UNIVERSITÄT BONN

Machine Learning Basics

Elena Trunz

September 15, 2023

Uni Bonn

Overview

Machine Learning Algorithms

Generalization

Parameter Estimation

Machine Learning Algorithms

What is machine learning?

Machine learning is a subfield of artificial intelligence with the goal of developing algorithms capable of learning from data automatically.

What do we mean by learning?

"A computer program is said to learn from *experience* E with respect to some class of *tasks* T and *performance measure* P, if its performance at tasks in T, as measured by P, improves with experience E." (Mitchell, 1997[1])

What is machine learning?

Machine learning is a subfield of artificial intelligence with the goal of developing algorithms capable of learning from data automatically.

What do we mean by learning?

"A computer program is said to learn from *experience* E with respect to some class of *tasks* T and *performance measure* P, if its performance at tasks in T, as measured by P, improves with experience E." (Mitchell, 1997[1])

What is machine learning?

Machine learning is a subfield of artificial intelligence with the goal of developing algorithms capable of learning from data automatically.

What do we mean by learning?

"A computer program is said to learn from *experience* E with respect to some class of *tasks* T and *performance measure* P, if its performance at tasks in T, as measured by P, improves with experience E." (Mitchell, 1997[1])

[1] Mitchell, T. M. (1997). Machine Learning. McGraw-Hill, New York.

Experience E

The experience E is the information that the algorithm can use during learning:

- *dataset*
- aka (training) data

Dataset contains *examples* (*data points*): collection of *features* that have been quantitatively measured from some object or event

- m-dimensional data point: $\mathbf{x} \in \mathbb{R}^m$
- each entry x_i is another feature

Dataset: example

Iris dataset[2]

- Collection of measurements of different parts of 150 iris plants
- Each individual plant coresponds to one example
- The features within each example are
  - sepal length
  - sepal width
  - petal length
  - petal width
- Recording, which species each plant belonged to
- Three different species of iris plants

[2] `https://archive.ics.uci.edu/ml/datasets/iris`

Supervised vs. unsupervised learning

Supervised learning:

- Each data point $\mathbf{x}$ comes with an associated *label* or *target* value $\mathbf{y}$
- The algorithm is learned by trying to match the targets, or predict $\mathbf{y}$ from $\mathbf{x}$
- From a probabilistic sense, this corresponds to estimating $P(\mathbf{y}|\mathbf{x})$

Unsupervised learning:

- Only data points $\mathbf{x}$, no *labels*
- The goal is to uncover the inherent structure within the data
- From a probabilistic sense, this corresponds to estimating *data generating distribution* $p_{\text{data}}(\mathbf{x})$

Machine learning tasks

Tasks specify how a machine learning system should process *examples*. Common tasks include

- **classification**: determine category of input i.e. $f : \mathbb{R}^m \to \{1, ..., K\}$
- **regression**: predict numerical value(s) for some input, i.e. $f : \mathbb{R}^m \to \mathbb{R}^p$
- **density estimation**: learn probability density (if $\mathbf{x}$ is continuous) or probability mass (if $\mathbf{x}$ is discrete) function $p_{\text{model}} : \mathbb{R}^m \to \mathbb{R}$ on the space that the examples were drawn from
- **dimensionality reduction**: find a compact, lower-dimensional representation of high-dimensional data $\mathbf{x} \in \mathbb{R}^D$, which is often easier to analyze than the original data

More machine learning tasks

- **imputation of missing values** or **hole filling**: predict values of missing entries x_i of $\mathbf{x} \in \mathbb{R}^m$
- **structured prediction**: vector output with well-defined relationships between the elements
- **synthesis**: generation of new examples that are similar to the training data
- **denoising**: get *clean* example $\mathbf{x} \in \mathbb{R}^m$ from *corrupted* example $\tilde{\mathbf{x}} \in \mathbb{R}^m$

Performance measure P

To evaluate a machine learning algorithm, we must design a quantitative measure of its performance.

Performance measure is task-spesific:

- Classification and imputation of missing inputs:
  - *Accuracy*: proportion of examples for which the model produces the correct output
  - *Error rate*: proportion of examples for which the model produces an incorrect output
- Density estimation:
  - Average log-probability the model assigns to examples

Generalization

We want to evaluate performance of our system on data that it has *not yet seen*.

