UNIL | Université de Lausanne

Intro to Machine Learning, and basics on Gaussian Process Regression

Simon Scheidegger
simon.scheidegger@unil.ch
January 23th, 2019

Cowles Foundation – Yale University

Today's Roadmap

I. Detour: What is Machine Learning? (Demystifying ML)

- Supervised Machine Learning
- Unsupervised Machine Learning
- Machine Learning in Dynamic Economic Models?

II. Introduction to Gaussian Process Regression

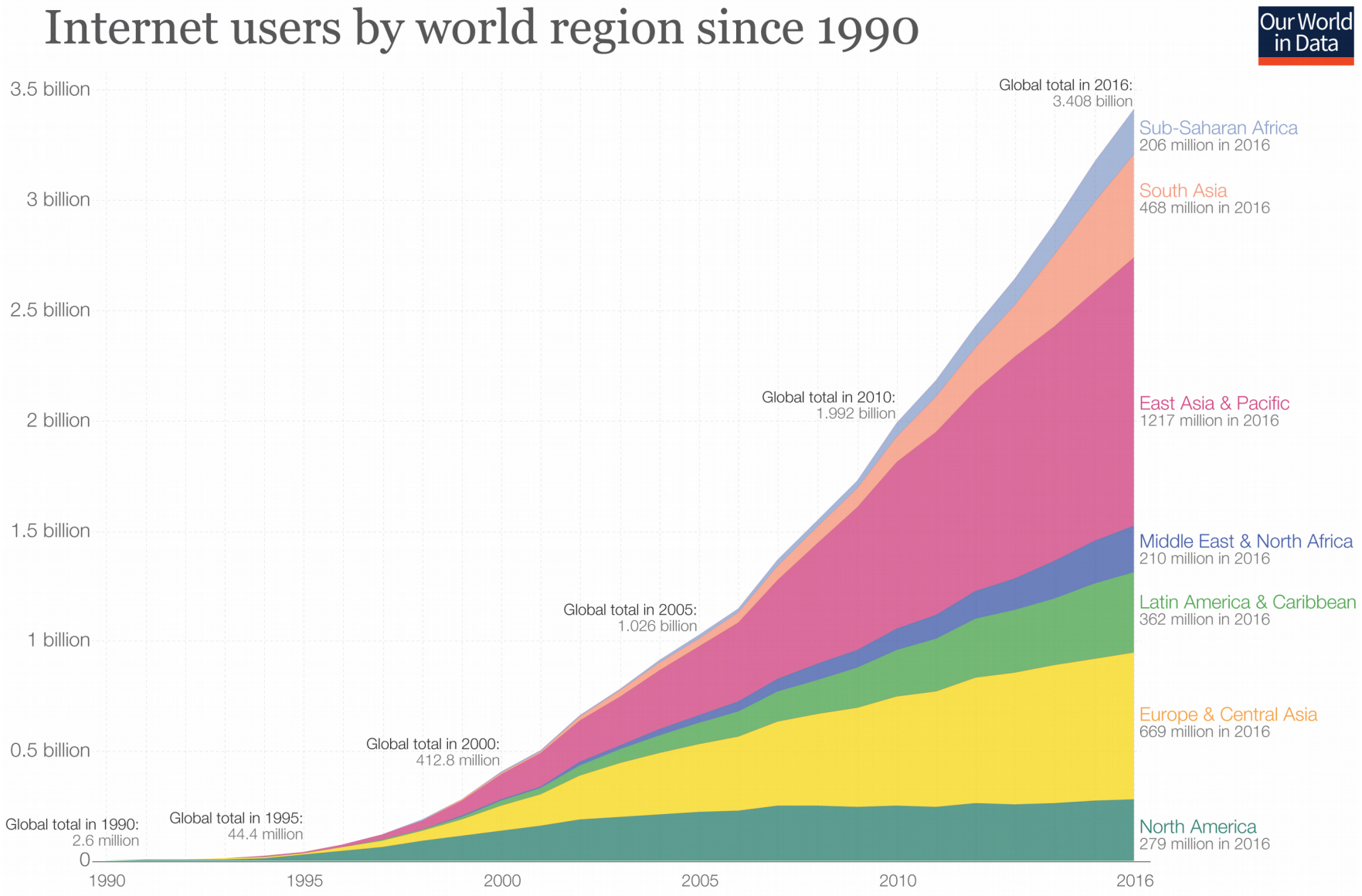
- Recall Gaussian Distributions
- Predictions using noise-free observations

We are drowning in information and starving for knowledge.

–John Naisbitt

Big Data and it's availability

https://ourworldindata.org/internet



Data source: Based on data from the World Bank and data from the International Telecommunications Union. Internet users are people with access to the worldwide network.
The interactive data visualization is available at OurWorldinData.org. There you find the raw data and more visualizations on this topic.

Big Data and it's availability (cont'd)

http://www.live-counter.com/how-big-is-the-internet

Size of the internet as we speak: **14.237.572 Petabytes**

1 Gigabyte	~ 1000 MB
1 Terabyte	~ 1000 GB
1 Petabyte	~ 1000 TB
1 Exabyte	~ 1000 PB
1 Zettabyte	~ 1000 EB

1 Gigabyte: An author takes 50 years for writing every week a book with about 190 pages, more specifically, with 383,561 characters (with spaces and sentence included). This would be a billion letters or bytes.

1 Exabyte: 212 million DVDs weighing 3,404 tons.

1 Zettabyte: 1,000,000,000,000,000,000.000 bytes or characters. Printed on graph paper (with one in letter in each mm^2 square) would be a paper measuring a billion km.The entire surface of the earth (510 million km^2) would be covered by a layer of paper almost twice.

Other sources of “Big Data”

- **Scientific experiments**
  - Cern (e.g., LHC)
    generates ~ **25 petabytes** per year (2012).
  - LIGO
    generates ~ **1 Petabyte** per year.

https://home.cern/

- **Numerical computations**

https://www.ligo.caltech.edu/

- …

https://www.olcf.ornl.gov/summit/

The need for Data Analytics

- Widespread use of personal computers and wireless communication leads to “big data”.
- We are both producers and consumers of data.
- Data is not random, it has structure, e.g., customer behavior.
- We need “big theory” to extract that structure from data for
  (a) Understanding the process.
  (b) Making predictions for the future.

→ **We need Data Analytics**

The purpose of Data Analytics

*Xia, B. S., & Gong, P. (2015). Review of business intelligence through data analysis. Benchmarking, 21(2), 300-311. doi:10.1108/BIJ-08-2012-0050

*Data analysis is a process of

- **inspecting**
- **cleansing**
- **transforming**
- **modeling data**

with the goal of **discovering useful information**, informing conclusions, and **supporting decision-making**.

Data analysis has multiple facets and approaches, encompassing diverse techniques under a variety of names, while being used in different business, science, and social science domains.

In today's business, data analysis is playing a role in making decisions more scientific and helping the business achieve effective operation.

Data Mining

- Data mining is the process of discovering patterns in large data sets involving methods at the intersection of machine learning, statistics, and database systems.

- **Retail:** Market basket analysis, Customer relationship management (CRM).
- **Finance:** Credit scoring, fraud detection.
- **Manufacturing:** Control, robotics, troubleshooting.
- **Medicine:** Medical diagnosis.
- **Telecommunications:** Spam filters, intrusion detection.
- **Bio-informatics:** Motifs, alignment.
- **Web mining:** Search engines.

Why Machine Learning?

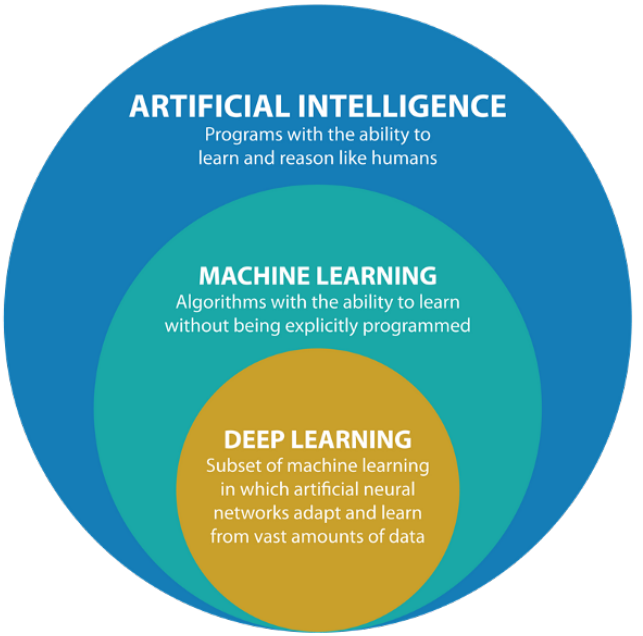
- Machine learning aims at gaining insights from data and **making predictions** based on it.
- Build a model that is a good and useful approximation to the data.
- Machine learning methods have been investigated for more than 60 years, but became mainstream only recently due to more data being available and advances in computing power ("*Moore's Law*").
- There is no need to "learn" to calculate for example the payroll.
- Learning is used when:
  - Human expertise does not exist (navigating on Mars).
  - Humans are unable to explain their expertise (speech recognition).
  - Solution changes in time (routing on a computer network).
  - Solution needs to be adapted to particular cases (user biometrics).
- You're relying on machine learning every day, maybe without being aware of it! → you certainly use a Smart Phone?

Some terminology

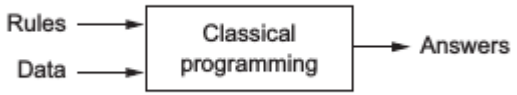
Figs. from F. Chollet (2018) – ML with Python

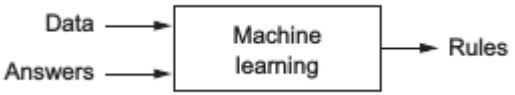
Artificial intelligence (AI)

Can computers be made to “think”—a question whose ramifications we’re still exploring today.
A concise definition of the AI field would be as follows:
the effort to automate intellectual tasks normally performed by humans.

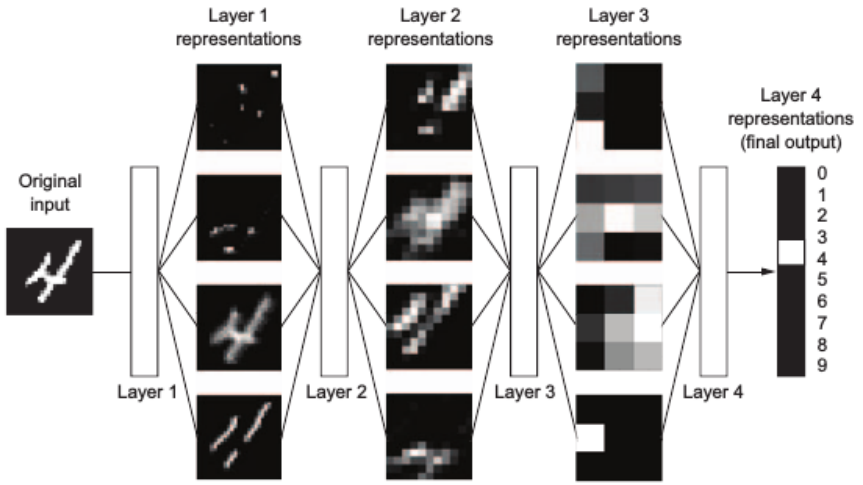


Machine learning (ML)



