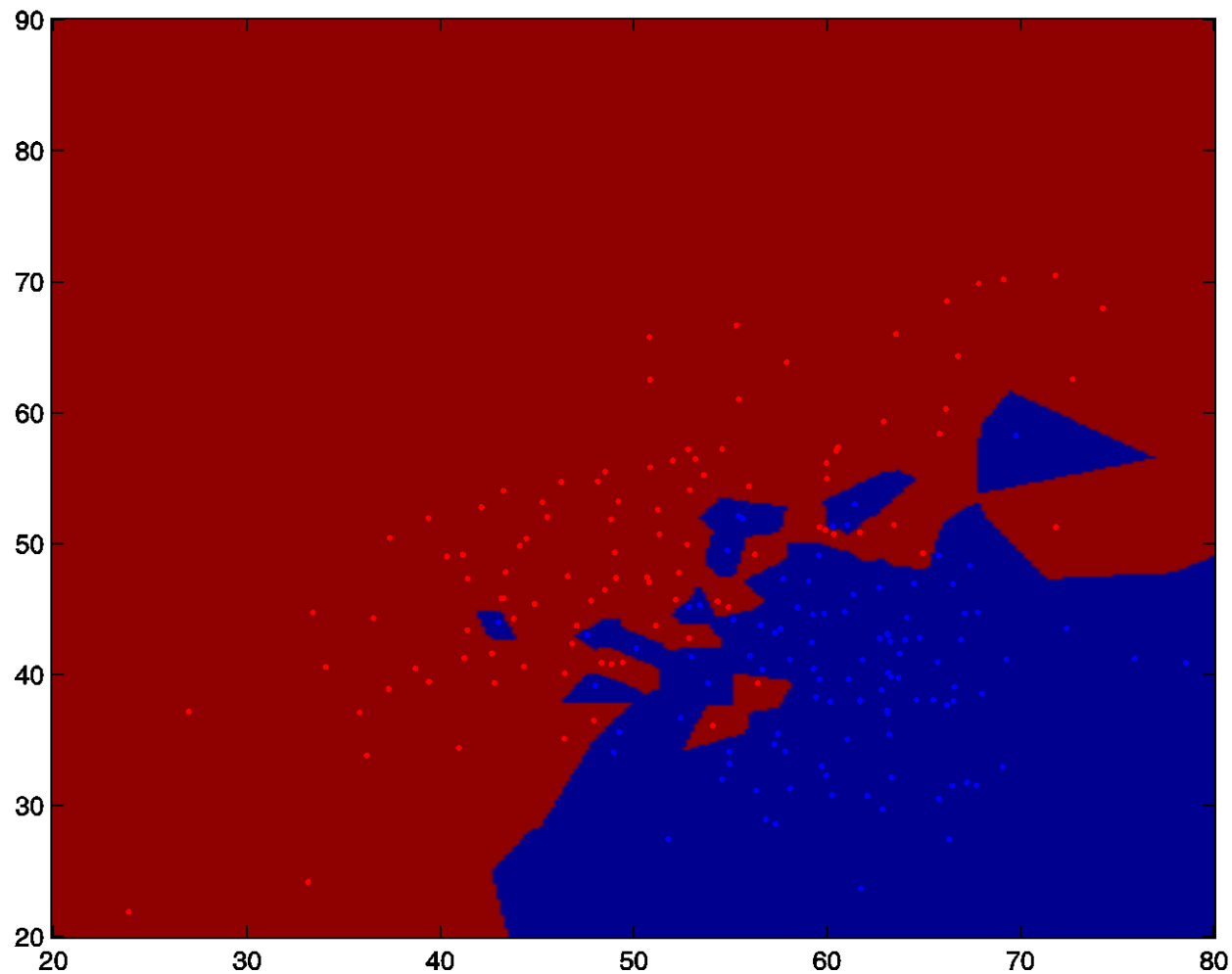


- *But can I even compare weight with redness?*
- *What if I start measuring weight in tons or milligrams?*
- *First, we need to normalize (scale) both features to have comparable values*



# 1- nearest neighbor

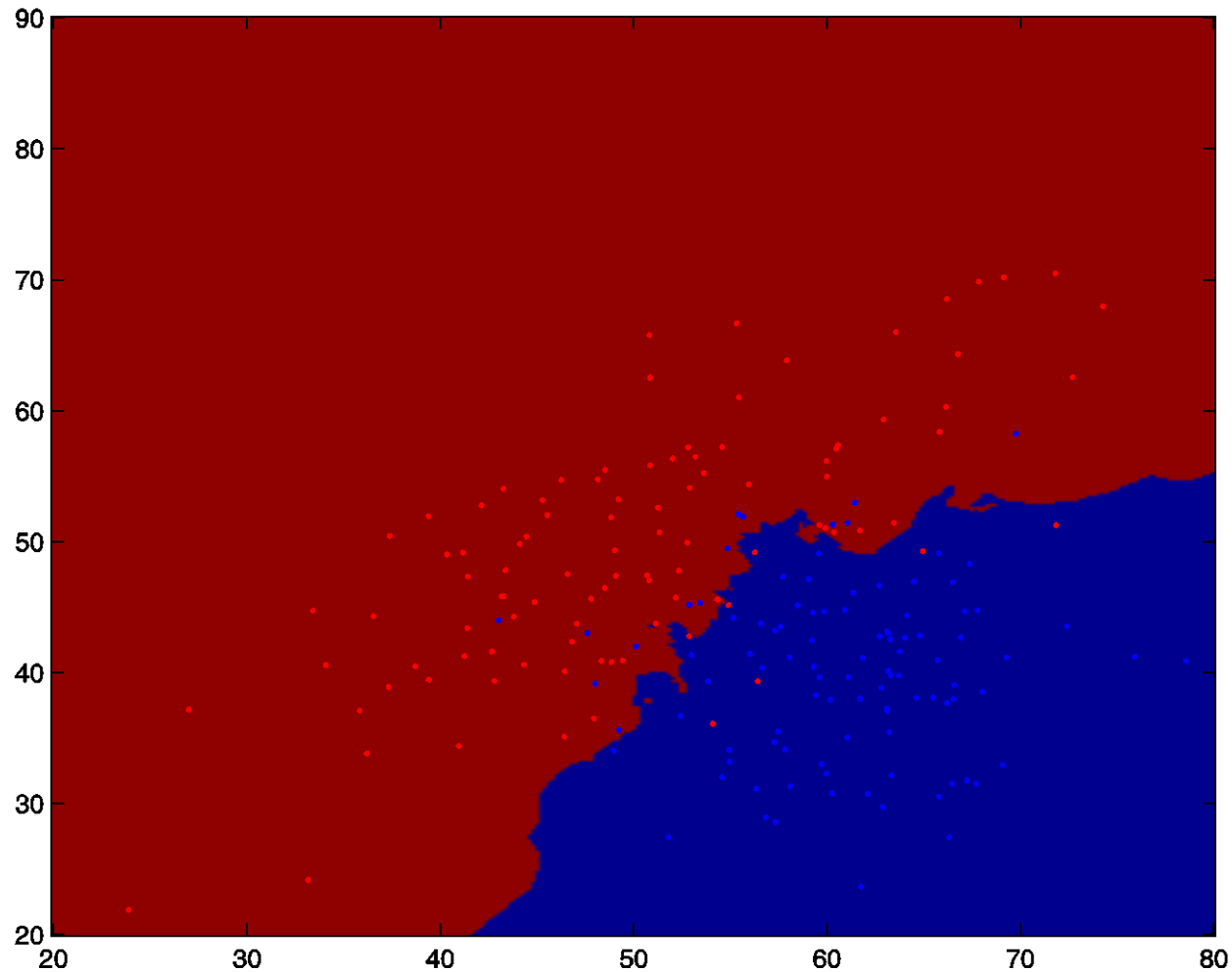
Again problem with generalization: too complex  
decision boundary





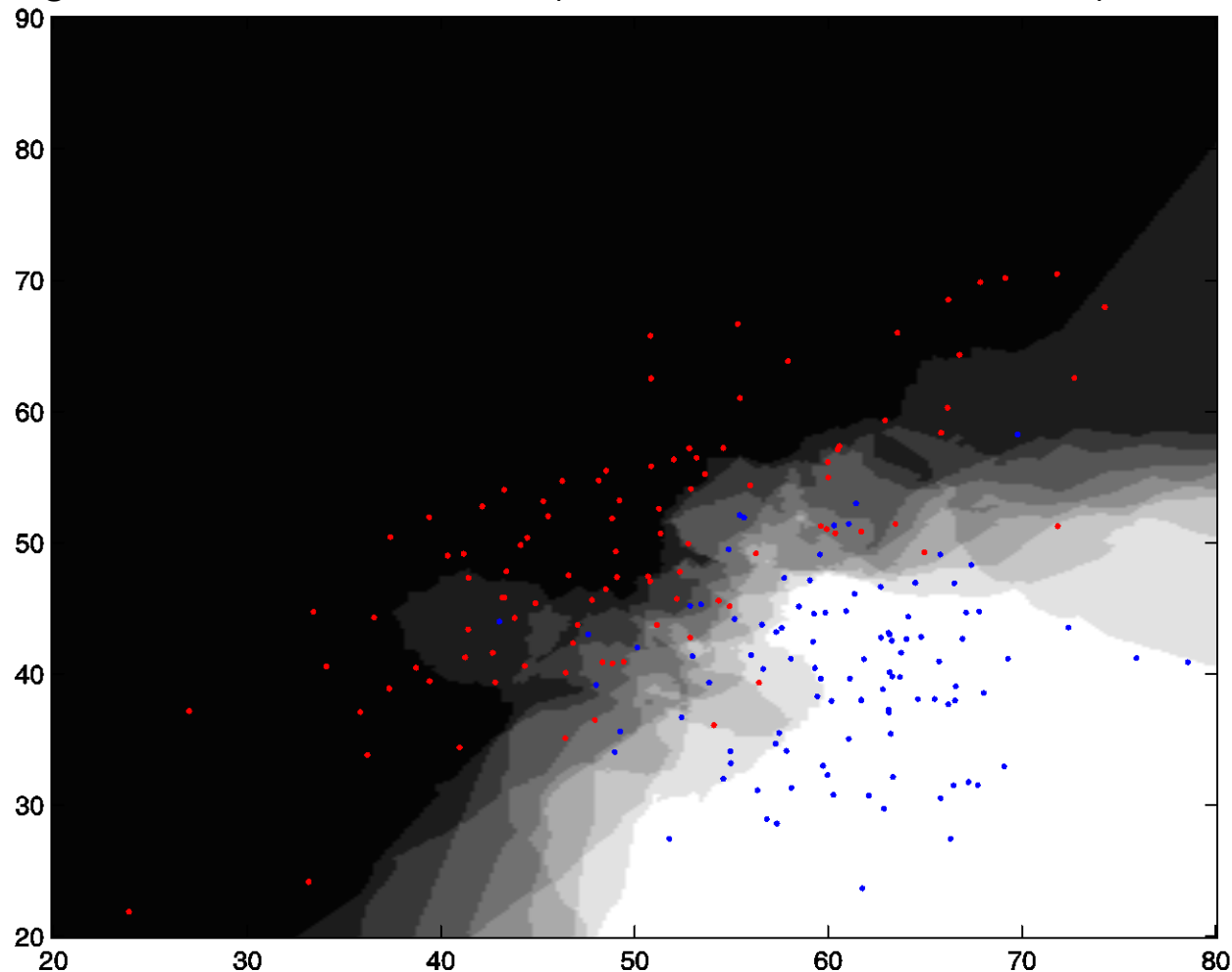
# 9-nearest neighbors

Decision boundary is already close to the optimal parabola



# 9- nearest neighbors soft decision

Ratio of the numbers of neighbors from the two classes can be used as an estimate of the probability that the classified patterns belongs to one of the classes (10 different levels for  $K=9$ ).



# Classification – Generative model

- With generative models, we model distribution of the observed data
- For simplicity, let's consider only discrete observations first.
- The task is to classify an object (*grenade* or *apple*) given an observation (discrete weight category)
  - It is heavy. Is it grenade or apple?
- Let's have 150 observations as training examples
  - Table of observation counts for each class and weight category



1	6	12	15	12	2	2	50
4	22	50	14	6	3	1	100
<i>lightest</i> 0.0 - 0.1	<i>lighter</i> 0.1 - 0.2	<i>light</i> 0.2 - 0.3	<i>middle</i> 0.3 - 0.4	<i>heavy</i> 0.4 - 0.5	<i>heavier</i> 0.5 - 0.6	<i>heaviest</i> 0.6 - 0.7	[kg]

# Marginal, joint and conditional probability

- $P(\text{class}, \text{observation})$  – joint probability
  - probability of a single field in our table
  - normalizing the counts by the total count gives **Maximum likelihood (ML) estimates** (see later):  $P(\text{grenade}, \text{heavy}) = \frac{12}{150}$
- $P(\text{class})$  – marginal probability
  - prior probability of a class
  - ML estimate:  $P(\text{grenade}) = \frac{50}{150}$
- $P(\text{class}|\text{observation})$  – conditional probability
  - posterior probability of a class given an observation
  - **Maximum a-posteriori classifier selects the most likely class**
  - ML estimate:  $P(\text{grenade}|\text{heavy}) = \frac{12}{12+6}$



1	6	12	15	12	2	2	50
4	22	50	14	6	3	1	100
<i>lightest</i> 0.0 - 0.1	<i>lighter</i> 0.1 - 0.2	<i>light</i> 0.2 - 0.3	<i>middle</i> 0.3 - 0.4	<i>heavy</i> 0.4 - 0.5	<i>heavier</i> 0.5 - 0.6	<i>heaviest</i> 0.6 - 0.7	[kg]

# Basic rules of probability theory – I.

Sum rule:

$$P(x) = \sum_y P(x, y)$$

Product rule:

$$P(x, y) = P(x|y)P(y) = P(y|x)P(x)$$

Bayes rule:

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)}$$

# Basic rules of probability theory – II.

- Sum rule:

$$P(\text{heavy}) = P(\text{grenade}, \text{heavy}) + P(\text{apple}, \text{heavy}) = \frac{12}{150} + \frac{6}{150} = \frac{18}{150}$$

$$P(\text{grenade}) = \sum_x P(\text{grenade}, x) = \frac{50}{150}$$

- Product rule:

$$P(\text{grenade}, \text{heavy}) = P(\text{grenade}|\text{heavy})P(\text{heavy}) = \frac{12}{18} \frac{18}{150} = \frac{12}{150}$$

