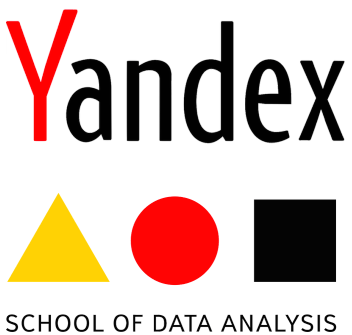
Machine Learning in Science and Industry

Day 1, lectures 1 & 2

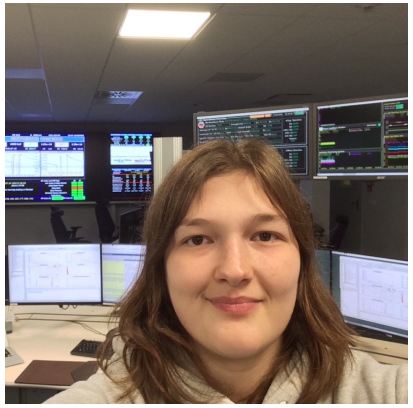
Alex Rogozhnikov, Tatiana Likhomanenko

Heidelberg, GradDays 2017



About authors

Alex Rogozhnikov



Tatiana Likhomanenko

- Yandex researchers (internet-company in Russia, mostly known for search engine)
- applying machine learning in CERN projects for 3 (Alex) and 4 (Tatiana) years
- graduated from Yandex School of Data Analysis

Intro notes

- 4 days course
  - but we'll cover most significant algorithms of machine learning
  - 2–2.5 hours for lecture, then optional practical session
  - we can be a bit out of schedule, but we'll try to fit
- all materials are published in repository: https://github.com/yandexdataschool/MLatGradDays
- we have a challenge!
  - know material? Spend more time on challenge!
  - you can participate in teams of two
  - use chat for questions on challenge

What is Machine Learning about?

- a method of teaching computers to make and improve predictions or behaviors based on some data?
- a field of computer science, probability theory, and optimization theory which allows complex tasks to be solved for which a logical/procedural approach would not be possible or feasible?
- a type of AI that provides computers with the ability to learn without being explicitly programmed?
- somewhat in between of statistics, AI, optimization theory, signal processing and pattern matching?

What is Machine Learning about

Inference of statistical dependencies which give us ability to predict

What is Machine Learning about

Inference of statistical dependencies which give us ability to predict

Data is cheap, knowledge is precious

Machine Learning is used in

- search engines
- spam detection
- security: virus detection, DDOS defense
- computer vision and speech recognition
- market basket analysis, customer relationship management (CRM), churn prediction
- credit scoring / insurance scoring, fraud detection
- health monitoring
- traffic jam prediction, self-driving cars
- advertisement systems / recommendation systems / news clustering
- handwriting recognition
- opinion mining

Machine Learning is used in

- search engines
- spam detection
- security: virus detection, DDOS defense
- computer vision and speech recognition
- market basket analysis, customer relationship management (CRM), churn prediction
- credit scoring / insurance scoring, fraud detection
- health monitoring
- traffic jam prediction, self-driving cars
- advertisement systems / recommendation systems / news clustering
- handwriting recognition
- opinion mining
- and hundreds more

Machine Learning in science

- In particle physics
  - High-level triggers
  - Particle identification
  - Tracking
  - Tagging
  - High-level analysis
- star / galaxy / quasar classification in astronomy
- analysis of gene expression
- brain-computer interfaces
- weather forecasting

Machine Learning in science

- In particle physics
  - High-level triggers
  - Particle identification
  - Tracking
  - Tagging
  - High-level analysis
- star / galaxy / quasar classification in astronomy
- analysis of gene expression
- brain-computer interfaces
- weather forecasting

In mentioned applications different data is used and different information is inferred, but the ideas beyond are quite similar.

General notion

In supervised learning the training data is represented as a **set of pairs**

$$x_i, y_i$$

- i is an index of an observation
- x_i is a vector of **features** available for an observation
- y_i is **a target** — the value we need to predict

features = observables = variables
observations = samples = events

Classification problem

$y_i \in Y$, where Y is finite set of labels.

Examples

- particle identification based on information about track:

$$x_i = (p, \eta, E, charge, \chi^2, ...)$$

$$Y = \{\text{electron, muon, pion, ... }\}$$

- binary classification:

  $Y = \{0, 1\}$ — 1 is signal, 0 is background

Regression problem

$$y \in \mathbb{R}$$

Examples:

- predicting price of a house by it's positions
- predicting money income / number of customers of a shop
- reconstructing real momentum of a particle

Regression problem

$$y \in \mathbb{R}$$

Examples:

- predicting price of a house by it's positions
- predicting money income / number of customers of a shop
- reconstructing real momentum of a particle

Why do we need automatic classification/regression?

- in applications up to thousands of features
- higher quality
- much faster adaptation to new problems

Classification based on nearest neighbours

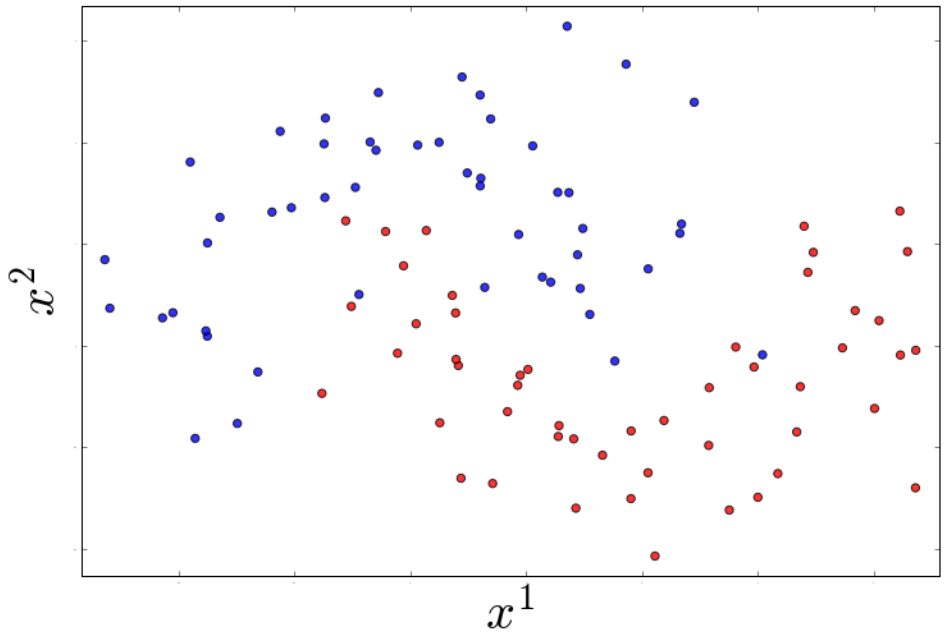
Given training set of objects and their labels $\{x_i, y_i\}$ we predict the label for the new observation x:

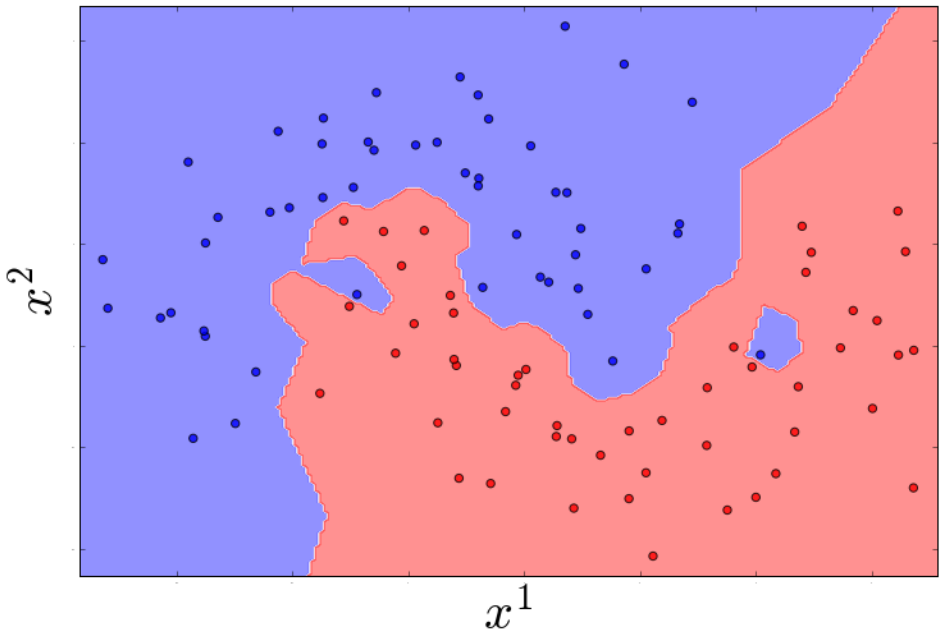
$$\hat{y} = y_j, \qquad j = \arg\min_i \rho(x, x_i)$$

Here and after $\rho(x, \tilde{x})$ is the distance in the space of features.

Visualization of decision rule

Consider a classification problem with 2 features:





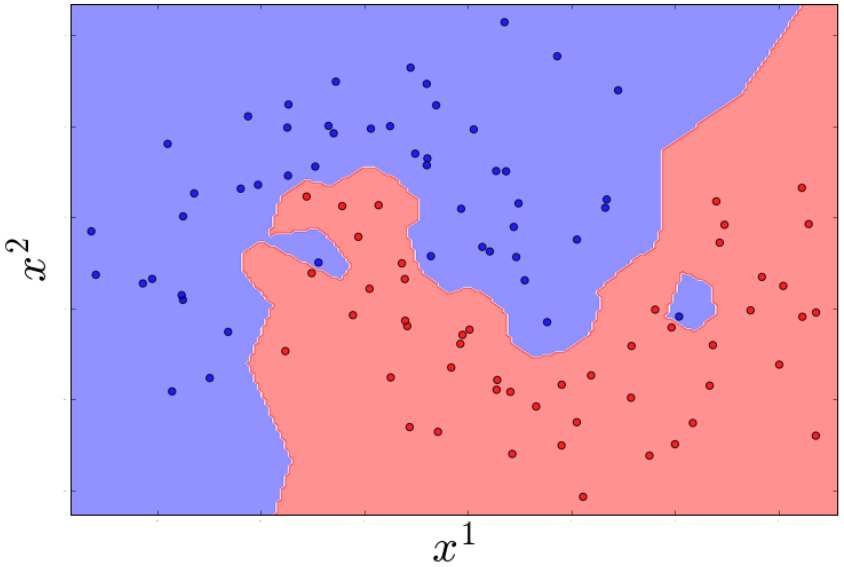
$x_i = (x_i^1, x_i^2), y_i \in Y = \{0, 1\}$

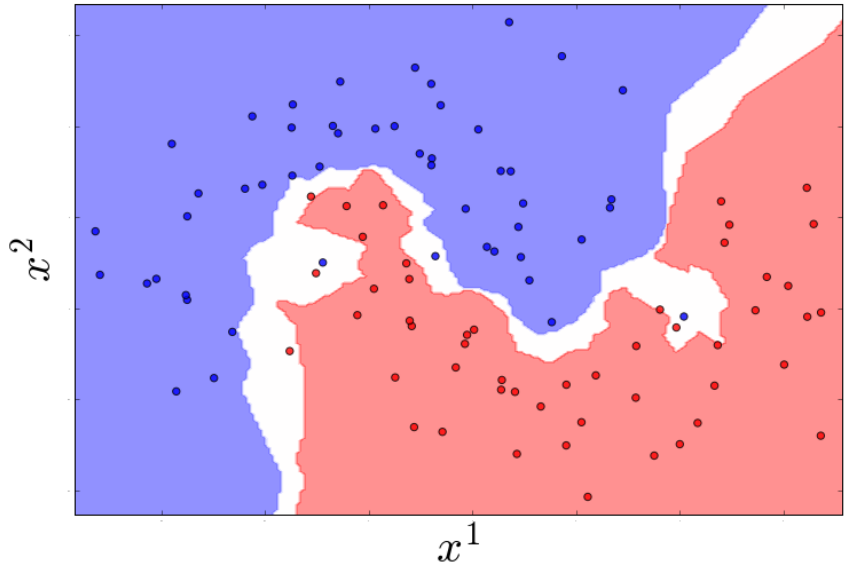
k Nearest Neighbours (kNN)

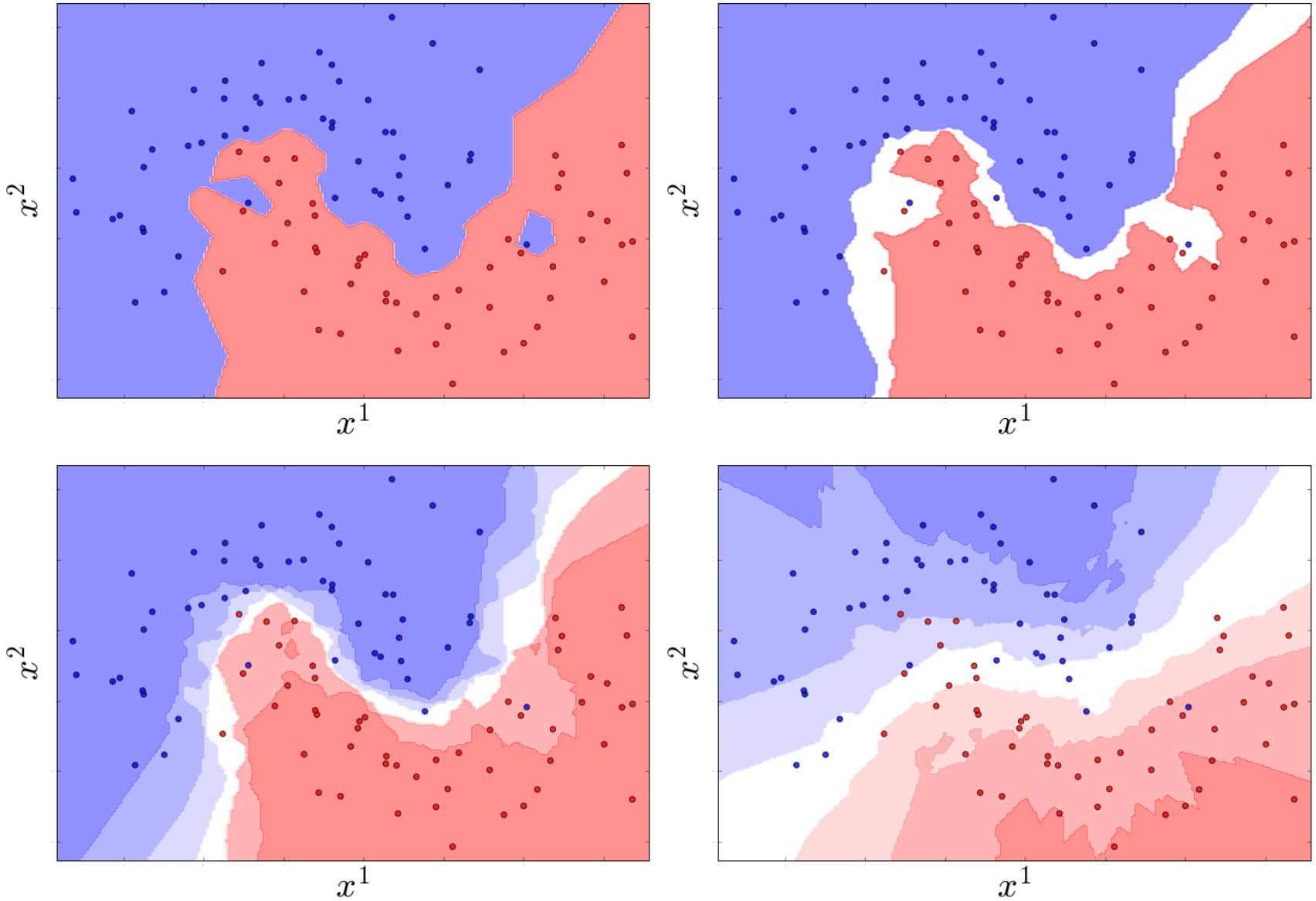
We can use k neighbours to compute probabilities to belong to each class:

$$p_{\tilde{y}}(x) = \frac{\#\text{ of knn of x in class } \tilde{y}}{k}$$

Question: why this may be a good idea?


x^2
x^1


x^2
x^1



k = 1, 2, 5, 30

Overfitting

Question: how accurate is classification on training dataset when $k = 1$?

Overfitting

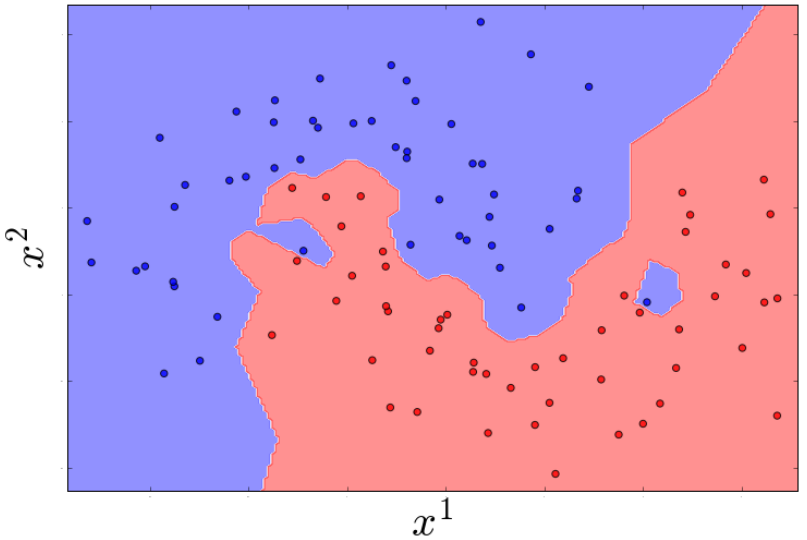
Question: how accurate is classification on training dataset when $k = 1$?

- answer: it is ideal (closest neighbor is observation itself)
- quality is lower when $k > 1$

Overfitting

Question: how accurate is classification on training dataset when $k = 1$?

- answer: it is ideal (closest neighbor is observation itself)
- quality is lower when $k > 1$
- this doesn't mean $k = 1$ is the best,
  it means we cannot use training observations to estimate quality
- when classifier's decision rule is too complex and captures details from training data that are not relevant to distribution, we call this *an overfitting*



Model selection

Given two models, which one should we select?