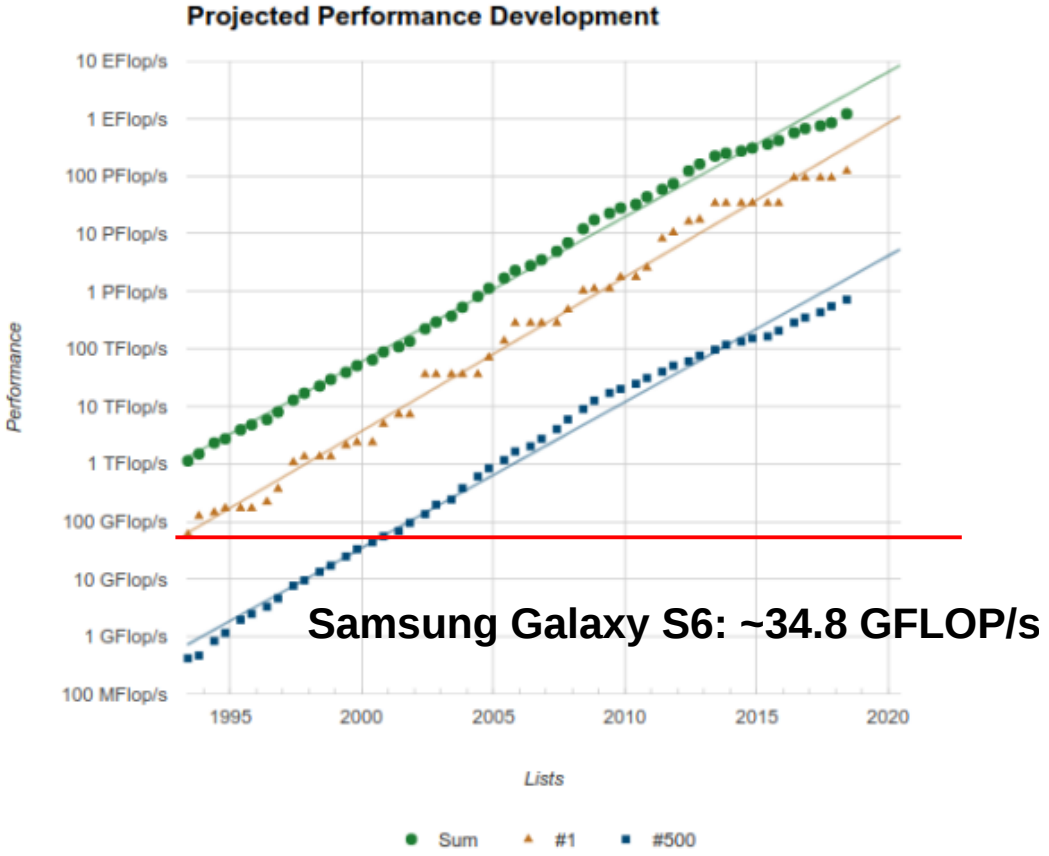


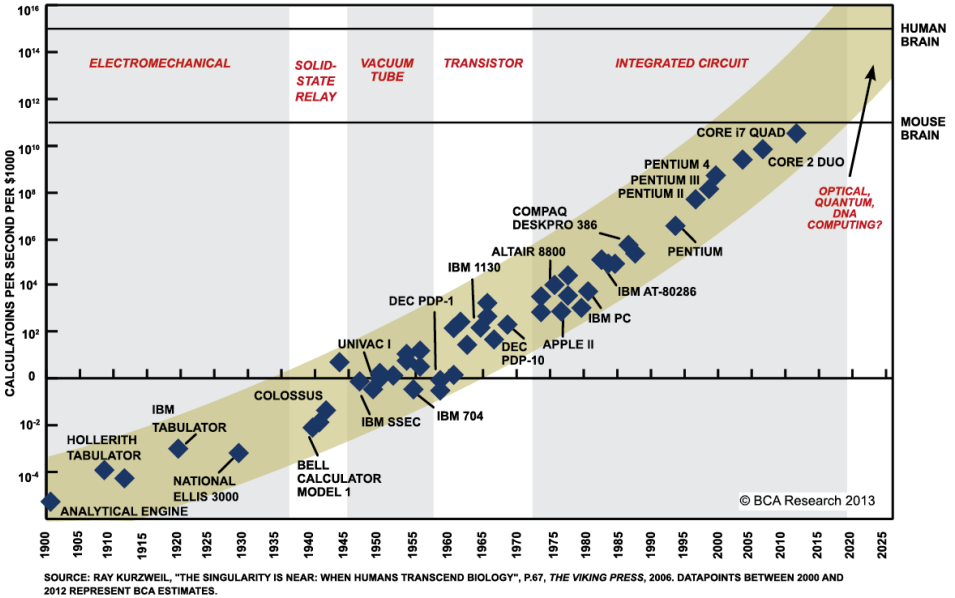
Deep Learning as a particular example of an ML technique



Moore's Law

http://www.extremetech.com/extreme/210872-extremetech-explains-what-is-moores-law
https://www.top500.org





→ Exponential growth in compute power allows us to move away from stylized models towards "realistically-sized" problems.

Structured Data

- We often deal with structured data.
- Example: Data about monthly rent of a real estate in a city.

Area	EstateType	DistanceToCenter	EnergyClass	MonthlyRent
58.20	Apartment	1.1	A	450
122.20	House	5.6	A	620
28.00	Apartment	0.5	B	320
8.00	Storage	6.3	B	150
75.78	House	2.2	C	375
66.00	Office	1.6	A	525
10.00	Storage	9.2	D	120

Unstructured and Semi-structured Data

- Many interesting datasets are unstructured, typically natural language texts written by humans, or semi-structured, interleaving structured and unstructured data.
- Example: Newspaper articles with assigned categories.

Category	Content
Sports	Bayern Munich was defeated by Real Madrid in the Champions League quarter finals...
Politics	Theresa May called for a snap general election on Monday...

Feature Types (I)

- When working with structured data, we distinguish different types of features, depending on which operations can be applied.

Area	EstateType	DistanceToCenter	EnergyClass	MonthlyRent
58.20	Apartment	1.1	A	450
122.20	House	5.6	A	620
28.00	Apartment	0.5	B	320
8.00	Storage	6.3	B	150
75.78	House	2.2	C	375
66.00	Office	1.6	A	525
10.00	Storage	9.2	D	120

Dataset consists of five features

Feature Types (II)

- **Nominal** features can be compared (=, !=) and counted (e.g., color of a car).
- **Ordinal** features can in addition be compared (<, >) (e.g., customer satisfaction level, energy class of car).
- **Numerical** features allow in addition for arithmetic operations (+, -, *, /), so that we can compute the difference between values, compute their mean, compute their variance etc. (e.g., fuel consumption of a car, income of a household).

Feature Types (III)

Feature types are different from data types in databases or programming languages, i.e., we don't care about aspects such as precision or character set.

Area	EstateType	DistanceToCenter	EnergyClass	MonthlyRent
58.20	Apartment	1.1	A	450
122.20	House	5.6	A	620
28.00	Apartment	0.5	B	320
8.00	Storage	6.3	B	150
75.78	House	2.2	C	375
66.00	Office	1.6	A	525
10.00	Storage	9.2	D	120
Numerical	**Nominal**	**Numerical**	**Ordinal**	**Numerical**

Applications of ML

- **Supervised Learning**
  assume that training data is available from which we can learn to predict a target feature based on other features (e.g., monthly rent based on area).

  → **Classification**
  → **Regression**

- **Unsupervised Learning**
  take a given dataset and aim at gaining insights by identifying patterns, e.g., by grouping similar data points.

- **Reinforcement Learning**

Supervised Regression – an example

- Regression aims at predicting a numerical target feature based on one or multiple other (numerical) features.

- Example: Price of a used car.

x : car attributes
y : price
$y = h(x \mid \theta)$
h ( ) : model
θ : parameters

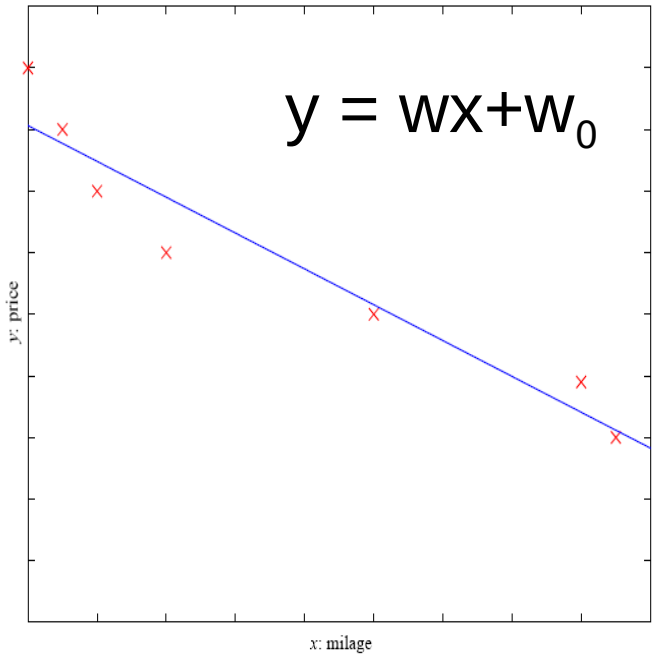


Fig. from Alpaydin (2014)

Supervised Classification – an example

Example 1: Spam Classification

- Decide which emails are Spam and which are not.
- Goal: Use emails seen so far to produce a good prediction rule for **future** data.

Example 2: Credit Scoring

Differentiating between low-risk and high-risk customers from their income and savings.

Discriminant: IF income > θ_2 AND savings > θ_2
THEN low-risk ELSE high-risk

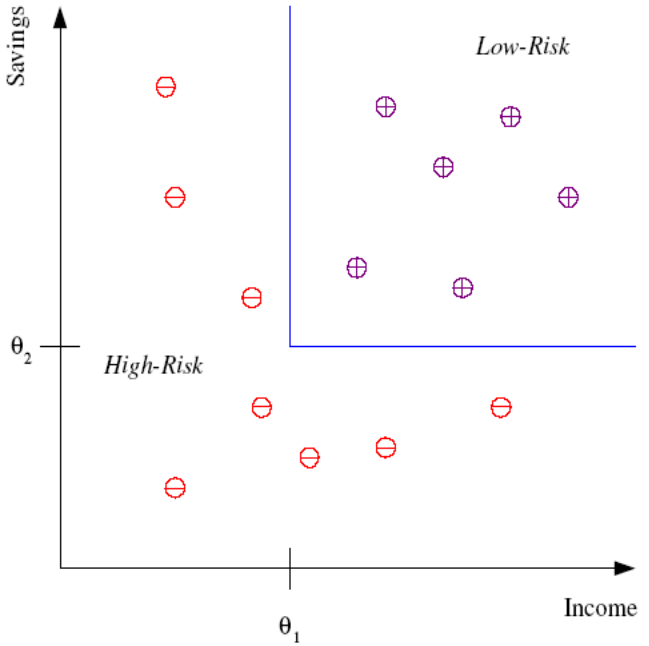
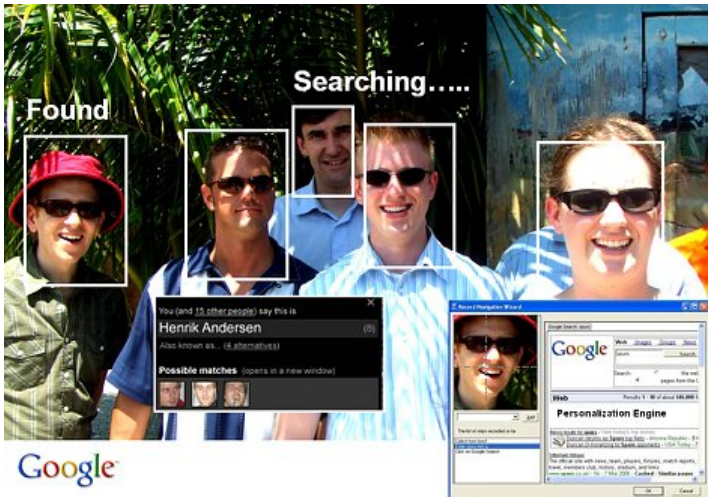


Fig. from Alpaydin (2014)

Classification: More applications

Face recognition:
Pose, lighting, occlusion (glasses, beard), make-up, hair style

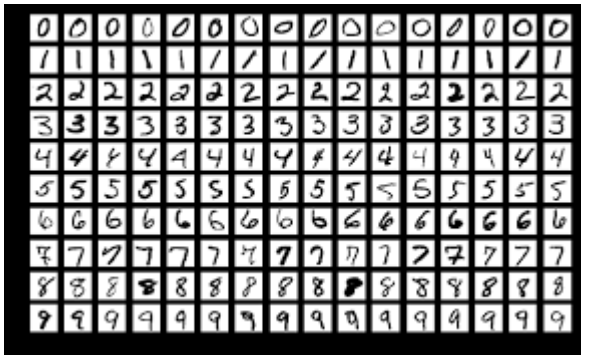


Character recognition:
Different handwriting styles.

Speech recognition:
Temporal dependency.

Medical diagnosis:
From symptoms to illnesses

Biometrics:
Recognition/authentication using physical and/or behavioral characteristics: Face, iris, signature, etc
Outlier/novelty detection:



Random sampling of MNIST

Handwritten Digit Classification

Movie – from the early 90ies

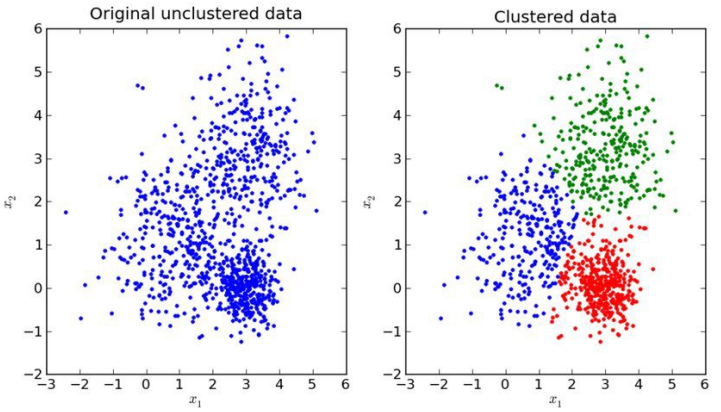
We have come a long way since then...

→ Handwritten Digit Classification – by Yann Lecun

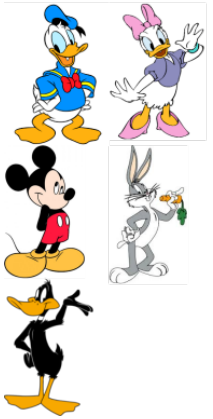
https://www.youtube.com/watch?v=yxuRnBEczUU

Unsupervised Learning

- Learning “what normally happens”.
- No output.
- Clustering: Grouping similar instances.

