An Introduction to Learning

CRT Foundations of Data Science 2020

Alessio Benavoli
Senior Lecturer
Computer Science and Information Systems (CSIS)
University of Limerick
alessio.benavoli@ul.ie
alessiobenavoli.com

17th September 2020

SFI Centre for Research Training
in Foundations of Data Science

What is (Machine) Learning?

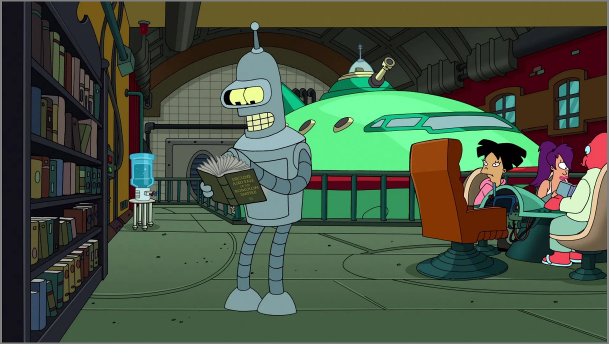


Table of Content

The number game

1. I tell you I am thinking of some simple arithmetical concept C.
2. I give you a series of **randomly** chosen positive examples

$$X = \{x_1, \dots, x_n\},$$

   that is numbers that belong to the concept C.
3. Then I ask you whether any other numbers (**test cases**) y belong to the concept C.

For simplicity, we assume that all numbers are between 1 and 100, so **your** task is to compute whether $y \in C$ given X, for $y \in \{1, \dots, 100\}$; we call it the **generalization task**.[1]

[1] This first part of the talk is a summary from Ch. 1 and 2 of Joshua Tenenbaum's thesis, "A Bayesian Framework for Concept Learning".

Example

I am thinking about “Fibonacci numbers” (but you do not know it) and I give you three examples of the concept:

$$1, 5, 13$$

You have to guess the arithmetical concept C I am thinking about.

Note that many concepts are compatible with the example, for instance

odd numbers

The **generalization task** consists in giving examples of other number belonging to the concept, for instance

7

(this is a wrong generalization because 7 is not Fibonacci).

Let us start the game

I tell you 16 is positive example of the concept.

What other numbers do you think are positive examples?

Is 17? 4? 32? 87? 31?

Example

Is 17? 4? 32? 87? 31?

Maybe, it seems that 17 is more likely to be a positive example of the concept than 87, based on its **proximity** to 16.

Maybe 32 is more likely to be a positive example of the concept than 31, because it shares more **arithmetical** properties with 16.

Similarly, 4 seems to me more likely than 5 or 6.

Maybe 6 is more likely than 4, because it shares the same last digit as 16.

Example

More examples of the concept:

$$\mathbf{16}, 8, 2, 64$$

Now your task should be easier!

Example

More examples of the concept:

$$\mathbf{16}, 8, 2, 64$$

Now your task should be easier!

For instance, it seems that

- 32 and 4 become much more likely to be positive examples than 31 or 6;
- both 17 and 87 seem quite unlikely.

Example

The majority would agree that given the four **random** positive examples

$$16, 8, 2, 64$$

the concept program seems most likely to select the powers of two,

$$2, 4, 8, 16, 32, 64$$

and so on.

Surprising

It is quite amazing that we can conclude that, because given 100 numbers

$$\{1, 2, 3, \ldots, 100\}$$

there are

$$2^{100}$$

possible subsets of numbers between 1 and 100.

Every time we give a positive example of the concept we cut in half the number of logically consistent subsets (that is those containing all the positive examples).

That means that there are still more than a billion billion subsets of numbers including $16, 8, 2, 64$.

Example of these subsets

Example $16, 8, 2, 64$:

- *all powers of two*
- *all even numbers*
- *all numbers less than 65*
- *all numbers less than 90*
- *all powers of two and also 25*
- *all powers of two except 32*
- ...

Despite this HUGE range of possibilities, we believe that we can identify numbers in the one subset that this program selects.

This belief comes after seeing just four random examples known to be in that set

- positive examples of the concept

and no negative examples.

A first solution to the concept learning problem

A rule-based approach[2] can provide a basic solution to our concept learning problem.
In a rule-based system, we

1. consider various hypotheses about what the concept could be (a **reasonable** subset of the 2^{100} possible subsets)
2. eliminate hypotheses which are not consistent with the examples observed.

We denote the initial set of reasonable hypotheses by

$$\mathcal{H}$$

and we call it the **hypothesis space**!
Since the size of $\mathcal{H}$ is less than 2^{100}, generalization from few positive examples becomes practical because most of the many ways to generalize are never even considered by the learner.

[2]The first AI expert systems were rule-based

Rule-based approach

Since the size of $\mathcal{H}$ is much less than 2^{100}, generalization from few positive examples becomes practical because most of the possible ways to generalize are never even considered by the learner.

$$\mathcal{H} = \{\textit{all powers of two}, \textit{all powers of three}, \textit{all powers of four}, \\ \textit{all even numbers}, \textit{all odd numbers}, \textit{all numbers less than 80}, \dots\}$$

After we see 16,

$$\mathcal{H}_1 = \{\textit{all powers of two}, \xcancel{\textit{all powers of three}}, \textit{all powers of four}, \\ \textit{all even numbers}, \xcancel{\textit{all odd numbers}}, \textit{all numbers less than 80}, \dots\}$$

After we see $16, 8, 2, 64$,

$$\mathcal{H}_4 = \{\textit{all powers of two}, \xcancel{\textit{all powers of three}}, \xcancel{\textit{all powers of four}}, \\ \textit{all even numbers}, \xcancel{\textit{all odd numbers}}, \textit{all numbers less than 80}, \dots\}$$

Rule-based approach

The observed examples $16, 8, 2, 64$ are consistent with more than one a priori reasonable hypothesis and, therefore, it is quite unlikely that we can end with a single hypothesis after few examples!