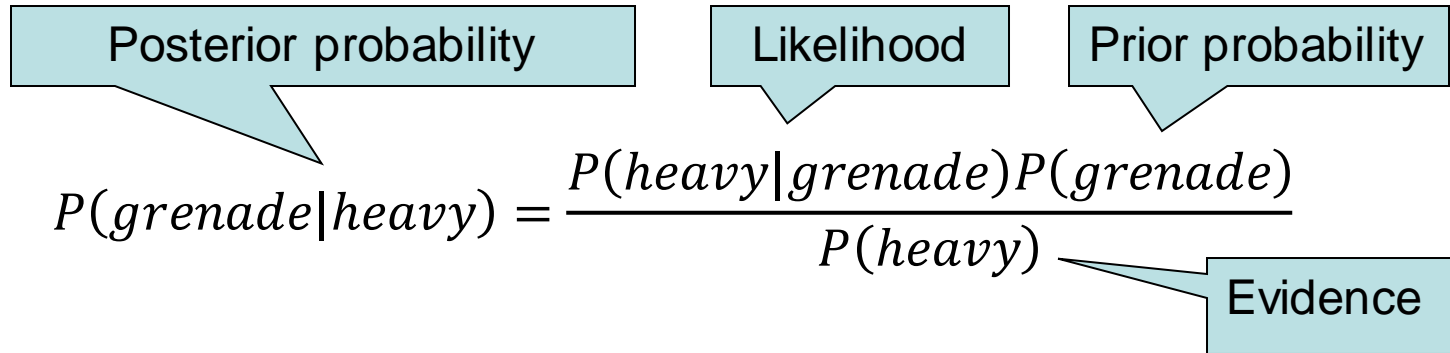
$$P(\text{grenade}, \text{heavy}) = P(\text{heavy}|\text{grenade})P(\text{grenade}) = \frac{12}{50} \frac{50}{150} = \frac{12}{150}$$



$\frac{1}{150}$	$\frac{6}{150}$	$\frac{12}{150}$	$\frac{15}{150}$	$\frac{12}{150}$	$\frac{2}{150}$	$\frac{2}{150}$	$\frac{50}{150}$
$\frac{4}{150}$	$\frac{22}{150}$	$\frac{50}{150}$	$\frac{14}{150}$	$\frac{6}{150}$	$\frac{3}{150}$	$\frac{1}{150}$	$\frac{100}{150}$
<i>lightest</i> 0.0 - 0.1	<i>lighter</i> 0.1 - 0.2	<i>light</i> 0.2 - 0.3	<i>middle</i> 0.3 - 0.4	<i>heavy</i> 0.4 - 0.5	<i>heavier</i> 0.5 - 0.6	<i>heaviest</i> 0.6 - 0.7	[kg]

# Basic rules of probability theory – III.

- Bayes rule:



The diagram shows the Bayes' rule formula with labels in light blue boxes. A box labeled 'Posterior probability' points to the left side of the equation. A box labeled 'Likelihood' points to the numerator. A box labeled 'Prior probability' points to the denominator. A box labeled 'Evidence' points to the denominator.

$$P(\text{grenade}|\text{heavy}) = \frac{P(\text{heavy}|\text{grenade})P(\text{grenade})}{P(\text{heavy})}$$

- The evidence can be evaluated using the sum and product rules in terms of likelihoods and priors:

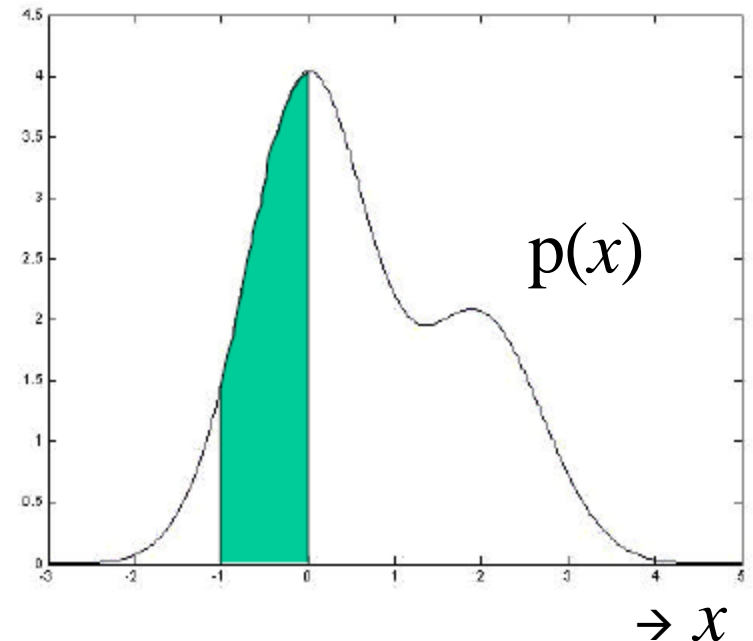
$$P(\text{heavy}) = P(\text{heavy}|\text{grenade})P(\text{grenade}) + P(\text{heavy}|\text{apple})P(\text{apple})$$

- Bayes rule for calculating the class posterior may not seem very useful now, but it will be useful in case continuous valued observations.

# Continuous random variables

- $P(x)$  –probability
- $p(x)$  –probability density function

$$P(x \in (a, b)) = \int_a^b p(x) dx$$

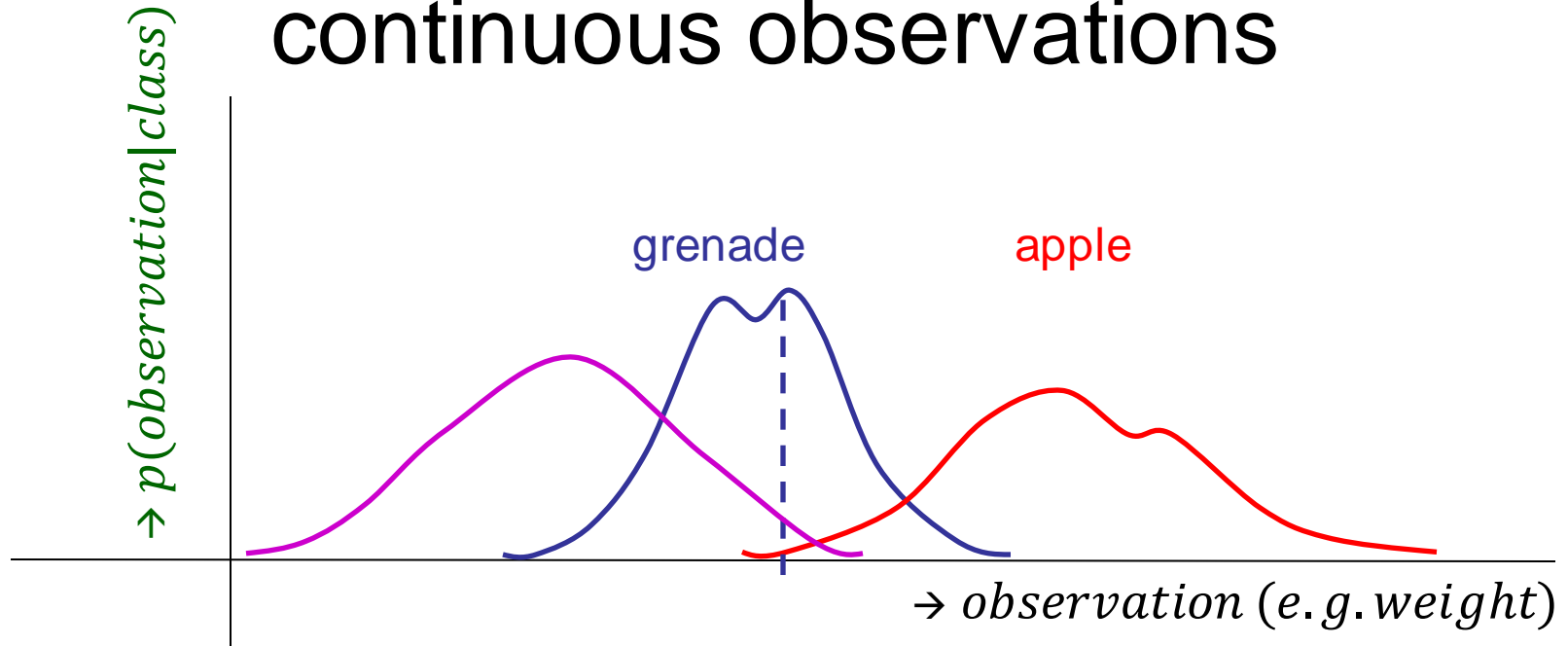


Sum rule:

$$p(x) = \int p(x, y) dy$$



# Classification - Generative model for continuous observations



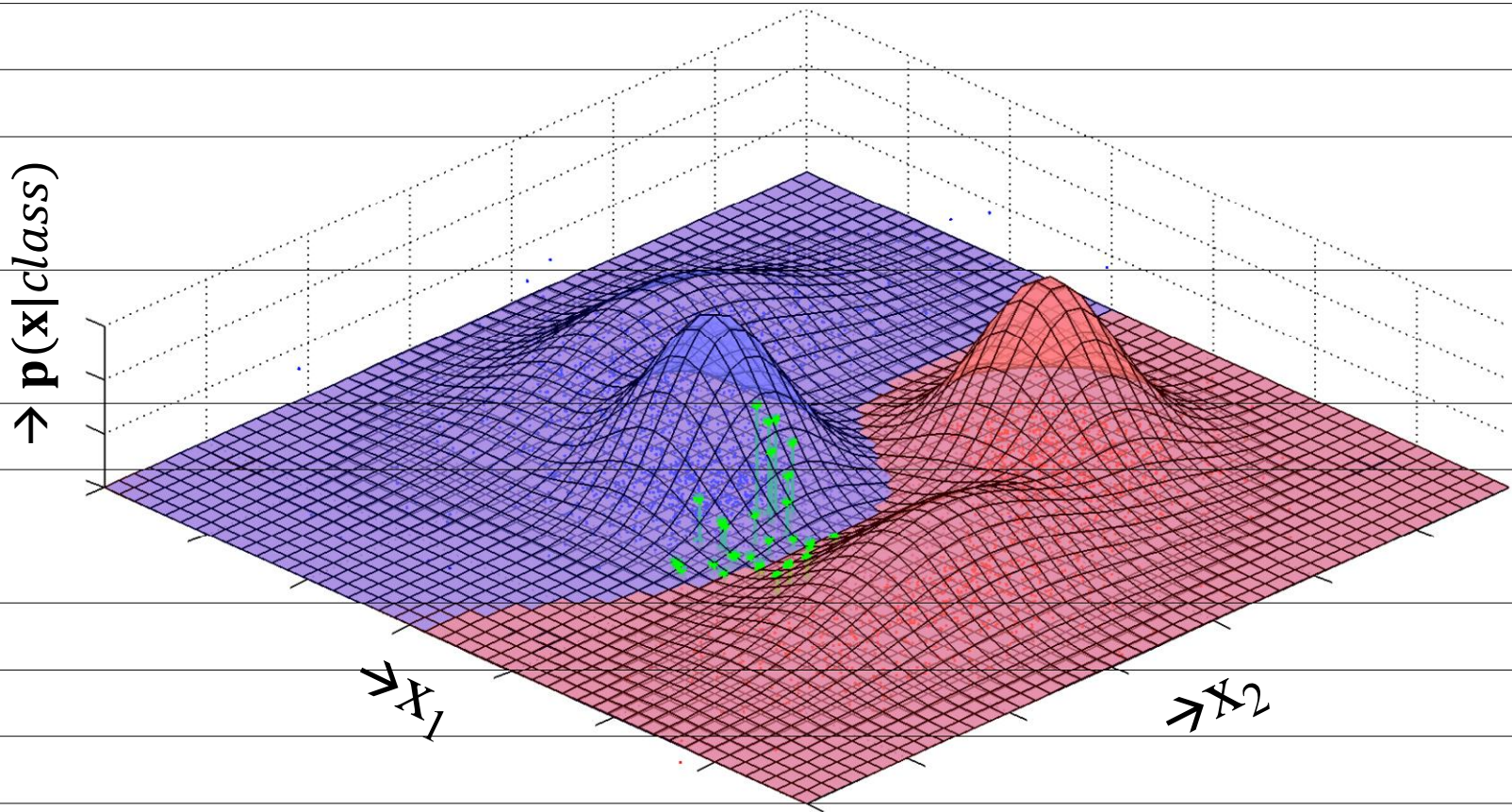
- Maximum a-posteriori classification rule says: “select the more likely class”

$$P(\text{class}|\text{observation}) = \frac{p(\text{observation}|\text{class})P(\text{class})}{p(\text{observation})}$$

$$P(\text{observation}) = \sum_{\text{class}} p(\text{observation}|\text{class})P(\text{class})$$

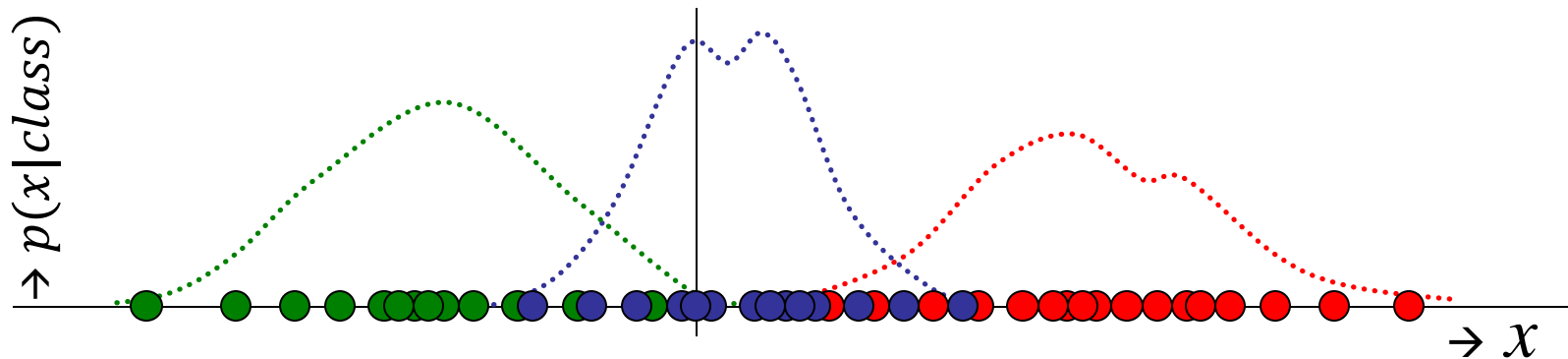
# Multivariate observations

From now, univariate observations will be denoted as  $x$  and multivariate as  $\mathbf{x} = [x_1, x_2, \dots, x_D] = [\textit{weight}, \textit{diameter}, \dots]$