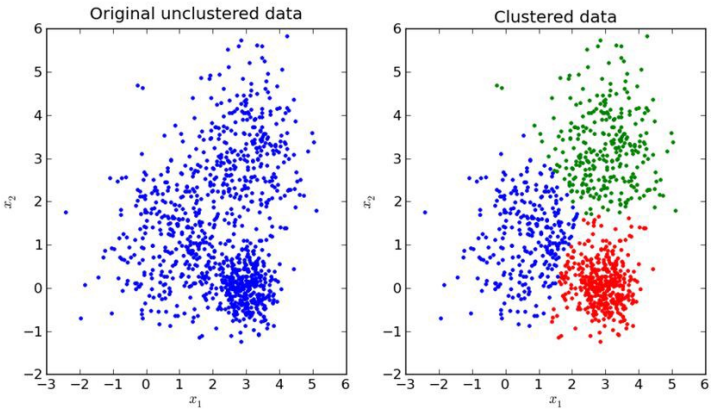


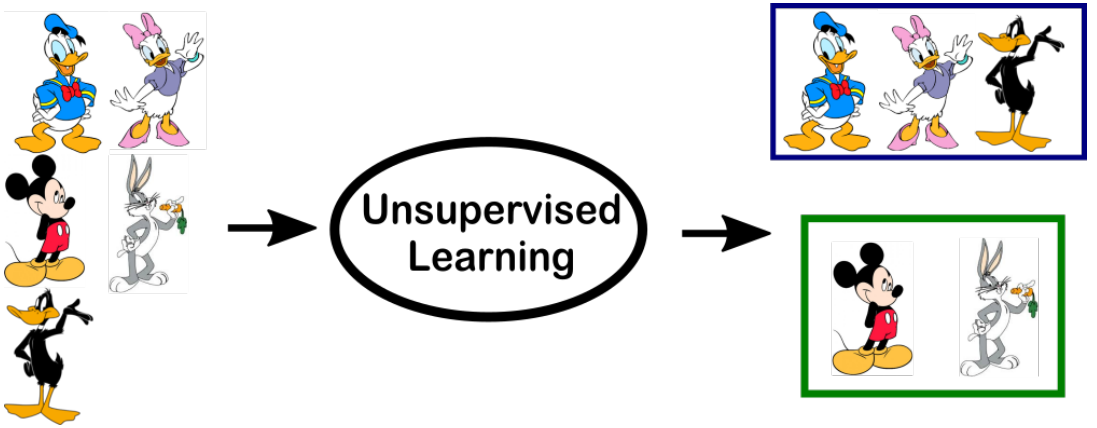
- Example applications:
  - Customer segmentation.
  - Image compression: Color quantization.
  - Bio-informatics: Learning motifs.



Unsupervised Learning

- Learning “what normally happens”.
- No output.
- Clustering: Grouping similar instances.
- Example applications:
  - Customer segmentation.
  - Image compression: Color quantization.
  - Bio-informatics: Learning motifs.





Reinforcement Learning

- Learning a policy: A sequence of outputs.
- No supervised output but delayed reward.
- Credit assignment problem.
- Game playing.
- Robot in a maze.
- Multiple agents, partial observability, …

https://www.youtube.com/watch?v=V1eYniJ0Rnk&vl=en

Self-driving Cars

Carnegie Mellon University – 1990ies
ALVINN: Autonomous Land Vehicle In a Neural Network

https://www.youtube.com/watch?v=ilP4aPDTBPE#action=share

Waymo / Google – 2017

https://www.youtube.com/watch?v=B8R148hFxPw

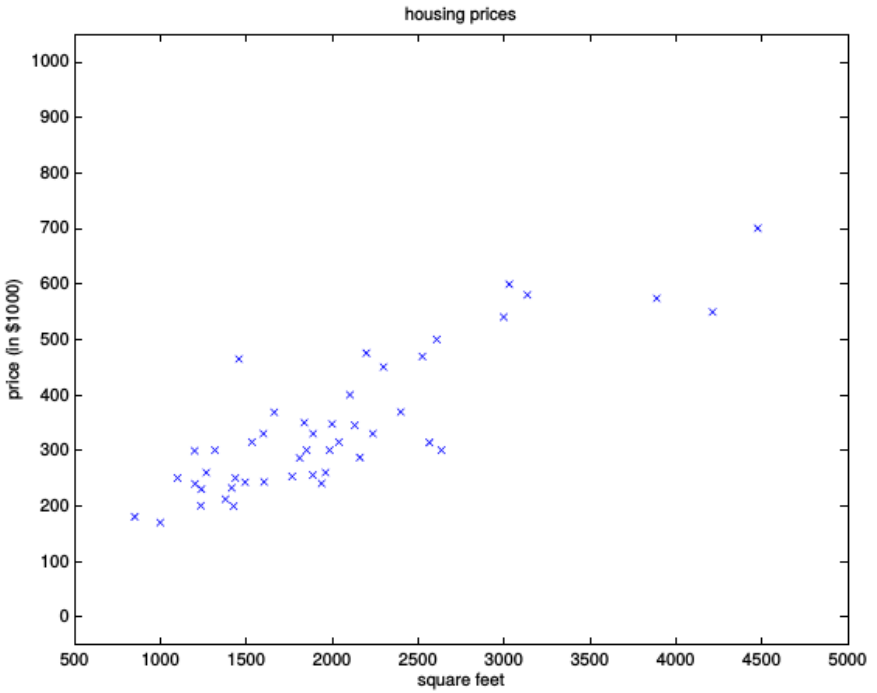
→ Classification
→ Regression
→ Reinforcement learning
→ Prediction

Building an ML Algorithm

- Optimize a performance criterion using example data or past experience.
- Role of Statistics: Inference from a sample.
- Role of Computer science: Efficient algorithms to
  - Solve the optimization problem.
  - Representing and evaluating the model for inference.

Building an ML Algorithm (II)

Living area (feet^2)	Price (1000$s)
2104	400
1600	330
2400	369
1416	232
3000	540
⋮	⋮



→ Given data like this, how can we learn to predict the prices of other houses as a function of the size of their living areas?

Building an ML Algorithm (III)

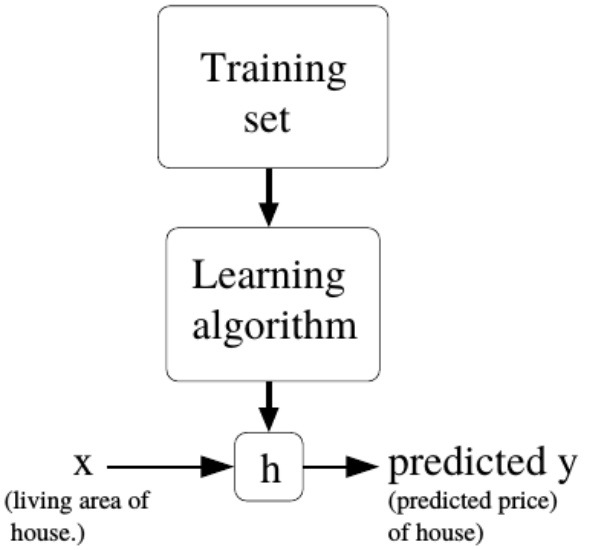
cf. Andrew Ng's ML course http://cs229.stanford.edu/

x(i): "input" variables (living area in this example),
also called input features

y(i): "output" / target variable that we are trying to predict (price).

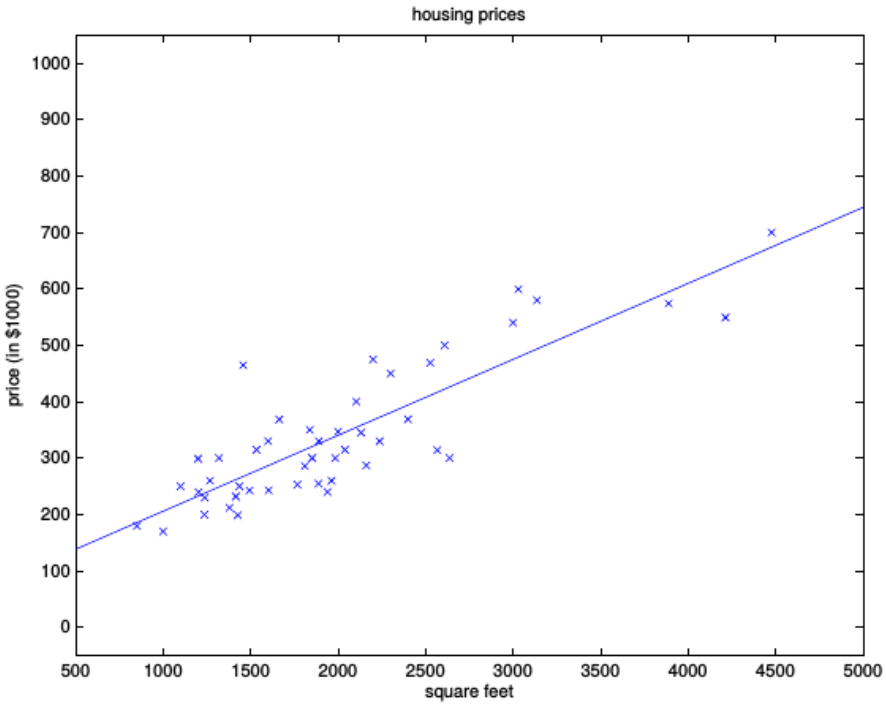
Training example:
a pair (x(i) , y(i)).

Training set:
a list of m training examples {(x(i), y (i)); i = 1, . . . , m}



→ To perform supervised learning, we must decide how we're going to represent **functions/hypotheses *h*** in a computer.

Building an ML Algorithm (IV)



Model / Hypothesis:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

θ_i's: parameters

Cost Function:

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

→ Minimize J(θ) in order to obtain the coefficients θ.

Building an ML Algorithm (V)

All Models are wrong, but some models are useful – (Box & Draper (1987), p.424)

In General: Machine learning (e.g., supervised) in 3 Steps:

- Choose a **model** $h(x|\theta)$.
- Define a **cost function** $J(\theta|x)$.
- **Optimization procedure** to find θ^* that minimizes $J(\theta)$.

Computationally, we need:
data, linear algebra, statistics tools, and optimization routines.

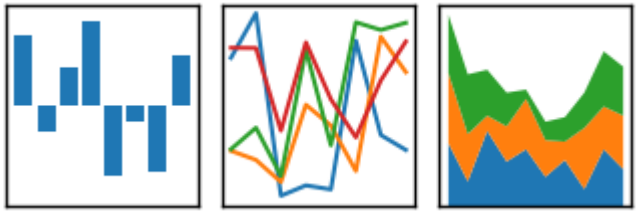
“Easy-to-use” Open Source Libraries

https://www.tensorflow.org/

https://keras.io/

https://scikit-learn.org/stable/

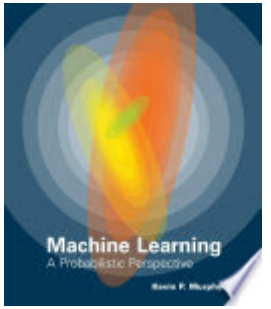
https://pandas.pydata.org/



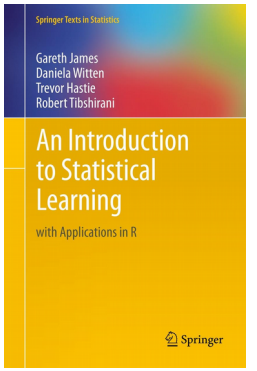
…

Some source materials

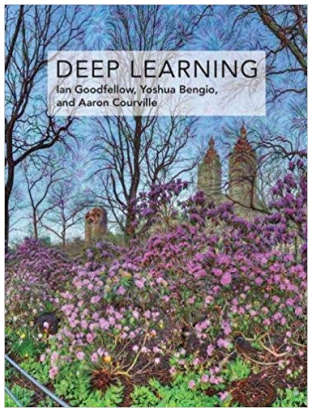
Machine Learning: a Probabilistic Perspective
K. Murphy, MIT Press, 2012
https://www.cs.ubc.ca/~murphyk/MLbook/index.html



An Introduction to Statistical Learning
Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani
Springer, 8th editon, 2017
https://www-bcf.usc.edu/~gareth/ISL/



Deep Learning
Ian Goodfellow and Yoshua Bengio and Aaron Courville
MIT Press 2016
http://www.deeplearningbook.org/



Machine Learning and dynamic models? Recall e.g., Dynamic Programming

The solution is approached in the limit as $j \to \infty$ by iterations on at every coordinate of the discretized grid.

$$V_{j+1}(x) = \max_u \{r(x,u) + \beta V_j(\tilde{x})\}$$

s.t.

$$\tilde{x} = g(x,u)$$

x: grid point, describes your system. State-space potentially **high-dimensional, and irregularly shaped, i.e., non-hypercubic.**

`old solution': high-dimensional function, approximated by **SOME** Interpolation method on which we Interpolate.

One way to use ML methods in dynamic models

Machine Learning to solve dynamic models?