> *ML is about inference of statistical dependencies, which give us ability to predict*

Model selection

Given two models, which one should we select?

> *ML is about inference of statistical dependencies, which give us ability to predict*

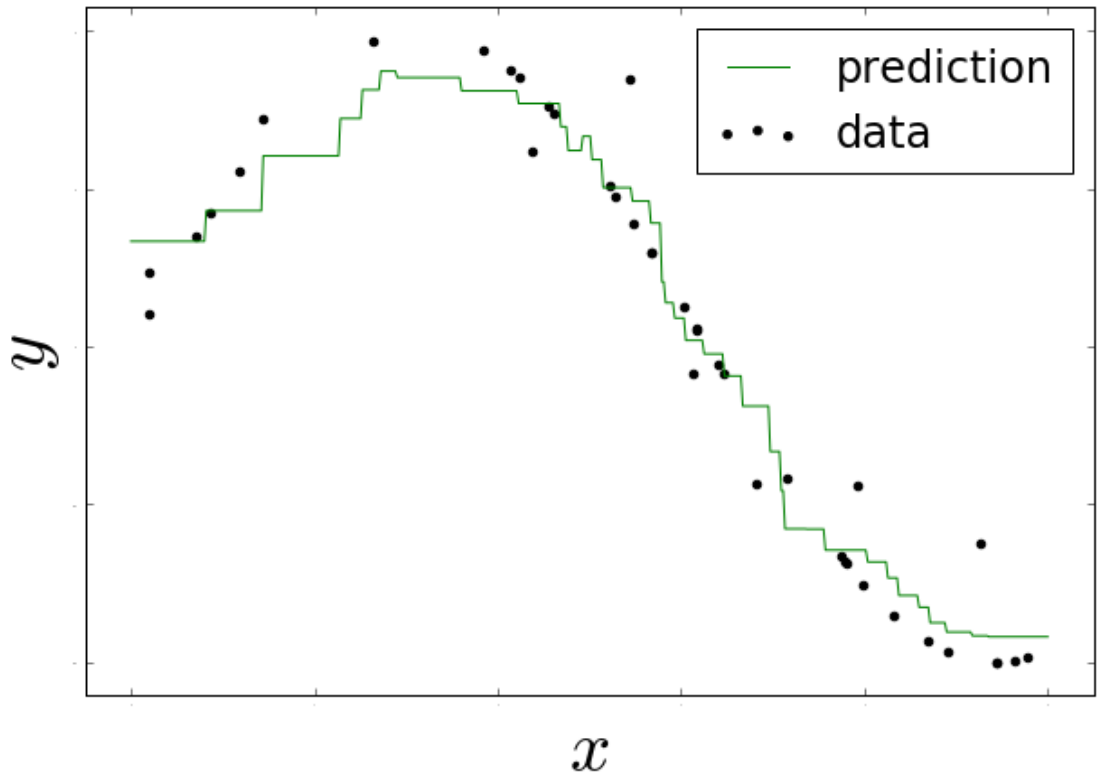
The best model is the model which gives better predictions for new observations.

Simplest way to control this is to check quality on a holdout — a sample not used during training (*cross-validation*). This gives unbiased estimate of quality for new data.

- estimates have variance
- multiple testing introduces bias (solution: train + validation + test, like kaggle)
  - very important factor if amount of data is small
- there are more approaches to cross-validation

Regression using kNN

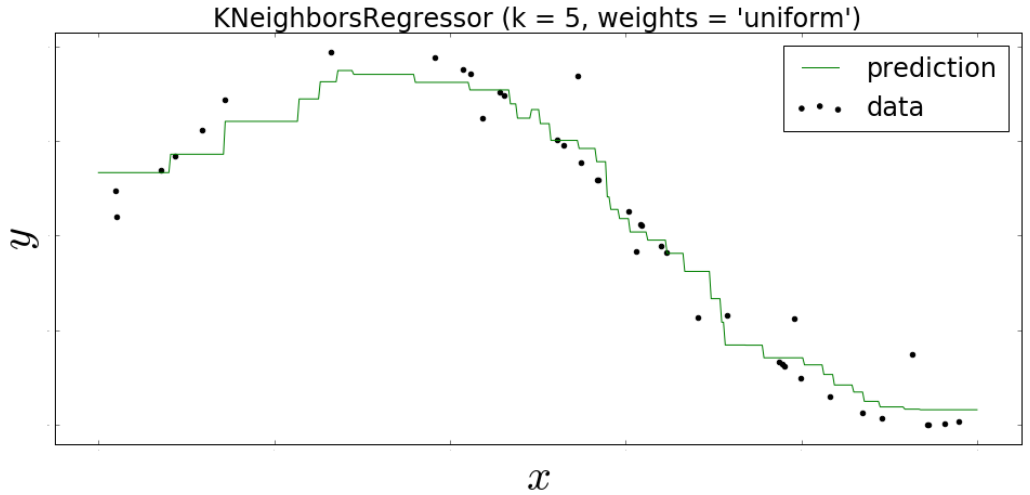
Regression with nearest neighbours is done by averaging of output

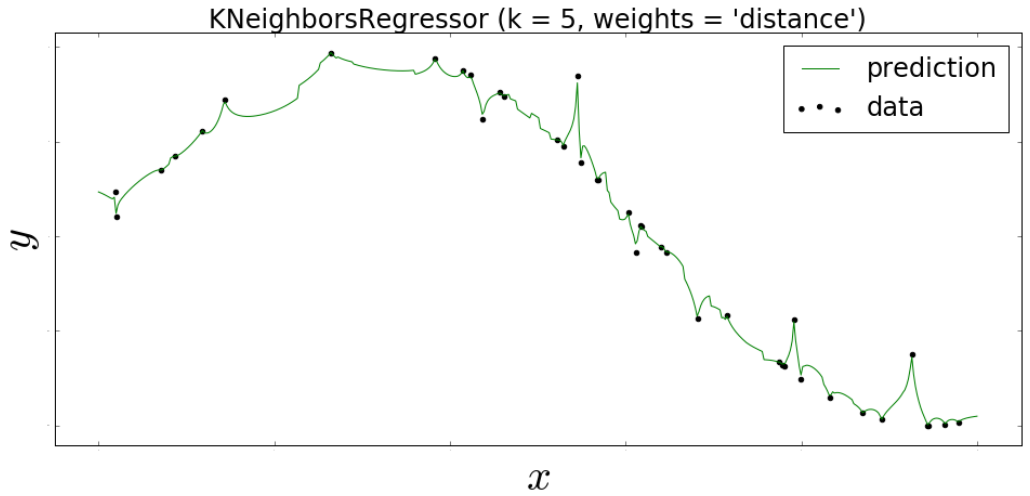


Model prediction:

$$\hat{y}(x) = \frac{1}{k} \sum_{j \in \text{knn}(x)} y_j$$

kNN with weights





Average neighbours' output with weights:

$$\hat{y} = \frac{\sum_{j \in \text{knn}(x)} w_j y_j}{\sum_{j \in \text{knn}(x)} w_j}$$

the closer the neighbour, the higher weights of its contribution, i.e.:

$$w_j = 1/\rho(x, x_j)$$

Computational complexity

Given that dimensionality of space is d and there are n training samples:

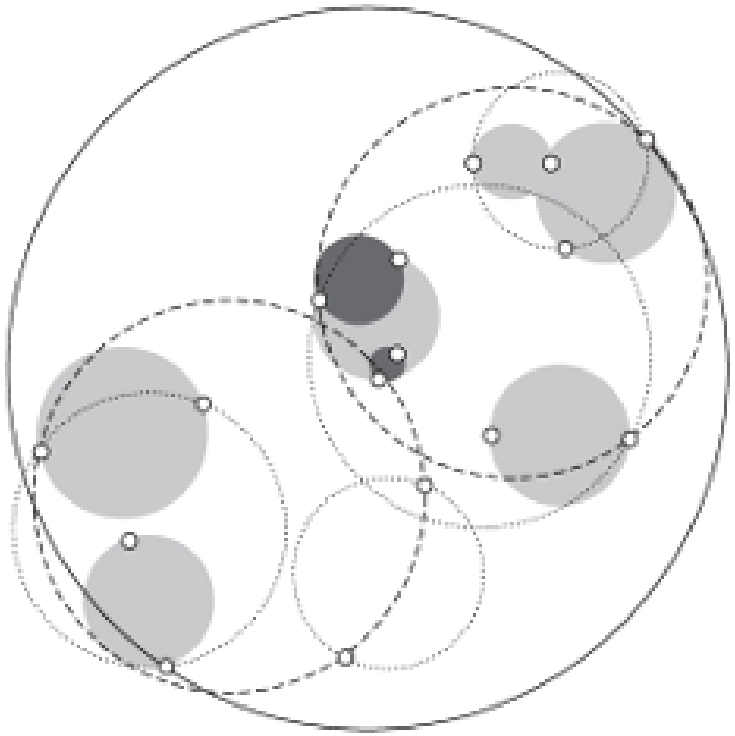
- training time: ~ O(save a link to the data)
- prediction time: $n \times d$ for each observation

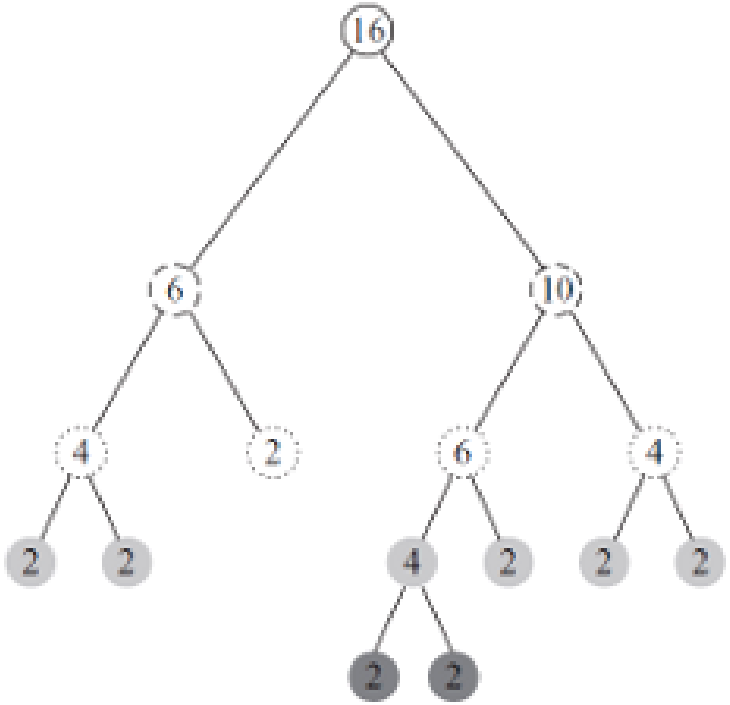
Prediction is extremely slow.

To resolve this one can build index-like structure to select neighbours faster.

Spacial index: ball tree

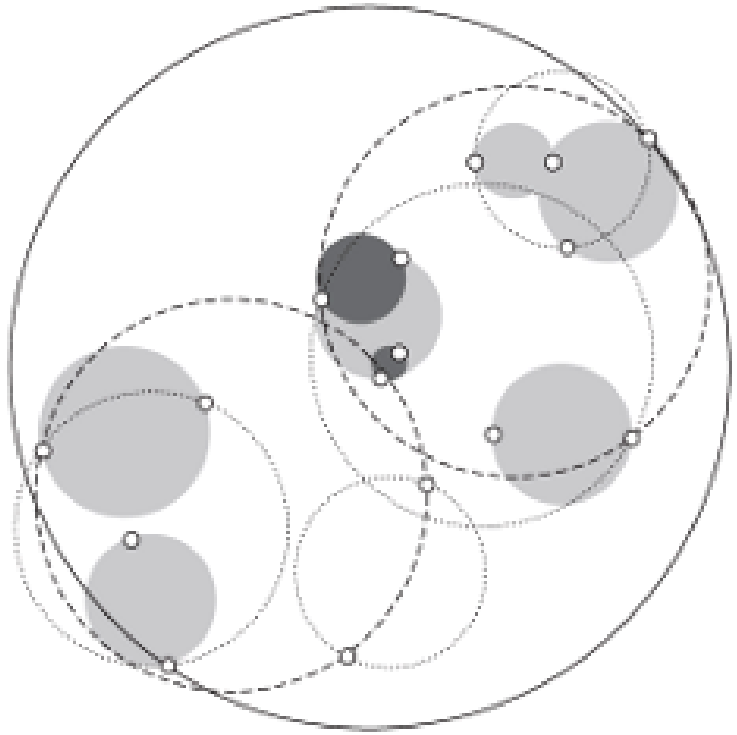
- training time
  ~ $O(d \times n \log(n))$
- prediction time
  ~ $\log(n) \times d$
  for each observation

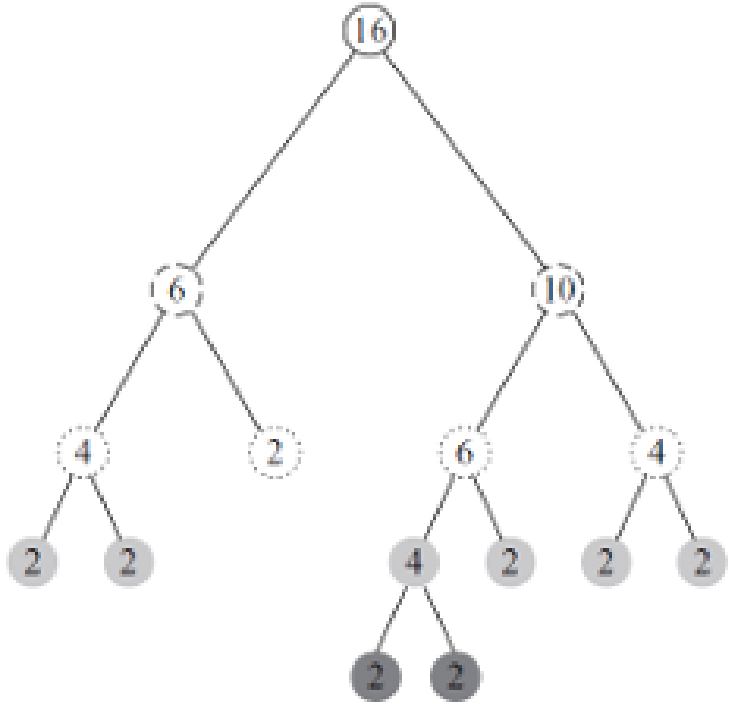




Spacial index: ball tree

- training time
  ~ $O(d \times n \log(n))$
- prediction time
  ~ $\log(n) \times d$
  for each observation





Other options exist:

- KD-tree
- FLANN
- FALCONN
- IMI, NO-IMI

Those are very useful for searching over large databases with items of high dimensions

Overview of kNN

- Awesomely simple classifier and regressor
- Provides too optimistic quality on training data
- Quite slow, though optimizations exist
- Too sensitive to scale of features
- Doesn't provide good results with data of high dimensions

Sensitivity to scale of features

Euclidean distance:

$$\rho(x, \tilde{x})^2 = (x_1 - \tilde{x}_1)^2 + (x_2 - \tilde{x}_2)^2 + ... + (x_d - \tilde{x}_d)^2$$

Sensitivity to scale of features

Euclidean distance:

$$\rho(x, \tilde{x})^2 = (x_1 - \tilde{x}_1)^2 + (x_2 - \tilde{x}_2)^2 + ... + (x_d - \tilde{x}_d)^2$$

Change scale of first feature:

$$\rho(x, \tilde{x})^2 = (10x_1 - 10\tilde{x}_1)^2 + (x_2 - \tilde{x}_2)^2 + ... + (x_d - \tilde{x}_d)^2$$
$$\rho(x, \tilde{x})^2 \sim 100(x_1 - \tilde{x}_1)^2$$

(assuming that $x_1, x_2, ..., x_n$ are of the same order)

Sensitivity to scale of features

Euclidean distance:

$$\rho(x, \tilde{x})^2 = (x_1 - \tilde{x}_1)^2 + (x_2 - \tilde{x}_2)^2 + ... + (x_d - \tilde{x}_d)^2$$

Change scale of first feature:

$$\rho(x, \tilde{x})^2 = (10x_1 - 10\tilde{x}_1)^2 + (x_2 - \tilde{x}_2)^2 + ... + (x_d - \tilde{x}_d)^2$$
$$\rho(x, \tilde{x})^2 \sim 100(x_1 - \tilde{x}_1)^2$$

(assuming that $x_1, x_2, ..., x_n$ are of the same order)

Scaling of features frequently increases quality.
Otherwise contribution from features with large scale is dominating.

Distance function matters

- Minkowski distance

$$\rho(x, \tilde{x})^p = \sum_l (x_l - \tilde{x}_l)^p$$

- Canberra

$$\rho(x, \tilde{x}) = \sum_l \frac{|x_l - \tilde{x}_l|}{|x_l| + |\tilde{x}_l|}$$

- Cosine metric

$$\rho(x, \tilde{x}) = \frac{< x, \tilde{x} >}{|x| \ |\tilde{x}|}$$

Problems with high dimensions

With higher dimensions $d >> 1$ the neighbouring points are further.

Example: consider n training data points being distributed uniformly in the unit cube:

- expected number of point in the ball of size r is proportional to the r^d
- to collect the same amount on neighbors, we need to put $r = \text{const}^{1/d} \to 1$

kNN suffers from *curse of dimensionality*.