# Estimation of parameters

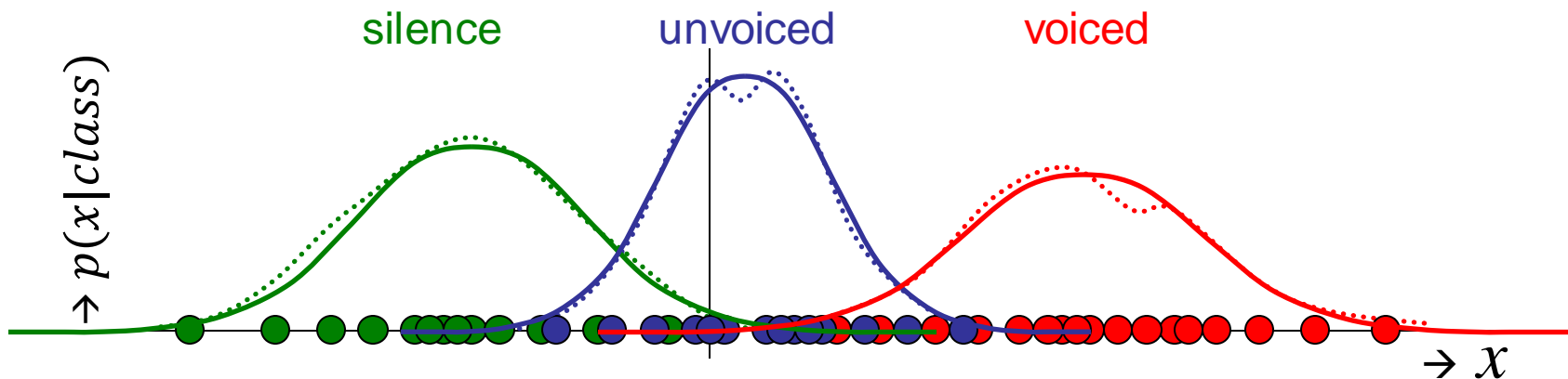
- Usually we do not know the true distributions  $p(x|class)$



# Estimation of parameters

... we only see some training examples.

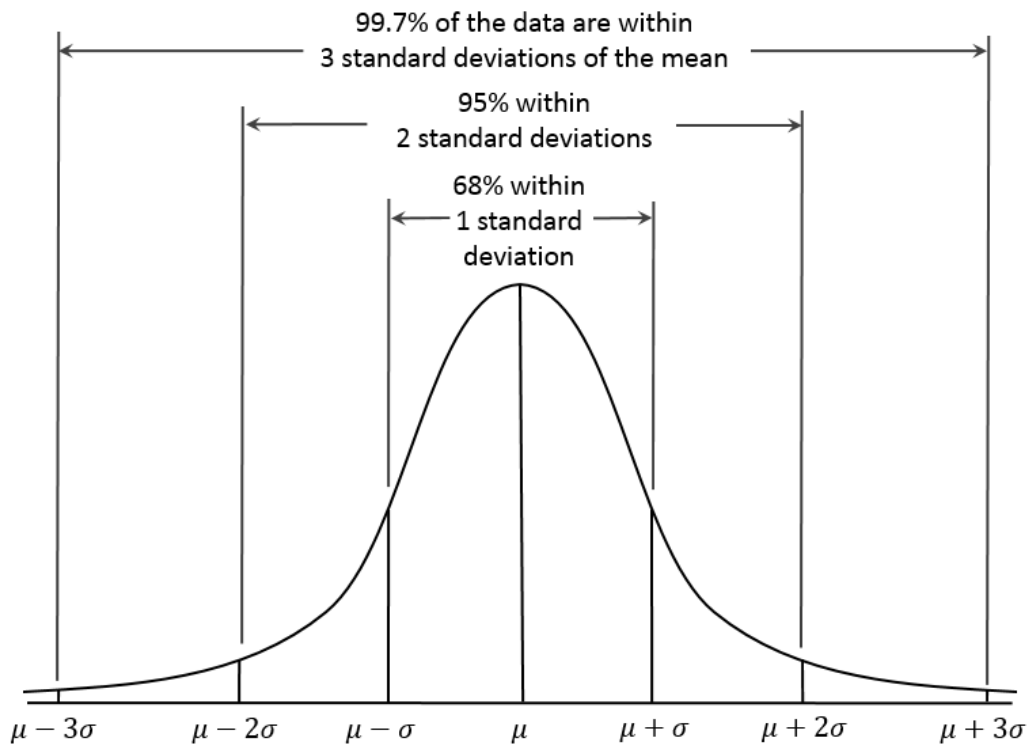
- Let's decide for some parametric model for  $p(x|class)$  (e.g. Gaussian distribution) and estimate its parameters from the data.



- From now, let's forget about classes. We will concentrate just on estimating probability density functions (e.g. one for each class).

# Gaussian distribution (univariate)

$$p(x) = \mathcal{N}(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$



**ML estimates of parameters**

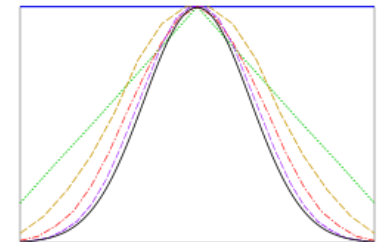
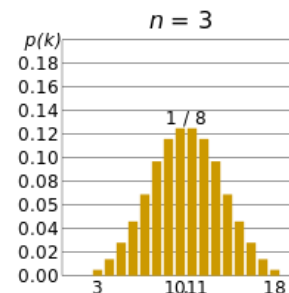
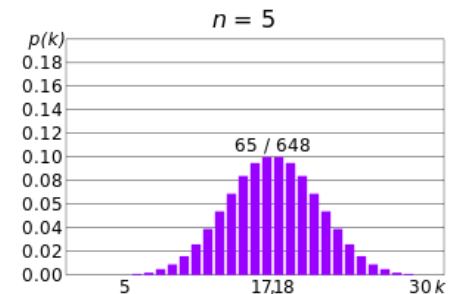
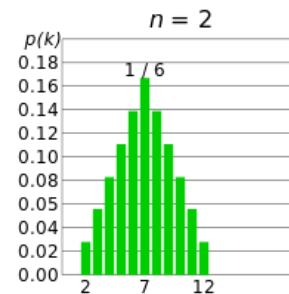
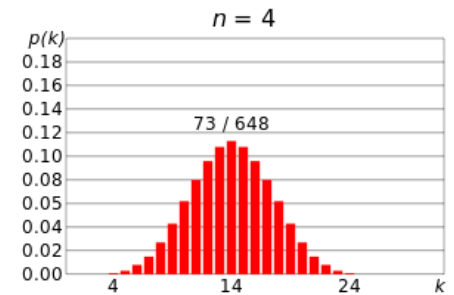
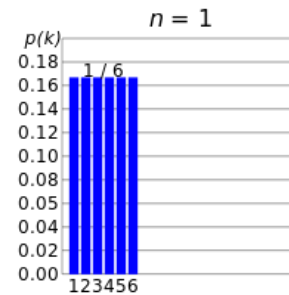
$$\mu = \frac{1}{N} \sum_n x_n$$

$$\sigma^2 = \frac{1}{N} \sum_n (x_n - \mu)^2$$

# Why Gaussian distribution?

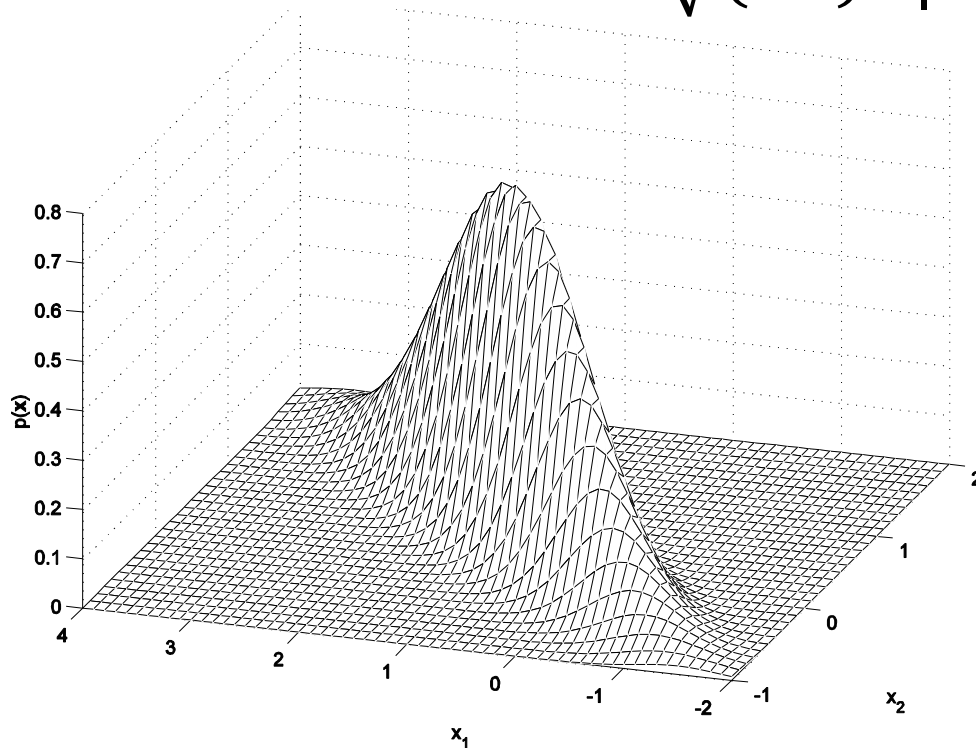
- Naturally occurring
- Central limit theorem: Summing values of many independently generated random variables gives Gaussian distributed observations
- Examples:
  - Summing outcome of N dices
  - Galton's board

<https://www.youtube.com/watch?v=03tx4v0i7MA>



# Gaussian distribution (multivariate)

$$p(x_1, \dots, x_D) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^D |\boldsymbol{\Sigma}|}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})}$$



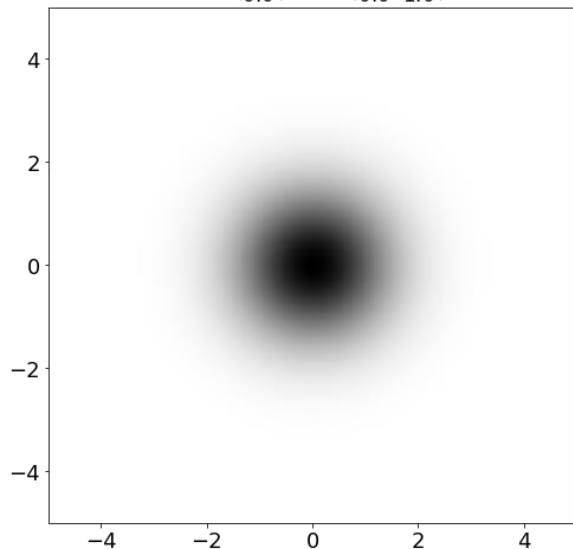
**ML estimates of parameters**

$$\boldsymbol{\mu} = \frac{1}{N} \sum_n \mathbf{x}_n$$

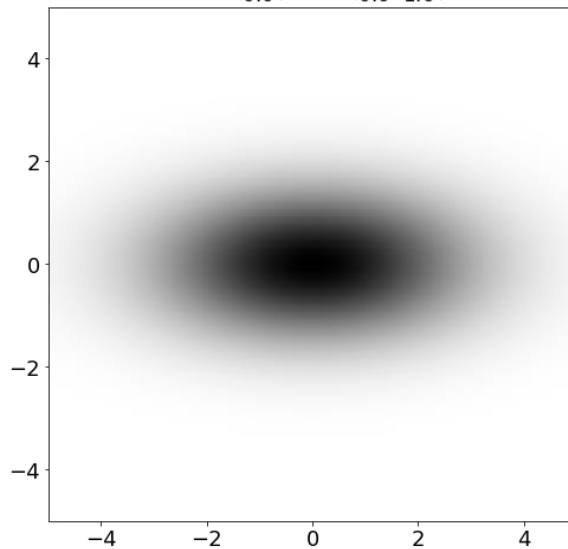
$$\boldsymbol{\Sigma} = \frac{1}{N} \sum_n (\mathbf{x}_n - \boldsymbol{\mu})(\mathbf{x}_n - \boldsymbol{\mu})^T$$

# Examples of Multivariate Gaussians

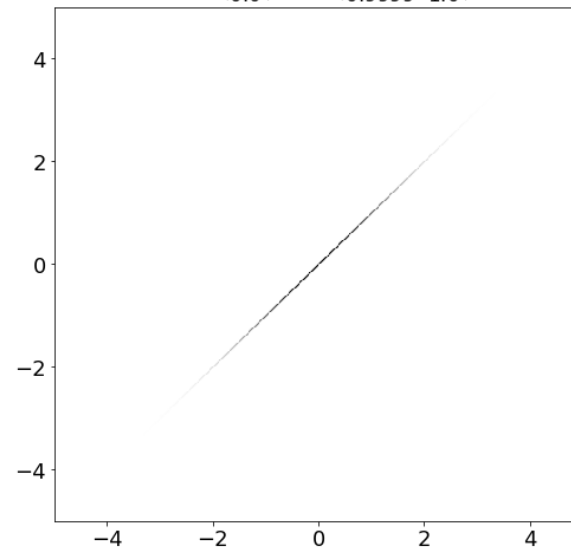
$$\mu = \begin{pmatrix} 0.0 \\ 0.0 \end{pmatrix} \Sigma = \begin{pmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \end{pmatrix}$$