Supervised ML:

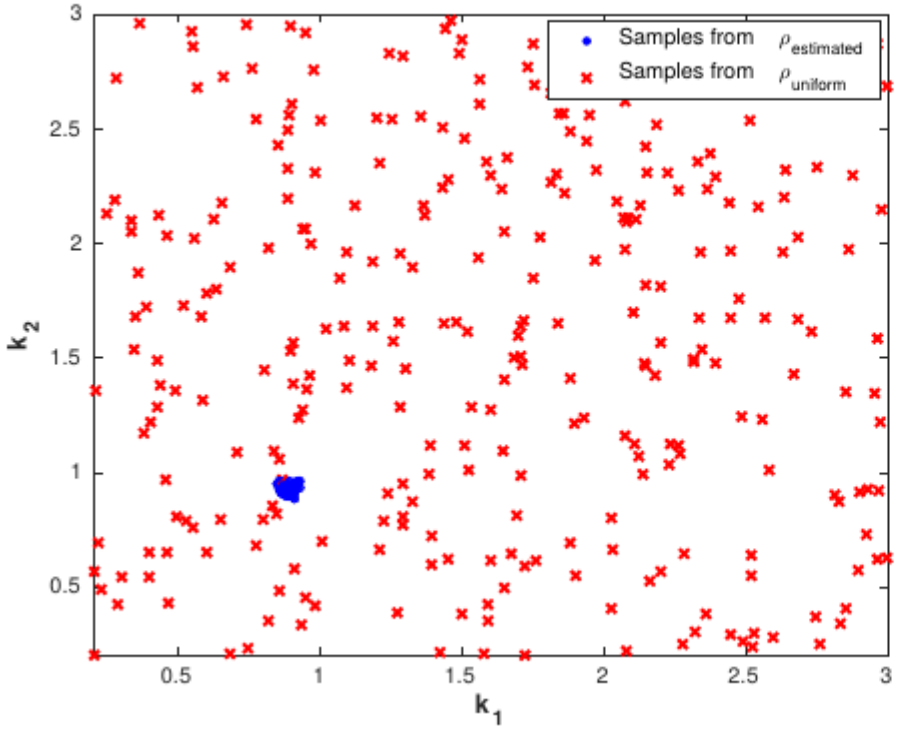
- Regression Methods → Approximate Functions.
- We are steering the data-generating process (→ “easy” to do with parallel programming).

Unsupervised ML:

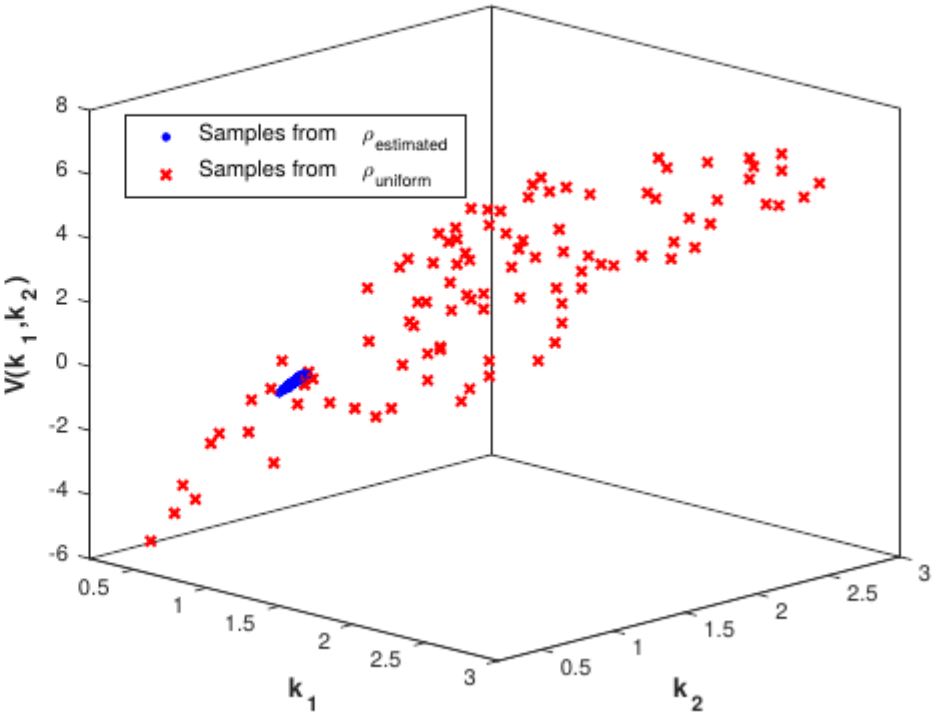
- Describe Ergodic & Feasible sets and sample from within them → Clustering Methods.
- Dimension-reduction → Deal with the Curse of Dimensionality.

Want to Solve e.g. Dynamic Models on high-dimensional, irregularly-shaped state-spaces

cf. Scheidegger & Bilionis (2017)

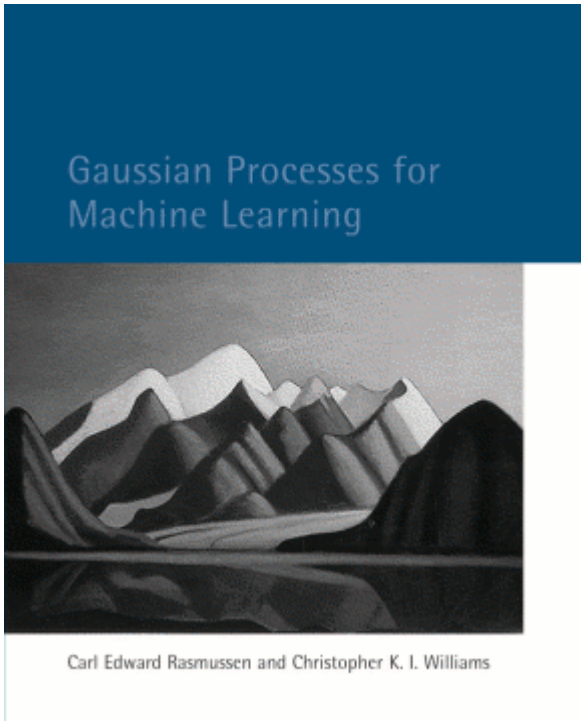


Blue: ergodic set.
Red: Computational domain



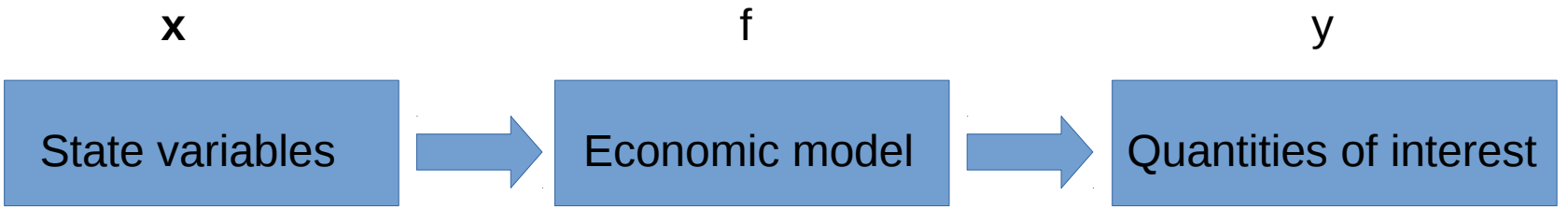
Blue: Value function, evaluated on ergodic set
Red: Value function, evaluated on the entire comp. domain

II. Introduction to Gaussian Process Regression



http://www.gaussianprocess.org/gpml/

Motivation



We'll think about it as a mathematical function:

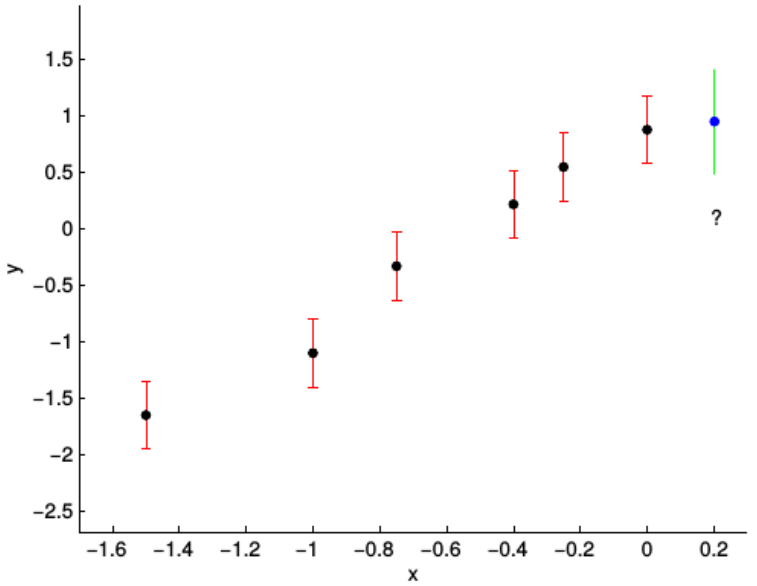
$$y = f(x)$$

Aim of Regression

- Given some (potential) noisy observations of a dependent variable at certain values of the independent variable x, what is our best estimate of the dependent variable y at a new value, x_*?
- Let *f* denote an (unknown) function which maps inputs x to outputs

$$f: X \rightarrow Y$$

- Modeling a function *f* means mathematically representing the relation between inputs and outputs.
- Often times, the shape of the underlying function might be unknown, the function can be hard to evaluate, or other requirements might complicate the process of information acquisition.



Choosing a model

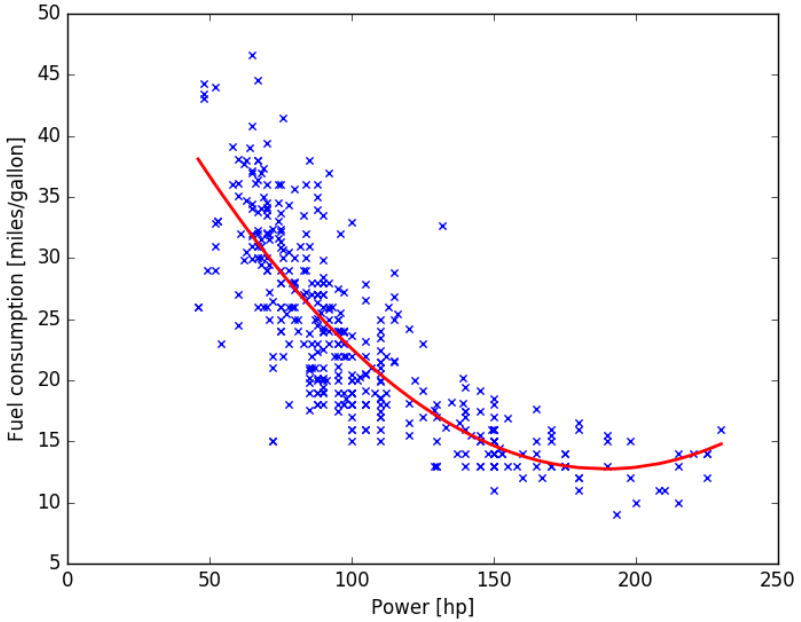
- If we expect the underlying function f(x) to be linear, and can make some assumptions about the input data, we might use a least-squares method to fit a straight line (linear regression).
- Moreover, if we suspect f(x) may also be quadratic, cubic, or even non-polynomial, we can use the principles of model selection to choose among the various possibilities.

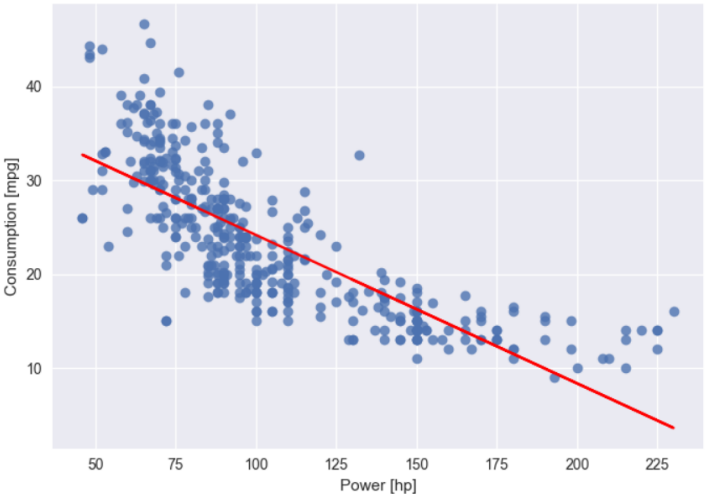
Model Selection

Example data set by https://archive.ics.uci.edu/ml/datasets/Auto+MPG

One common approach to reliably assess the quality of a machine learning model and avoid over-fitting is to randomly split the available data into

- training data (~70% of the data)
  is used for determining optimal coefficients.
- validation data (~20% of the data) is used for model selection (e.g., fixing degree of polynomial, selecting a subset of features, etc.)
- test data (~10% of the data) is used to measure the quality that is reported.