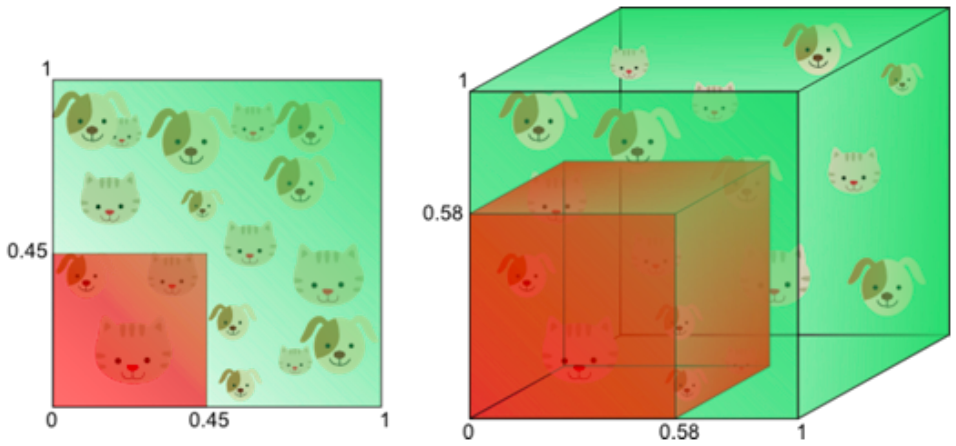


image source

Nearest neighbours applications

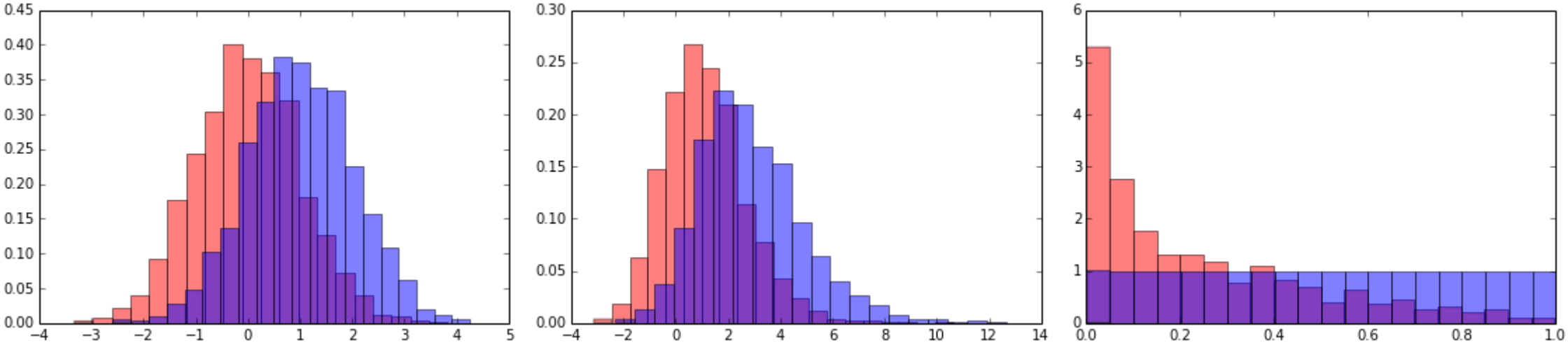
Classification and regression based on kNN used quite rarely, but the nearest neighbours search is very widespread, used when you need to find something similar in the database.

- finding documents and images
- finding similar proteins by using 3d-structure
- search in chemical databases
- person identification by photo or by speech
- user identification by behavior on the web

Similarity (distance) is very important and isn't defined trivially in most cases.

Measuring quality of binary classification

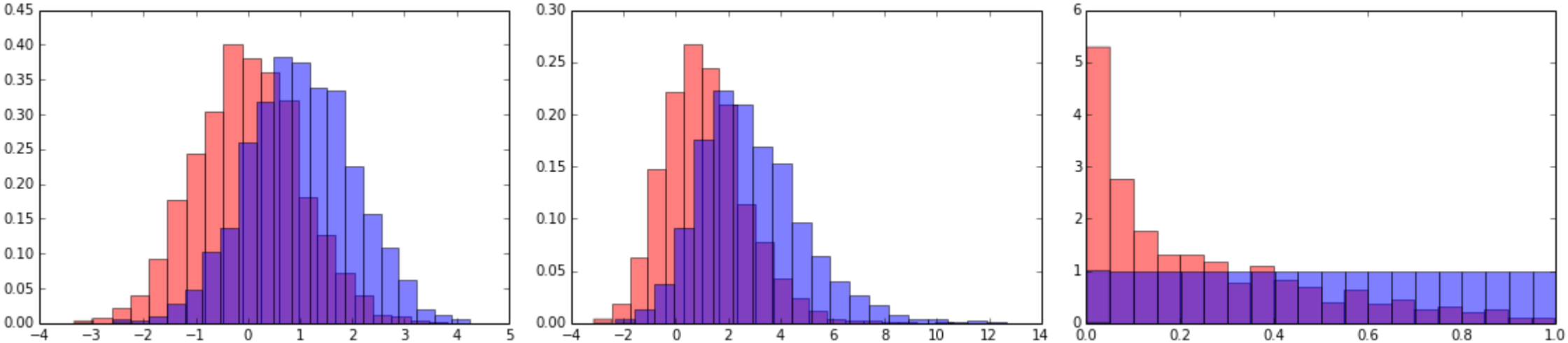
The classifier's output in binary classification is real variable (say, signal is blue and background is red)



- Which classifier provides better discrimination?

Measuring quality of binary classification

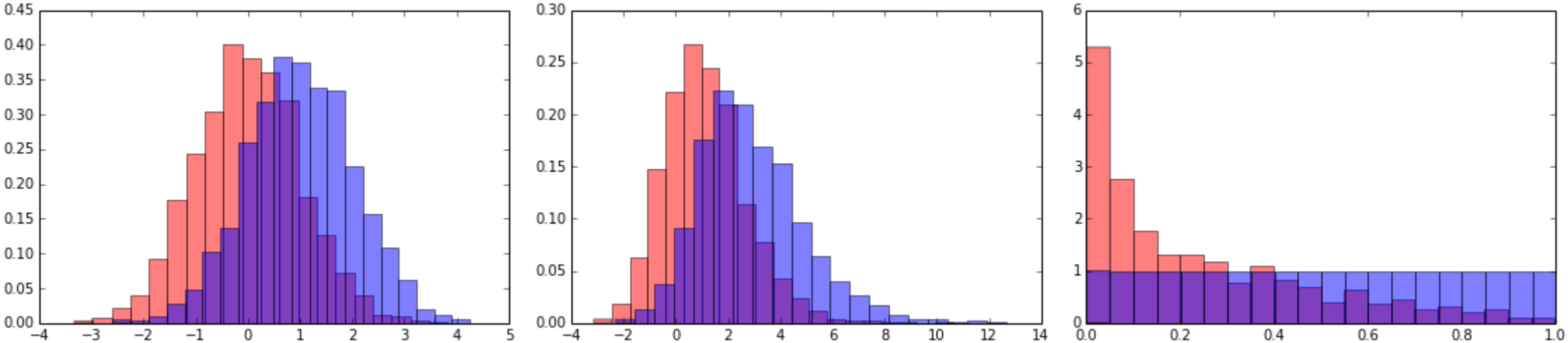
The classifier's output in binary classification is real variable (say, signal is blue and background is red)



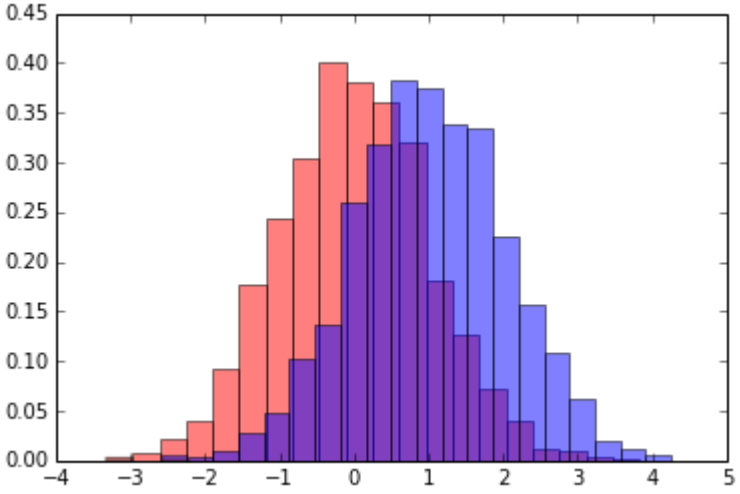
- Which classifier provides better discrimination?
- Discrimination is identical in all three cases

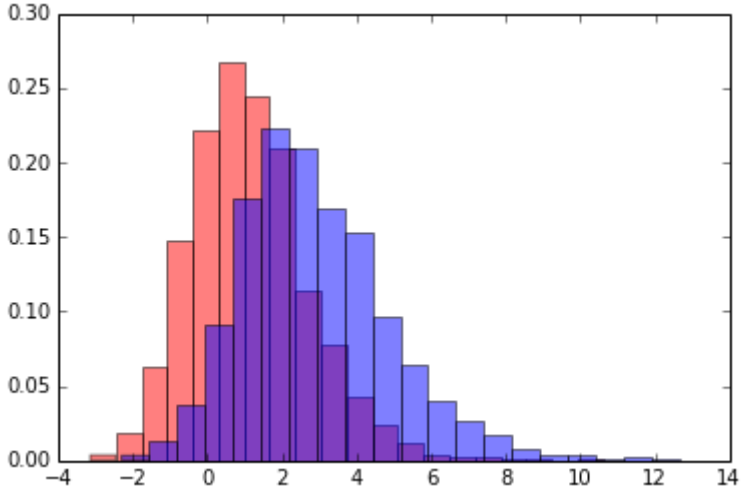
ROC curve demonstration

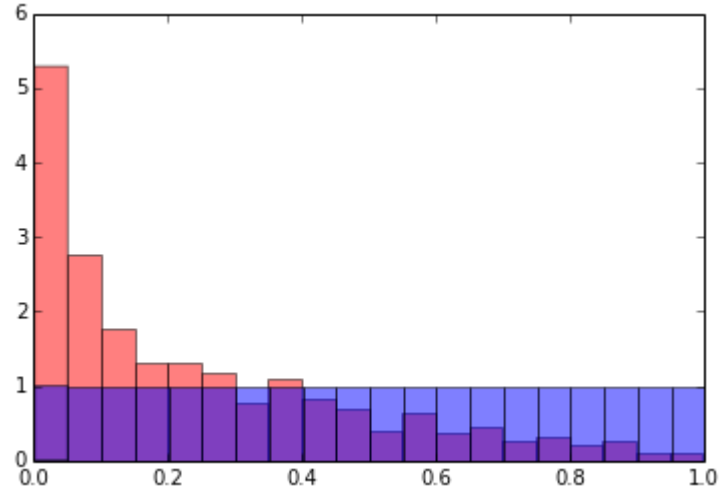
ROC curve


0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−4 −3 −2 −1 0 1 2 3 4 5
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−4 −2 0 2 4 6 8 10 12 14
6
5
4
3
2
1
0
0.0 0.2 0.4 0.6 0.8 1.0

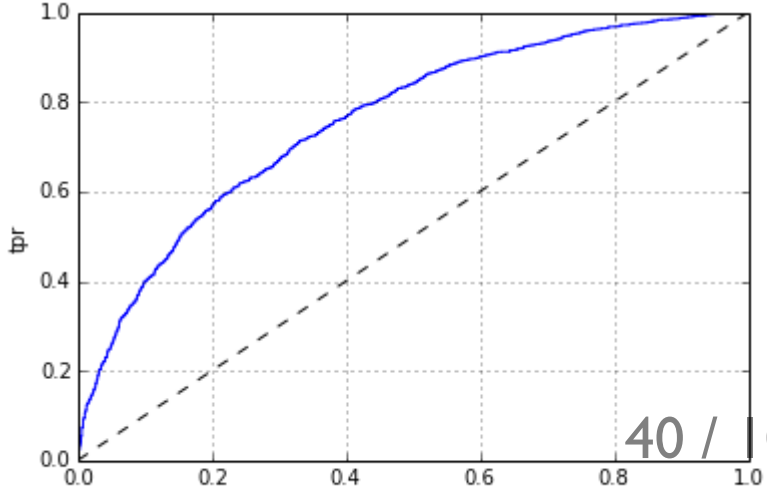
ROC curve







These distributions have the same ROC curve:
(ROC curve is passed signal vs passed bck dependency)



ROC curve

- Defined only for binary classification
- Contains important information:
  all possible combinations of signal and background efficiencies you may achieve by setting a threshold
- Particular values of thresholds (and initial pdfs) don't matter, ROC curve doesn't contain this information
- ROC curve = information about order of observations' predictions:

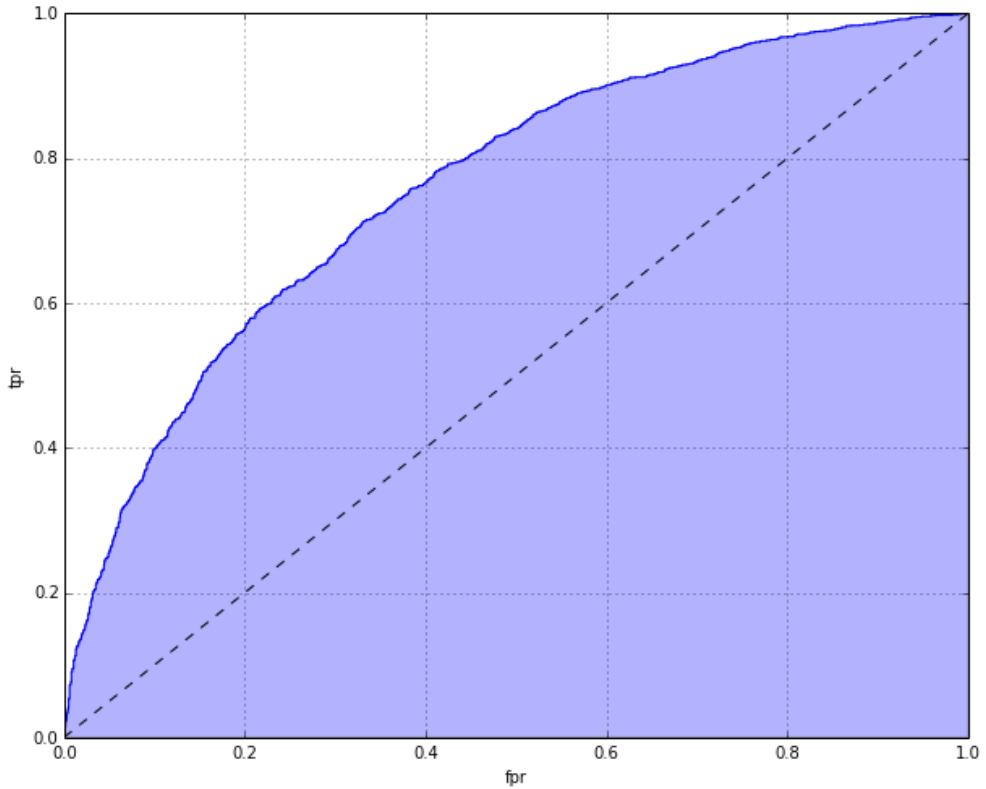
```
b b s b s b  ...  s s b s s
```

- Comparison of algorithms should be based on the information from ROC curve.

ROC AUC (area under the ROC curve)

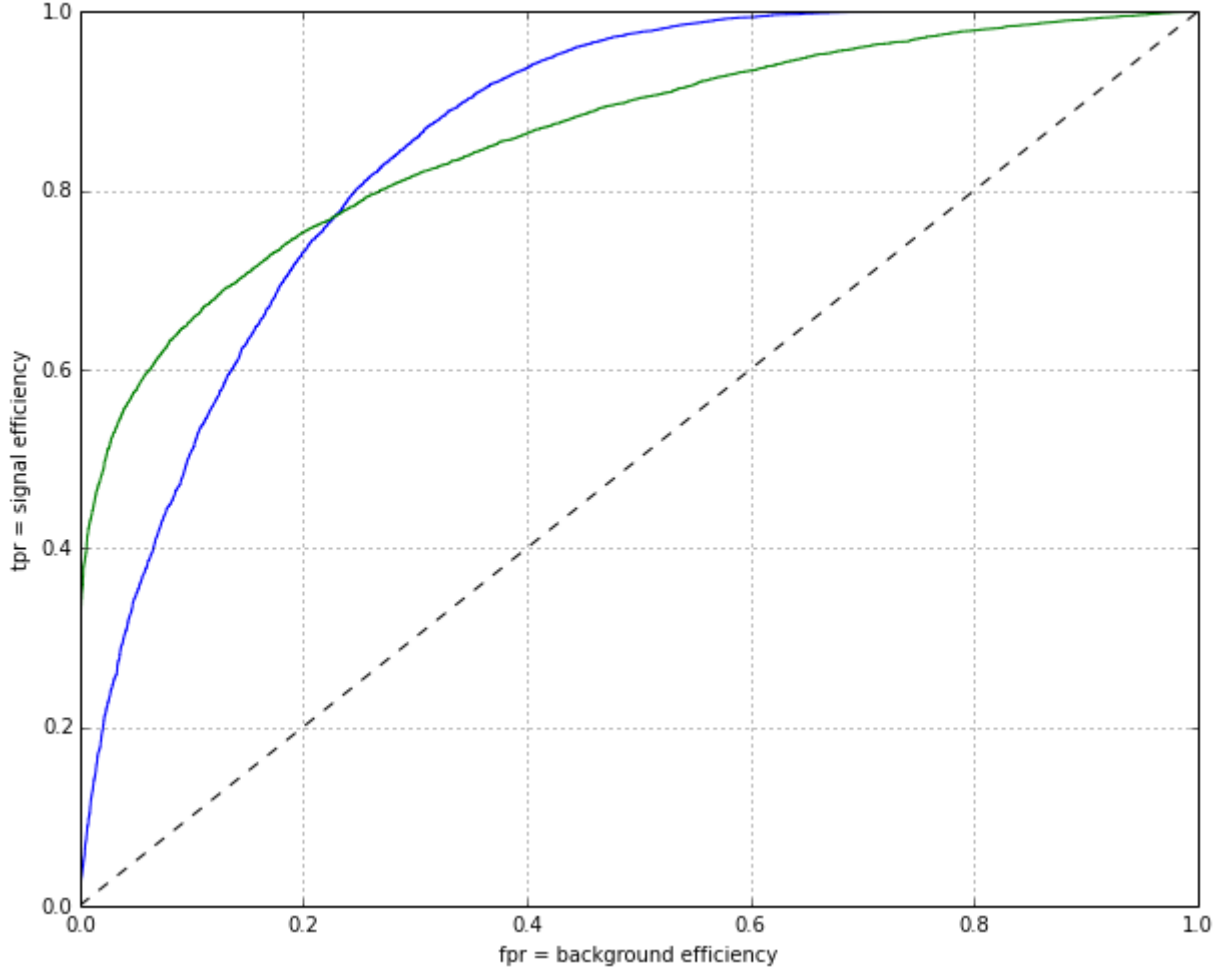
$ROC\ AUC = P(r_b < r_s)$

where r_b, r_s are predictions of random background and signal observations.