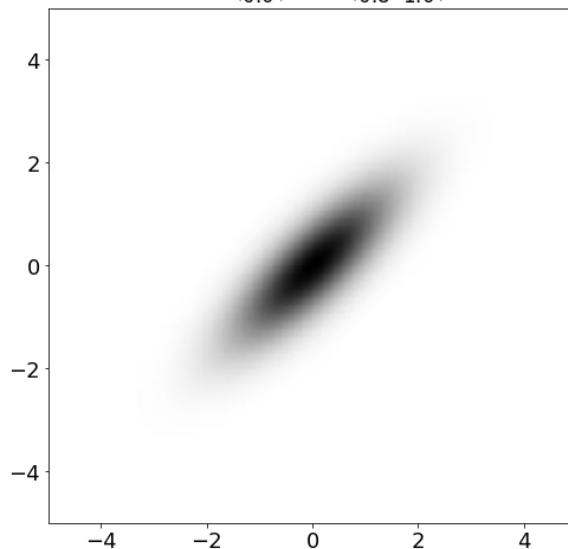
$$\mu = \begin{pmatrix} 0.0 \\ 0.0 \end{pmatrix} \Sigma = \begin{pmatrix} 4.0 & 0.0 \\ 0.0 & 1.0 \end{pmatrix}$$



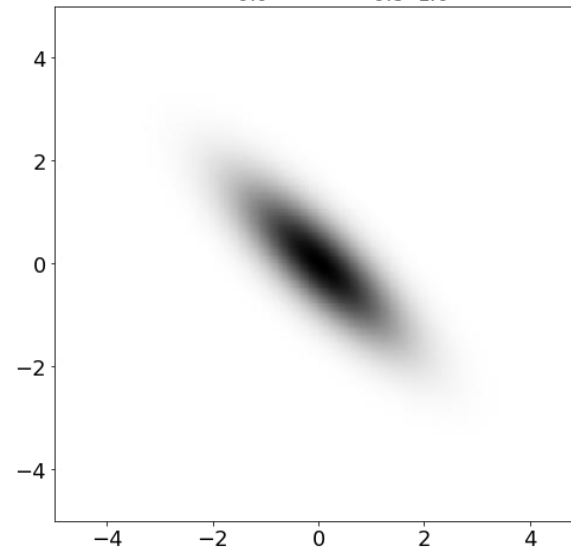
$$\mu = \begin{pmatrix} 0.0 \\ 0.0 \end{pmatrix} \Sigma = \begin{pmatrix} 1.0 & 0.9999 \\ 0.9999 & 1.0 \end{pmatrix}$$



$$\mu = \begin{pmatrix} 0.0 \\ 0.0 \end{pmatrix} \Sigma = \begin{pmatrix} 1.0 & 0.8 \\ 0.8 & 1.0 \end{pmatrix}$$



$$\mu = \begin{pmatrix} 0.0 \\ 0.0 \end{pmatrix} \Sigma = \begin{pmatrix} 1.0 & -0.8 \\ -0.8 & 1.0 \end{pmatrix}$$



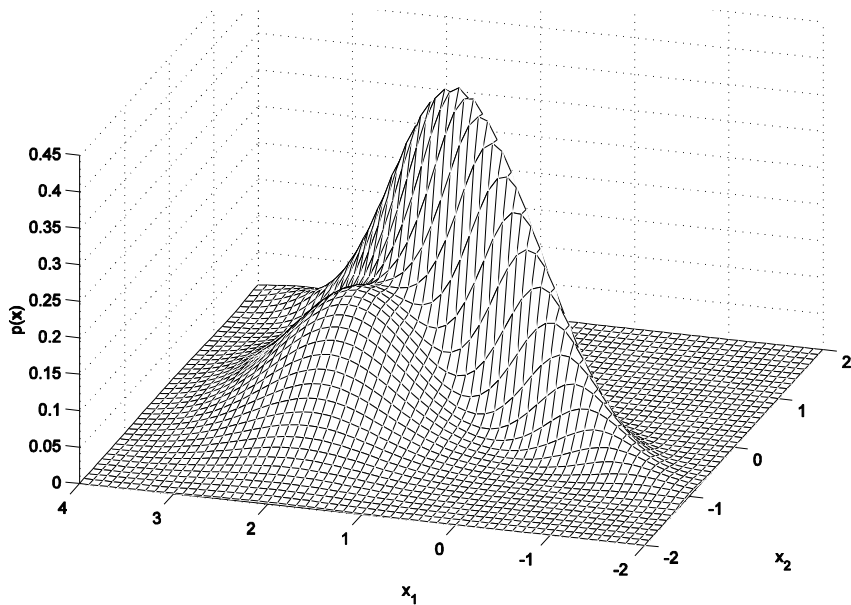


# Multivariate Gaussian Mixture model

$$p(\mathbf{x}) = \sum_c \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c) \pi_c$$

where

$$\sum_c \pi_c = 1$$



- More complicated distributions can be modelled, for example, using weight sum of multiple Gaussian distributions
- However, this is a topic for another class ...

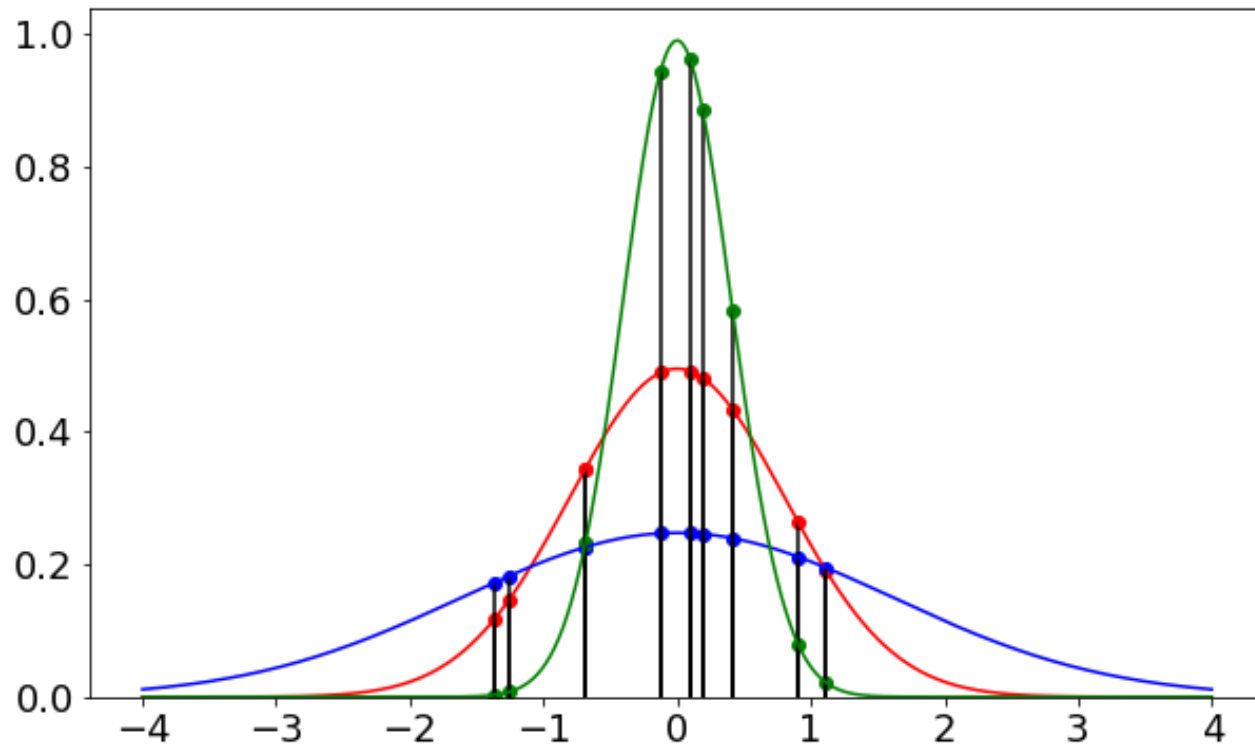
# Maximum likelihood estimation of parameters

- Let's choose a parametric distribution  $p(\mathbf{x}|\boldsymbol{\eta})$  with parameters  $\boldsymbol{\eta}$ 
  - Gaussian distribution with parameters  $\mu, \sigma^2$
- ... and let's have some observed training data  $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N]$ , which we assume to be i.i.d. generated from this distribution.
- We might obtain maximum likelihood estimates of the parameters  $\hat{\boldsymbol{\eta}}^{ML}$  by maximizing the likelihood of the observed data

$$\hat{\boldsymbol{\eta}}^{ML} = \arg \max_{\boldsymbol{\eta}} p(\mathbf{X}|\boldsymbol{\eta}) = \arg \max_{\boldsymbol{\eta}} \prod_{n=1}^N p(\mathbf{x}_n|\boldsymbol{\eta})$$

- Assuming that any values of the parameters are equally likely before observing any data (so-called flat prior), ML estimate gives us the most likely parameters.

# ML estimate for Gaussian



- Black horizontal lines are the training examples
- The red Gaussian corresponds to the **maximum likelihood estimate**
  - Product of heights of the red dots will be larger compared to the blue and green ones

# ML estimate for Gaussian

$$\begin{aligned}\arg \max_{\mu, \sigma^2} p(\mathbf{x}|\mu, \sigma^2) &= \arg \max_{\mu, \sigma^2} \log p(\mathbf{x}|\mu, \sigma^2) = \arg \max_{\mu, \sigma^2} \sum_n \log \mathcal{N}(x_n; \mu, \sigma^2) \\ &= \arg \max_{\mu, \sigma^2} \left( -\frac{1}{2\sigma^2} \sum_n x_n^2 + \frac{\mu}{\sigma^2} \sum_n x_n - N \frac{\mu^2}{2\sigma^2} - \frac{\log(2\pi)}{2} \right)\end{aligned}$$

$$\begin{aligned}\frac{\partial}{\partial \mu} \log p(\mathbf{x}|\mu, \sigma^2) &= \frac{\partial}{\partial \mu} \left( -\frac{1}{2\sigma^2} \sum_n x_n^2 + \frac{\mu}{\sigma^2} \sum_n x_n - N \frac{\mu^2}{2\sigma^2} - \frac{\log(2\pi)}{2} \right) \\ &= \frac{1}{\sigma^2} \left( \sum_n x_n - N\mu \right) = 0 \quad \Rightarrow \quad \hat{\mu}^{ML} = \frac{1}{N} \sum_n x_n\end{aligned}$$

$$\text{and similarly: } \widehat{\sigma^2}^{ML} = \frac{1}{N} \sum_n (x_n - \mu)^2$$

# Categorical distribution



4	22	50	14	6	3	1	100
<i>lightest</i> 0.0 - 0.1	<i>lighter</i> 0.1 - 0.2	<i>light</i> 0.2 - 0.3	<i>middle</i> 0.3 - 0.4	<i>heavy</i> 0.4 - 0.5	<i>heavier</i> 0.5 - 0.6	<i>heaviest</i> 0.6 - 0.7	[kg]

$$p(x|\boldsymbol{\pi}) = \text{Cat}(x|\boldsymbol{\pi}) = \pi_x$$

- Also referred to as **Discrete distribution**
- Special binary case is **Bernoulli distribution**
- $x \in \{\textit{lightest}, \textit{lighter}, \textit{light}, \textit{middle}, \textit{heavy}, \textit{heavier}, \textit{heaviest}\}$   
or  $x$  can be simply the index of a category  $\mathbf{x} \in \{1, 2, \dots, C\}$
- $\boldsymbol{\pi} = [\pi_1, \pi_2, \dots, \pi_C]$  - probabilities of the categories are the parameters
- Likelihood of an observed training set  $\mathbf{x} = [x_1, x_2, \dots, x_N]$

$$P(\mathbf{x}|\boldsymbol{\pi}) = \prod_n \text{Cat}(\mathbf{x}_n|\boldsymbol{\pi}) = \prod_n \pi_{x_n} = \prod_c \pi_c^{m_c}$$

where  $m_c$  is number of observations from category  $c$ .

- (e.g. the numbers from the table)

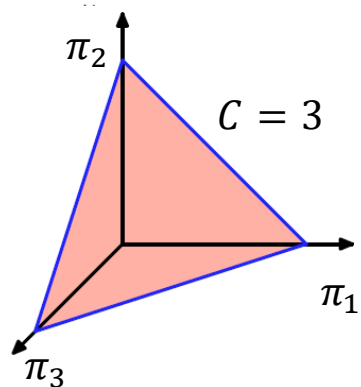
# ML estimate for Categorical

$$\begin{aligned}\arg \max_{\boldsymbol{\pi}} p(\mathbf{x}|\boldsymbol{\pi}) &= \arg \max_{\boldsymbol{\pi}} \log p(\mathbf{x}|\boldsymbol{\pi}) = \arg \max_{\boldsymbol{\pi}} \log \prod_{n=1}^N \text{Cat}(x_n|\boldsymbol{\pi}) \\ &= \arg \max_{\boldsymbol{\pi}} \log \prod_c \pi_c^{m_c} = \arg \max_{\boldsymbol{\pi}} \sum_c m_c \log \pi_c\end{aligned}$$