Theoretically, this approach can only work if we have a small hypothesis space and a large enough set of examples to guarantee that only one hypothesis will survive elimination.

After we see $16, 8, 2, 64$,

$$\mathcal{H}_4 = \{\textit{all powers of two}, \xcancel{\textit{all powers of three}}, \xcancel{\textit{all powers of four}}, \textit{all even numbers}, \xcancel{\textit{all odd numbers}}, \textit{all numbers less than 80}, \dots\}$$

What do we choose?

Ranked-hypothesis elimination

The rule-based approach is too simple. Maybe it can be improved if the initial hypothesis space are allowed to be soft, not hard.

That is, instead of just allowing or disallowing candidate hypotheses, we can rank all possible hypotheses using some a-priori idea of **reasonableness**, and the learner chooses the highest ranked hypothesis that is consistent with the examples.

After we see $16, 8, 2, 64$ and given the hypothesis space:

$$\mathcal{H} = \{\textit{all powers of two}, \textit{all powers of three}, \textit{all powers of four}, \\ \textit{all even numbers}, \textit{all odd numbers}, \textit{all numbers less than 80}, \ldots\}$$

under the ranked rule-based view, to justify the preference for *powers of two* over *even numbers*, we would have to assume that *powers of two* has a-priori higher rank than even numbers.

However there is a problem, how can we explain that after seeing

$$16, 8$$

and then

$$16, 8, 2, 64$$

in the second case, we are much more **sure** that *powers of two* is more likely than *even numbers*. The **ranked rule-based** cannot explain that.

Summarising

Both

- rule-based
- rank rule-based

are not good enough!.

We need a different way of learning:

- a model that is able to explain why some consistent hypotheses seem increasingly more likely than others as more examples are observed
- how generalization can be based on multiple hypotheses of (perhaps) varying plausibilities

The solution

1. how does the learner infer, from a small set of positive examples of a concept, which other objects are likely to fall under that concept?
2. why do we infer that, given the random yes examples of $16, 8, 2, 64$, the program probably accepts all and only the powers of two?
3. First, why should the learner settle on one hypothesis, e.g. all powers of two , over others that are equally consistent with the observations $16, 8, 2, 64$ and that seem equally or more natural a priori, such as all even numbers or all numbers less than 100 ?
4. Second, how should the learner generalize when no single hypothesis is clearly more compelling than all others, as after the one example 16?

Avoiding suspicious coincidences

Why do we choose

$h_1 =$ *powers of two* and not for instance $h_2 =$ *even numbers*

after seeing $X = \{16, 8, 2, 64\}$, given that both hypotheses are consistent with the evidence?

The key idea is that we want to **avoid suspicious coincidences**.

If the true concept were *even numbers*, why we did not see any numbers that were not *powers of two*?

Avoiding suspicious coincidences

The fact that $X = \{16, 8, 2, 64\}$ is considered "suspicious" is because we are implicitly making the **strong sampling assumption**, that is that the examples were chosen randomly from the concept.

Under the **strong sampling assumption**, the probability of independently sampling n items (with replacement) from h is given by

$$\textbf{Likelihood:} \quad p(X|h) = \left[\frac{1}{\text{size}(h)}\right]^n$$

This equation is called the **size principle**: it is a form of **Ockhams razor**, which says one should pick the simplest explanation that is consistent with the data.

Example

Let X $= \{16\}$, then

$$p(X|h = \textit{powers of two}) = \frac{1}{6}$$

because there are only 6 powers of 2 less than 100.

This is more likely than:

$$p(X|h = \textit{even numbers}) = \frac{1}{50}$$

and much more likely than inconsistent concepts

$$p(X|h = \textit{odd numbers}) = 0.$$

Example

After seeing $X = \{16, 8, 2, 64\}$,

$$p(X|h = \textit{powers of two}) = \frac{1}{6^4} = 7.7 \cdot 10^{-4}$$

This is much more likely than:

$$p(X|h = \textit{even numbers}) = \frac{1}{50^4} = 1.6 \cdot 10^{-7}$$

This is a likelihood ratio of almost 5000 : 1 in favor of *powers of two*. This confirms mathematically our earlier intuition that $X = \{16, 8, 2, 64\}$ would be a very suspicious coincidence if generated by *even numbers*.

What is the least suspicious hypothesis?

The is the hypothesis that has the maximum likelihood given $X = \{16, 8, 2, 64\}$:

$$\max_h p(X|h) = \frac{1}{\text{size}(h)^4}$$

Note that the most likely hypothesis is not *powers of two*! For instance, the rather unreasonable hypothesis, *powers of two except 32* has

$$p(X|h = \textit{powers of two except 32}) = \frac{1}{5^4} = 1.6 \cdot 10^{-3}$$

It has higher likelihood because it does not need to explain the (small) coincidence that we did not see 32.