For completeness: Polynomial Regression in Python

Code and data set here: global_solution_yale19/Lecture_4/polyreg.py

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model, preprocessing

# read cars dataset, a clean data set
cars = pd.read_csv('auto-mpg.data.txt',header=None, sep='\s+')


# extract mpg values
y = cars.iloc[:,0].values

# extract horsepower values
X = cars.iloc[:,[3]].values
X = X.reshape(X.size, 1)

# precompute polynomial features
poly = preprocessing.PolynomialFeatures(2)
Xp = poly.fit_transform(X)

# fit linear regression model
reg = linear_model.LinearRegression()
reg.fit(Xp,y)

# coefficients
reg.intercept_ # 56.900099702112925
reg.coef_ # [-0.46618963, 0.00123054]

# compute correlation coefficient
np.corrcoef(reg.predict(Xp),y) # 0.82919179 (from 0.77842678)

# compute mean squared error (MSE)
sum((reg.predict(Xp) - y)**2) / len(y) # 18.984768907617223 ( from 23.943662938603104)

### plot

hp = cars.iloc[:,3].values
mpg = cars.iloc[:,0].values

hps = np.array(sorted(hp))
hps = hps.reshape(hps.size, 1)
hpsp = poly.fit_transform(hps)

plt.scatter(hp, mpg, color='blue', marker='x')
plt.plot(hps, reg.predict(hpsp), color='red', lw=2)
plt.xlabel('Power [hp]')
plt.ylabel('Fuel consumption [miles/gallon]')
plt.show()
```

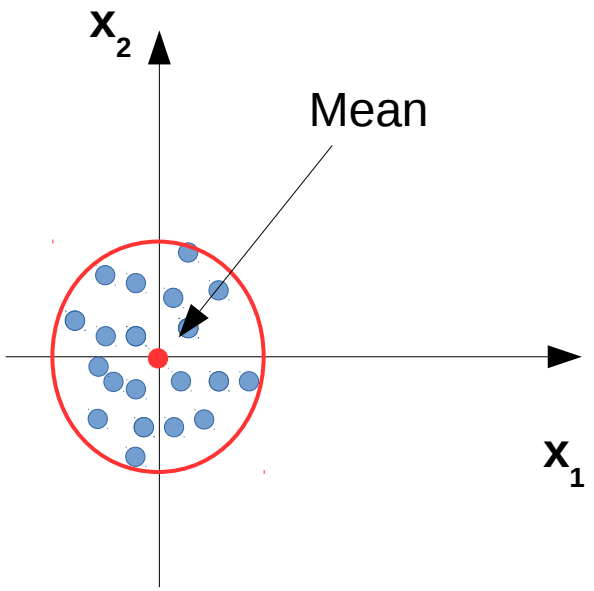
Degree of regression

Why Gaussian Process Regression?

- There are infinitely many projections possible.
- We have to choose one either a priori or by model comparison with a set of possible projections.
- Especially if the problem is to explore and exploit a completely unknown function, this approach will not be beneficial as there is little guidance to which projections we should try.

Gaussian process regression offers a principled solution to this problem in which projections are chosen implicitly, effectively leading "**the data decide**" on the complexity of the function.

Recall Multivariate Gaussians



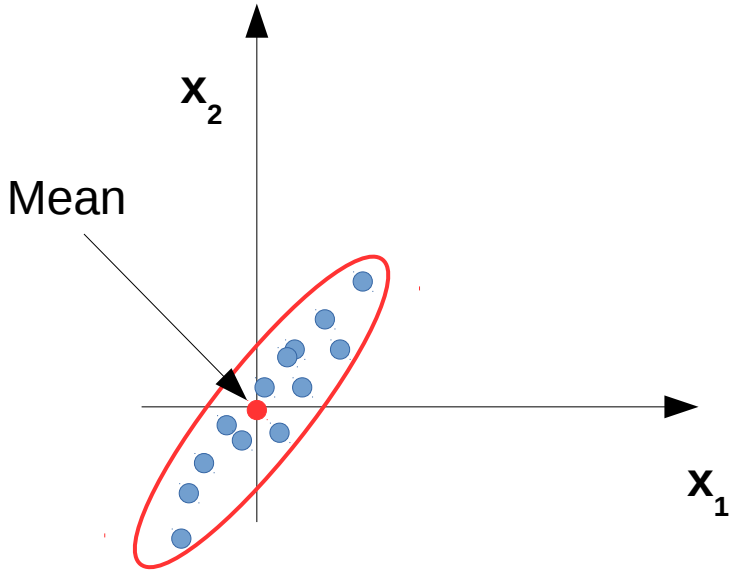
Fit a Gaussian that with a covariance that is circular.

Say you measure two variables

- x_1:height
- x_2: weight

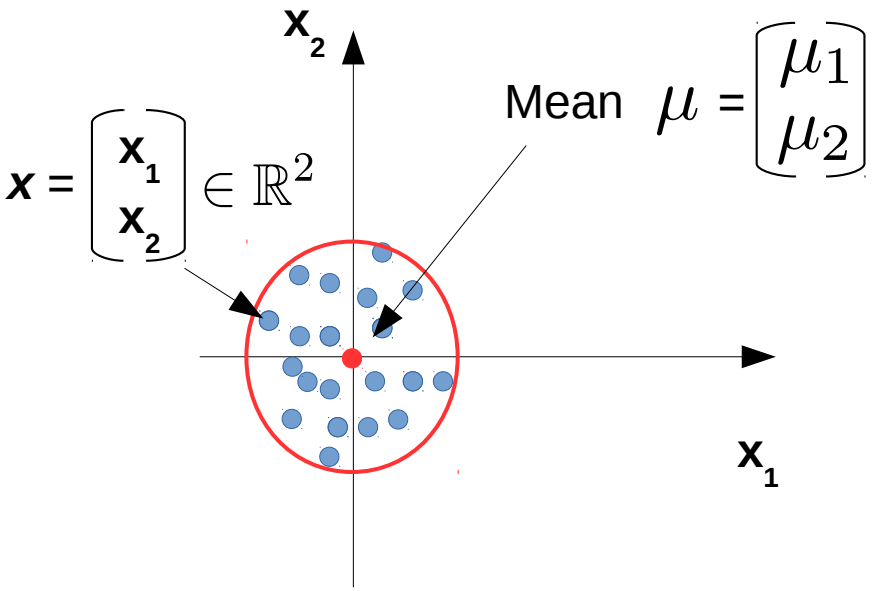
→ plot

→ we want to fit a Gaussian to these points.

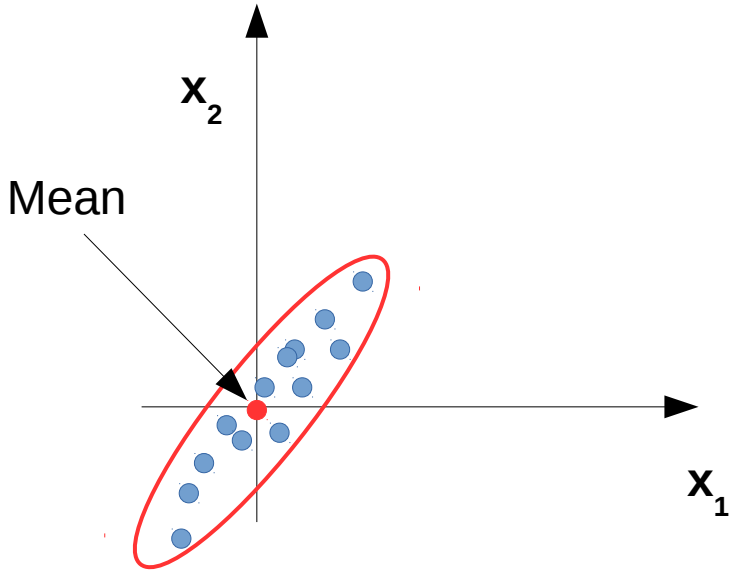


Fit a Gaussian that with a covariance that is an ellipse.

Recall Multivariate Gaussians (II)



Fit a Gaussian that with a covariance that is circular.



Fit a Gaussian that with a covariance that is an ellipse.

Multivariate Gaussians (III)

- Assume the points are Gaussian distributed (this is our “model”).
- How do points relate to each other? (“how does increasing $\mathbf{x}_1$ increase $\mathbf{x}_2$?)

→ The variable to describe this is called “Covariance*” (cov)

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \sim N\left(\begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix} \right)$$

Mean Covariance

- If the entries in the column vector $\mathbf{X} = (X_1, X_2, \ldots, X_n)^{\mathrm{T}}$ are random variables, each with finite variance and expected value, then the covariance matrix $\mathbf{K}_{xx}$ is the matrix whose (i ,j) entry is the covariance.

$$\mathrm{K}_{X_i X_j} = \operatorname{cov}[X_i, X_j] = \mathrm{E}[(X_i - \mathrm{E}[X_i])(X_j - \mathrm{E}[X_j])]$$

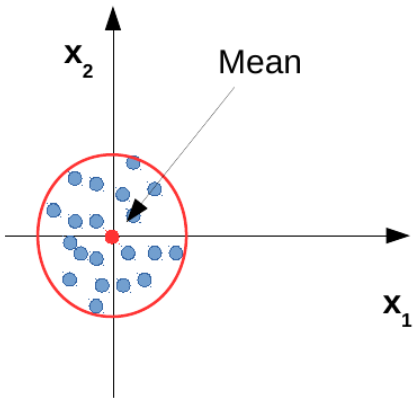
*Covariance matrix: positive definite (Matrix symmetric, its Eigenvalues positive).

Multivariate Gaussian (IV)

- Assume for a moment that E[ ] = 0 → Covariance E[$\mathbf{x}_1\mathbf{x}_2$] is a “dot” product.
- Assume two points [1 0] $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$ = 1 → **Covariance is a measure similarity**.

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \sim N\left(\begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right)$$

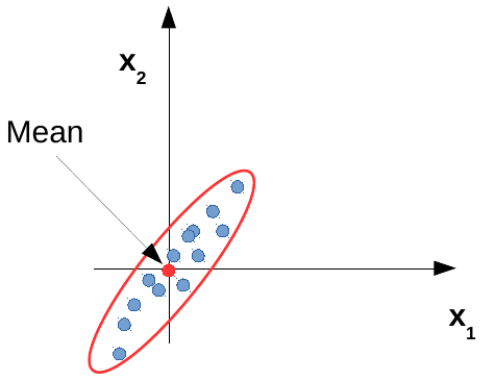
Knowing about $\mathbf{x}_1$ does not provide any information about $\mathbf{x}_2$ as they are uncorrelated.



Multivariate Gaussians (V)

- Assume for a moment that **E**[ ] = 0 → Covariance **E**[$\mathbf{x}_1\mathbf{x}_2$] is a "dot" product.
- Assume two points [1 0] $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$ = 1 → Covariance measure similarity.

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \sim N\left(\begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} 1 & 0.6 \\ 0.6 & 1 \end{bmatrix} \right)$$



→ Knowing about $\mathbf{x}_1$ DOES provide information about $\mathbf{x}_2$.

→ if $\mathbf{x}_1$ is positive, $\mathbf{x}_2$ is with great probability.