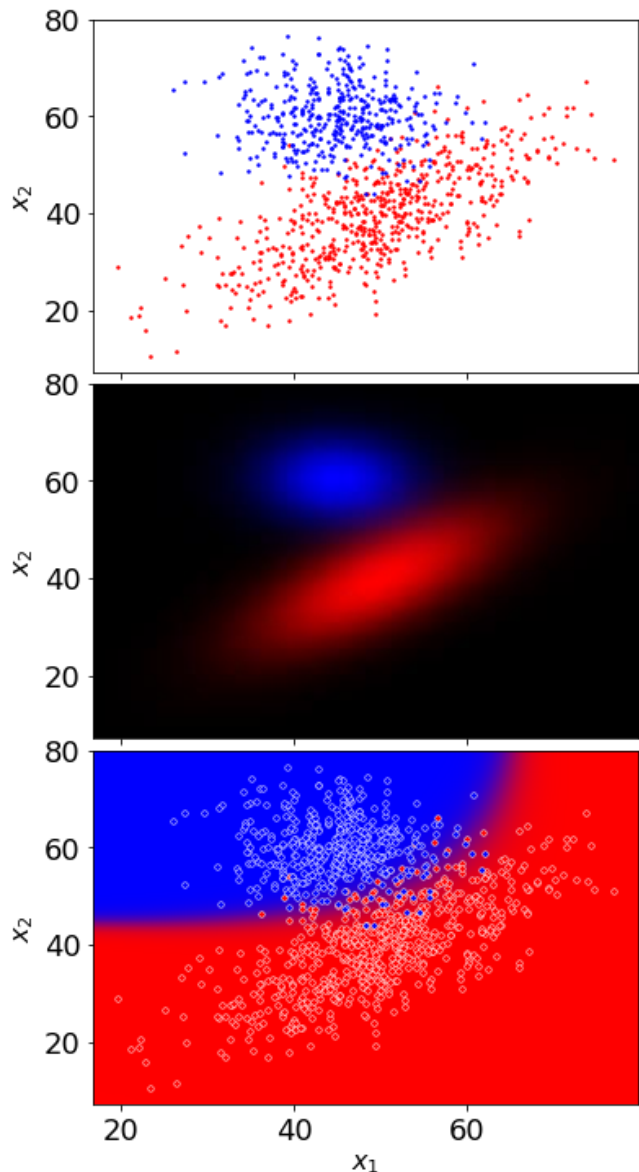
We need to use Lagrange multiplier  $\lambda$  to enforce the constraint  $\sum_k \pi_k = 1$

$$\frac{\partial}{\partial \pi_c} \log p(\mathbf{x}|\boldsymbol{\pi}) = \frac{\partial}{\partial \pi_c} \left( \sum_k m_k \log \pi_k - \lambda \left( \sum_k \pi_k - 1 \right) \right) = \frac{m_c}{\pi_c} - \lambda = 0$$

$$\Rightarrow \pi_c = \frac{m_c}{\lambda} = \frac{m_c}{N}$$



# Gaussian classifier – 2D observation



- Class priors can be ML estimated as the proportions of the example counts

$$P(c) = \frac{N_c}{\sum_k N_k} \quad P(\text{blue}) = \frac{400}{400 + 600}$$

- Probability density function for each class is assumed to be 2D Gaussian

$$p(\mathbf{x}|c) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$$

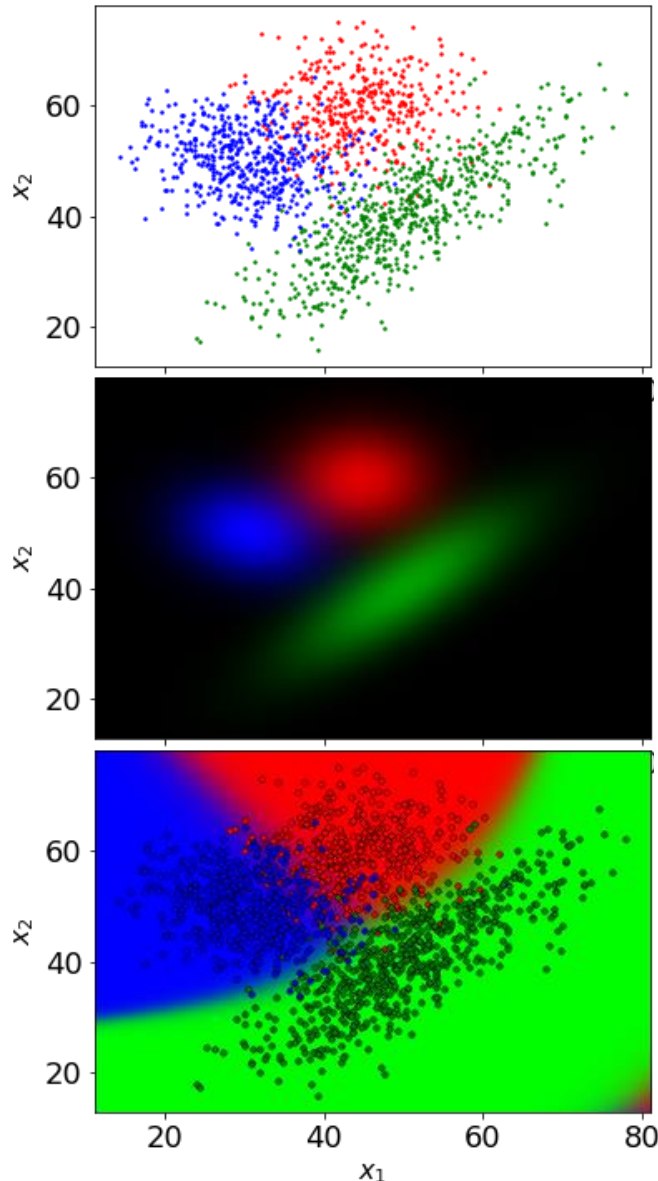
and its parameters ML estimated as

$$\boldsymbol{\mu}_c = \frac{1}{N_c} \sum_{n=1}^{N_c} \mathbf{x}_{cn} \quad \boldsymbol{\Sigma}_c = \frac{1}{N_c} \sum_{n=1}^{N_c} (\mathbf{x}_{cn} - \boldsymbol{\mu}_c)(\mathbf{x}_{cn} - \boldsymbol{\mu}_c)^T$$

- Class posterior probability for new observations is obtained from the prior and class pdf-s using Bayes rule:

$$P(c|\mathbf{x}) = \frac{p(\mathbf{x}|c)P(c)}{\sum_k p(\mathbf{x}|k)P(k)}$$

# Gaussian classifier – more classes



- Class priors can be ML estimated as the proportions of the example counts

$$P(c) = \frac{N_c}{\sum_k N_k} \quad P(\text{blue}) = \frac{400}{400 + 600}$$

- Probability density function for each class is assumed to be 2D Gaussian

$$p(\mathbf{x}|c) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$$

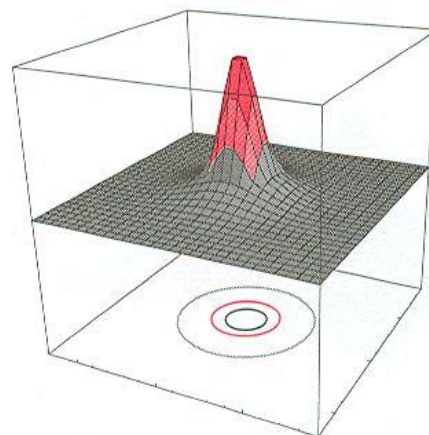
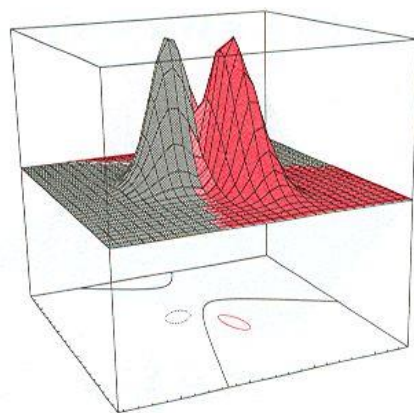
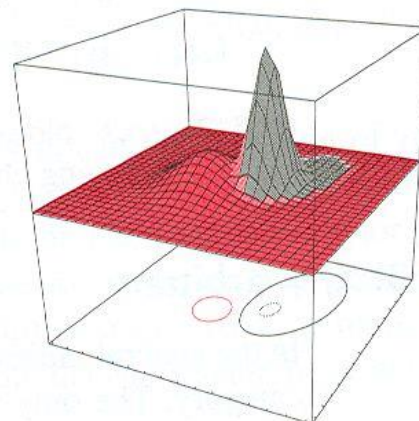
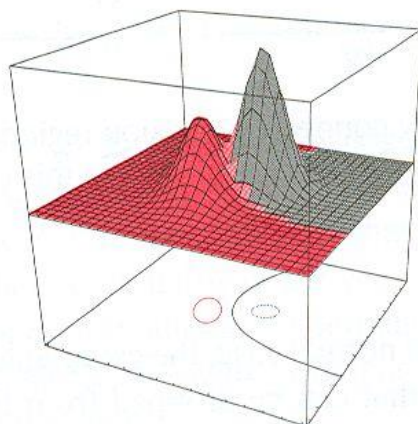
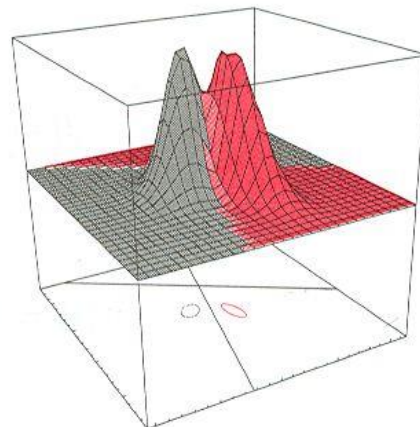
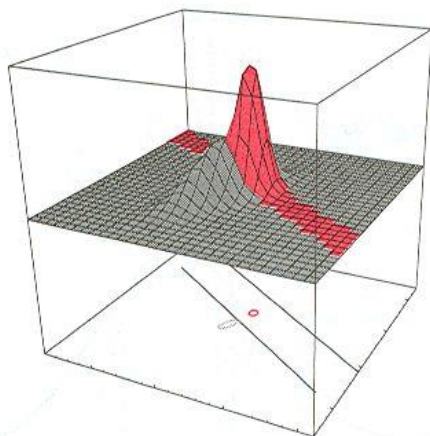
and its parameters ML estimated as

$$\boldsymbol{\mu}_c = \frac{1}{N_c} \sum_{n=1}^{N_c} \mathbf{x}_{cn} \quad \boldsymbol{\Sigma}_c = \frac{1}{N_c} \sum_{n=1}^{N_c} (\mathbf{x}_{cn} - \boldsymbol{\mu}_c)(\mathbf{x}_{cn} - \boldsymbol{\mu}_c)^T$$

- Class posterior probability for new observations is obtained from the prior and class pdf-s using Bayes rule:

$$P(c|\mathbf{x}) = \frac{p(\mathbf{x}|c)P(c)}{\sum_k p(\mathbf{x}|k)P(k)}$$





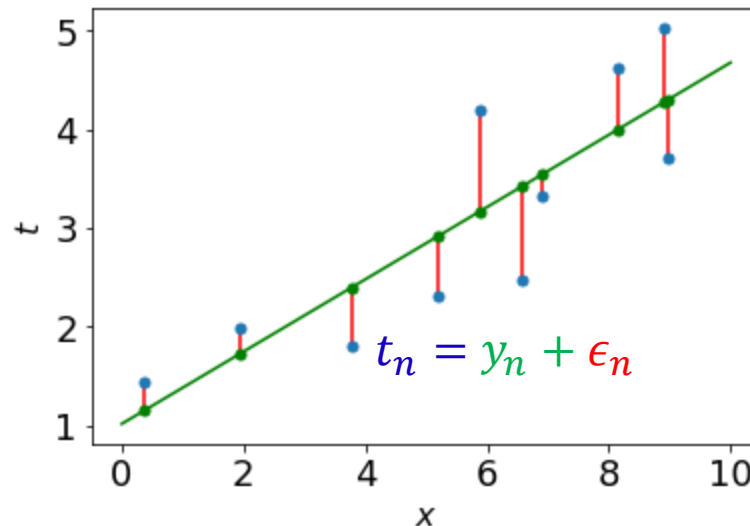
# Linear Regression – I.

- The task is to learn (parameters  $w_0$  and  $w_1$  of) a linear function

$$y = f(x) = w_1 x + w_0$$

from training examples (pairs of inputs  $x_n$  and desired outputs  $t_n$  represented by the blue dots).

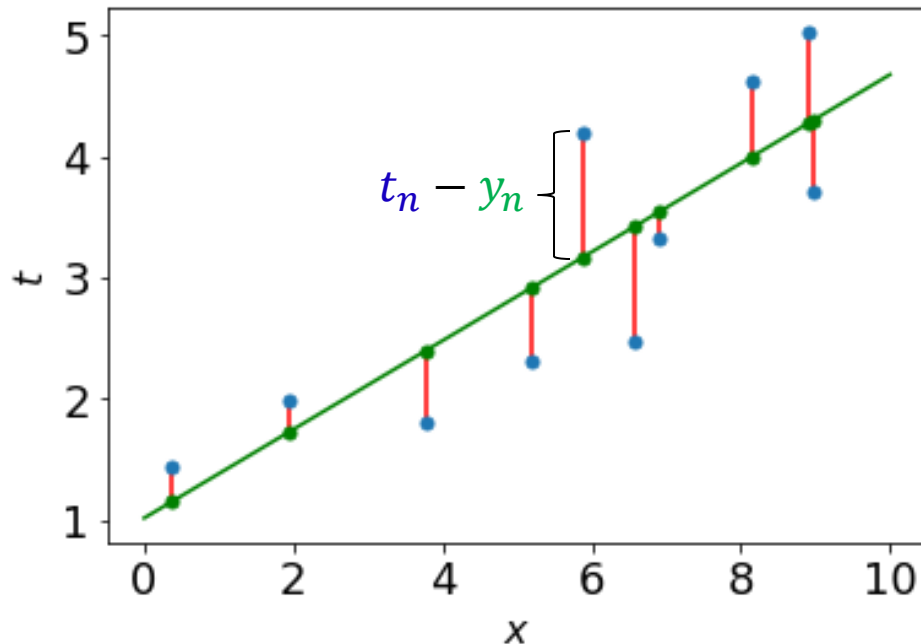
- We assume that there is a linear trend in the data, which we are trying to learn, but expect some random (Gaussian) noise to be added to the outputs (i.e.  $t = y + \epsilon$ , where  $\epsilon$  is Gaussian random variable).