To rule out such unreasonable concepts, we need to use some prior knowledge.

Prior

After seeing $X = \{16, 8, 2, 64\}$, Why do we (humans) choose

$$h_1 = \textit{powers of two}$$

and not, for instance,

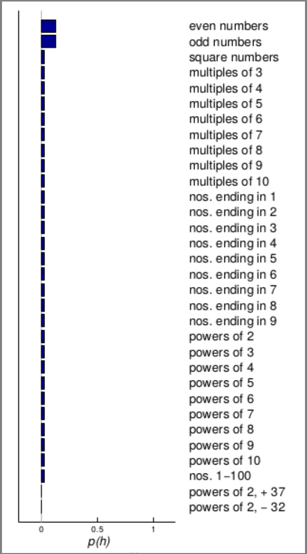
$$h_2 = \textit{powers of two except 32}$$

After all, h_2 has higher likelihood!

However, h_2 is much less likely than h_1 a-priori, because it is **conceptually an unnatural hypothesis**.

It is the combination of the likelihood and the prior knowledge that can explain why we (humans) select h_1.

Prior

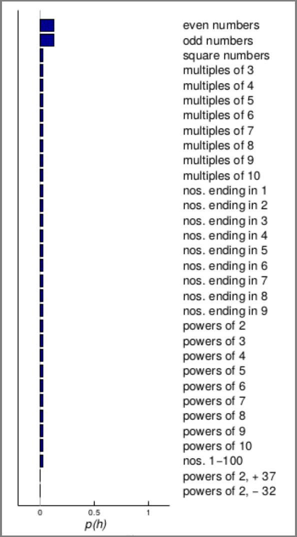


This is my prior, it puts less weight on unnatural concepts such as *powers of two except 32*, and more weight on very simple concepts like *even numbers*.

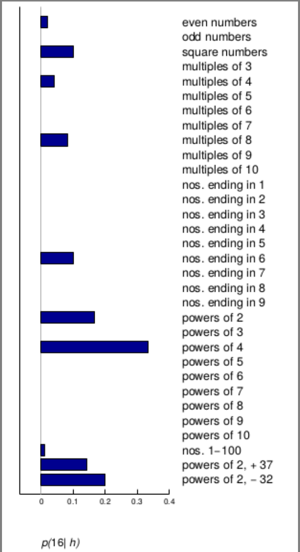
You may have a different prior. In general, the prior encodes the background knowledge on the "concept" we aim to learn.

Prior and Likelihood

Prior

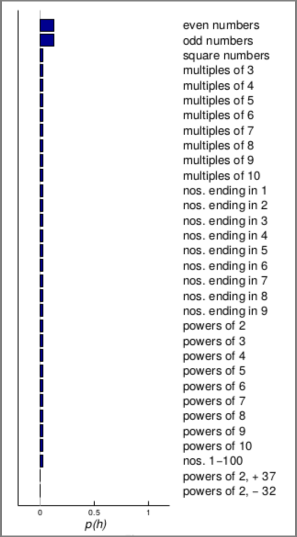


Likelihood ($p(16|h) = \frac{1}{\text{size}(h)}$)

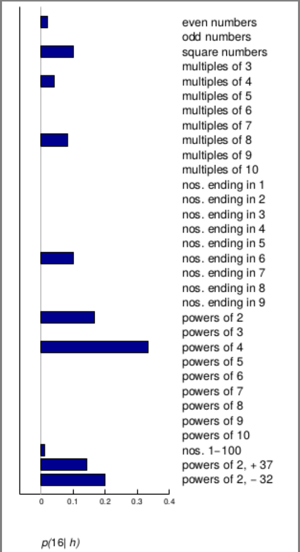


How do we combine them?

Prior



Likelihood $(p(16|h) = \frac{1}{\text{size}(h)})$



How do we combine them?

We have defined

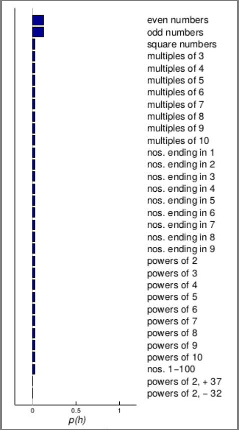
- $p(h)$ is the prior probability on the hypotheses, it tells us how probable we think it is that h is the true hypothesis before we have observed any examples.
- $p(X|h)$ is likelihood, it tells us the probability that we would observe the examples X if h were the true hypothesis.
- we want $p(h|X)$, the probability that the true hypothesis is h given we have observed X. This is called posterior and measures our belief in h after we observed the examples in X.

We want to compute:

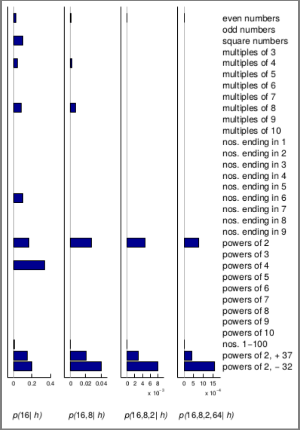
$$p(h|X) = \frac{p(X|h)p(h)}{p(X)} = \frac{p(X|h)p(h)}{\sum_{h'} p(X|h')p(h')}$$

that is Bayes' rule.