→ knowing something about $\mathbf{x}_1$ allows us to know something about $\mathbf{x}_2$.

Joint Gaussian distributions

see, e.g., Rasmussen et al. (2005), Murphy (2012)

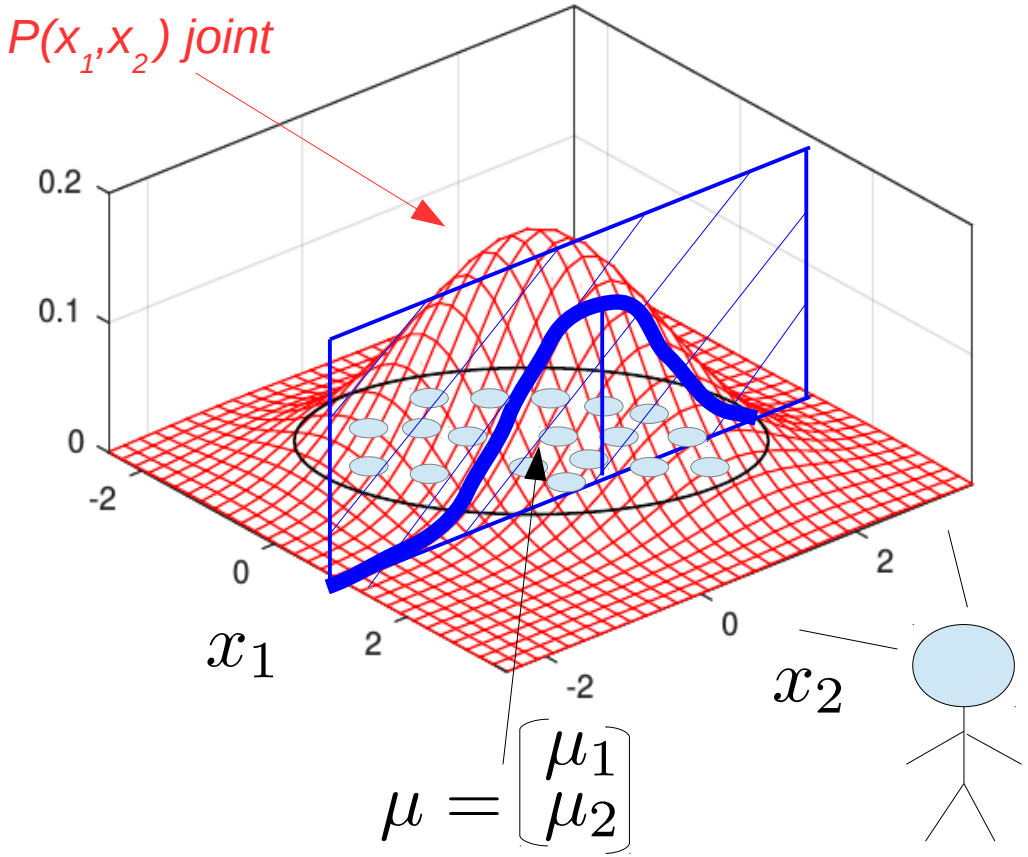
$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \sim N\left(\begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix} \right)$$

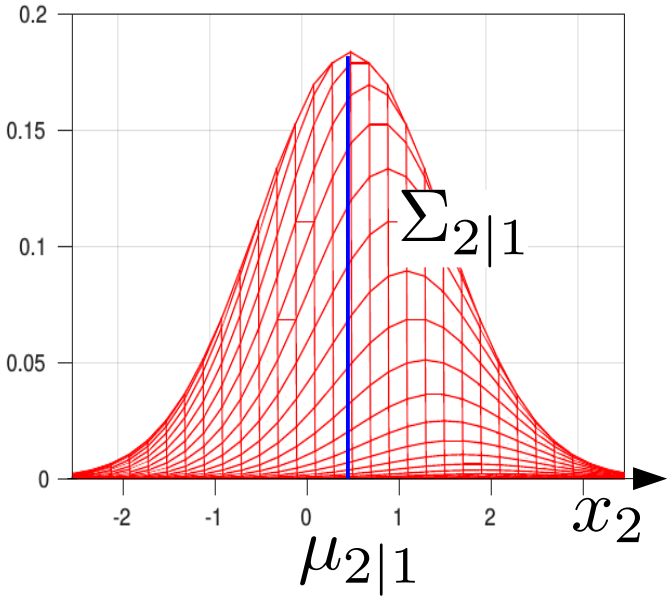
Mean

Covariance

Conditional distribution

$$P(x_2 | X_1 = x_1)$$





Joint Gaussian distributions (II)

see, e.g., Murphy (2012), chapter 4.

Theorem 4.3.1 (Marginals and conditionals of an MVN). *Suppose* $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)$ *is jointly Gaussian with parameters*

$$\boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{pmatrix}, \quad \boldsymbol{\Sigma} = \begin{pmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{pmatrix}, \quad \boldsymbol{\Lambda} = \boldsymbol{\Sigma}^{-1} = \begin{pmatrix} \boldsymbol{\Lambda}_{11} & \boldsymbol{\Lambda}_{12} \\ \boldsymbol{\Lambda}_{21} & \boldsymbol{\Lambda}_{22} \end{pmatrix} \tag{4.67}$$

Then the marginals are given by

$$\begin{aligned} p(\mathbf{x}_1) &= \mathcal{N}(\mathbf{x}_1 | \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_{11}) \\ p(\mathbf{x}_2) &= \mathcal{N}(\mathbf{x}_2 | \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_{22}) \end{aligned} \tag{4.68}$$

and the posterior conditional is given by

$$\begin{aligned} p(\mathbf{x}_1 | \mathbf{x}_2) &= \mathcal{N}(\mathbf{x}_1 | \boldsymbol{\mu}_{1|2}, \boldsymbol{\Sigma}_{1|2}) \\ \boldsymbol{\mu}_{1|2} &= \boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\mathbf{x}_2 - \boldsymbol{\mu}_2) \\ &= \boldsymbol{\mu}_1 - \boldsymbol{\Lambda}_{11}^{-1}\boldsymbol{\Lambda}_{12}(\mathbf{x}_2 - \boldsymbol{\mu}_2) \\ &= \boldsymbol{\Sigma}_{1|2}\left(\boldsymbol{\Lambda}_{11}\boldsymbol{\mu}_1 - \boldsymbol{\Lambda}_{12}(\mathbf{x}_2 - \boldsymbol{\mu}_2)\right) \\ \boldsymbol{\Sigma}_{1|2} &= \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21} = \boldsymbol{\Lambda}_{11}^{-1} \end{aligned} \tag{4.69}$$

This Theorem allows you to go from joint to conditional distributions.

Basics on Gaussians

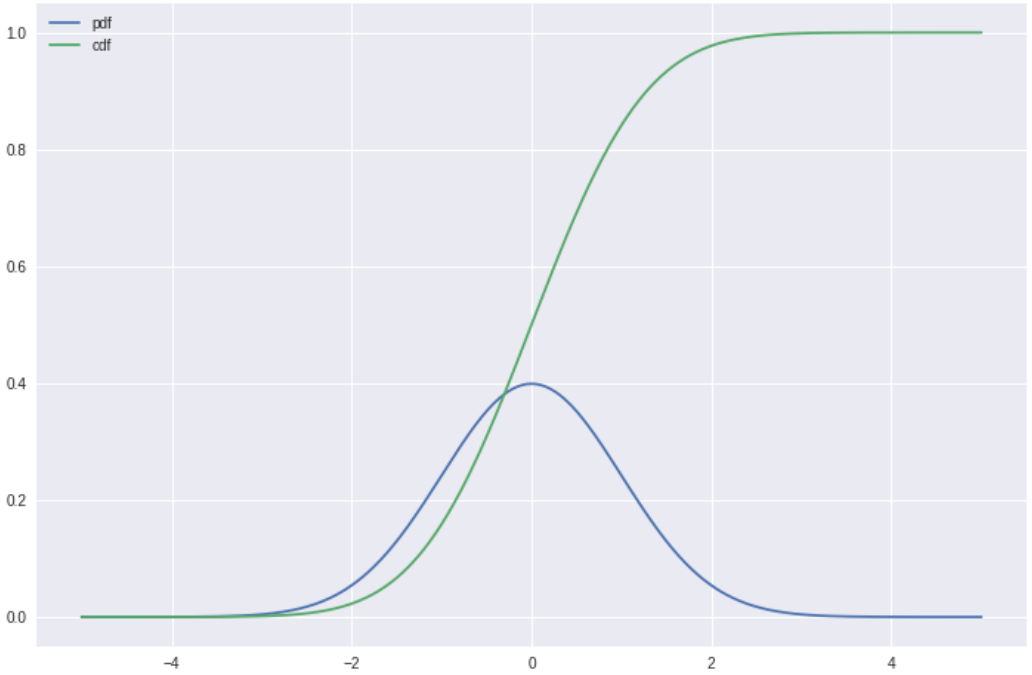
$$x_i \sim \mathcal{N}(0, 1)$$

$$x_i \sim \mathcal{N}(\mu, \sigma^2) \sim \mu + \sigma\mathcal{N}(0, 1)$$

As we have the capability of drawing *1-dim* random numbers from a Gaussian, we can also do this in a multivariate case.

→ We need a way to take “square roots” from matrices.

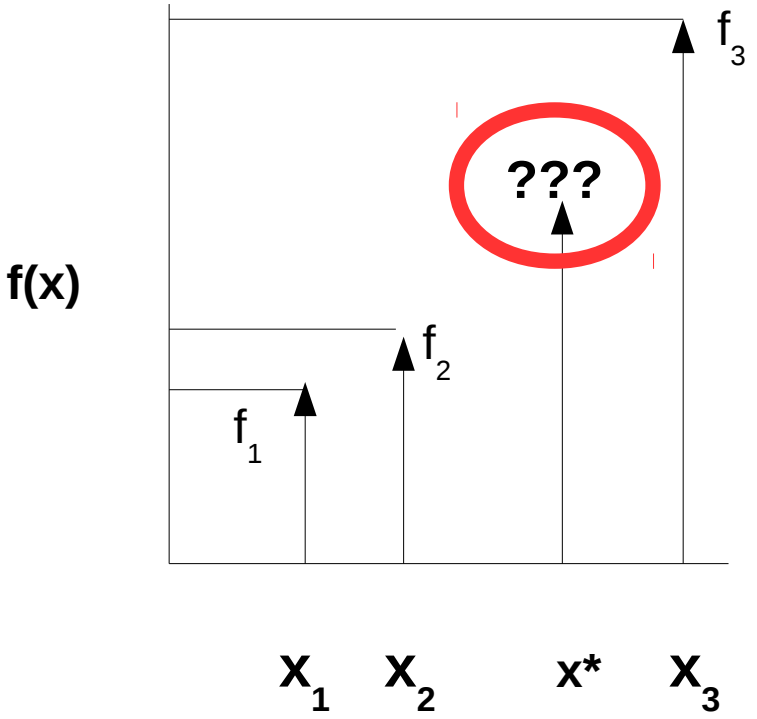
→ ***Cholesky*** decomposition: $\Sigma = LL^T$



$$\longrightarrow \quad \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \sim N\left(\begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix} \right) \quad \longrightarrow \quad x \sim \mu + L\mathcal{N}(0, I)$$

$\mathbf{x}$ μ Σ

Observations → Interpolation



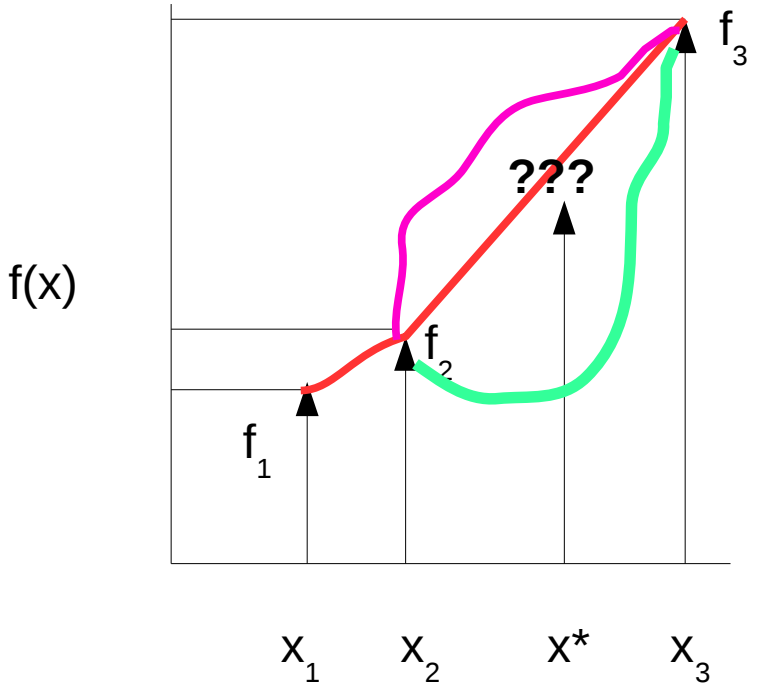
We have 3 observations at $\mathbf{x_i}$ for $\mathbf{f(x_i)}$

→ **Given the data pairs**
$D = \{ (x_1,f_1), (x_2,f_2), (x_3,f_3)\}$

→ **want to find/learn the function that describes the data, i.e., for a "new" x*, we want to know what f(x*) would be!**

Observations → Interpolation (II)

We assume that **f's (the height) are Gaussian distributed**, with **zero – mean** and some **covariance matrix K**.



$$\begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix} \sim N \left(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} K_{11}, K_{12}, K_{13} \\ K_{21}, K_{22}, K_{23} \\ K_{31}, K_{32}, K_{33} \end{bmatrix} \right)$$

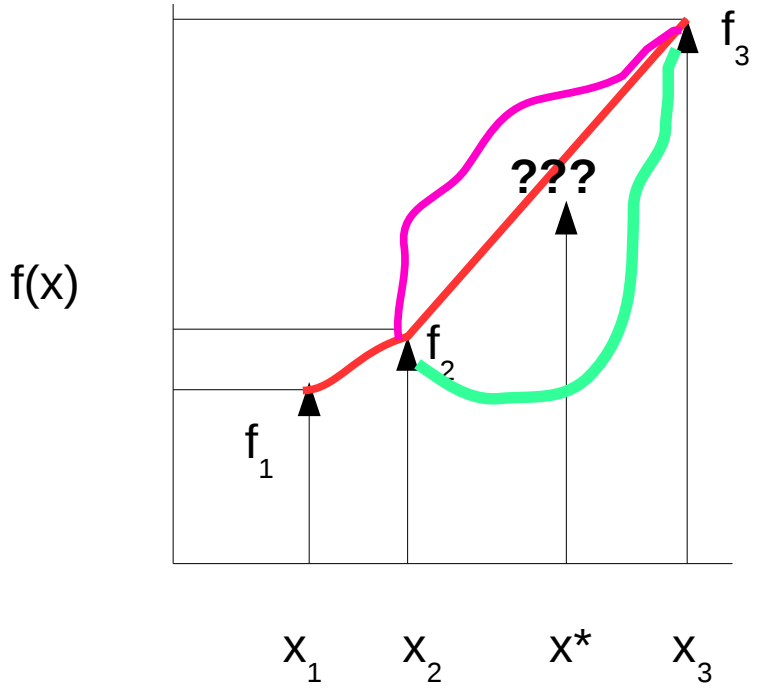
Note: f_1 and f_2 should probably be more correlated, as they are nearby (compared to f_1 and f_3).

→ The prior mean function μ reflects the expected function value at input x: $\mu(x) = \mathbb{E}(f(x))$

→ It is often set to 0.

Observations → Interpolation (II)

We assume that **f's (the height) are Gaussian distributed**, with **zero – mean** and some **covariance matrix K**.



$$\begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix} \sim N\left(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} K_{11}, K_{12}, K_{13} \\ K_{21}, K_{22}, K_{23} \\ K_{31}, K_{32}, K_{33} \end{bmatrix} \right)$$

Note: f_1 and f_2 should probably be more correlated, as they are nearby (compared to f_1 and f_3), e.g.,

$$\sim N\left(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1, 0.7, 0.2 \\ 0.7, 1, 0.6 \\ 0.2, 0.6, 1 \end{bmatrix} \right)$$