# Linear Regression – II.

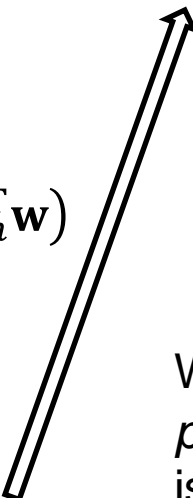
- For “simplicity”, we rewrite  $y = w_1 x + w_0 = \hat{\mathbf{x}}^T \mathbf{w}$  where  $\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$  and  $\hat{\mathbf{x}} = \begin{bmatrix} 1 \\ x \end{bmatrix}$
- We search for parameter  $w_0$  and  $w_1$  that minimizes sum-of-squares error objective function:

$$E(w_0, w_1) = \frac{1}{2} \sum_{n=1}^N (t_n - y_n)^2 = \frac{1}{2} \sum_{n=1}^N (t_n - \hat{\mathbf{x}}_n^T \mathbf{w})^2$$



# Linear Regression – III.

- We can minimize the objective by setting its gradient equal to zero and solving for the parameters  $w_0$  and  $w_1$

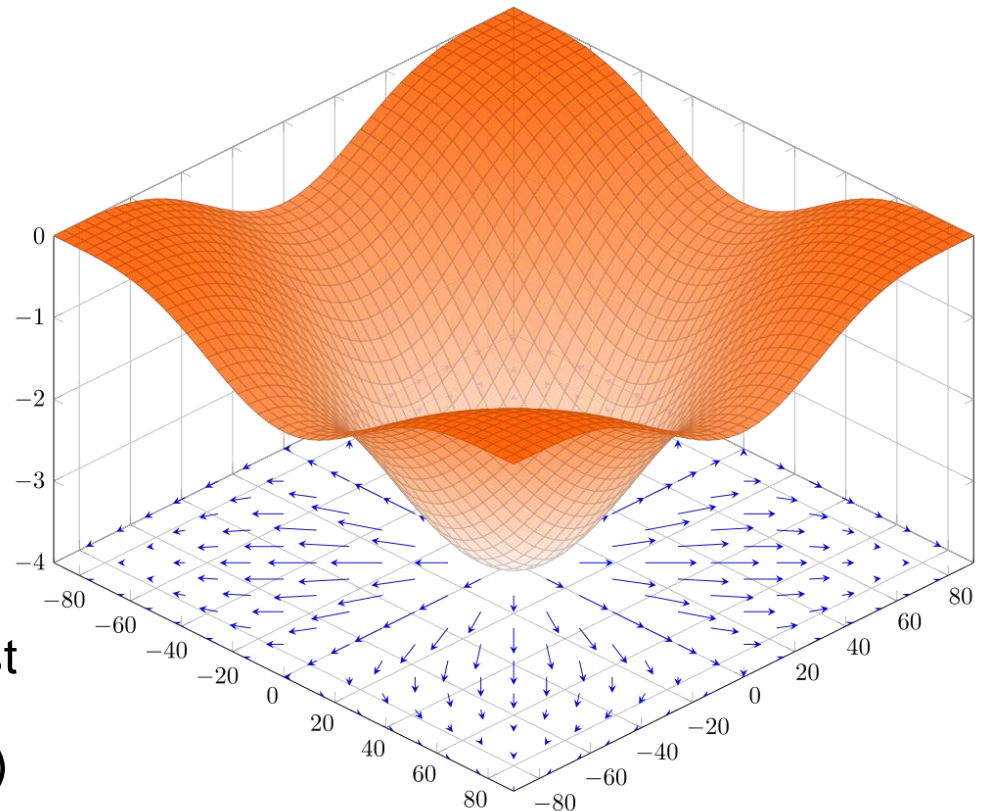
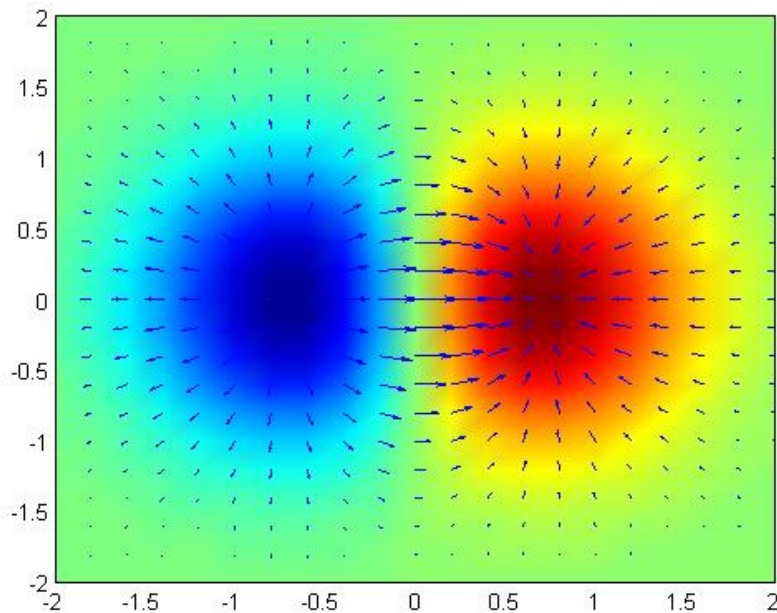
$$\begin{aligned}\frac{\partial}{\partial \mathbf{w}} E(\mathbf{w}) &= \frac{\partial}{\partial \mathbf{w}} \frac{1}{2} \sum_{n=1}^N (t_n - \hat{\mathbf{x}}_n^T \mathbf{w})^2 \\&= \frac{1}{2} \sum_{n=1}^N \frac{\partial}{\partial \mathbf{w}} (t_n - \hat{\mathbf{x}}_n^T \mathbf{w})^2 \\&= \sum_{n=1}^N (t_n - \hat{\mathbf{x}}_n^T \mathbf{w}) \frac{\partial}{\partial \mathbf{w}} (t_n - \hat{\mathbf{x}}_n^T \mathbf{w}) \\&= \sum_{n=1}^N (t_n - \hat{\mathbf{x}}_n^T \mathbf{w}) \hat{\mathbf{x}}_n \\&= \sum_{n=1}^N t_n \hat{\mathbf{x}}_n - \sum_{n=1}^N \hat{\mathbf{x}}_n \hat{\mathbf{x}}_n^T \mathbf{w} = 0\end{aligned}$$

$$\begin{aligned}\left( \sum_{n=1}^N \hat{\mathbf{x}}_n \hat{\mathbf{x}}_n^T \right) \mathbf{w}^* &= \sum_{n=1}^N t_n \hat{\mathbf{x}}_n \\ \mathbf{w}^* &= \left( \sum_{n=1}^N \hat{\mathbf{x}}_n \hat{\mathbf{x}}_n^T \right)^{-1} \sum_{n=1}^N t_n \hat{\mathbf{x}}_n \\&= (\hat{\mathbf{X}} \hat{\mathbf{X}}^T)^{-1} \hat{\mathbf{X}} \mathbf{t} \\&= \hat{\mathbf{X}}^\dagger \mathbf{t}\end{aligned}$$

Where  $\dagger$  means *Moore-Penrose pseudo-inverse*,  $\hat{\mathbf{X}} = [\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2, \dots, \hat{\mathbf{x}}_N]$  is matrix of all extended inputs and  $\mathbf{t} = [t_1, t_2, \dots, t_N]^T$  is vector of all desired outputs

# Gradient

$$\nabla E(\mathbf{w}) = \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} = \begin{bmatrix} \frac{\partial E(\mathbf{w})}{\partial w_0} \\ \frac{\partial E(\mathbf{w})}{\partial w_1} \end{bmatrix}$$

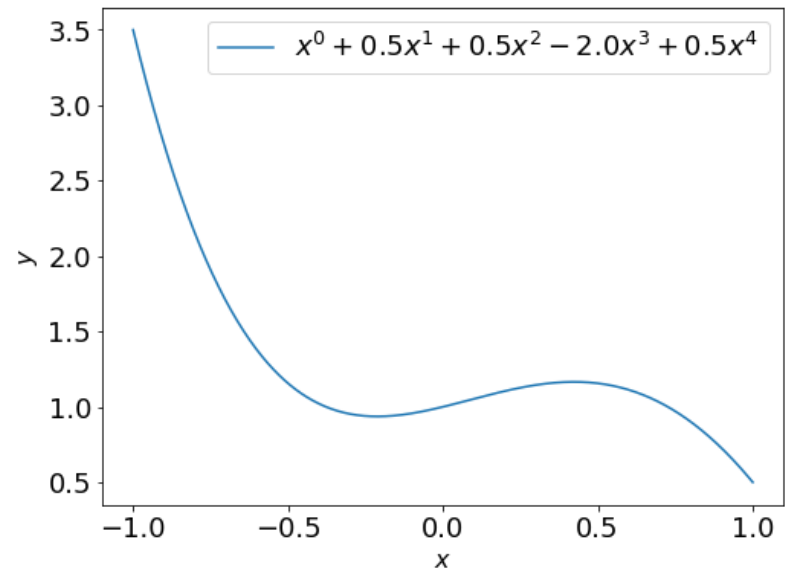
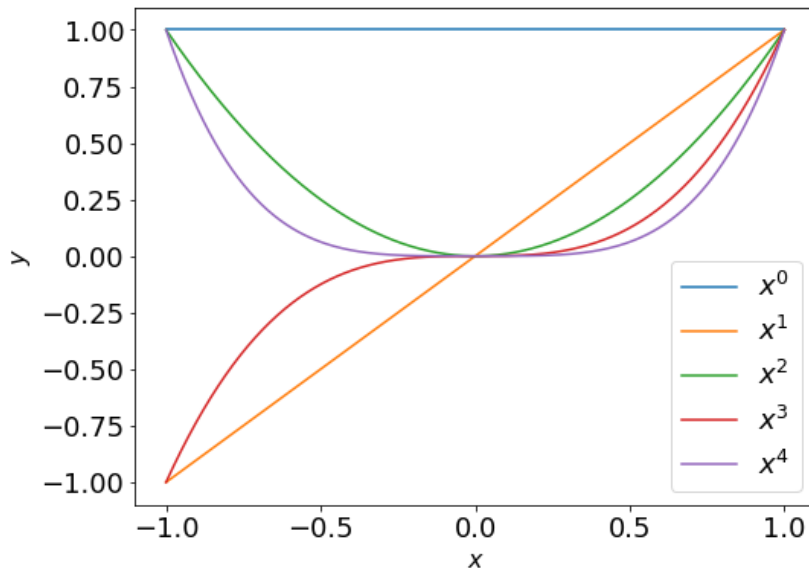
- For each  $\mathbf{w}$ ,  $\nabla E(\mathbf{w})$  is a vector showing the “most uphill” direction and the length of this vector is the “steepness of the slope” in that direction



- For linear regression,  $E(\mathbf{w})$  is just multivariate-quadratic function (i.e. simpler than these examples)

# Learning nonlinear function using Linear Regression

- Any non-linear function can be represented as linear combination of other “simpler” nonlinear functions (e.g. polynomial functions)



- We can use linear regression to learn the combination weights

# Polynomial Regression – I.

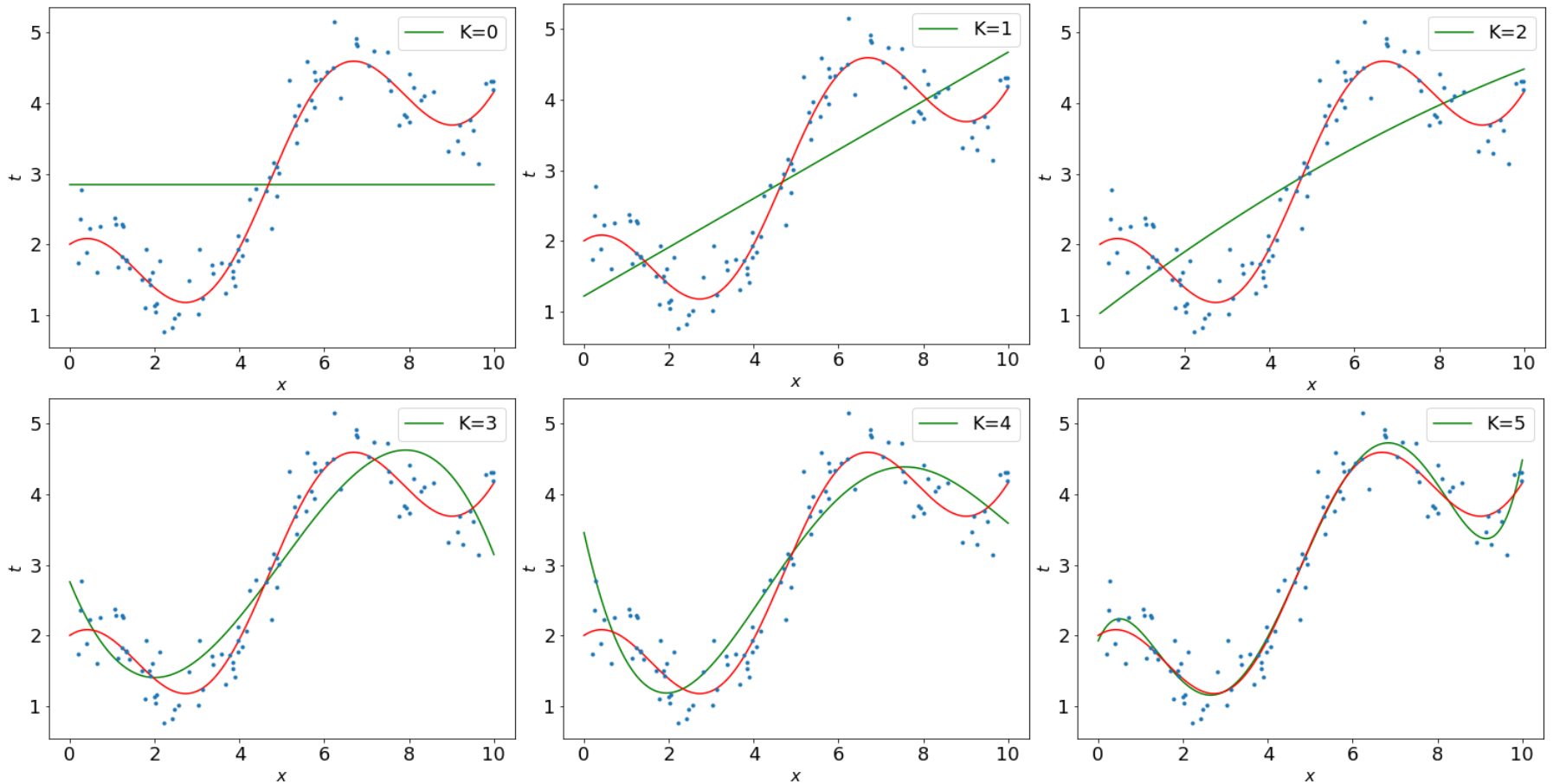
- Let make  $\hat{\mathbf{x}} = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^K \end{bmatrix}$  and  $\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_K \end{bmatrix}$
- Using the same formulation of the problem as for the linear case, we now get polynomial regression as a special case of linear regression