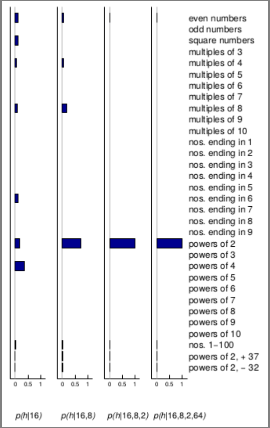
Prior



Likelihood



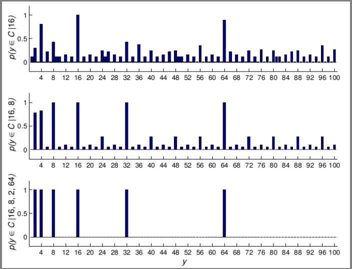
Posterior



Problem of generalization

In the Bayesian setting, the problem of generalization reduces to computing $p(y \in C|X)$, that is the probability that a new number y belongs to concept C, given the set X of previously observed. (Bayesian) Model averaging:[3]

$$p(y \in C|X) = \sum_h p(y \in C|h)p(h|X)$$



[3] $p(y \in C|h) = 1$ if $y \in h$ and zero otherwise.

Summarising the Bayesian framework

1. A constrained hypothesis space $\mathcal{H}$. Otherwise, it is impossible to generalize from a finite data set, because any hypothesis consistent with the evidence is possible.
2. An informative prior, that ranks members of the hypothesis space.
3. The size principle, which is the likelihood function of a strong sampling model.
4. Hypothesis averaging, i.e., integrating out h when making predictions:

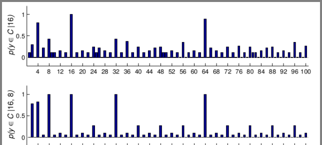
$$p(y \in C|X) = \sum_h p(y \in C|h)p(h|X)$$

Maximum Likelihood Estimation (MLE)

It has only ingredients 1 and 3.

- 1. A constrained hypothesis space $\mathcal{H}$. Otherwise, it is impossible to generalize from a finite data set, because any hypothesis consistent with the evidence is possible.
- 3. The size principle, which is the likelihood function of a strong sampling model.

Issues: given the example 16, the MLE hypothesis is *all powers of four*, so only 4 and 64 receive a nonzero probability of generalization, while given two examples $\{16, 8\}$, all $4, 8, 16, 64$ receive a nonzero probability of generalization. MLE can make very sharp decisions after only one (few) observation (**overfitting**), while for the Bayesian method this is never the case:



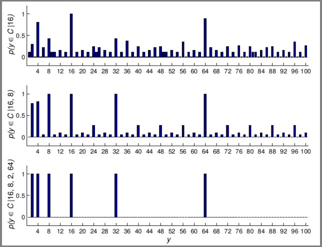
Maximum A-Posteriori (MAP)

It has ingredients 1, 2 and 3.

- 2. An informative prior, that ranks members of the hypothesis space.

It returns the hypothesis that has the highest posterior probability: $\max_h p(h|X)$.

Issues: without averaging, MAP does not account for the uncertainty, that is the fact that there may be alternative hypotheses that have very close posterior probability.



How is this related to Machine Learning?

The vast majority of the algorithms (for regression, classification and clustering) you have used (will use) are based on

- MLE (minimum loss function)
- MAP (regularisation)

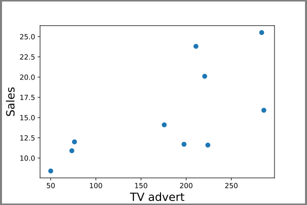
Scikit-learn the most used machine learning library in Python: Linear regression (MLE), Logistic regression (MAP), Neural Network (MAP)...

Keras is the most used deep learning framework (it uses MLE by default).

Issue: MLE leads to overfitting and MLE/MAP do not consider the uncertainty.

Parametric regression: linear model

Consider this dataset: the x represents yearly money invested in TV advertisements and y the sales of a product.



Our goal is to learn what is a relationship between x and y. In parametric regression, we first define the parametric form of this relationship and then learn the parameters from data. The most common functional form is parametric linear model:

$$y = \beta x + \alpha,$$

with $\alpha, \beta \in \mathbb{R}$.

Bayesian learning framework: prior

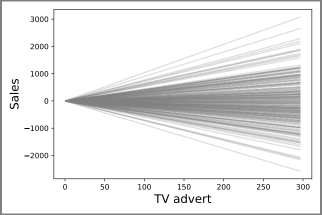
Ingredient 1:

$$y = \beta x + \alpha,$$

$\mathcal{H} = \{\alpha, \beta \in \mathbb{R}\}$.

Ingredient 2: prior hypothesis space $\mathcal{H}$

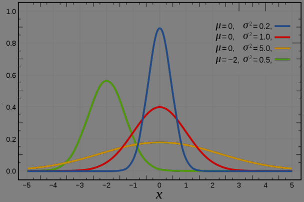
$$p(h) = p(\alpha, \beta) \quad \Big(= N(\alpha, 0, 10) N(\beta, 0, 10) \Big)$$



Gaussian PDF

Gaussian (or normal) distribution has PDF of a continuous variable x that can take values in the possibility space $\Omega = \mathbb{R}$ (a real number).
It is denoted as $N(x; \mu, \sigma)$ and defined as

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



Bayesian learning framework: likelihood

Ingredient 3: likelihood $p(data|h) = p(data|\alpha, \beta)$.