Classifier have the same ROC AUC, but which is better ...

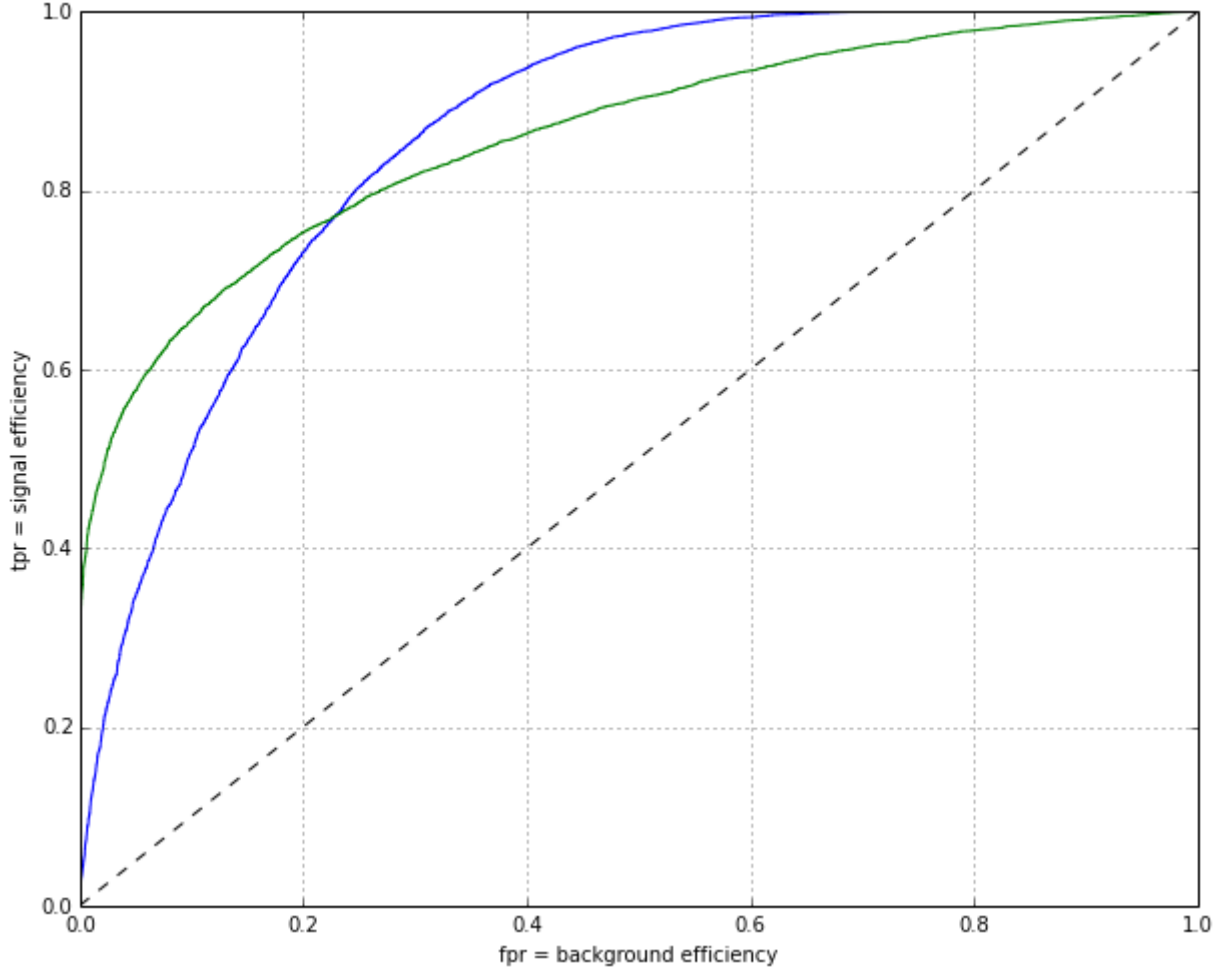
- if the problem is spam detection? (putting normal letter in spam is very undesireable)
- for background rejection system at the LHC? (we need to pass very few background)



Classifier have the same ROC AUC, but which is better ...

- if the problem is spam detection? (putting normal letter in spam is very undesireable)
- for background rejection system at the LHC? (we need to pass very few background)

Applications frequently demand different metric.



Statistical Machine Learning

Machine learning we use in practice is based on statistics

- Main assumption: the data is generated from probabilistic distribution:

$$p(x, y)$$

- Does there really exist the distribution of people / pages / texts?
- Same question for stars / proteins / viruses?

Statistical Machine Learning

Machine learning we use in practice is based on statistics

- Main assumption: the data is generated from probabilistic distribution:

$$p(x, y)$$

- Does there really exist the distribution of people / pages / texts?
- Same question for stars / proteins / viruses?
- In HEP these distributions do exist

Statistical framework turned out to be quite helpful to build models.

Optimal regression

Assuming that we know real distributions $p(x, y)$ and the goal is to minimize squared error:

$$\mathcal{L} = \mathbb{E}(y - \hat{y}(x))^2 \to \min$$

An optimal prediction is a marginalized average

$$\hat{y}(x_0) = \mathbb{E}(y|x = x_0)$$

Optimal classification. Bayes optimal classifier

Assuming that we know real distributions $p(x, y)$ we reconstruct using Bayes' rule

$$p(y|x) = \frac{p(x,y)}{p(x)} = \frac{p(y)p(x|y)}{p(x)}$$

$$\frac{p(y=1\,|\,x)}{p(y=0\,|\,x)} = \frac{p(y=1)\,p(x\,|\,y=1)}{p(y=0)\,p(x\,|\,y=0)}$$

Optimal classification. Bayes optimal classifier

Assuming that we know real distributions $p(x, y)$ we reconstruct using Bayes' rule

$$p(y|x) = \frac{p(x,y)}{p(x)} = \frac{p(y)p(x|y)}{p(x)}$$

$$\frac{p(y=1\,|\,x)}{p(y=0\,|\,x)} = \frac{p(y=1)\,p(x\,|\,y=1)}{p(y=0)\,p(x\,|\,y=0)}$$

Lemma (Neyman–Pearson):

The best classification quality is provided by $\frac{p(y=1\,|\,x)}{p(y=0\,|\,x)}$ (*Bayes optimal classifier*)

Optimal Binary Classification

- Bayes optimal classifier has the highest possible ROC curve.
- Since the classification quality depends only on order, $p(y = 1 \mid x)$ gives optimal classification quality too!

$$\frac{p(y = 1 \mid x)}{p(y = 0 \mid x)} = \frac{p(y = 1)}{p(y = 0)} \times \frac{p(x \mid y = 1)}{p(x \mid y = 0)}$$

Optimal Binary Classification

- Bayes optimal classifier has the highest possible ROC curve.
- Since the classification quality depends only on order, $p(y = 1 \mid x)$ gives optimal classification quality too!

$$\frac{p(y = 1 \mid x)}{p(y = 0 \mid x)} = \frac{p(y = 1)}{p(y = 0)} \times \frac{p(x \mid y = 1)}{p(x \mid y = 0)}$$

How can we estimate from data terms from this expression?

- $p(y = 1), p(y = 0)$?

Optimal Binary Classification

- Bayes optimal classifier has the highest possible ROC curve.
- Since the classification quality depends only on order, $p(y = 1 \mid x)$ gives optimal classification quality too!

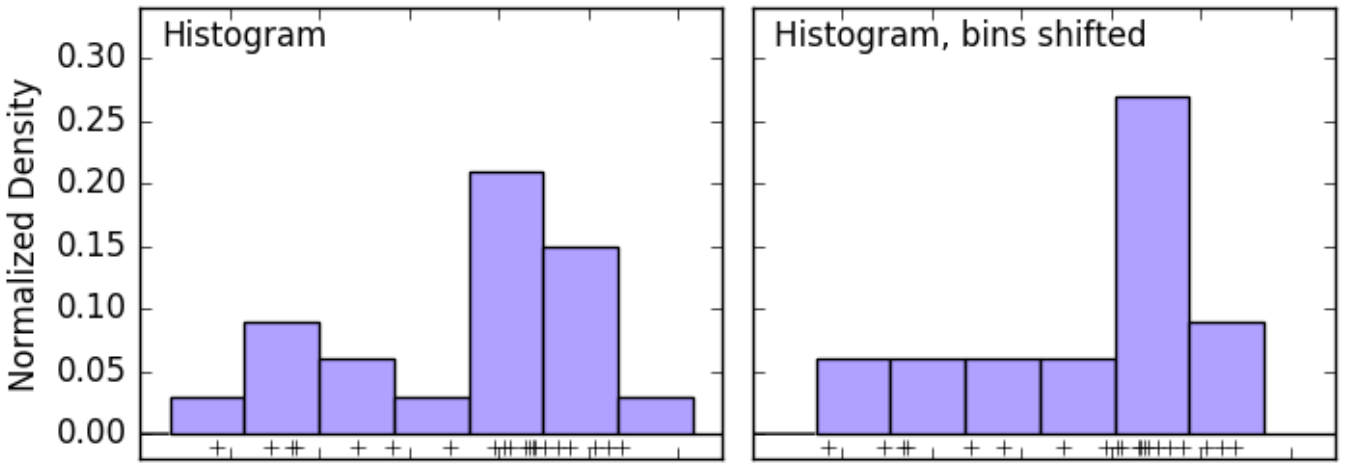
$$\frac{p(y = 1 \mid x)}{p(y = 0 \mid x)} = \frac{p(y = 1)}{p(y = 0)} \times \frac{p(x \mid y = 1)}{p(x \mid y = 0)}$$

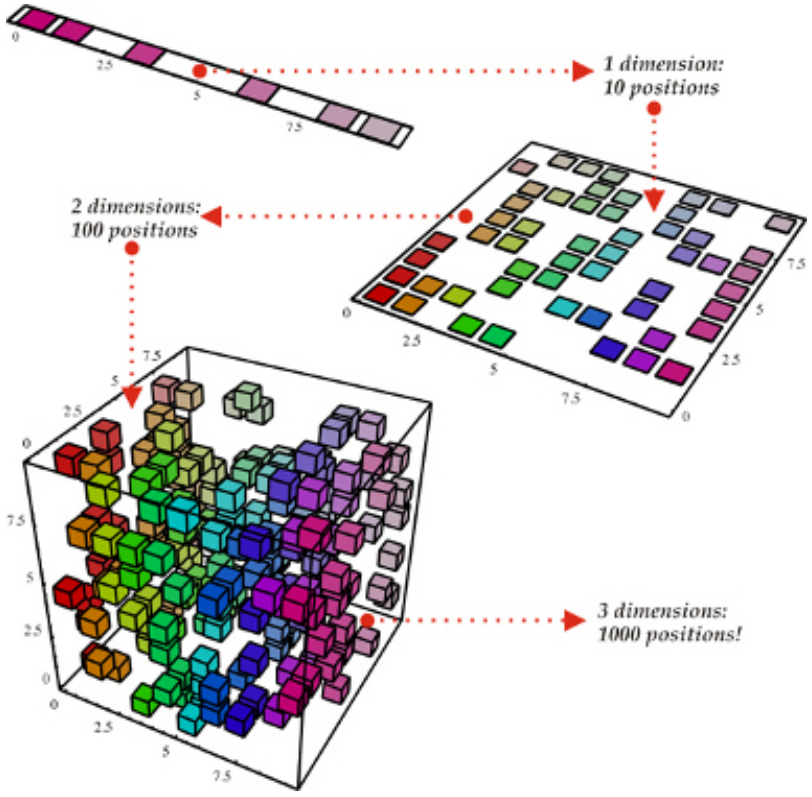
How can we estimate from data terms from this expression?

- $p(y = 1), p(y = 0)$?
- $p(x \mid y = 1), p(x \mid y = 0)$?

Histograms density estimation

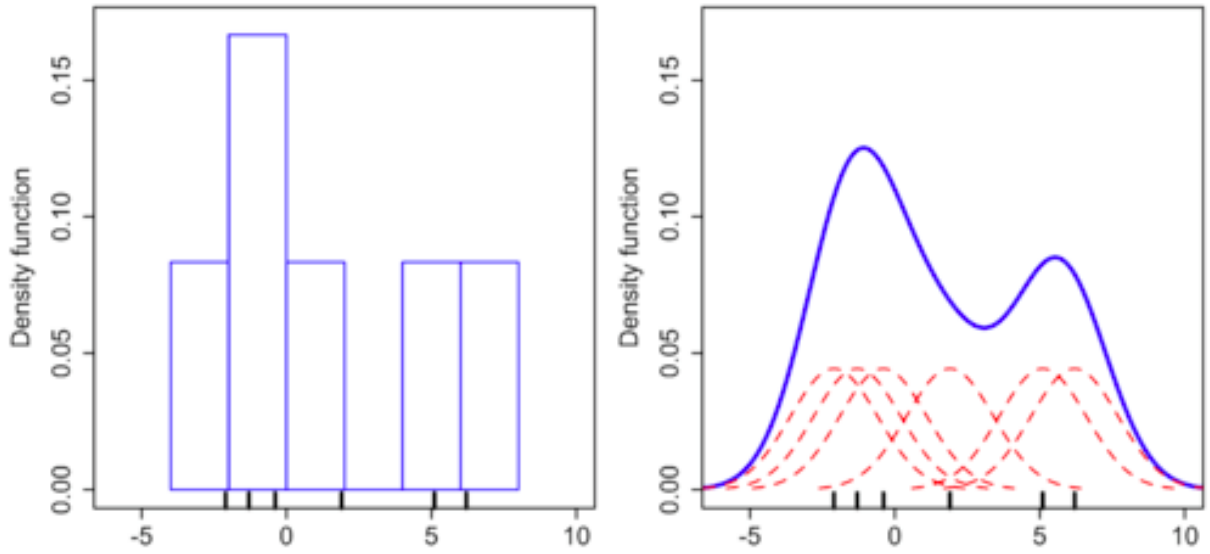
Counting number of samples in each bin and normalizing.





- fast
- choice of binning is crucial
- number of bins grows exponentially $\rightarrow$ curse of dimensionality

Kernel density estimation

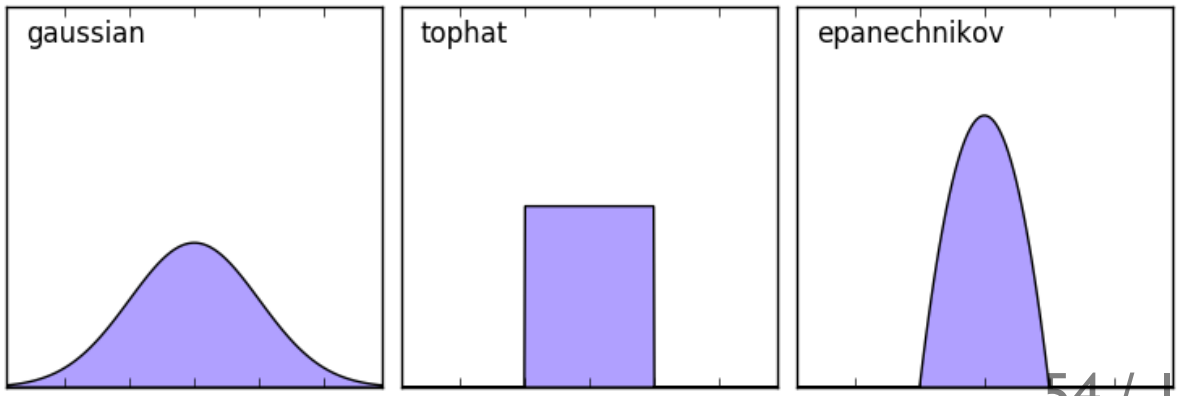


$$f(x) = \frac{1}{nh} \sum_i K\left(\frac{x - x_i}{h}\right)$$

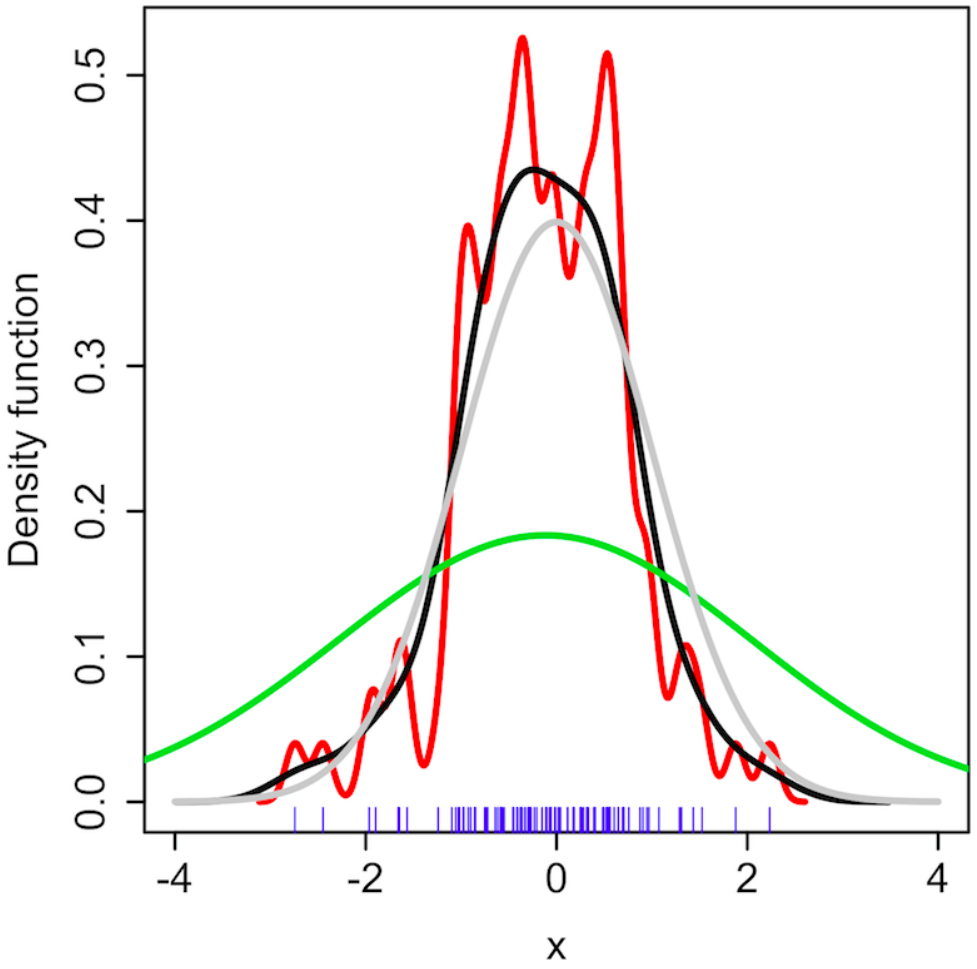
$K(x)$ is kernel, h is bandwidth

Typically, gaussian kernel is used, but there are many others.