$$y = \hat{\mathbf{x}}^T \mathbf{w} = w_0 + w_1 x + w_2 x^2 + \cdots + w_K x^K$$

- The optimal weights  $\mathbf{w}^*$  can be found using the same formulas as before. In fact, the linear case considered before is just the special case for  $K = 1$
- Polynomial expansion (polynomial base functions) is just one possibility. We can choose to use any other non-linear bases:

$$\hat{\mathbf{x}} = [f_1(x), f_2(x), \dots, f_K(x)]^T$$

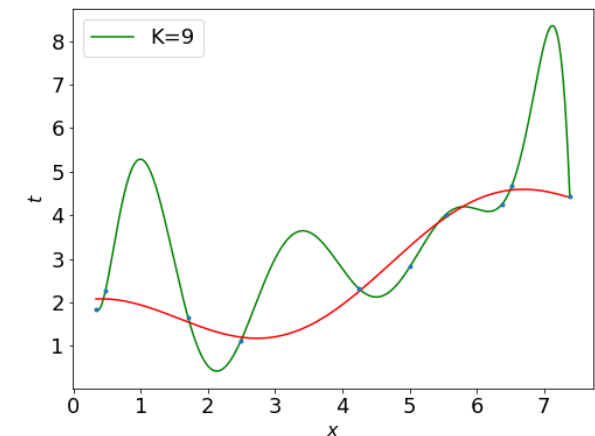
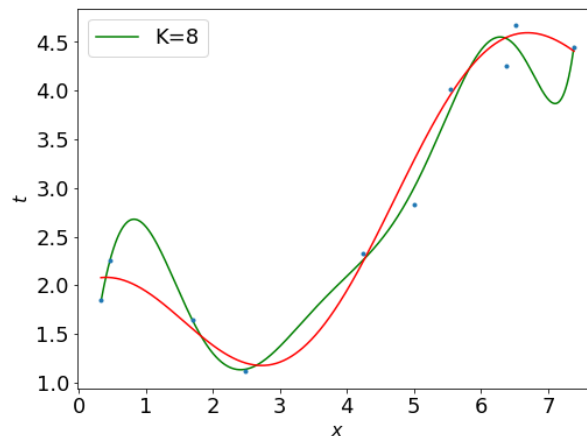
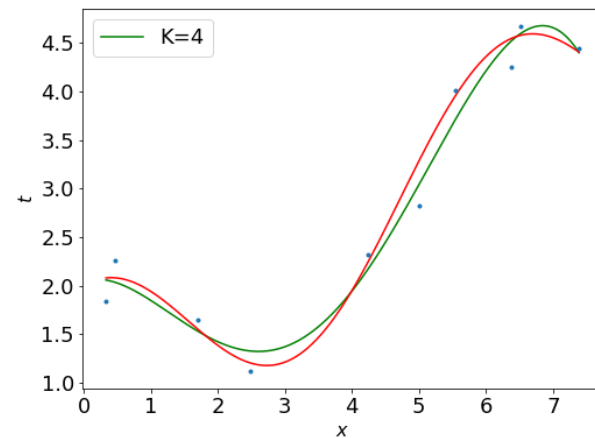
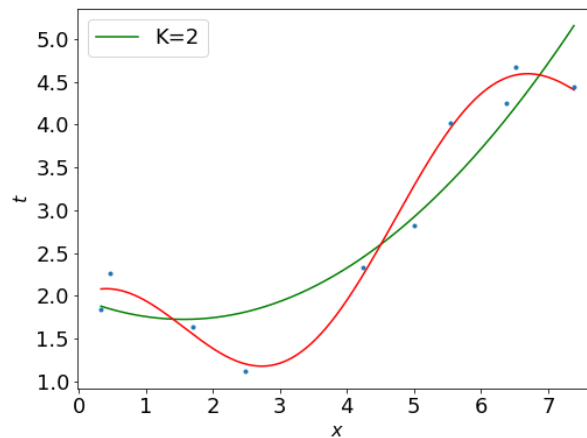
# Polynomial Regression – II.





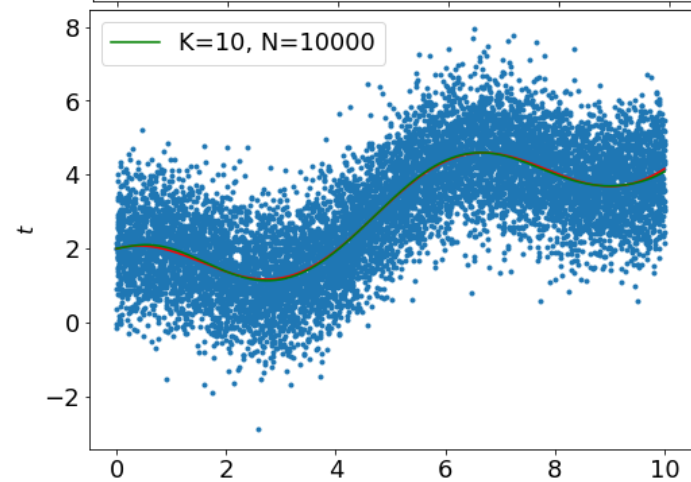
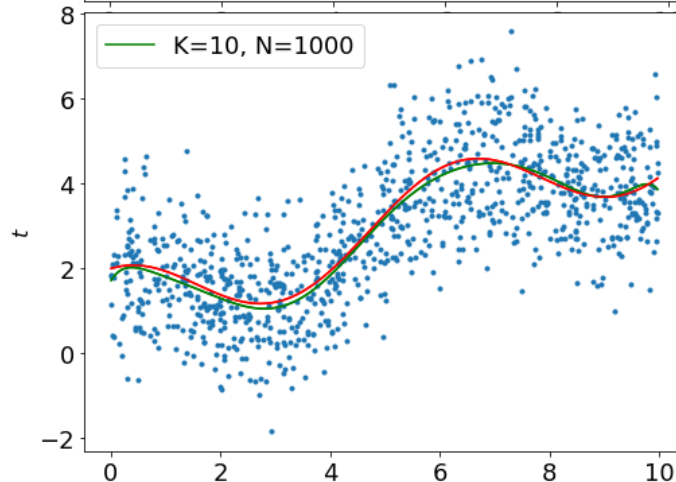
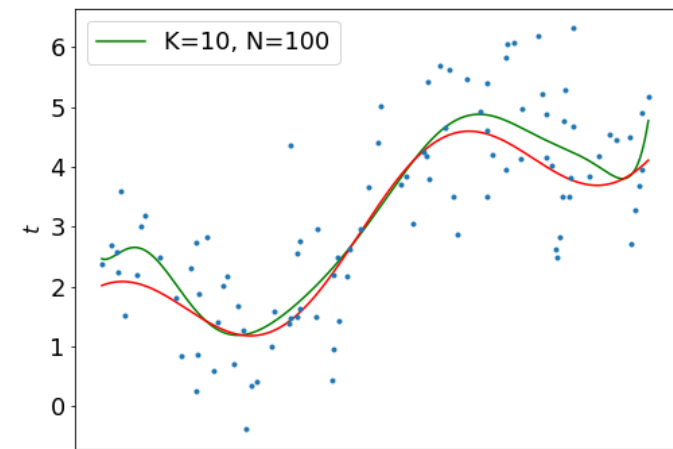
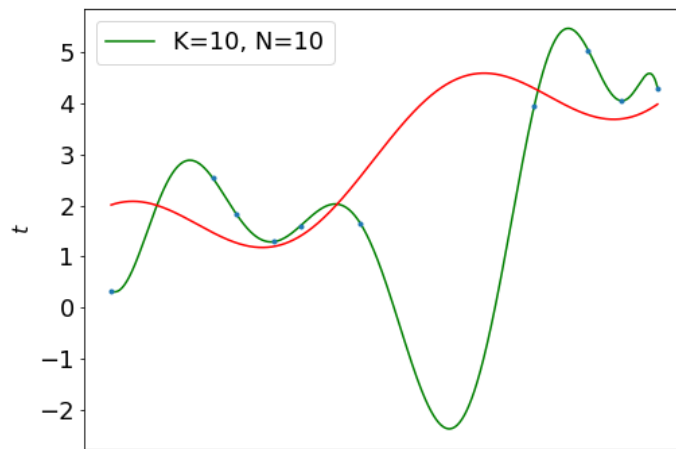
# Polynomial Regression – III.

- The model must have the right complexity otherwise, we underfit or overfit (so-called bias-variance tradeoff)



# Why sum-of-squares error function?

- If the output is only corrupted by Gaussian noise, sum-of-squares error objective tends to learn the correct underlying function
  - as the number of training data increases
  - given the sufficient modelling capacity (e.g. large enough  $K$ )



# Why sum-of-squares error function?

- We learn function

$$y = f(x) = \hat{\mathbf{x}}^T \mathbf{w}$$

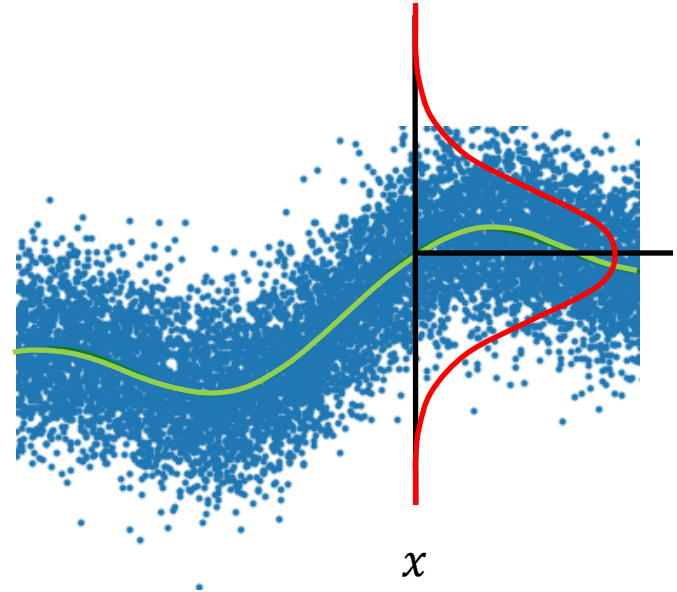
- but assume output distributed as

$$t = y + \epsilon$$

where  $\epsilon$  is Gaussian noise  $\mathcal{N}(\epsilon; 0, \sigma^2)$

- In other words

$$p(t|x) = \mathcal{N}(t; y, \sigma^2) = \mathcal{N}(t; \hat{\mathbf{x}}_n^T \mathbf{w}, \sigma^2)$$



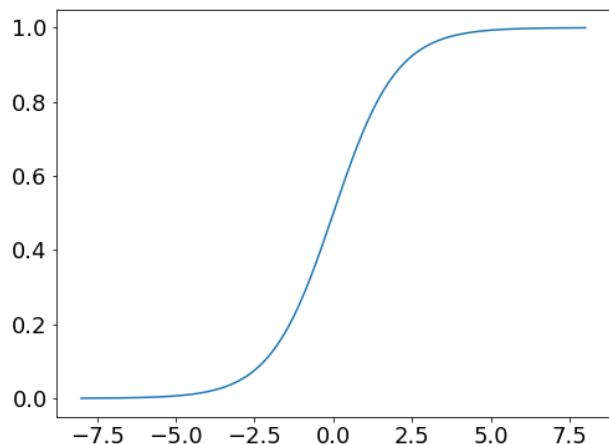
- To obtain maximum likelihood estimates of parameters  $\mathbf{w}$  (and  $\sigma^2$ ) given training examples  $x_n, t_n$ , we need to maximize log likelihood

$$\begin{aligned} \log p(\mathbf{t}|\mathbf{X}) &= \sum_{n=1}^N \log p(t_n|\mathbf{x}_n) = \sum_{n=1}^N \log \mathcal{N}(t_n; \hat{\mathbf{x}}_n^T \mathbf{w}, \sigma^2) \\ &= -\frac{N}{2} \log 2\pi\sigma^2 - \frac{1}{2\sigma^2} \sum_{n=1}^N (t_n - \hat{\mathbf{x}}_n^T \mathbf{w})^2 \end{aligned}$$

which corresponds to **sum-of-squares objective** for learning  $\mathbf{w}$

# Classification – Discriminative approach

- With generative models (e.g. Gaussian classifier)
  - we learn distribution of observations for each class  $p(\mathbf{x}|c)$
  - and class prior probabilities  $p(c)$
  - and use Bayes rule to derive posterior probability of each class  $p(c|\mathbf{x})$
  - we try to precisely model  $p(\mathbf{x}|c)$  even at places far from the decision boundary  
⇒ possible waste of parameter; model that is not compact
- Let's concentrate directly at modeling the posterior  $p(c|\mathbf{x})$ 
  - For now, let's consider only binary classifier (i.e.  $c \in \{0,1\}$ ), where the posterior  $p(c|\mathbf{x})$  is just a function, with output constrained to be in the range of 0 and 1
  - We were able to learn, functions from data using linear regression.
  - Let's use linear regression-like model with the output mapped to the 0 – 1 range using logistic sigmoid function  $\sigma(a)$ :



$$\sigma(a) = \frac{1}{1 + \exp(-a)}$$

$$P(c = 1|\mathbf{x}) = \sigma(\hat{\mathbf{x}}^T \mathbf{w})$$

# Linear logistic regression – I.

- 2-class linear logistic regression models directly posterior probability of class  $c = 1$  as

$$P(c = 1|\mathbf{x}) = \sigma(\hat{\mathbf{x}}^T \mathbf{w})$$

- Probability of the other class  $c = 0$  is

$$P(c = 0|\mathbf{x}) = 1 - P(c = 1|\mathbf{x})$$

- To train the model (i.e. the parameter  $\mathbf{w}$ ), we need training examples
  - $\mathbf{x}_n$ , which we expand as before to  $\hat{\mathbf{x}}_n$  (e.g. using the polynomial basis)
  - $t_n$  encodes the correct class i.e. (has value 1 or 0)
- Sum-of-squares error objective can be used again as the training objective, ...
- but it is better to use Maximum Likelihood approach as we try to predict the probability  $P(c|\mathbf{x})$ 
  - We maximize the same objective as for ML estimate of Categorical distribution, except that the discrete probability is now conditioned on  $\mathbf{x}$ :

$$P(\mathbf{t}|\mathbf{X}) = \prod_n P(t_n|\mathbf{x}_n) = \prod_n \sigma(\hat{\mathbf{x}}_n^T \mathbf{w})^{t_n} (1 - \sigma(\hat{\mathbf{x}}_n^T \mathbf{w}))^{(1-t_n)}$$

# Linear logistic regression – II.

- Instead of maximizing

$$P(\mathbf{t}|\mathbf{X}) = \prod_n P(t_n|\mathbf{x}_n) = \prod_n \sigma(\hat{\mathbf{x}}_n^T \mathbf{w})^{t_n} (1 - \sigma(\hat{\mathbf{x}}_n^T \mathbf{w}))^{(1-t_n)}$$

the machine learning people prefer to minimize equivalent “error”

$$E(\mathbf{w}) = -\ln P(\mathbf{t}|\mathbf{X}) = \sum_{n=1}^N t_n \ln \sigma(\hat{\mathbf{x}}_n^T \mathbf{w}) + (1 - t_n) \ln (1 - \sigma(\hat{\mathbf{x}}_n^T \mathbf{w}))$$

called (binary) **cross-entropy**.