We start from the premise that, although the relation between x and y may be linear, the data may not follow it exactly (there is some *noise*). The main reason is that the observation mechanism is usually **imperfect**.
This means that

$$y_{observed} \neq y_{real}$$

We account for the imperfect observation mechanism with a probabilistic model

$$p(y_{observed}|y_{real})$$

Bayesian learning framework: likelihood

In linear regression, it is usually assumed that this PDF is Gaussian:

$$p(y_{observed}|y_{real}) = N(y_{observed}; y_{real}, \sigma) = N(y_{observed}; \beta\, x + \alpha, \sigma)$$

We usually denote $y_{observed}$ simply with y.
We have n observations (training data):

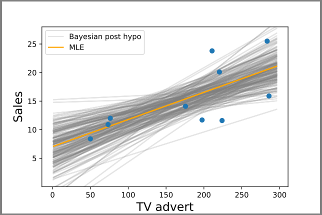
$$D = \{(x_i, y_i) : \ i = 1, \ldots, n\}$$

Assuming the observations are **conditional independent** given the parameters,

$$p(D|\alpha, \beta, \sigma) = \prod_{i=1}^{n} N(y_i; \beta\, x_i + \alpha, \sigma)$$

Bayesian learning framework: posterior

$$\underbrace{p(\alpha,\beta,\sigma|\text{data})}_{\text{posterior}} = \frac{\underbrace{p(\text{data}|\alpha,\beta,\sigma)}_{\text{likelihood}}\underbrace{p(\alpha,\beta,\sigma)}_{\text{prior}}}{\underbrace{p(\text{data})}_{\text{evidence}}}$$

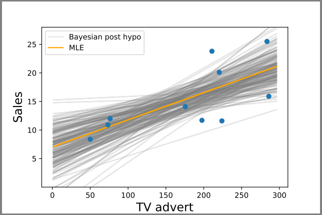


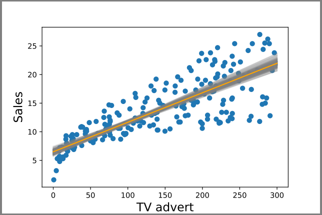
Compared with

$$\text{MLE:} \quad \arg\max_{\alpha,\beta} p(\text{data}|\alpha,\beta,\sigma) = \arg\min_{\alpha,\beta} \sum_{i=1}^{n} (y_i - \alpha - \beta x_i)^2$$

Bayesian learning framework: posterior

$$\underbrace{p(\alpha, \beta, \sigma | \text{data})}_{\text{posterior}} = \frac{\overbrace{p(\text{data}|\alpha, \beta, \sigma)}^{\text{likelihood}} \overbrace{p(\alpha, \beta, \sigma)}^{\text{prior}}}{\underbrace{p(\text{data})}_{\text{evidence}}}$$





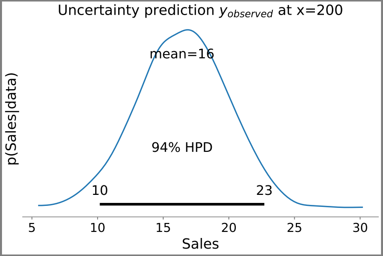
The MLE is the same in the two cases, but the uncertainty is smaller in the second case (because there are more data points).

Bayesian learning framework: averaging

Ingredient 4: averaging

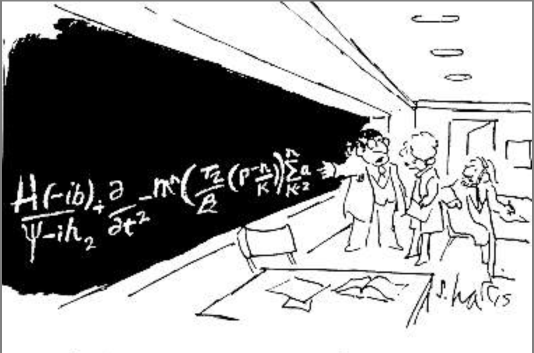
$$p(y|x^*, \text{data}) = \int p(y|x^*, \alpha, \beta, \sigma)p(\alpha, \beta, \sigma|\text{data})$$

What are the predicted Sales if we spend 200 in TV advertisement?



The MLE prediction is 16.5.

How do we get those lines?



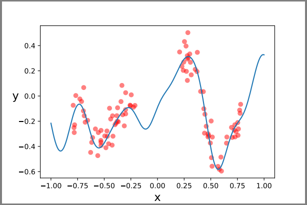
"But this *is* the simplified version for the general public."

It is Bayesian Inference and as you learned we need to solve integrals. In this case, we can do it in closed form, but more in general we can use Stan or PyMC3 or any other probabilistic programming language.