Approach is very close to weighted kNN.



Kernel density estimation

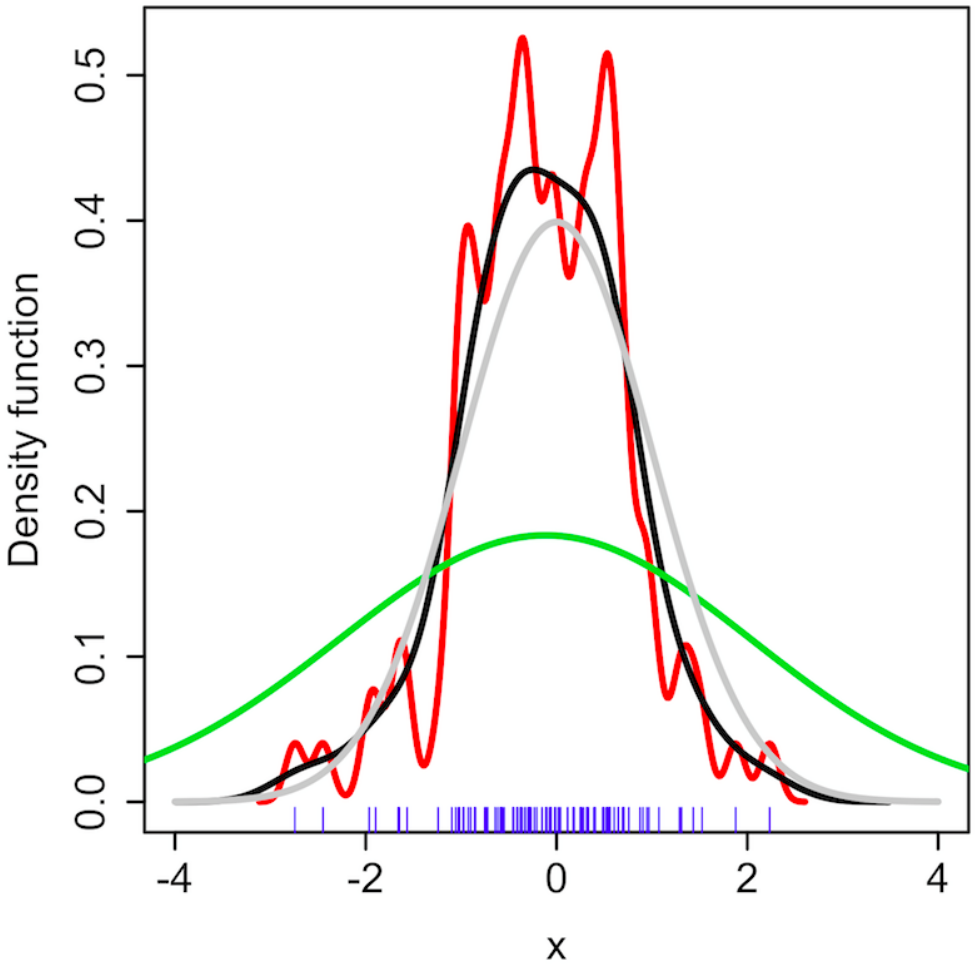


- bandwidth selection
- Silverman's rule of thumb:

$$h = \hat{\sigma} \left(\frac{4}{3n} \right)^{\frac{1}{5}}$$

$\hat{\sigma}$ is an empirical standard deviation

Kernel density estimation



- bandwidth selection
- Silverman's rule of thumb:

$$h = \hat{\sigma} \left(\frac{4}{3n} \right)^{\frac{1}{5}}$$

  $\hat{\sigma}$ is an empirical standard deviation
- may be irrelevant if the data is far from being gaussian

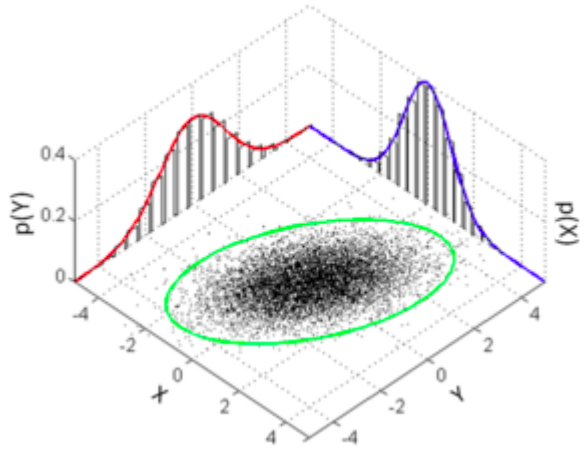
n-minutes break

Recapitulation

1. Statistical ML: applications and problems
2. k nearest neighbours classifier and regressor
    - distance function
    - overfitting and model selection
    - speeding up neighbours search
3. Binary classification: ROC curve, ROC AUC
4. Optimal regression and optimal classification
5. Density estimation
    - histograms
    - kernel density estimation

Parametric density estimation

Family of density functions: $f(x;\theta)$.



Problem: estimate parameters of a Gaussian distribution.

$$f(x;\mu,\Sigma) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

QDA (Quadratic discriminant analysis)

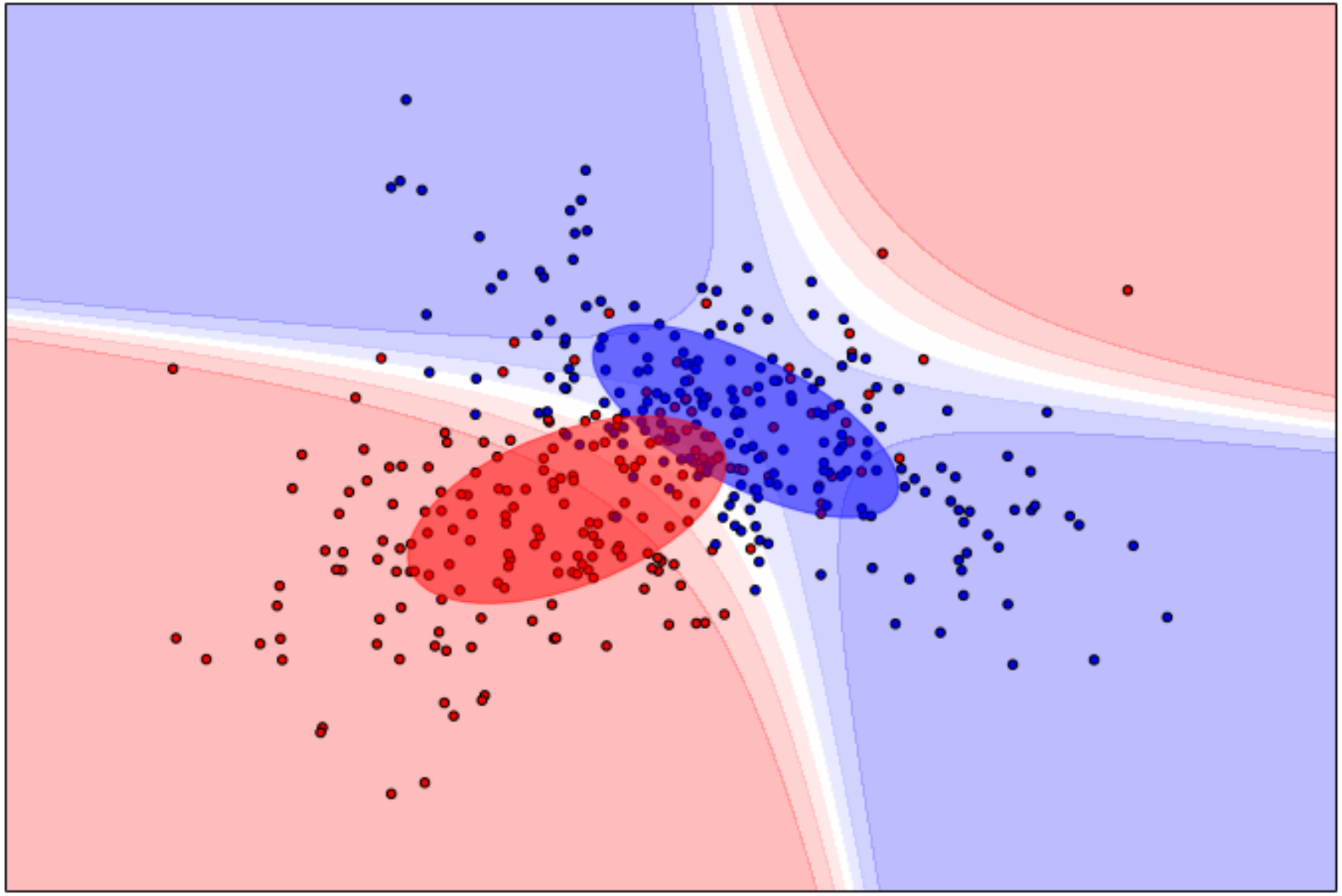
Reconstructing probabilities $p(x \mid y = 1), p(x \mid y = 0)$ from data, assuming those are multidimensional normal distributions:

$$p(x \mid y = 0) \sim \mathcal{N}(\mu_0, \Sigma_0)$$

$$p(x \mid y = 1) \sim \mathcal{N}(\mu_1, \Sigma_1)$$

$$\frac{p(y = 1 \mid x)}{p(y = 0 \mid x)} = \frac{p(y = 1)}{p(y = 0)} \frac{p(x \mid y = 1)}{p(x \mid y = 0)} = \frac{n_1}{n_0} \text{const} \frac{\exp\left(-\frac{1}{2}(x - \mu_1)^T \Sigma_1^{-1} (x - \mu_1)\right)}{\exp\left(-\frac{1}{2}(x - \mu_0)^T \Sigma_0^{-1} (x - \mu_0)\right)} =$$

$$= \exp\left(-\frac{1}{2}(x - \mu_1)^T \Sigma_1^{-1} (x - \mu_1) + \frac{1}{2}(x - \mu_0)^T \Sigma_0^{-1} (x - \mu_0) + \text{const}\right)$$



QDA computational complexity

n samples, d dimensions

- training consists of fitting $p(x \mid y = 0)$ and $p(x \mid y = 1)$ takes $O(nd^2 + d^3)$
  - computing covariance matrix $O(nd^2)$
  - inverting covariance matrix $O(d^3)$
- prediction takes $O(d^2)$ for each sample spent on computing dot product

QDA overview

- simple analytical formula for prediction
- fast prediction
- many parameters to reconstruct in high dimensions
  - thus, estimate may become non-reliable
- data almost never has gaussian distribution

Gaussian mixtures for density estimation

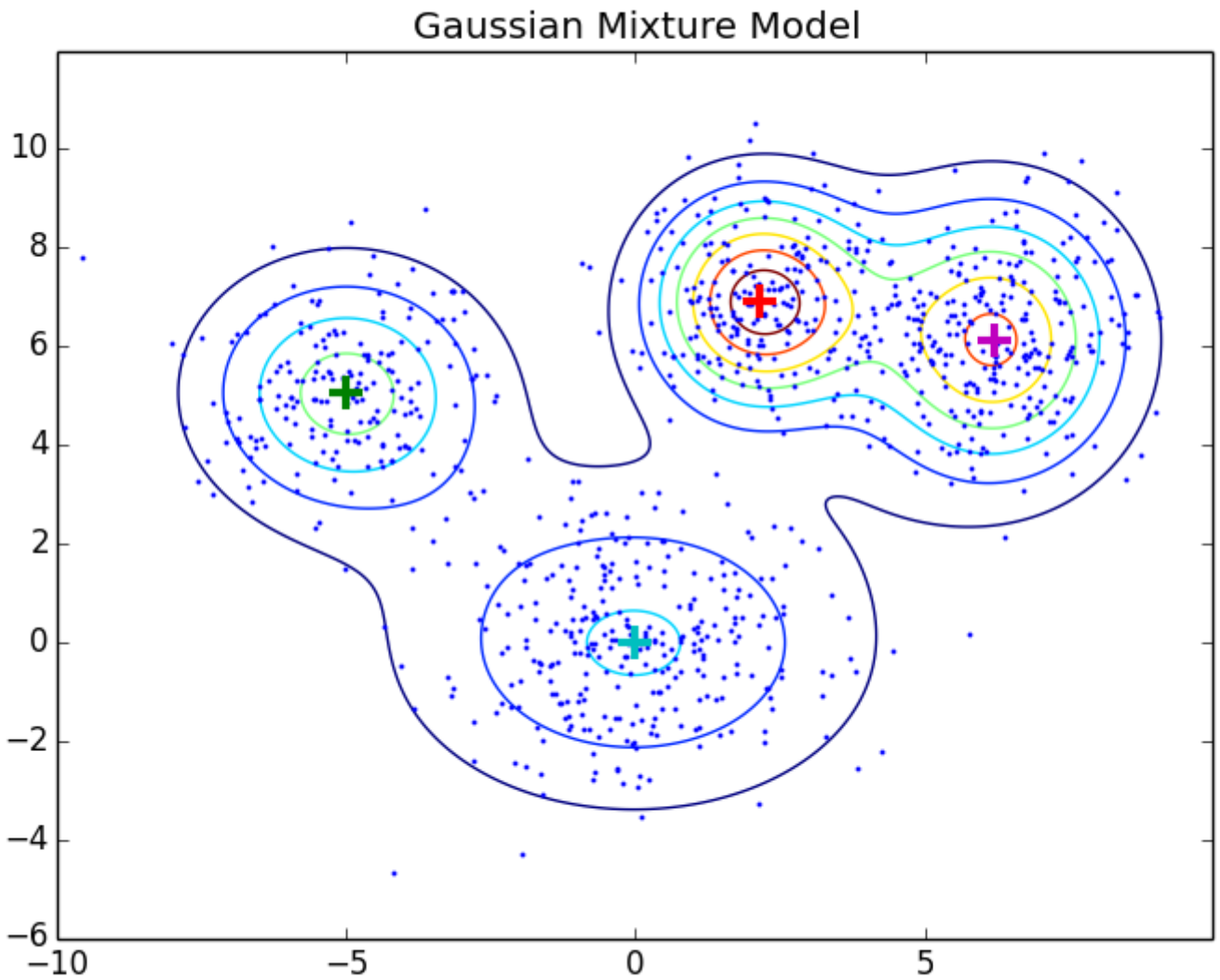
Mixture of distributions:

$$f(x) = \sum_{c-\text{components}} \pi_c \; f_c(x, \theta_c); \qquad \sum_{c-\text{components}} \pi_c = 1$$

Mixture of Gaussian distributions:

$$f(x) = \sum_{c-\text{components}} \pi_c \; f(x; \mu_c, \Sigma_c)$$

Parameters to be found: $\pi_1, ..., \pi_C$, $\mu_1, ..., \mu_C$, $\Sigma_1, ..., \Sigma_C$


Gaussian Mixture Model
10
8
6
4
2
0
−2
−4
−6
−10
−5
0
5

Gaussian mixtures: finding parameters

Criterion is maximizing likelihood (using MLE to find optimal parameters)

$$\sum_i \log f(x_i; \theta) \to \max_\theta$$

- no analytic solution
- we can use general-purpose optimization methods

Gaussian mixtures: finding parameters

Criterion is maximizing likelihood (using MLE to find optimal parameters)

$$\sum_i \log f(x_i; \theta) \to \max_\theta$$

- no analytic solution
- we can use general-purpose optimization methods

In mixtures parameters are split in two groups:

- $\theta_1, ..., \theta_C$ — parameters of components
- $\pi_1, ..., \pi_C$ — contributions of components

Expectation-Maximization algorithm [Dempster et al., 1977]

Idea: introduce set of hidden variables $\pi_c(x)$

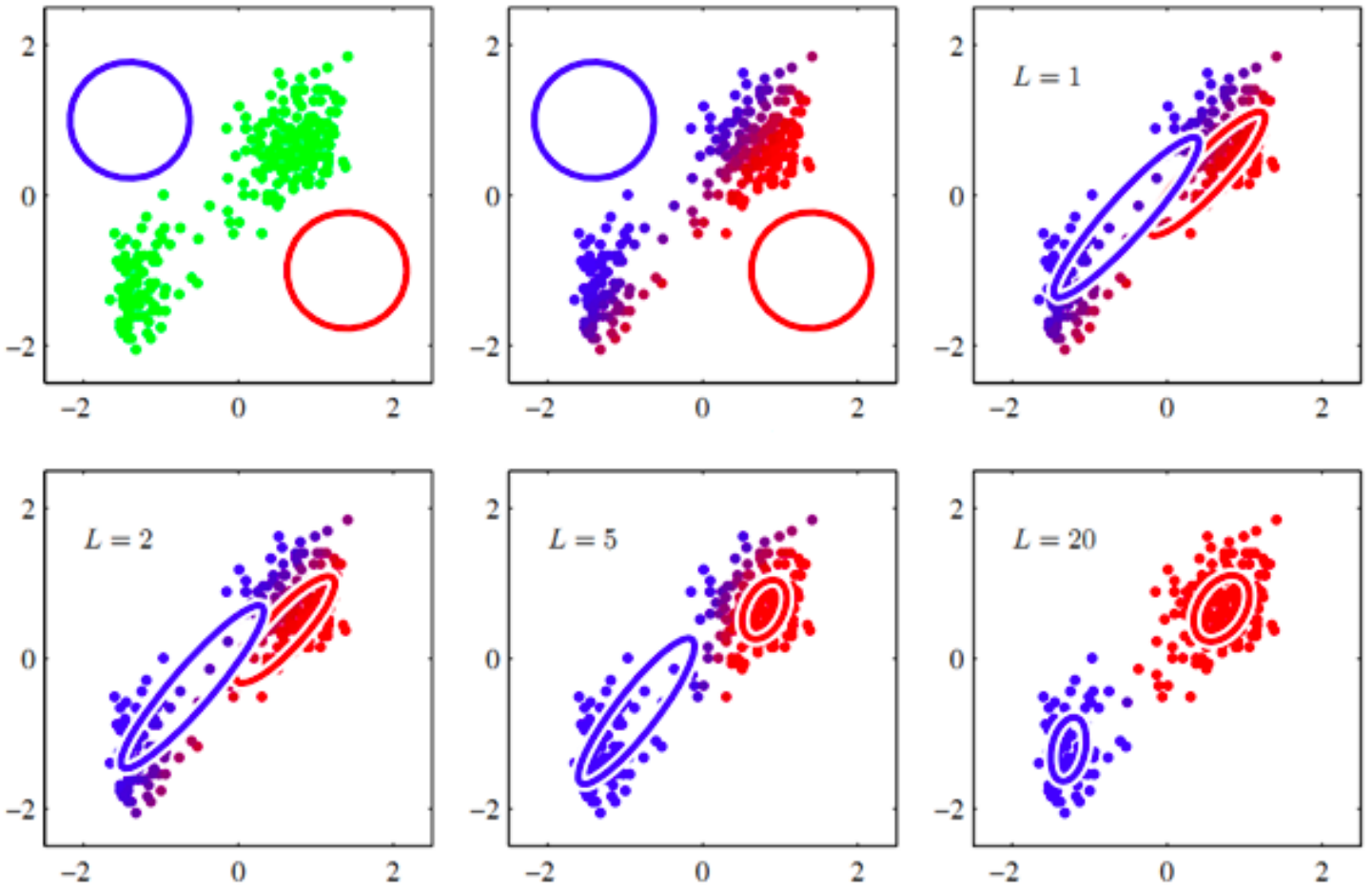
- Expectation: $\pi_c(x) = p(x \in c) = \dfrac{\pi_c f_c(x; \theta_c)}{\sum_{\tilde{c}} \pi_{\tilde{c}} f_{\tilde{c}}(x; \theta_{\tilde{c}})}$
- Maximization:

$$\pi_c = \sum_i \pi_c(x_i)$$
$$\theta_c = \arg\max_\theta \sum_i \pi_c(x_i) \log f_c(x_i, \theta_c)$$

Maximization step is trivial for Gaussian distributions.
EM-algorithm is more stable and has good convergence properties.