- By taking its derivative w.r.t.  $\mathbf{w}$ , we obtain gradient

$$\nabla E(\mathbf{w}) = \sum_{n=1}^N (\sigma(\hat{\mathbf{x}}_n^T \mathbf{w}) - t_n) \hat{\mathbf{x}}_n$$

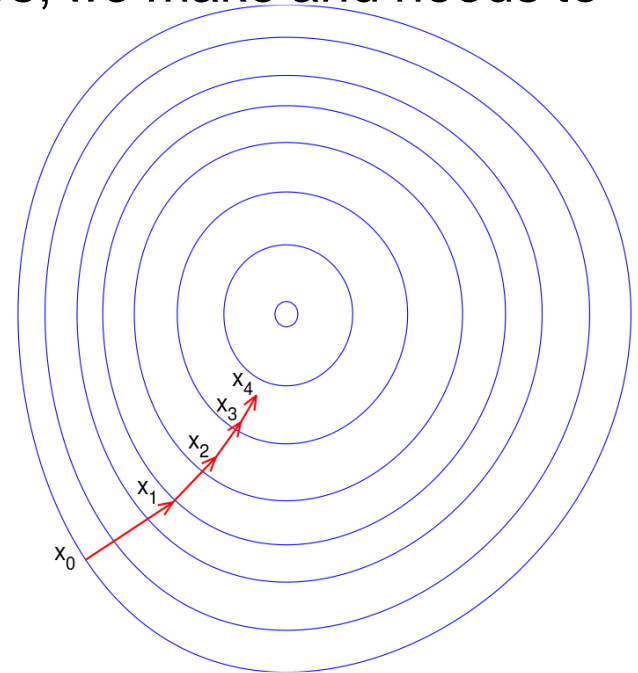
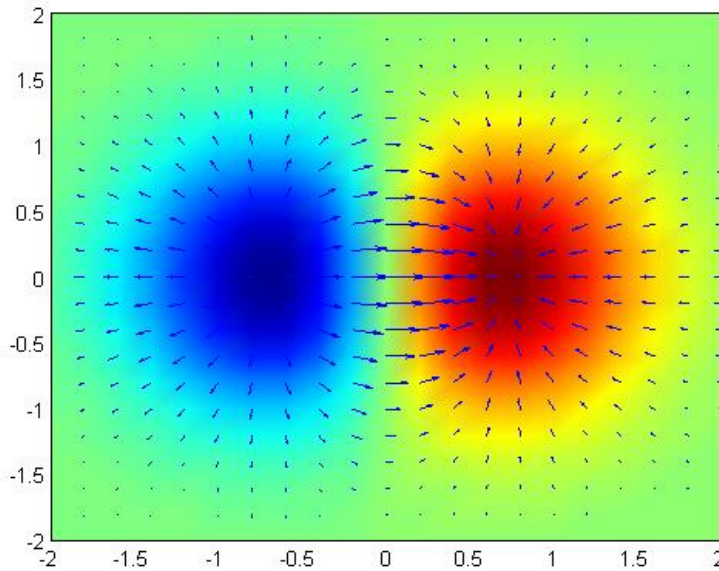
- Unfortunately, this objective does not have closed-form solution as it was in the case of linear regression.
  - We need to resort to numerical optimization (e.g. gradient descent)
  - Quadratic optimization is often better choice, but we do not consider it here

# Gradient descent optimization

- Repeatedly update the parameters  $\mathbf{w}$  by moving in small steps in the direction opposite to the gradient  $\nabla E(\mathbf{w})$  (i.e. downhill) until reaching the minimum of the objective function (i.e.  $\nabla E(\mathbf{w}) = \mathbf{0}$ ).

$$\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} - \eta \nabla E(\mathbf{w}^{\tau})$$

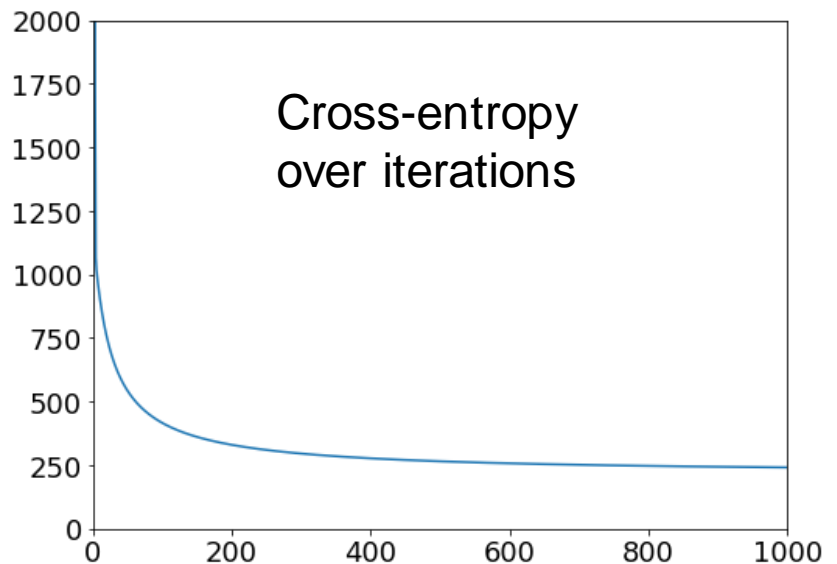
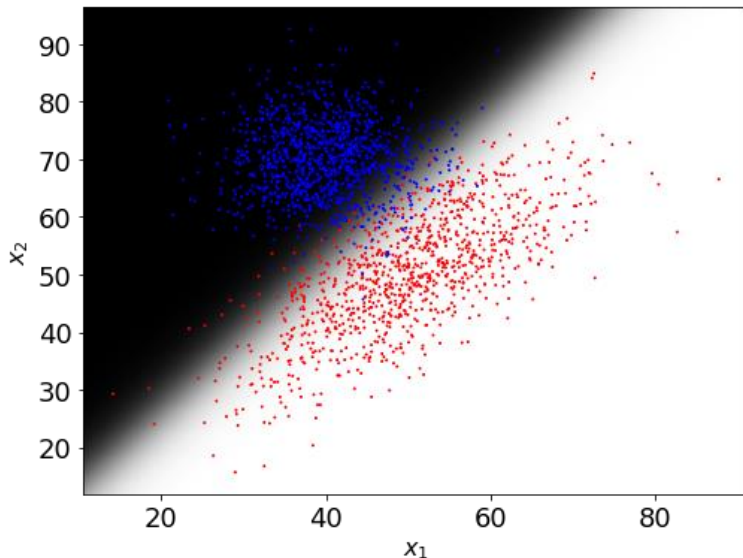
- The *learning rate*  $\eta$  controls how big steps, we make and needs to be tuned for good convergence



# Logistic regression – example

- Classifier trained using linear logistic regression without any polynomial expansion
  - only offers linear decision boundary
  - only 3 parameters to train (Gaussian classifier had 10)

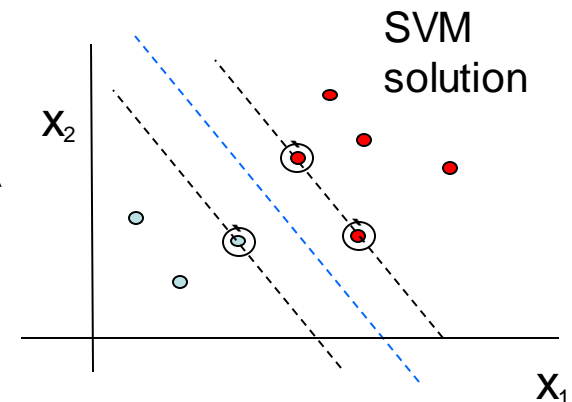
$$P(c = 1|\mathbf{x}) = \sigma(\hat{\mathbf{x}}^T \mathbf{w}) = \sigma(w_1 x_1 + w_2 x_2 + w_0)$$





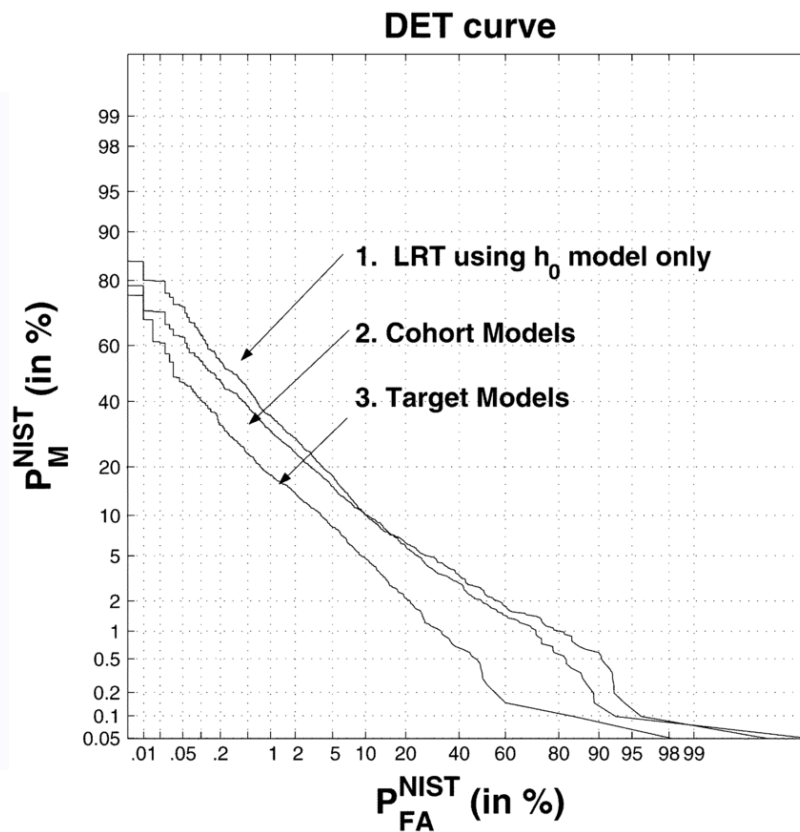
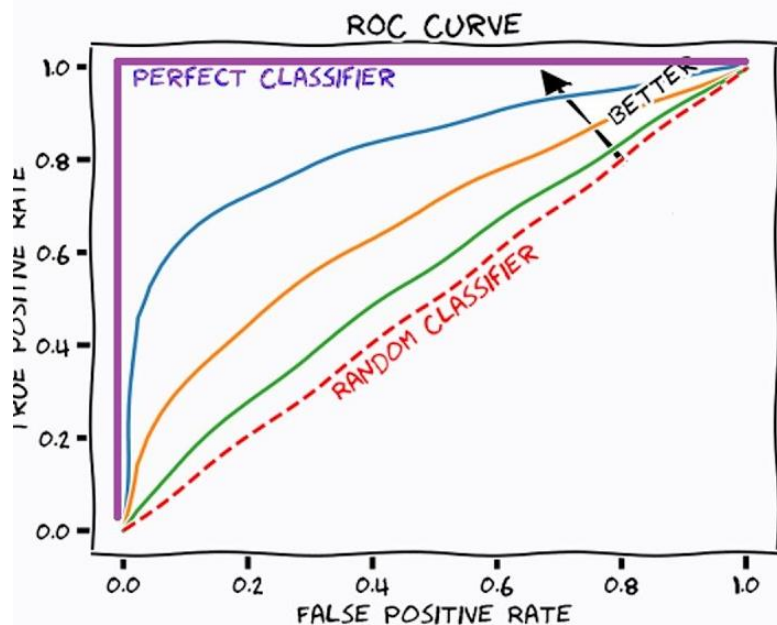
# Discriminative vs Generative

- For generative models we learn  $p(\mathbf{x}|c)$  and  $p(c)$  and use Bayes rule to derive  $p(c|\mathbf{x})$
- Logistic regression is discriminative approach, where we learn to estimate directly  $p(c|\mathbf{x})$
- There are other discriminative training strategies that do not even try to estimated any probability
- For example, Support Vector Machines directly finds decision boundary that maximizes margin between the classes
- Generative models
  - Usually less prone to overfitting with small amount of training data
  - Modular: complex models build out of simple ones
- Discriminative training
  - Less waste of parameters
  - Usually better performance with enough data
  - Allows end-to-end solution



# Evaluating unbalanced classes

- 99.99... % of authentication attempts are legitimate
- 99.99... % of random pairs of utterances come from different people
- What good is accuracy then?



# Summary

- Machine learning: automatically learning from **training data** to accomplish certain **task**
- **Generalization** to data unseen during training is the core objective
- There are many different tasks, models, and strategies and algorithms
  - **Supervised, unsupervised**, reinforcement, ... learning
  - **Generative** models vs. **discriminative** training
- The learning algorithms usually optimizes some **objective function** which should correlate with the actual task objective
  - **Maximum likelihood** objective for learning distributions
  - **Sum-of-squares** error for regression
  - **Cross-entropy** for classification
  - We only considered simple models (linear/logistic regression), but the same objectives and the same optimization strategies (e.g. **gradient descent**) are used with more powerful models such as neural networks.
- The objective function is usually optimized w.r.t. some **parameters** of the underlying model, which we use to accomplished the desired task