How do we get those lines?

```
import pymc3 as pm
regmodel=pm.Model()
with regmodel:
    #prior hypothesis
    alpha = pm.Normal('alpha',0.0,30)
    beta  = pm.Normal('beta',0.0,30)
    sigma = pm.Uniform('sigma',0.0001,20)
    mu = pm.Deterministic('mu',beta*x+alpha)
    #likelihood
    yo=pm.Normal('Like',mu,sigma,observed=y)
# it computes samples from the posterior using HMC
with regmodel:
    samples=pm.sample(5000)
```

Parametric Nonlinear regression



The assumption of linearity is very restrictive. A very simple idea to overcome this problem is to first project the inputs x into some high dimensional space using a set of basis functions and then apply the linear model in this space instead of directly on the inputs themselves.

For example, a scalar input x $(m = 1)$ could be projected into the space of powers:

$$\phi(x) = (1, x, x^2, x^3, \dots, x^d)$$

obtaining polynomial regression.
Another example: x is a scalar input, we project it into the space of functions of x:

$$\phi(x) = (\tanh(w_{11}^{(1)} + w_{12}^{(1)}x), \tanh(w_{21}^{(1)} + w_{22}^{(1)}x), \dots, \tanh(w_{d1}^{(1)} + w_{d2}^{(1)}x))$$

where $wij^{(1)} \in \mathbb{R}$ are parameters, obtaining a Neural Network (Multilayer perceptron).

Polynomial regression

$$\phi(x) = (1, x, x^2, x^3, \ldots, x^d)$$

and, therefore,

$$\begin{aligned} f(x) = \phi(x)^T w &= \sum_{i=0}^{d} w_i x^i \\ &= w_0 + w_1 x + w_2 x^2 + \cdots + w_d x^d \end{aligned}$$

This is our polynomial regression function with unknown parameters w_i.

$$\begin{aligned} p(h) &= p(w_0, w_1, \ldots, w_d) \\ p(\text{data}|h) &= \prod_{i=1}^{n} N(y_i; w_0 + w_1 x + w_2 x^2 + \cdots + w_d x^d, \sigma) \end{aligned}$$

MLP

$$\phi(x) = [\tanh(w_{11}^{(1)} + w_{12}^{(1)} x), \tanh(w_{21}^{(1)} + w_{22}^{(1)} x), \dots, \tanh(w_{\ell 1}^{(1)} + w_{\ell 2}^{(1)} x)]^T$$

and

$$f(x) = \phi(x, W^{(1)})^T w^{(2)} = \sum_{i=1}^{\ell} w_i^{(2)} \tanh(w_{i1}^{(1)} + w_{i2}^{(1)} x)$$

Prior

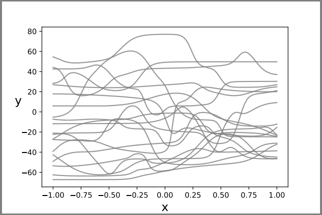
Ingredient 1:

$$y = \sum_{i=1}^{\ell} w_i^{(2)} \tanh(w_{i1}^{(1)} + w_{i2}^{(1)} x),$$

$\mathcal{H} = \{w_{ij}^{(1)}, w_{ij}^{(2)} \in \mathbb{R}\}.$

Ingredient 2: prior hypothesis space $\mathcal{H}$

$$p(h) = p(w_{ij}^{(k)}) \quad \Big(= N(w_{ij}^{(k)}, 0, 10)\Big)$$



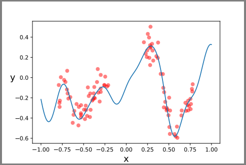
Likelihood

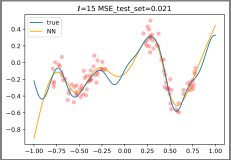
Ingredient 3:

$$p(\text{data}|h) = \prod_{i=1}^{n} N\left(y_i; \sum_{i=1}^{\ell} w_i^{(2)} \tanh(w_{i1}^{(1)} + w_{i2}^{(1)} x), \sigma\right)$$

.

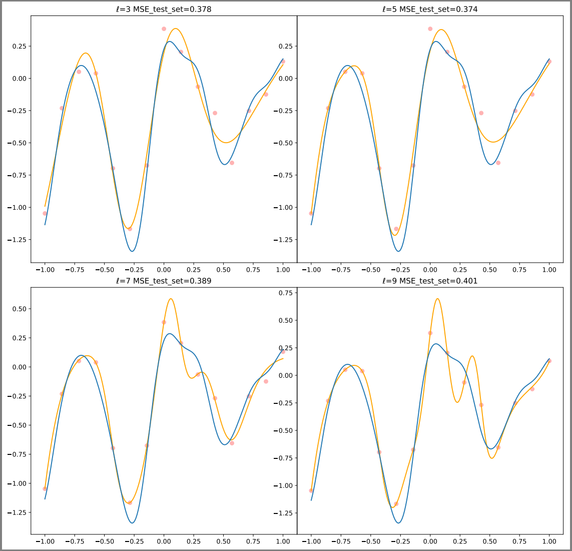
MLE: Ingredient 1 and 3





The fitting error is good in the regions with data and bad in the regions without data. This is reasonable, but NN should tell us that the prediction for $x \in [-0.2, 0.2]$ is less reliable than that for $x \in [0.2, 0.8]$.