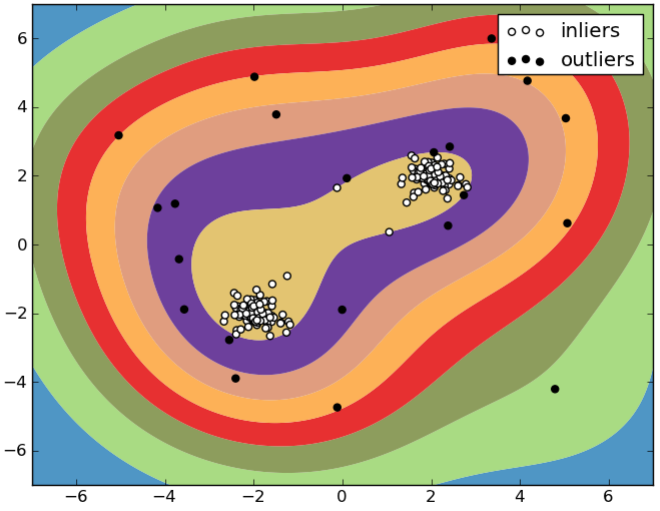
EM algorithm

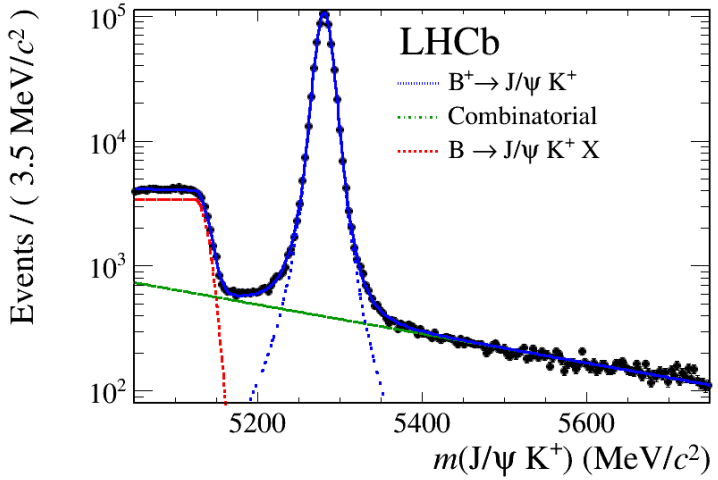
EM algorithm



Density estimation applications

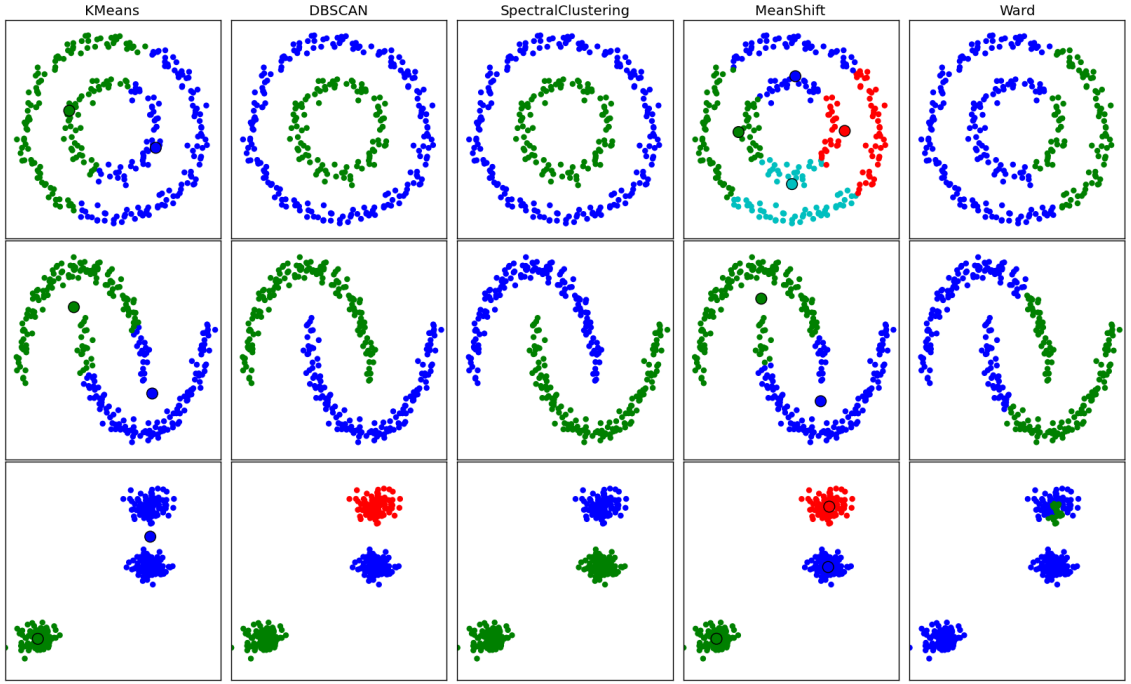
- parametric fits in HEP
  - specifically, mixtures of signal + background
- outlier detection
  - if a new observation has low p w.r.t. to current distribution, it is considered as an outlier
- photometric selection of quasars
- gaussian mixtures are used as a smooth version of clustering (discussed later)





Clustering (unsupervised learning)

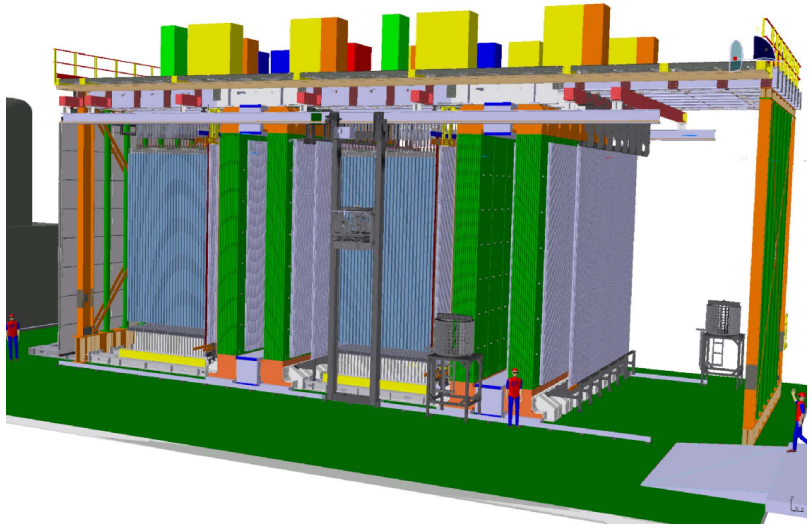
Gaussian mixtures are considered as a smooth form of clustering.
However in some cases clusters have to be defined strictly (that is, each observation can be assigned only to one cluster).



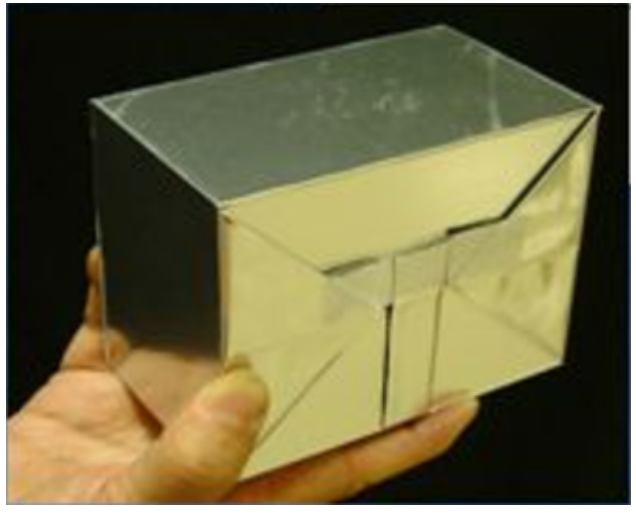
Demo of k-means and DBSCAN clusterings

(shown interactively)

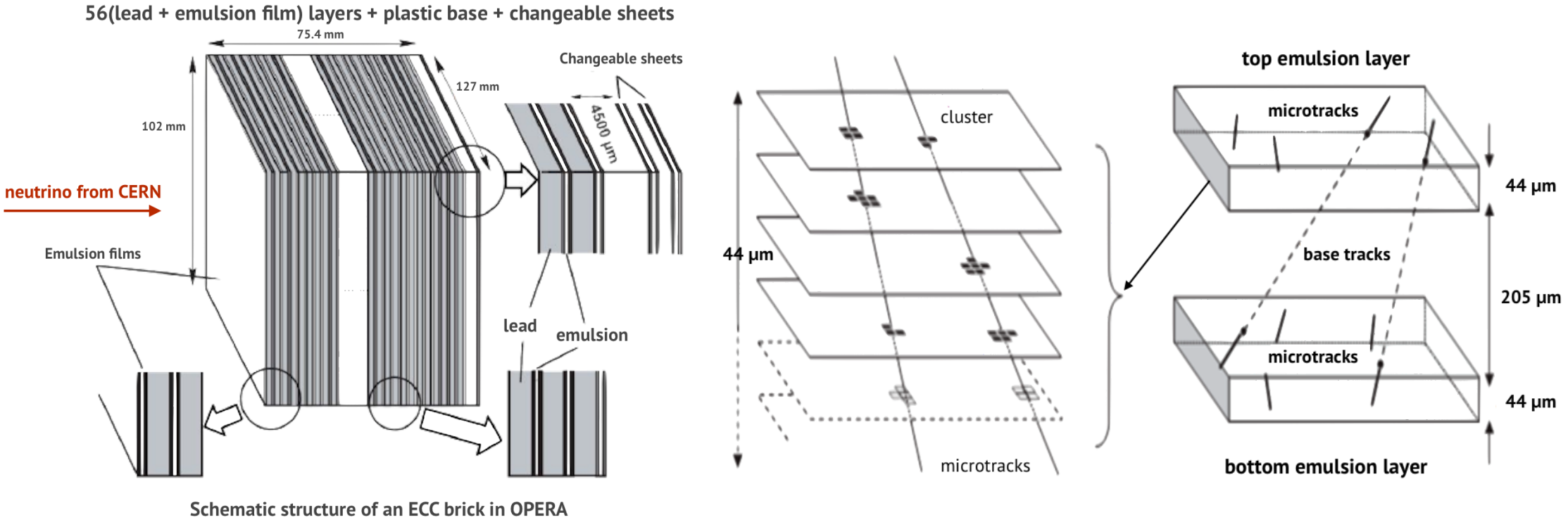
OPERA (operated in 2008-2012)



- Oscillation Project with Emulsion-tRacking Apparatus
- neutrinos sent from CERN to Gran Sasso (INFN, Italy)
- looking for neutrino oscillations (Nobel prize 2015)
  - hence, neutrinos aren't massless
- OPERA detector is placed under the ground,
  consists of 150'000 bricks 8.3 kg each
- brick consists of interleaving photo emulsion and lead layers

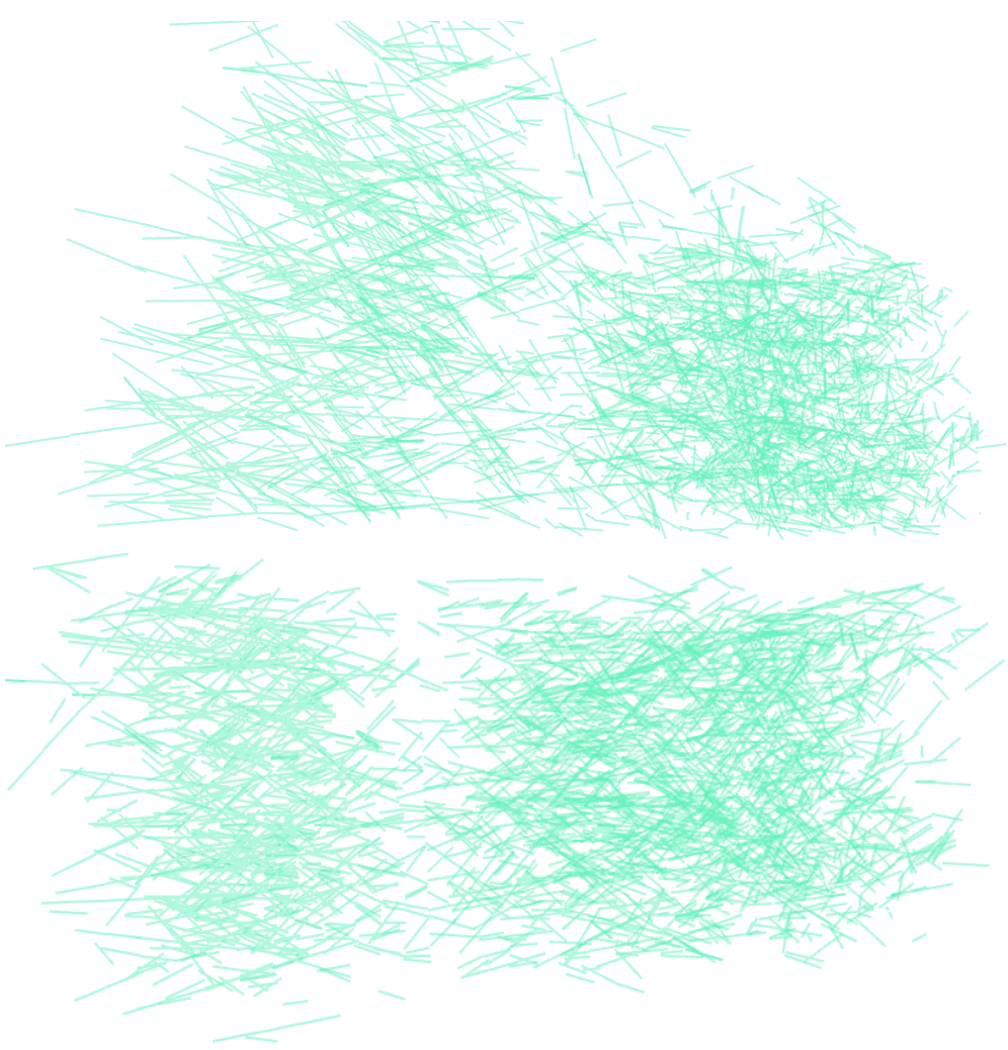


OPERA: brick and scanning



- bricks are taken out and scanned only if there are signs of something interesting
- one brick = billions of base-tracks

OPERA: Tracking with DBSCAN clustering



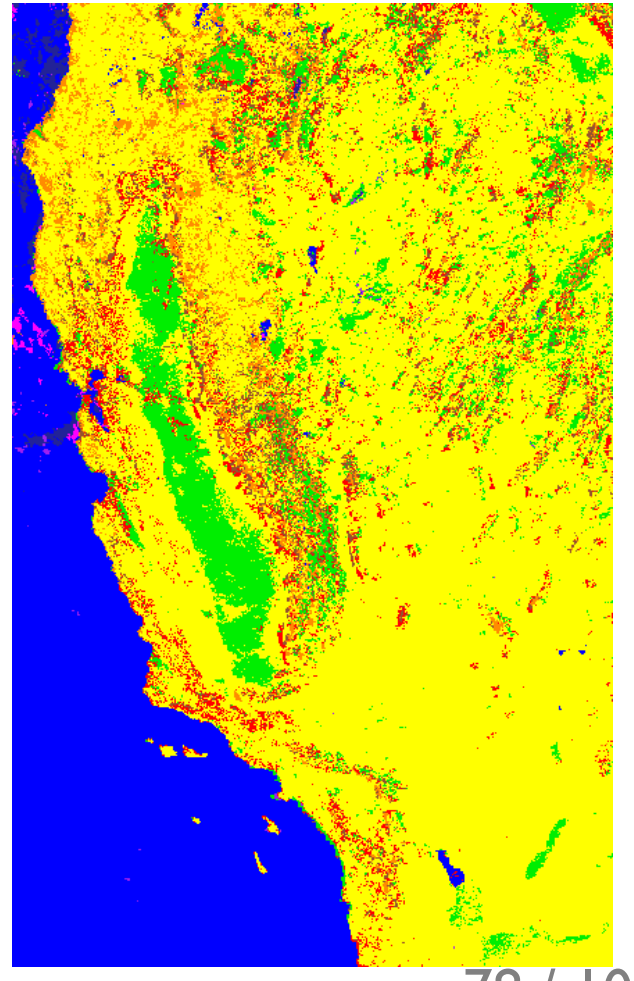
OPERA clustering demo

(shown interactively)

- different metrics can be used in clustering
- the appropriate choice of metric is crucial

Clustering

- news clustering in news aggregators
    - each topic is covered in dozens of media, those are joined in one group to avoid showing lots of similar results
- clustering of surface based on the spectra
  (in the picture, 5 spectral channels for each point were taken from satellite photos)
- clustering in calorimeters in HEP
- clustering of gamma ray bursts
  in astronomy



Classification model based on mixtures density estimation is called *MDA* (mixture discriminant analysis)

Generative approach

Generative approach: trying to reconstruct $p(x, y)$, then use Bayes classification formula to predict.