This estimates how well the system will *generalize*, in comparison to performance on already seen data used during the learning process.

- Unseen data makes up our *test set* or test data
- Already seen data is the *training set* or training data

During training we want to reduce the *training error* – this is simply an optimization problem.

What distinguishes learning from optimization is the interest in also reducing the *test error* (*generalization error*).

Linear regression: experience

Recall the regression problem: the goal is to build a system that

- takes a vector $\mathbf{x} \in \mathbb{R}^m$ as input and
- predicts the value of a scalar $y \in \mathbb{R}$ as its output.

Our experience E are the input examples $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ and the corresponding target values $\mathbf{y} \in \mathbb{R}^N$.

Thus, the dataset consists of N example pairs
$\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$

Linear regression: task

Let $\hat{y}$ be the value that our model predicts y should take on. We define the output to be

$$\hat{y} = \mathbf{w}^\top \mathbf{x}$$

where $\mathbf{w}$ is a vector of model *parameters*.

Parameters are values that control the behavior of the system.

$\rightarrow$ Our task T: predict y from $\mathbf{x}$ by outputting $\hat{y} = \mathbf{w}^\top \mathbf{x}$.

Linear regression: performance measure

Now we need to define our performance measure P.

Suppose we have M example inputs together with the correct target values in the test dataset.

One way of measuring the performance of the model for a regression task, is to compute the *mean squared error* (MSE):

$$\begin{aligned} MSE_{test} &= \frac{1}{M} \sum_i (\hat{\mathbf{y}}^{(test)} - \mathbf{y}^{(test)})_i^2 \\ &= \frac{1}{M} \|\hat{\mathbf{y}}^{(test)} - \mathbf{y}^{(test)}\|_2^2 \end{aligned}$$

So the error increases whenever the Euclidean distance between the predictions and the targets increases.

Linear regression: ML algorithm

To make a machine learning algorithm, we need to design an algorithm that will improve the parameters $\mathbf{w}$ in a way that reduces MSE_{test} when the algorithm is allowed to gain experience by observing the training set $(\mathbf{X}^{(train)}, \mathbf{y}^{(train)})$.

One way to do it is to minimize MSE_{train}. We can simply solve for where the gradient of MSE_{train} is $\mathbf{0}$:

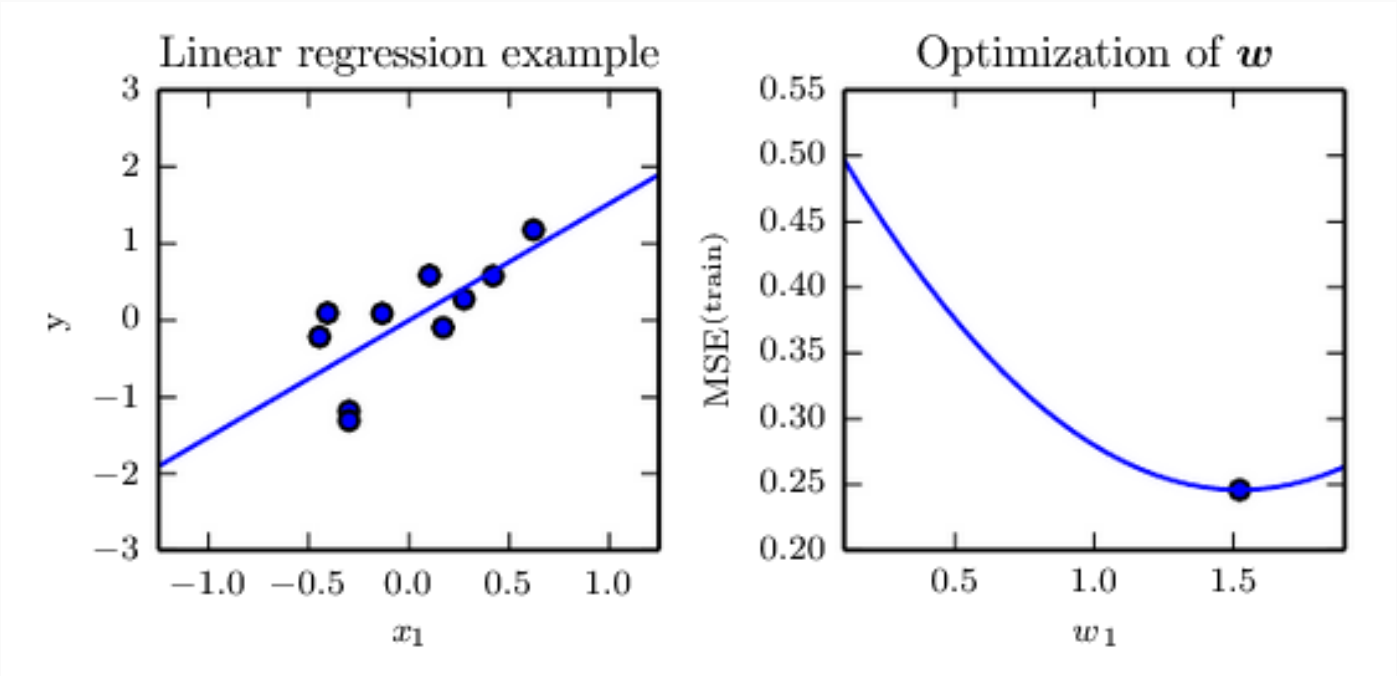
$$\nabla_{\mathbf{w}} MSE_{train} = \mathbf{0}$$