Overfitting



MLE can lead to Overfitting

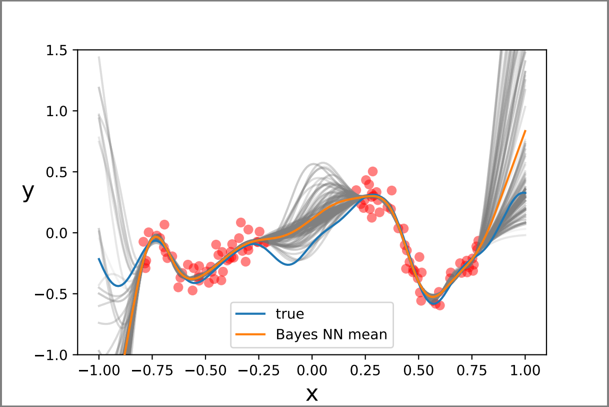
NNs can lead to overfitting.

This happens when the number of parameters is “too large” compared to the size of the data.

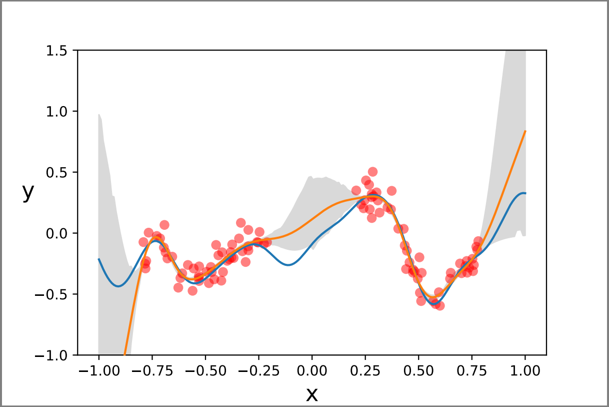
We can select model complexity via cross-validation.

Traditional NN learning does not provide any measure of uncertainty of the estimate and prediction.

Bayesian learning

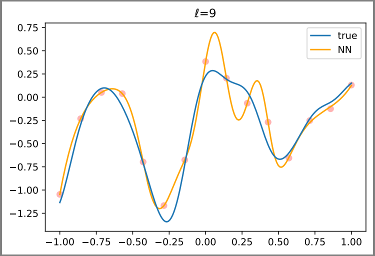


95% HPD

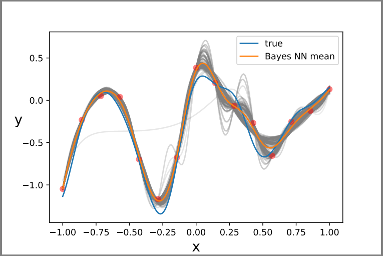


This is the uncertainty about the true function, no about the prediction of future data.

MLE vs. Bayesian learning



MLE



Bayesian framework

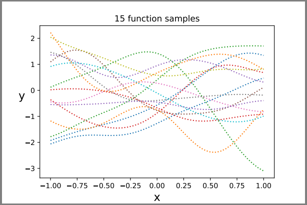
How do we get those lines?

```
import pymc3 as pm
l = 15
with pm.Model() as neural_network:
    ann_input = pm.Data('ann_input', x)
    ann_output = pm.Data('ann_output', y)
    #Priors
    weights_in_1 = pm.Normal('w_1', 0, sigma=10,
                             shape=(x.shape[1], l))
   weights_2_out = pm.Normal('w_0', 0, sigma=10,
                              shape=(l,))
  act_1 = pm.math.tanh(pm.math.dot(ann_input,weights_in_1))
  act_out = pm.Deterministic('act_out',pm.math.dot(act_1,weights_2_out)
    sigma = pm.HalfCauchy('sigma',5)
    # Normal likelihood
    out = pm.Normal('out',act_out,sigma=sigma,observed=y)
#this can be slow because there are many parameters
with neural_network:
    posterior = pm.fit(100000)#advi
```

Bayesian Nonparametric regression

Can we place a prior directly on the unknown function f without using any parametric form?

So it means that our prior hypothesis space is infinite, the space of functions.

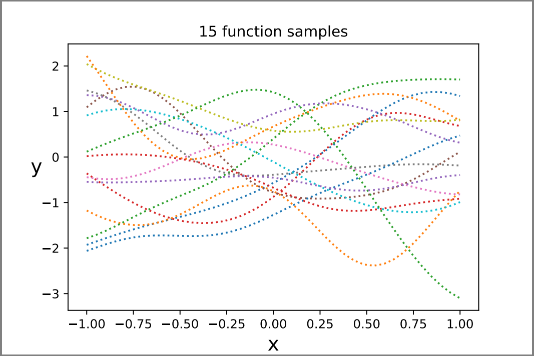


Ingredient 1: $y = f(x)$ and $\mathcal{H} = \{f \in \mathcal{F}\}$.