QDA, MDA are generative classifiers.

Classification model based on mixtures density estimation is called *MDA* (mixture discriminant analysis)

Generative approach

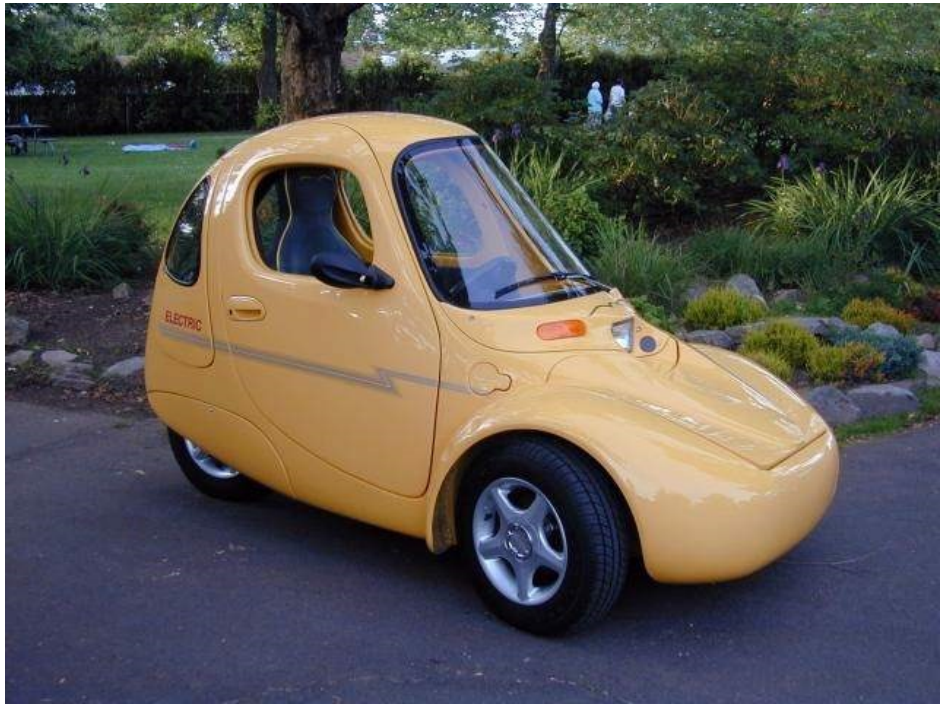
Generative approach: trying to reconstruct $p(x, y)$, then use Bayes classification formula to predict.

QDA, MDA are generative classifiers.

Problems of generative approach

- Real life distributions hardly can be reconstructed
- Especially in high-dimensional spaces
- So, we switch to *discriminative* approach: guessing directly $p(y|x)$

Classification: truck vs car



Density estimation requires reconstructing dependency between all known parameters, which requires much data, but most of these dependencies aren't needed in application.

If we can avoid multidimensional density estimation, we'd better do it.

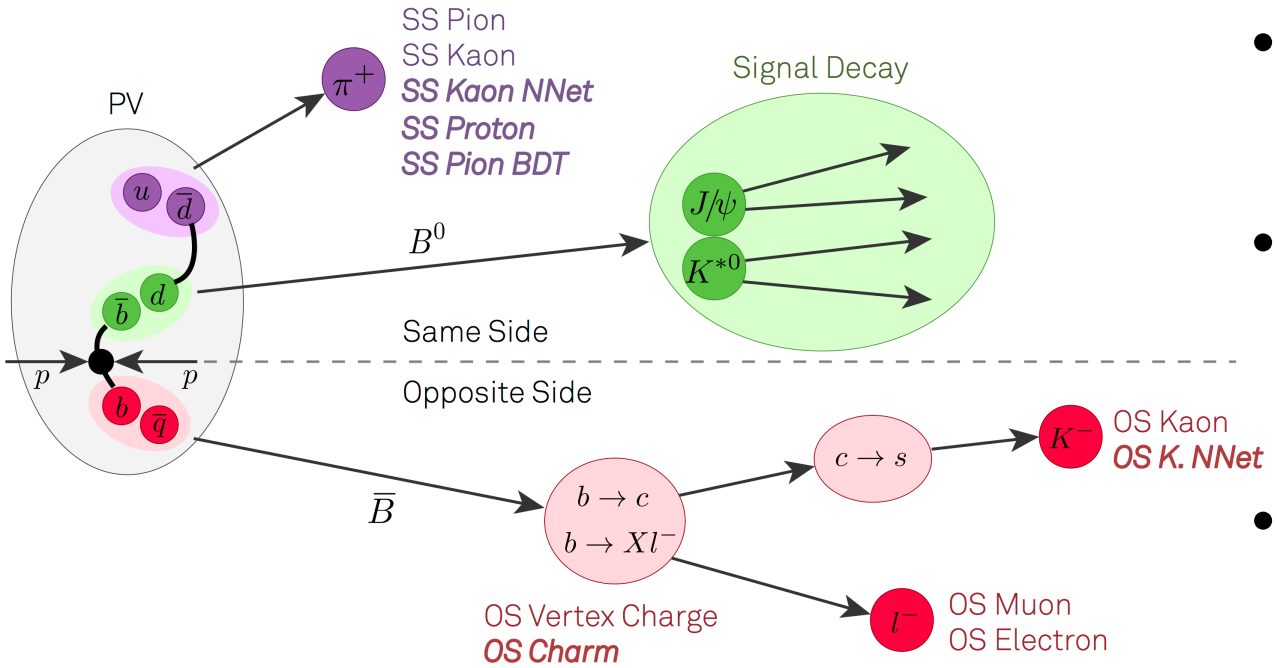
Naive Bayes classification rule

Assuming that features are independent for each class, one can avoid multidimensional density fitting:

$$\frac{p(y=1\,|\,x)}{p(y=0\,|\,x)} = \frac{p(y=1)}{p(y=0)}\frac{p(x\,|\,y=1)}{p(x\,|\,y=0)} = \frac{p(y=1)}{p(y=0)}\prod_i \frac{p(x_i\,|\,y=1)}{p(x_i\,|\,y=0)}$$

Contribution of each feature is independent, one-dimensional fitting is much easier.

Application: B-tagging (picture from LHCb)



- flavour tagging is important in estimating CKM-matrix elements
- to predict which meson was produced (B^0 or $\bar{B}^0$), non-signal part of collision is analyzed (tagging)
- signal part (green) provides information about decayed state
- there are dozens of tracks that aren't part of this scheme

Inclusive tagging idea

- use all (non-signal) tracks from collision
- let machine guess which tracks are tagging, how their charge is connected to meson flavour, and estimate probability

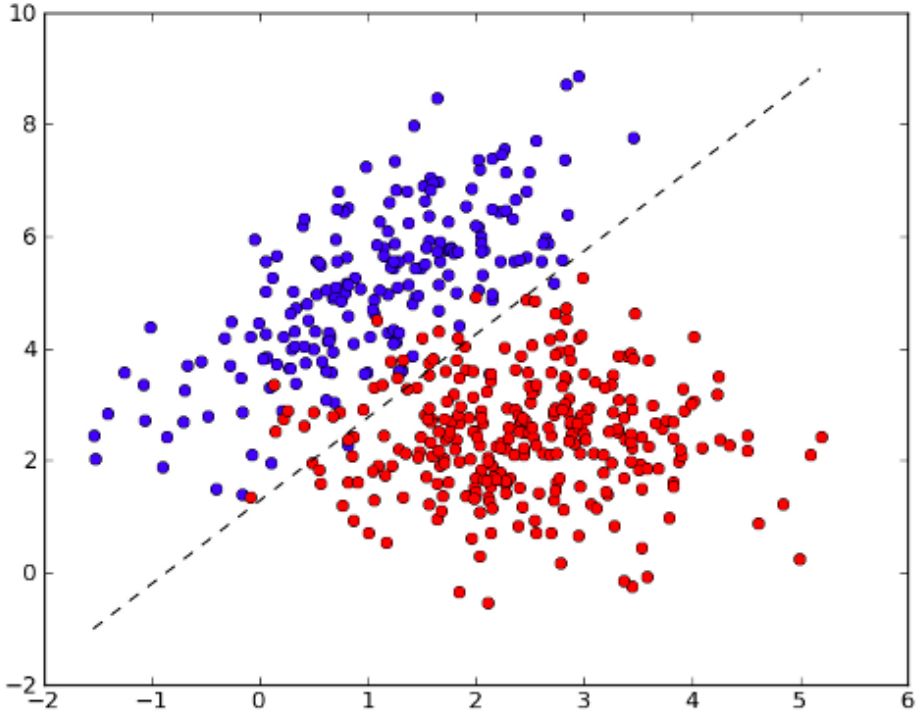
We can make a naive assumption, that information from tracks is independent:

$$\frac{p(B^0|tracks)}{p(\bar{B}^0|tracks)} = \frac{p(B^0)}{p(\bar{B}^0)} \prod_{track} \frac{p(track|B^0)}{p(track|\bar{B}^0)} = \left(\frac{p(\bar{B}^0)}{p(B^0)}\right)^{n_{\text{tracks}}-1} \prod_{track} \frac{p(B^0|track)}{p(\bar{B}^0|track)}$$

Number of terms in the product varies (compared to typical Naive Bayes).
Ratio for each track in the right side is estimated with ML techniques discussed later.

This simple technique provides very good results.

Linear decision rule



Decision function is linear:

$$d(x) =< w, x > +w_0$$

$$\begin{cases} d(x) > 0 \rightarrow & \hat{y} = +1 \\ d(x) < 0 \rightarrow & \hat{y} = -1 \end{cases}$$

for brevity:

$$\hat{y} = \text{sgn}(d(x))$$

This is a parametric model (finding parameters w, w_0).
QDA & MDA are parametric as well.

Finding Optimal Parameters

- A good initial guess: get such w, w_0, that error of classification is minimal:

$$\mathcal{L} = \sum_{i \in \text{samples}} \mathbf{1}_{y_i \neq \hat{y}_i} \qquad \hat{y}_i = \text{sgn}(d(x_i))$$

  Notion: $\mathbf{1}_{true} = 1,\ \mathbf{1}_{false} = 0.$

- Discontinuous optimization (arrrrgh!)

Finding Optimal Parameters

- A good initial guess: get such w, w_0, that error of classification is minimal:

$$\mathcal{L} = \sum_{i \in \text{samples}} \mathbf{1}_{y_i \neq \hat{y}_i} \qquad \hat{y}_i = \text{sgn}(d(x_i))$$

  Notion: $\mathbf{1}_{true} = 1,\ \mathbf{1}_{false} = 0.$

- Discontinuous optimization (arrrrgh!)
  - solution: let's make decision rule **smooth**

$$p(y = +1|x) = p_{+1}(x) = f(d(x))$$
$$p(y = -1|x) = p_{-1}(x) = 1 - p_{+1}(x)$$

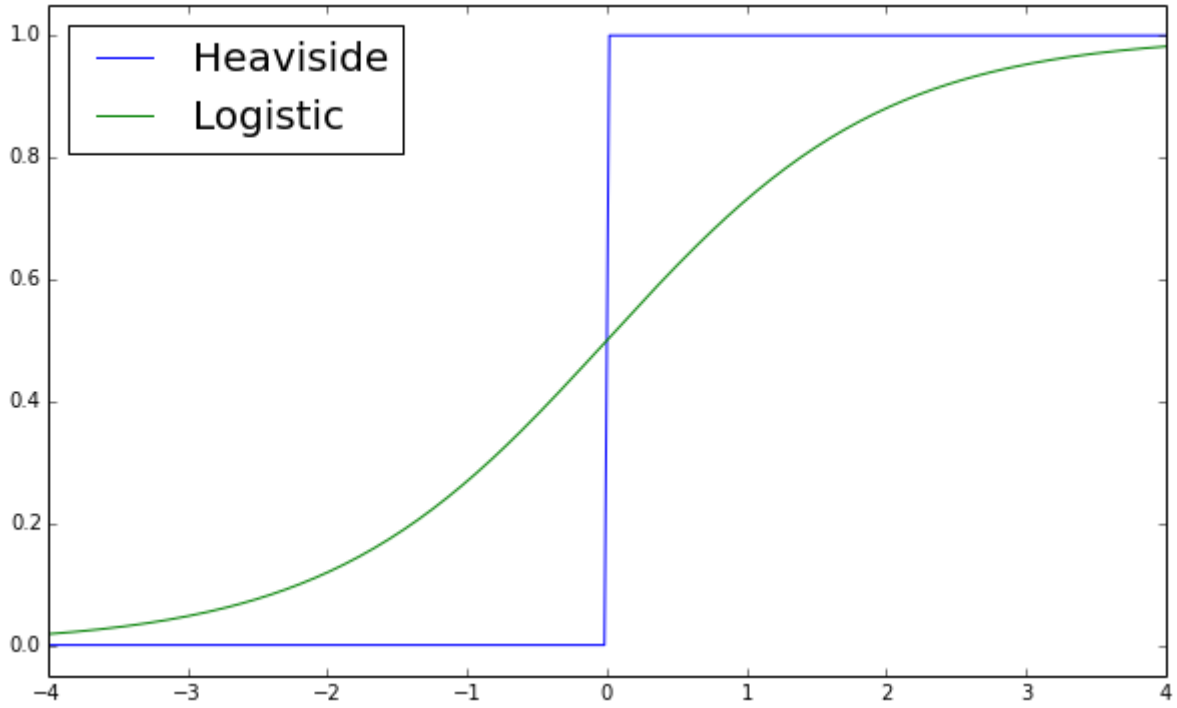
$$\begin{cases} f(0) = 0.5 \\ f(x) > 0.5 & x > 0 \\ f(x) < 0.5 & x < 0 \end{cases}$$

Logistic function

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

Properties

1. monotonic, $\sigma(x) \in (0,1)$
2. $\sigma(x) + \sigma(-x) = 1$
3. $\sigma'(x) = \sigma(x)(1-\sigma(x))$
4. $2\sigma(x) = 1 + \tanh(x/2)$



Logistic regression

Define probabilities obtained with logistic function

$$\begin{aligned} p_{+1}(x) &= \sigma(d(x)) \\ p_{-1}(x) &= \sigma(-d(x)) \end{aligned} \qquad \text{with } d(x) =< w, x > + w_0$$

and optimize log-likelihood:

$$\mathcal{L} = \sum_{i \in \text{observations}} -\ln(p_{y_i}(x_i)) = \sum_i L(x_i, y_i) \to \min$$

Logistic loss (LogLoss)

Term *loss* refers to somewhat we are minimizing. Losses typically estimate our risks, denoted as $\mathcal{L}$.

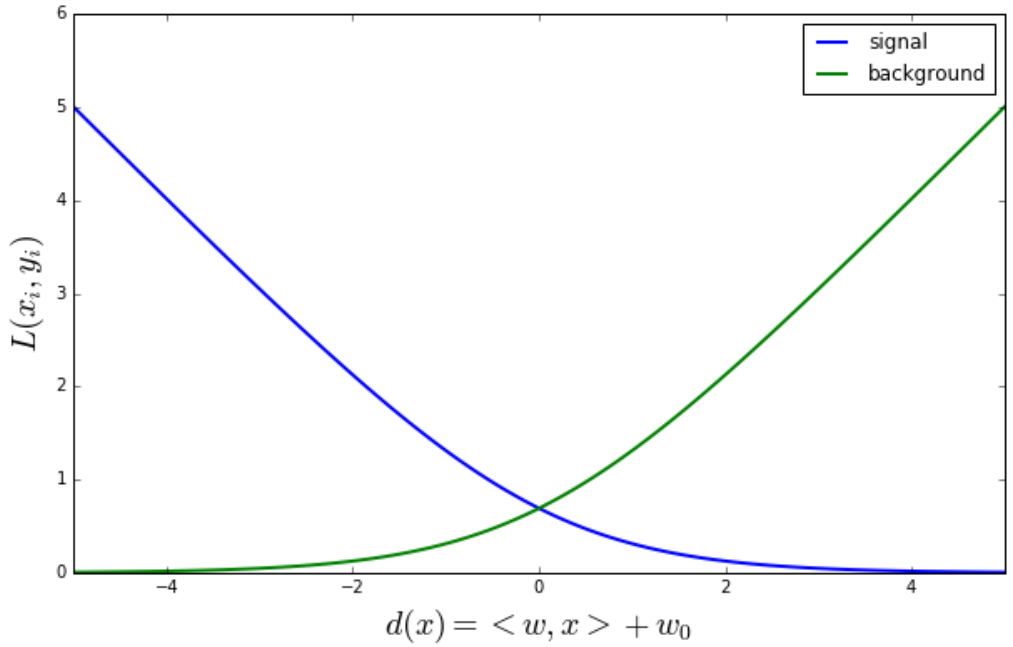
$$\mathcal{L} = \sum_{i \in \text{observations}} -\ln(p_{y_i}(x_i)) = \sum_i L(x_i, y_i) \to \min$$

LogLoss penalty for single observation:

$$L(x_i, y_i) = -\ln(p_{y_i}(x_i)) = \begin{cases} \ln(1 + e^{-d(x_i)}), & y_i = +1 \\ \ln(1 + e^{d(x_i)}), & y_i = -1 \end{cases} = \ln(1 + e^{-y_i d(x_i)})$$

Margin $y_i d(x_i)$ is expected to be high for all observations.