$$\nabla_{\mathbf{w}} \frac{1}{M} \|\hat{\mathbf{y}}^{(train)} - \mathbf{y}^{(train)}\|_2^2 = \mathbf{0}$$

$$\frac{1}{M} \nabla_{\mathbf{w}} \|\mathbf{X}^{(train)}\mathbf{w} - \mathbf{y}^{(train)}\|_2^2 = \mathbf{0}$$

$$\Rightarrow \mathbf{w} = (\mathbf{X}^{(train)\top}\mathbf{X}^{(train)})^{-1}\mathbf{X}^{(train)\top}\mathbf{y}^{(train)}$$

Linear regression: result



Training sets consists of ten examples, each containing one feature. So there is only one parameter to learn, w_1 (= slope).

Image was taken from [GBC16]

Linear regression: notes

Usually linear regression refers to a model with an additional parameter – an intercept term b:

$$\hat{y} = \mathbf{w}^\top \mathbf{x} + b$$

This means that the plot of the model's predictions need not pass though the origin.

We can continue to use the model $\hat{y} = \mathbf{w}^\top \mathbf{x}$, by implicitly including b:

- Augment $\mathbf{x}$ with an extra entry that is always set to 1
- The weight corresponding to the extra 1 entry plays the role of the bias parameter b

Loss function

A *loss function* is a function that measures the difference (or loss) between a predicted label and a true label (e.g. MSE_{train} in the regression example).

It is closely related to performance measure:

- Loss function is used *during* the training to *train* the model
- Performance measure is used *after* the training to *evaluate* the model

Sometimes both can be represented by the same function (as it was in the case of regression), but for some other tasks (e.g. classification) we have different measures (e.g. cross-entropy vs. accuracy).

Generalization

Underfitting and overfitting

How well a machine learning algorithm will perform, depends on its ability to

1. Make the training error small
2. Make the gap between training and test error small

Algorithms which *underfit*, cannot satisfy the first criteria.

Algorithms which *overfit*, can satisfy the first criteria but not the second.

Model Capacity

Underfitting and overfitting can be controlled by adjusting the *model capacity* in the learning algorithm.

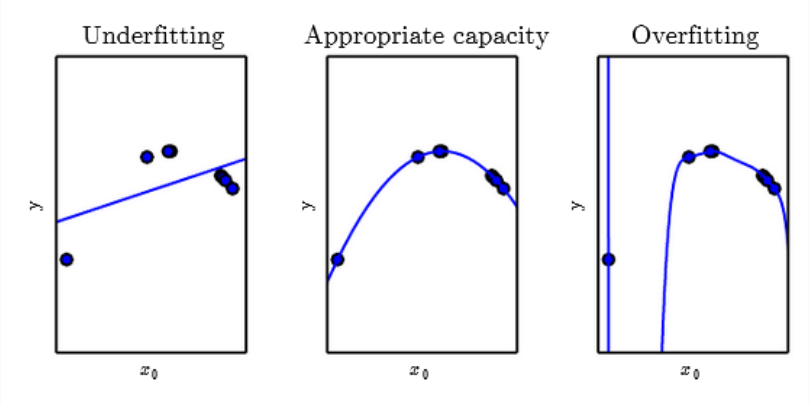
A model's capacity is its ability to fit a wide variety of functions.