Ingredient 2: prior hypothesis space $\mathcal{H}$

$$p(h) = p(f) \quad \Big(f(x) \sim GP(0, k(x, x)\Big)$$

Example: prior over f

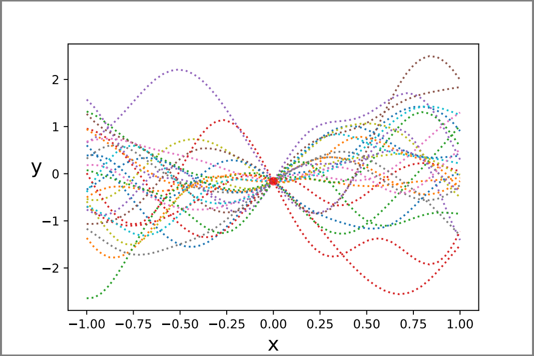


Example: posterior after 1 observation

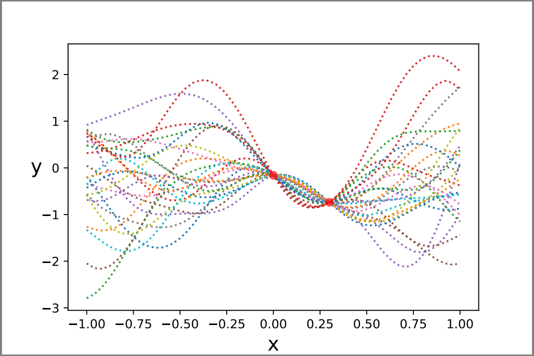
Assume that there is not noise

$$y = f(x)$$

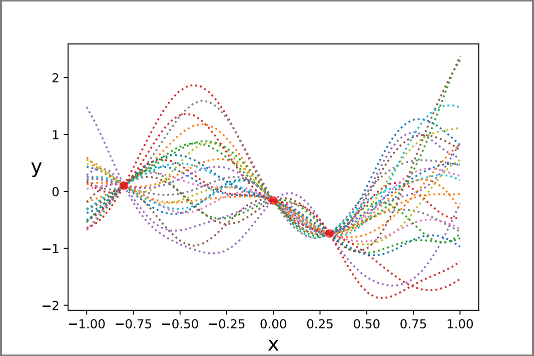
that is $\sigma = 0$.



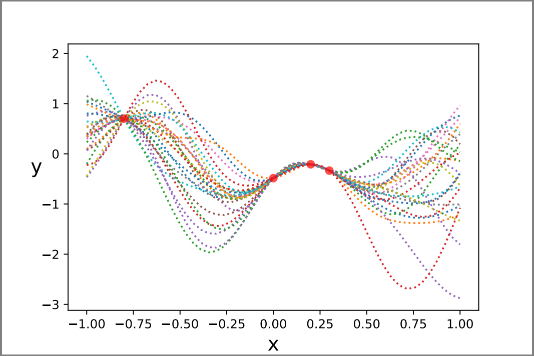
Example: posterior after 2 observations



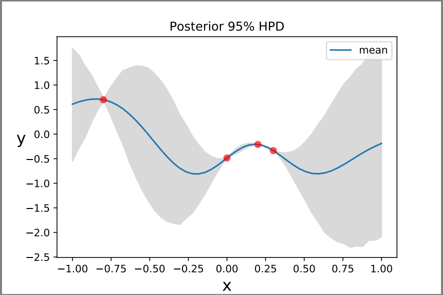
Example: posterior after 3 observations



Example: posterior after 4 observations



Example: HPD after 3 observations



This model is called “Gaussian Process Regression” and we can compute the posterior distribution in **closed form**. It is called nonparametric because our hypothesis space is infinite, that means that actually has infinite parameters:

$$f(x) = \sum_{i=1}^{\infty} w_i^{(2)} \tanh(w_{i1}^{(1)} + w_{i2}^{(1)} x)$$

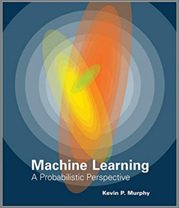
Conclusions

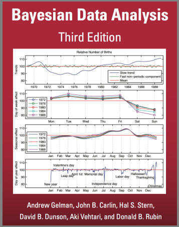
Future of AI and Machine Learning

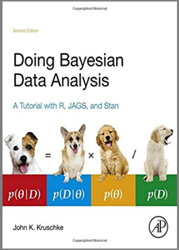
- Probabilistic Computing Takes Artificial Intelligence to the Next Step;
- Causal Learning.

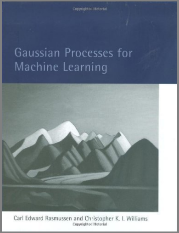
My favourite books

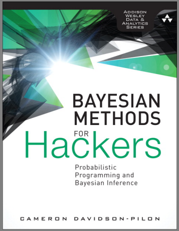
but also the books you should read

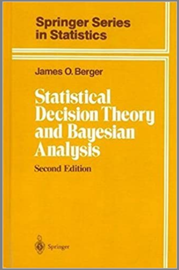



Bayesian Data Analysis
Third Edition
Andrew Gelman, John B. Carlin, Hal S. Stern,
David B. Dunson, Aki Vehtari, and Donald B. Rubin




Copyrighted Material
Gaussian Processes for Machine Learning
Carl Edward Rasmussen and Christopher K. I. Williams
Copyrighted Material


Addison Wesley Data & Analytics Series
BAYESIAN METHODS FOR Hackers
Probabilistic Programming and Bayesian Inference
CAMERON DAVIDSON-PILON


Springer Series in Statistics
James O. Berger
Statistical Decision Theory and Bayesian Analysis
Second Edition
Springer

References

- The number game is from Joshua Tenenbaum's thesis, "A Bayesian Framework for Concept Learning", MIT, 1999.
- PyMC3 `https://docs.pymc.io/`
- Short intro to the theory behind GP regression `https://gregorygundersen.com/blog/2019/06/27/gp-regression/`
- Practical Implementation of GPs `https://gregorygundersen.com/blog/2019/09/12/practical-gp-regression//`