Covariance matrix **constructed** by some "**measure of similarity**", i.e., a **kernel function** (parametric ansatz), such as "squared exponential". Parameters can be obtained e.g. via MLE (later).

$$\kappa(x, x') = \sigma_f^2 \exp\left(-\frac{1}{2\ell^2}(x - x')^2\right)$$

σ_f^2 – controls vertical variation.

ℓ – controls horizontal length scale.

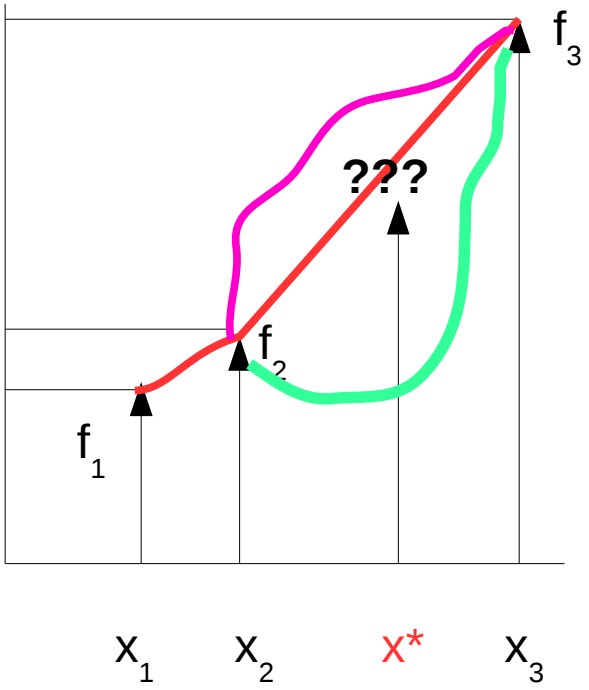
Observations → Interpolation (III)

Given data D = { (x_1,f_1), (x_2,f_2), (x_3,f_3)} → **f(x*) =f_*** ?

→ Assume f ~ *N*(0 , K(·,·))

→ Assume f(x*) ~ *N*(0, K(x*,x*))

3d-Covariance K from the training data



$$\begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_* \end{pmatrix} \sim N \left(\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} , \begin{bmatrix} K_{11}, K_{12}, K_{13}, K_{1*} \\ K_{21}, K_{22}, K_{23}, K_{2*} \\ K_{31}, K_{32}, K_{33}, K_{3*} \\ K_{*1}, K_{*2}, K_{*3}, K_{**} \end{bmatrix} \right)$$

→ Joint distribution over f and f_* .

→ **We need the conditional of f_* given f.**

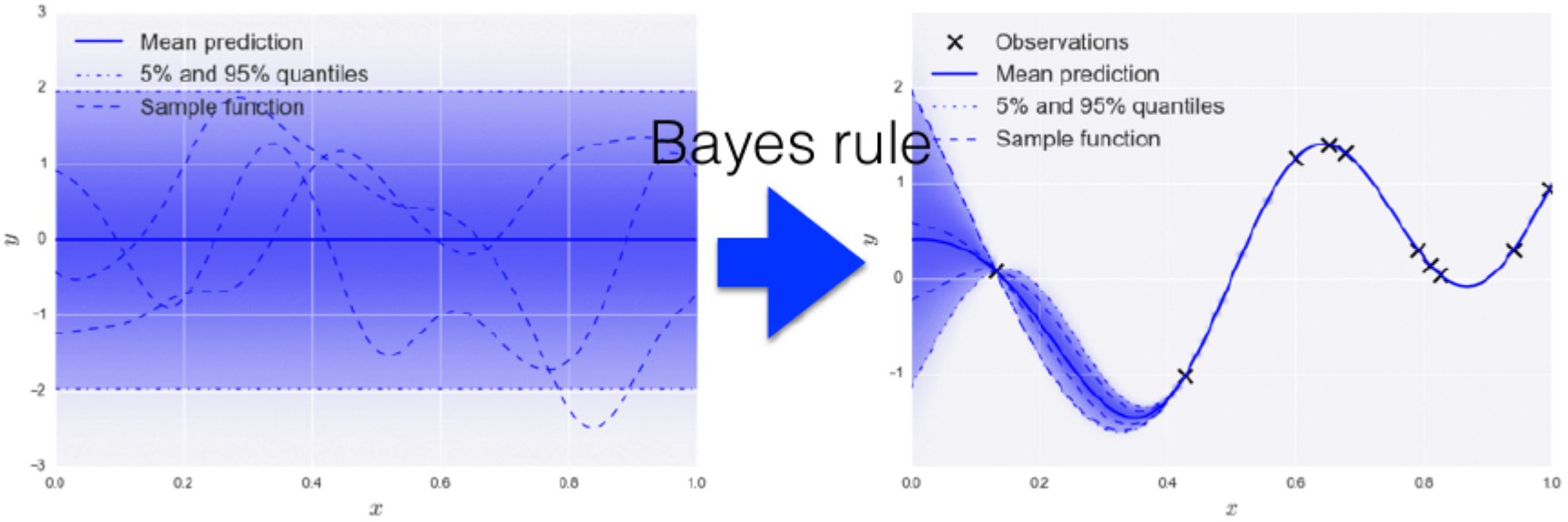
→ In this example, we “cut” in 3 dimensions.

→ What is left is a 1-dimensional Gaussian, i.e., the Gaussian for f_*

$$K(x_1, x_*) = K_{1*}$$

Interpolation → Noiseless GPR

(see, e.g., Rasmussen & Williams (2006), with references therein)



Prior GP

Posterior GP

Training set: $D = \{(\mathbf{x}_i, y_i) | i = 1, \ldots, n\}$

$$\begin{pmatrix} \mathbf{f} \\ \mathbf{f}_* \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \boldsymbol{\mu} \\ \boldsymbol{\mu}_* \end{pmatrix}, \begin{pmatrix} \mathbf{K} & \mathbf{K}_* \\ \mathbf{K}_*^T & \mathbf{K}_{**} \end{pmatrix} \right)$$

$$p(\mathbf{f}_* | \mathbf{X}_*, \mathbf{X}, \mathbf{f}) = \mathcal{N}(\mathbf{f}_* | \boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*)$$

$$\boldsymbol{\mu}_* = \boldsymbol{\mu}(\mathbf{X}_*) + \mathbf{K}_*^T \mathbf{K}^{-1} (\mathbf{f} - \boldsymbol{\mu}(\mathbf{X}))$$

$$\boldsymbol{\Sigma}_* = \mathbf{K}_{**} - \mathbf{K}_*^T \mathbf{K}^{-1} \mathbf{K}_*$$

Test point = interpolation at X*

→ **predictive mean** $\mu_* = \mathbb{E}(f_*)$

→ **Confidence Intervals!**

Where we have data, we have high confidence in our predictions.

Where we do not have data, we cannot be to confident about our predictions.

Algorithmic Complexity!

- The central computational operation in using Gaussian processes will involve the inversion of a matrix of size N × N, for which standard methods require $O(N^3)$ computations.

- The matrix inversion must be performed once for the given training set.

  → For large training data sets, however, the direct application of Gaussian process methods can become infeasible.

  → Sparse Methods (cf. Rasmussen et. al (2006) and references therein).

A note on the predictive mean

Note that the predictive mean can (in general) also be written as

$$\mu = m(x) + \sum_{i=1}^{N} a_i k(x_i, x)$$

where $\boldsymbol{a} = (a_1, \ldots, a_N) = \left(\mathbf{K} + s_n^2 \mathbf{I}_N\right)^{-1} (\mathbf{t} - \mathbf{m})$
and ***t*** being the N observations.

→ We can think of the GP posterior mean as an approximation of f(·) using *N* symmetric basis functions centered at each observed input.

→ by choosing a covariance function that vanishes when x and x' are separated by a lot, for example the squared exponential covariance function, we see that an observed input-output will only affect the approximation locallv.

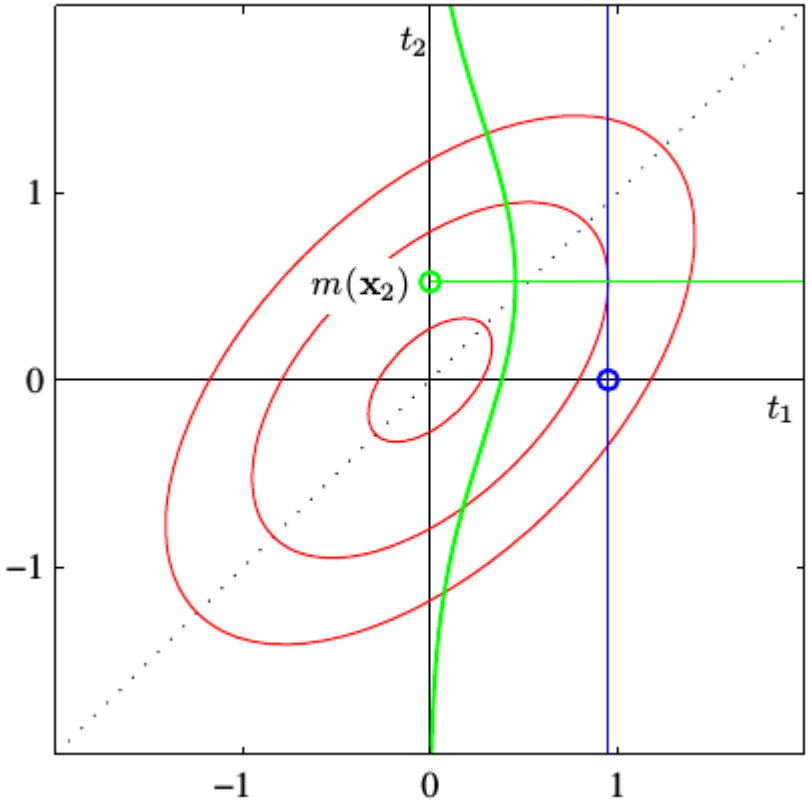
$$k_{\mathrm{SE}}(\mathbf{x}, \mathbf{x}'; \boldsymbol{\theta}) = s^2 \exp\left\{ -\frac{1}{2} \sum_{i=1}^{D} \frac{(x_i - x_i')^2}{\ell_i^2} \right\}$$

Illustration of GPR at work

Fig. from Bishop (2006)

- Illustration of the mechanism of GPR for the case of one training point and one test point, in which the red ellipses show contours of the joint distribution $p(t_1, t_2)$.

- Here t_1 is the training data point, and conditioning on the value of t_1, corresponding to the vertical blue line, we obtain $p(t_2 | t_1)$ shown as a function of t_2 by the green curve.



GP – a distribution over functions

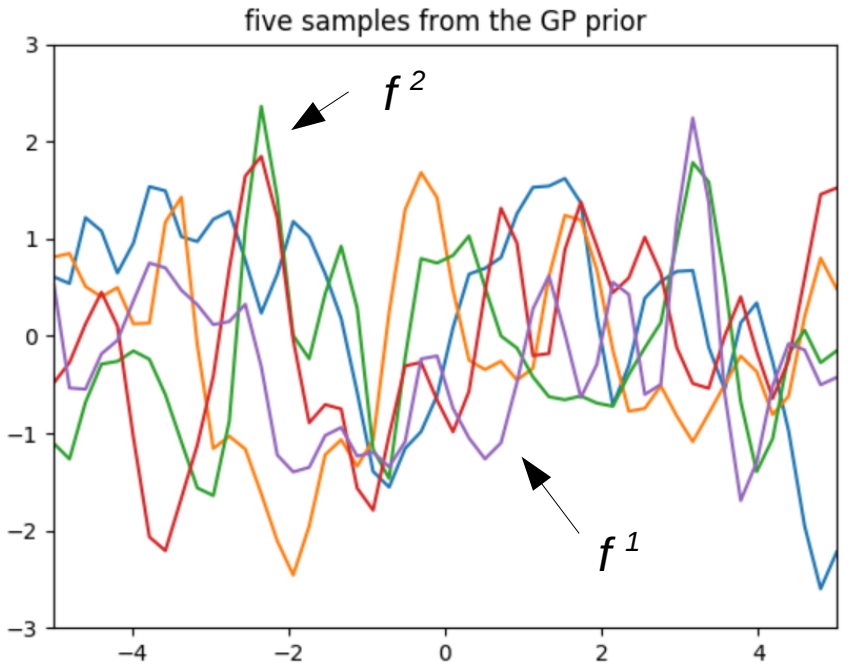
see global_solution_yale19/Lecture_4/code/1d_gp_example.py

$$f(x) \sim GP(\mu(x), k(x, x'))$$

$$\mu(x) = \mathbb{E}[f(x)]$$

$$k(x, x') = \mathbb{E}[(f(x) - \mu(x))((f(x') - \mu(x'))^T]$$

$$k(x, x') = exp(-\frac{1}{2}(x - x')^2)$$



Procedure

Create vector $X_{1:N}$

$$\mu = 0, K_{N \times N}$$

$K = LL^T$

$f^i \sim N(0, K) \sim L\, N(0,I)$

GPR Example code & numerical stability

Look at global_solution_yale19/Lecture_4/code/1d_gp_example.py and Murphy, Chapter 15