Logistic loss

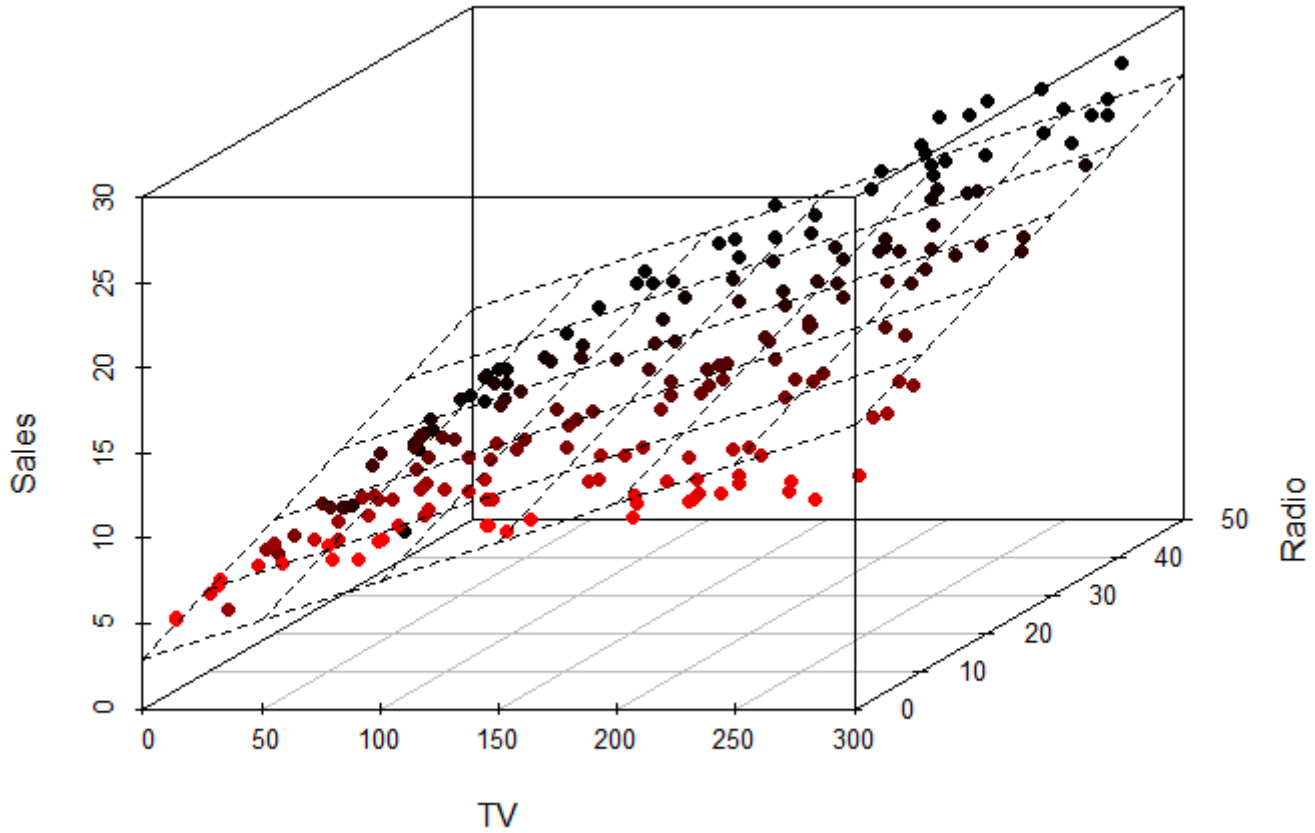


$L(x_i, y_i)$ is a convex function.

Simple analysis shows that $\mathcal{L}$ is a sum of convex functions w.r.t. to w, so the optimization problem has at most **one optimum**.

Visualization of logistic regression

Linear model for regression



How to use linear function for regression?

$$d(x) =< w, x > +w_0$$

Simplification of notion:
$x_0 = 1, x = (1, x_1, ..., x_d)$.

$$d(x) =< w, x >$$

Linear regression (ordinary least squares)

We can use linear function for regression:

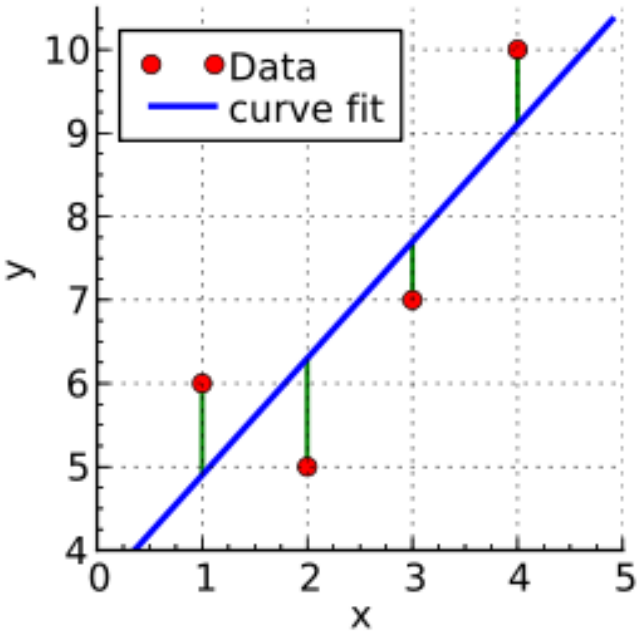
$$d(x_i) = y_i \qquad d(x) =< w, x >$$

This is a linear system with $d + 1$ variables and n equations.

Minimize *OLS* aka *MSE* (mean squared error):

$$\mathcal{L} = \sum_i (d(x_i) - y_i)^2 \to \min$$

Explicit solution: $\left(\sum_i x_i x_i^T\right) w = \sum_i y_i x_i$



Linear regression

- can use some other loss $\mathcal{L}$ different from MSE
  - but no explicit solution in other cases

- demonstrates properties of linear models
  - reliable estimates when $n >> d$
  - able to completely fit to the data if $n = d$
  - undefined when $n < d$

Regularization: motivation

When the number of parameters is high (compared to the number of observations)

- hard to estimate reliably all parameters
- linear regression with MSE:
  - in d-dimensional space you can find hyperplane through any d points
  - non-unique solution if $n < d$
  - the matrix $\sum_i x_i x_i^T$ degenerates

Solution 1: manually decrease dimensionality of the problem (select more appropriate features)

Solution 2: use regularization

Regularization

When number of parameters in model is high, overfitting is very probable

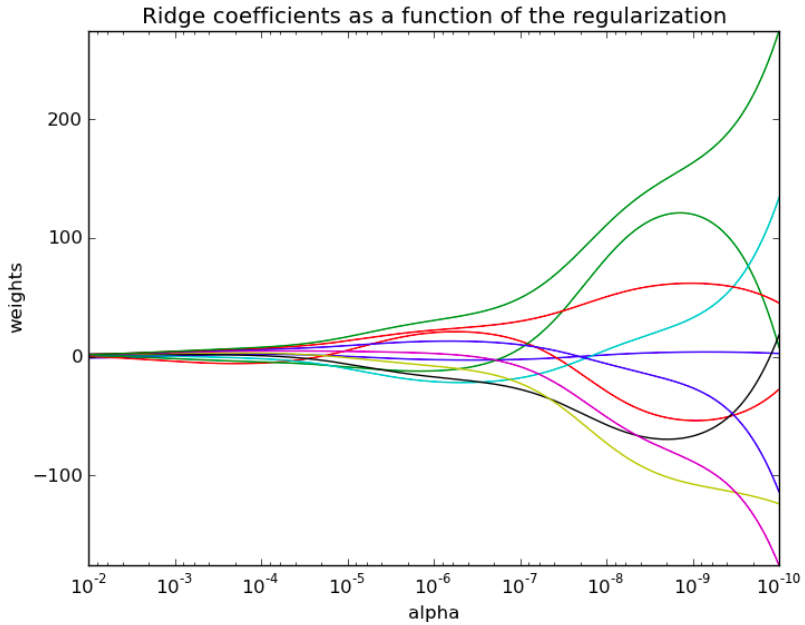
Solution: add a *regularization term* to the loss function:

$$\mathcal{L} = \frac{1}{N} \sum_i L(x_i, y_i) + \mathcal{L}_{\text{reg}} \to \min$$

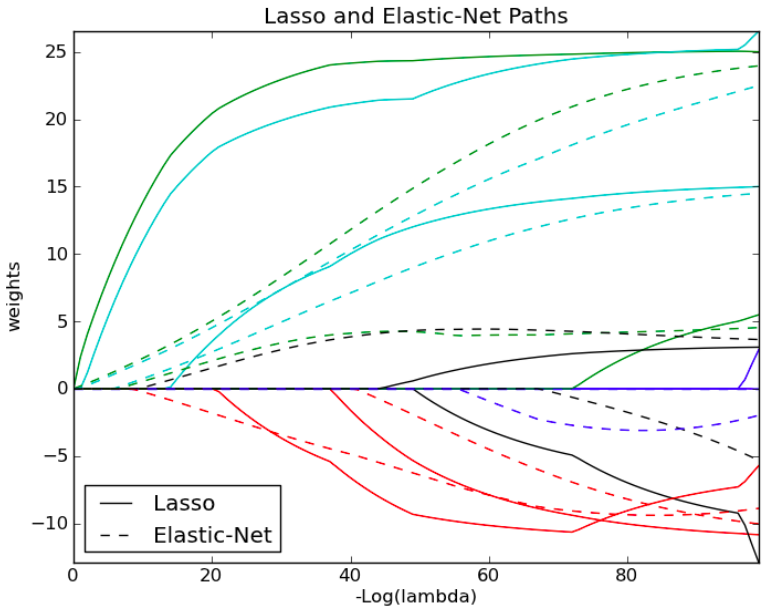
- L_2 regularization: $\mathcal{L}_{\text{reg}} = \alpha \sum_j |w_j|^2$
- L_1 regularization: $\mathcal{L}_{\text{reg}} = \beta \sum_j |w_j|$
- $L_1 + L_2$ regularization: $\mathcal{L}_{\text{reg}} = \alpha \sum_j |w_j|^2 + \beta \sum_j |w_j|$

L_2, L_1 – regularizations

Dependence of parameters (components of w) on the regularization (stronger regularization to the left)



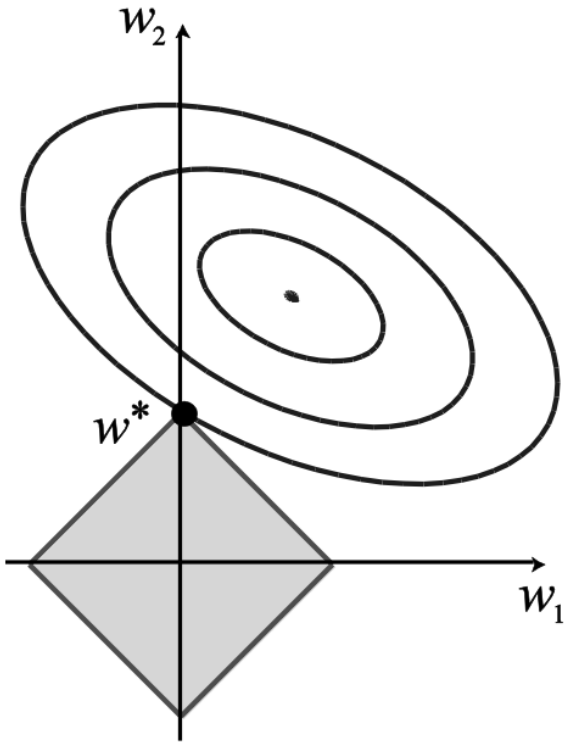
L_2 regularization



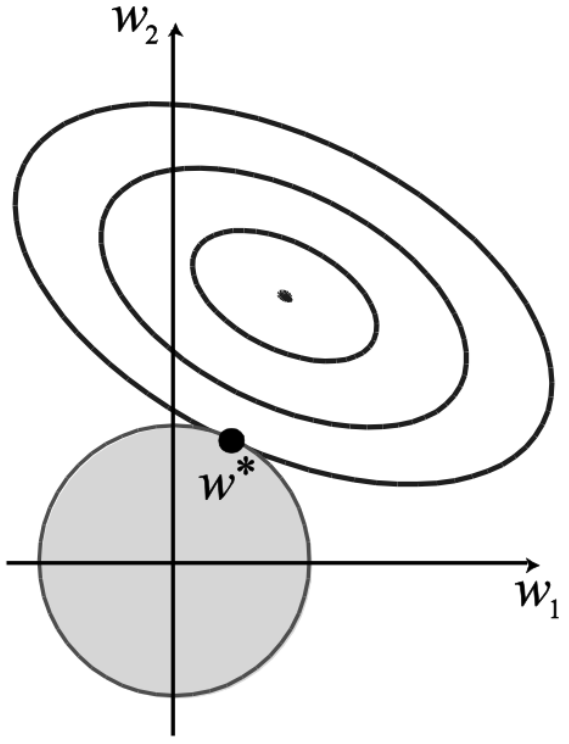
L_1 (solid), $L_1 + L_2$ (dashed)

Regularizations

L_1 regularization encourages sparsity (many coefficients in w turn to zero)



L1

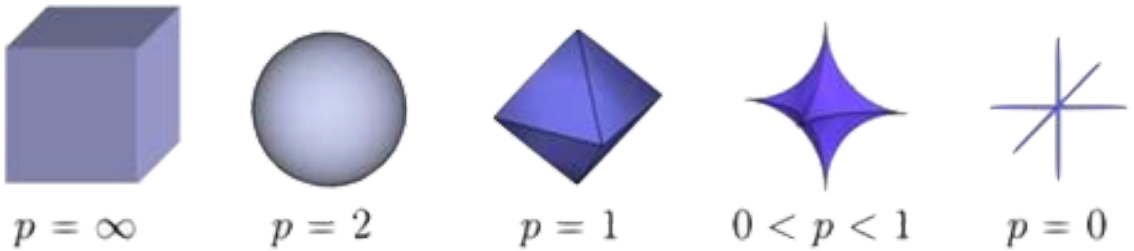


L2

L_p regularizations

We can consider a more general case:

$$\mathcal{L}_p = \sum_i w_i^p$$



Expression for L_0: $L_0 = \sum_i 1_{w_i \neq 0}$

- exactly penalizes number of non-zero coefficients
- but nobody uses L_0. Why?

L_p regularizations

We can consider a more general case:

$$\mathcal{L}_p = \sum_i w_i^p$$

Expression for L_0: $L_0 = \sum_i 1_{w_i \neq 0}$

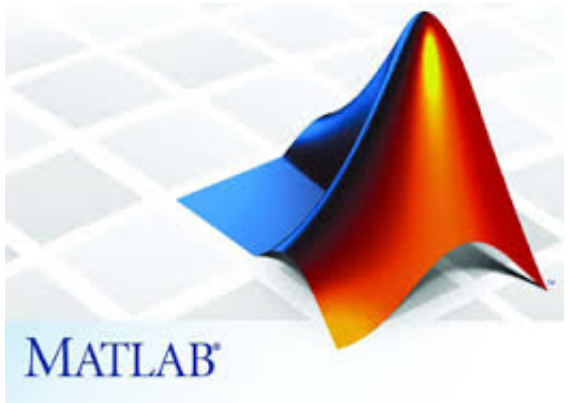
- exactly penalizes number of non-zero coefficients
- but nobody uses L_0. Why?
- even $L_p,\ 0 < p < 1$ not used. Why?

Regularization summary

- important tool to fight overfitting (= poor generalization on a new data)
- different modifications for other models
- makes it possible to handle really many features
- machine learning should detect important features itself
- from mathematical point:
  - turning convex problem to strongly convex (NB: only for linear models)
- from practical point:
  - softly limiting the space of parameters
- breaks scale-invariance of linear models

n-minutes break

Data Scientist Tools



- Experiments in appropriate high-level language or environment
- After experiments are over — implement final algorithm in low-level language (C++, CUDA, FPGA)

Second point is typically unnecessary

Scientific Python

NumPy

vectorized computations in python

Matplotlib

for drawing

Scikit-learn

most popular library for machine learning

Scientific Python

Scipy

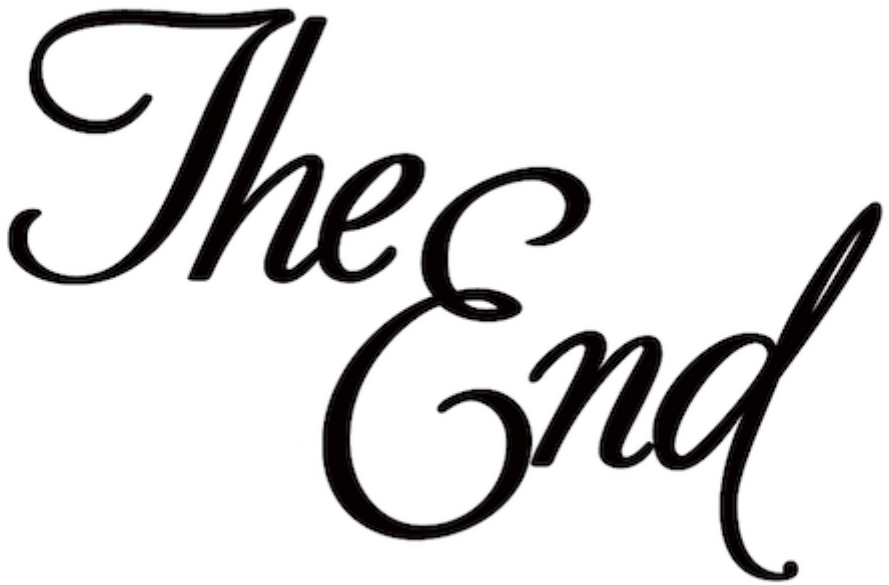
libraries for science and engineering

Root_numpy

convenient way to work with ROOT files

Astropy

A community python library for astronomy


The End