```python
import numpy as np
import matplotlib.pyplot as pl

""" A code to illustrate the workings of GP regression in 1d.
    We assume a zero mean GP Prior """


# This is the true unknown, one-dimensional function we are trying to approximate
f = lambda x: np.sin(0.9*x).flatten()


# This is the kernel function
def kernel_function(a, b):
    """ GP squared exponential kernel function """
    kernelParameter = 0.1
    sqdist = np.sum(a**2,1).reshape(-1,1) + np.sum(b**2,1) - 2*np.dot(a, b.T)
    return np.exp(-.5 * (1/kernelParameter) * sqdist)


# Here we run the test
N = 10          # number of training points.
n = 50         # number of test points.
s = 0.00005    # noise variance.

# Sample some input points and noisy versions of the function evaluated at
# these points.
X = np.random.uniform(-5, 5, size=(N,1))
y = f(X) + s*np.random.randn(N)  #add some noise

K = kernel_function(X, X)
L = np.linalg.cholesky(K + s*np.eye(N))

# points we're going to make predictions at.
TestPoint = np.linspace(-5, 5, n).reshape(-1,1)

# compute the mean at our test points.
Lk = np.linalg.solve(L, kernel_function(X, TestPoint))
mu = np.dot(Lk.T, np.linalg.solve(L, y))

# compute the variance at our test points.
K_ = kernel_function(TestPoint, TestPoint)
s2 = np.diag(K_) - np.sum(Lk**2, axis=0)
s = np.sqrt(s2)
```

$$\mathbb{E}[f(x_*)] = \mu = k_*^T K_y^{-1} y$$

$$K_y = LL^T$$

$$\alpha = K_y^{-1} y = L^{-T} L^{-1} y$$

$$m = L^{-1} y$$

$$Lm = y$$

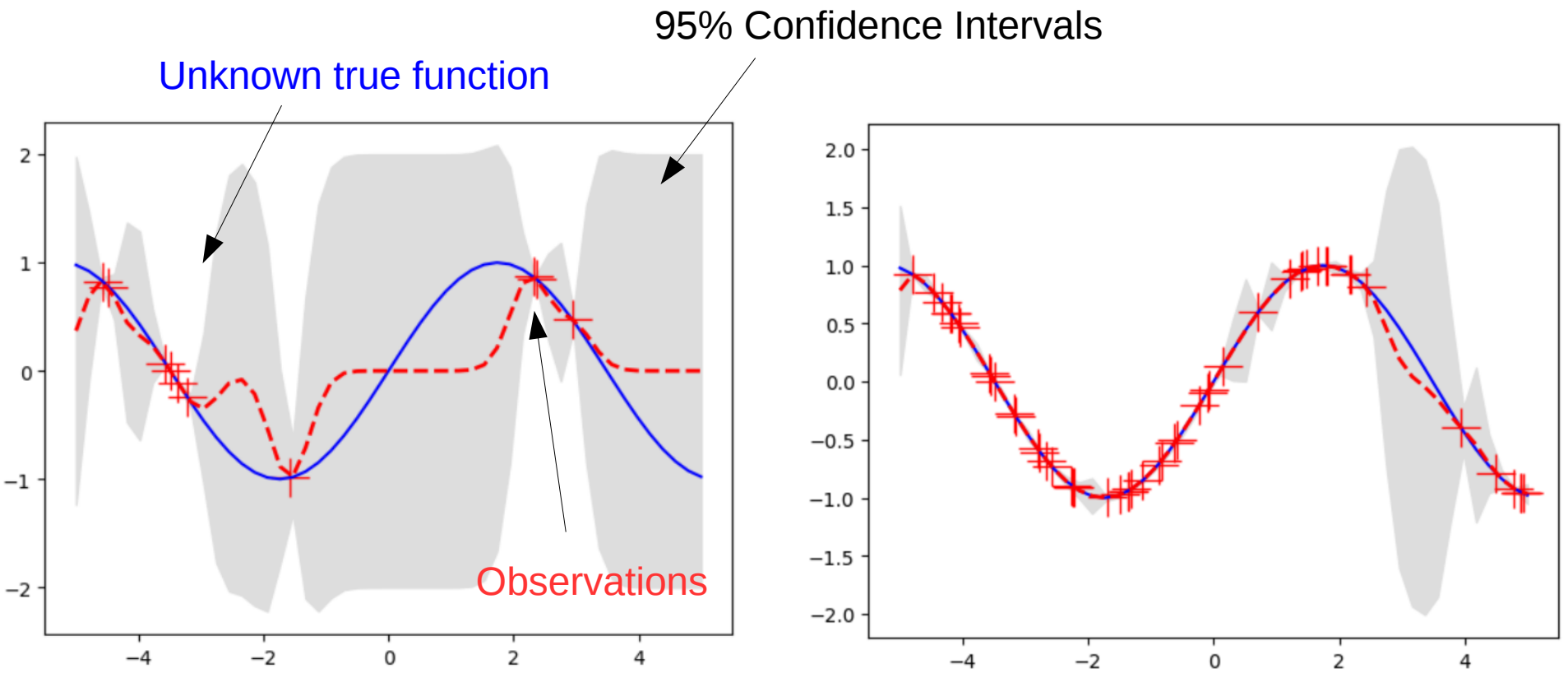
Algorithm 15.1: GP regression

1. $\mathbf{L} = \text{cholesky}(\mathbf{K} + \sigma_y^2 \mathbf{I})$;
2. $\boldsymbol{\alpha} = \mathbf{L}^T \setminus (\mathbf{L} \setminus \mathbf{y})$;
3. $\mathbb{E}[f_*] = \mathbf{k}_*^T \boldsymbol{\alpha}$;
4. $\mathbf{v} = \mathbf{L} \setminus \mathbf{k}_*$;
5. $\text{var}[f_*] = \kappa(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{v}^T \mathbf{v}$;
6. $\log p(\mathbf{y}|\mathbf{X}) = -\frac{1}{2}\mathbf{y}^T \boldsymbol{\alpha} - \sum_i \log L_{ii} - \frac{N}{2} \log(2\pi)$

Alg. 15: Numerical more stable.

Prediction & Confidence Intervals

Note: if you don't fix the seed, these pictures vary every time you run of the code.



10 Training Points **50 Training Points**

We see that the model perfectly interpolates the training data, and that the predictive uncertainty increases as we move further away from the observed data.

Non-hypercubic domain: GPR versus ASG

More on this in Lecture 6!

$$\mathrm{Vol}_\Delta = \frac{1}{D!}$$

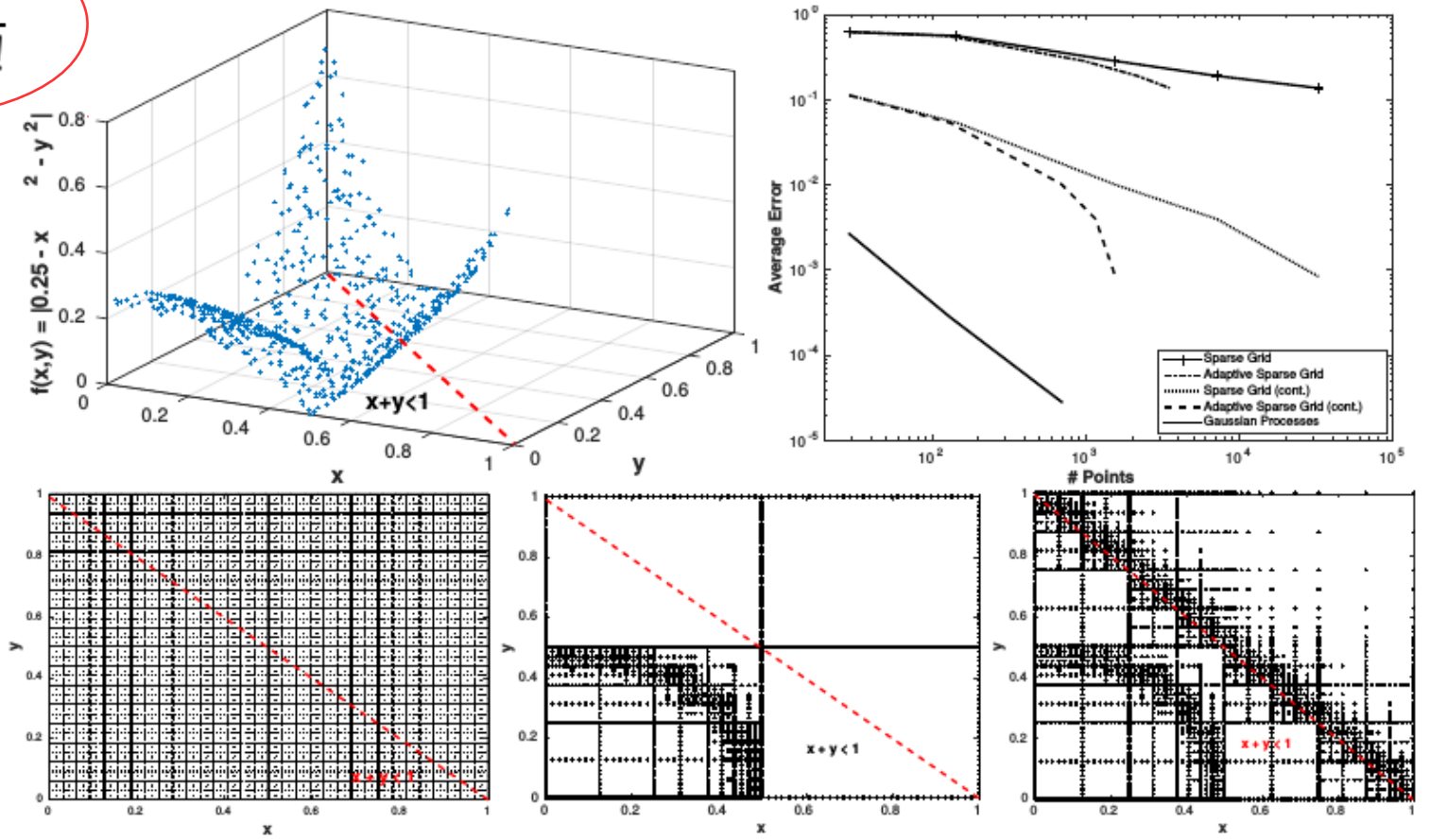


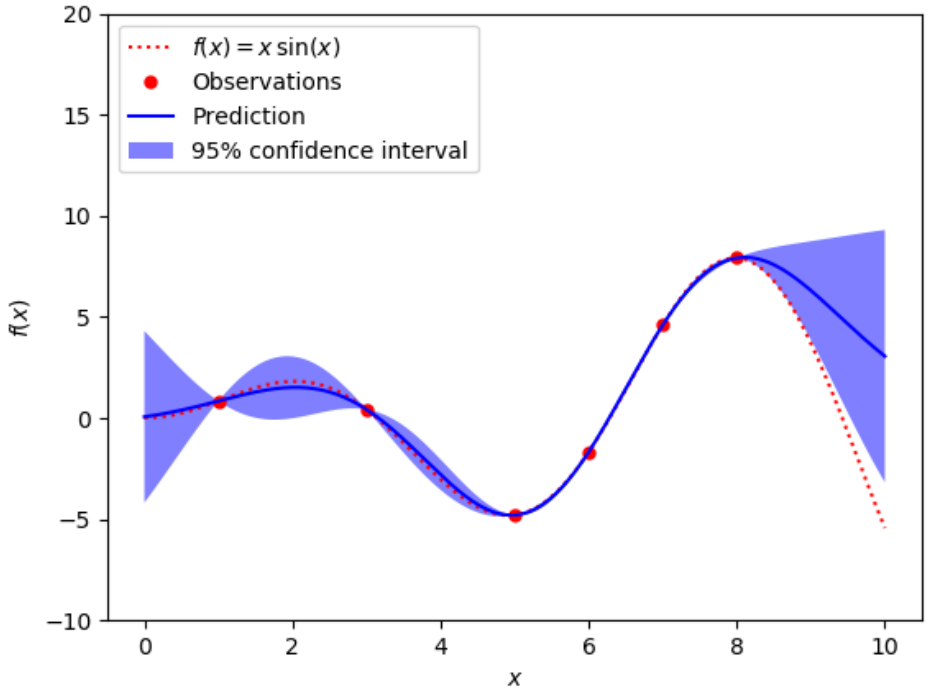
Figure: The upper left panel shows the analytical test function evaluated at random test points on the simplex. The upper right panel displays a comparison of the interpolation error for GPs, sparse grids, and adaptive sparse grids of varying resolution and constructed under the assumption that a continuation value exists, (denoted by cont"), or that there is no continuation value. The lower left panel displays a sparse grid consisting of 32,769 points. The lower middle panel shows an adaptive sparse grid (cont) that consists of 1,563 points, whereas the lower right panel shows an adaptive sparse grid, constructed with 3,524 points and under the assumption that the function outside Δ is not known.

GPR in scikit-learn.org

https://scikit-learn.org/stable/modules/gaussian_process.html

https://scikit-learn.org/stable/auto_examples/gaussian_process/plot_gpr_noisy_targets.html#sphx-glr-auto-examples-gaussian-process-plot-gpr-noisy-targets-py

Look at global_solution_yale19/Lecture_4/1d_GPR.py



More GP packages (next to scikit-learn.org)

Name	Algorithm	Language	Author
GPML	GP Toolbox	Matlab	Rasmussen and Nickisch (2010)
SFO	Submodular Optimization	Matlab	Krause (2010)
GPy	GP Toolbox	Python	Sheffield ML group (since 2012)
tgp	Tree GPs, GP regression	R	Gramacy et al. (2007)

Get the code e.g. from here

http://sheffieldml.github.io/GPy/

$ pip install GPy

Questions?