Models with (too) low capacity are not rich or expressive enough to capture the training data and hence underfit.

Models with (too) high capacity simply memorize the training data and will therefore overfit.

Model Capacity (2)

- Capacity can be controlled by the choice of the *hypothesis space*. This space determines the set of functions that the learning algorithm can select as being the solution.
- The linear regression algorithm has the set of all linear functions of its input as its hypothesis space.
- We can increase the model's capacity by generalizing linear regression to include polynomials in its hypothesis space.

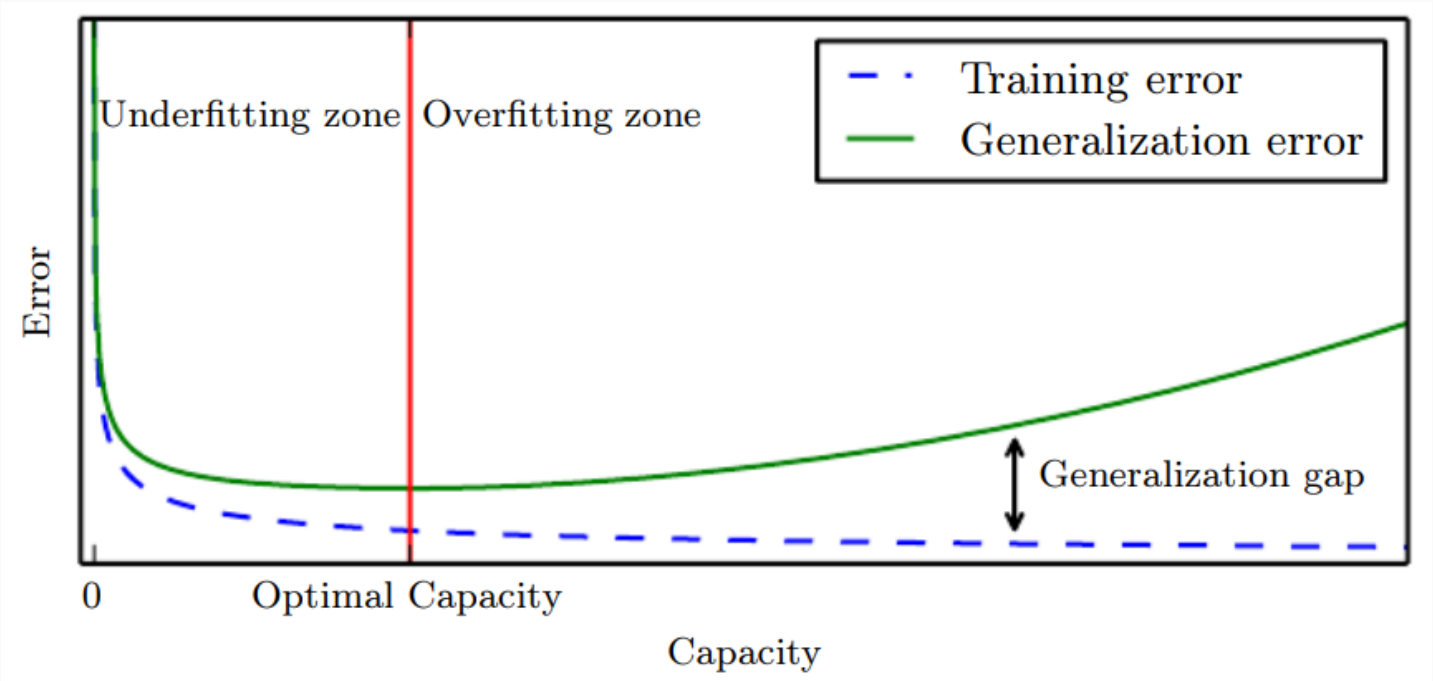


Capacity here is determined by the polynomial degree.

Image was taken from [GBC16]

Capacity vs. error

Depending on the model capacity training and test error behave differently:



Typical relationship between capacity and error.

Image was taken from [GBC16]

Regularization

Another way to control underfitting and overfitting is to drastically increase the capacity and use a *regularizer* so as to not overfit.

Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.

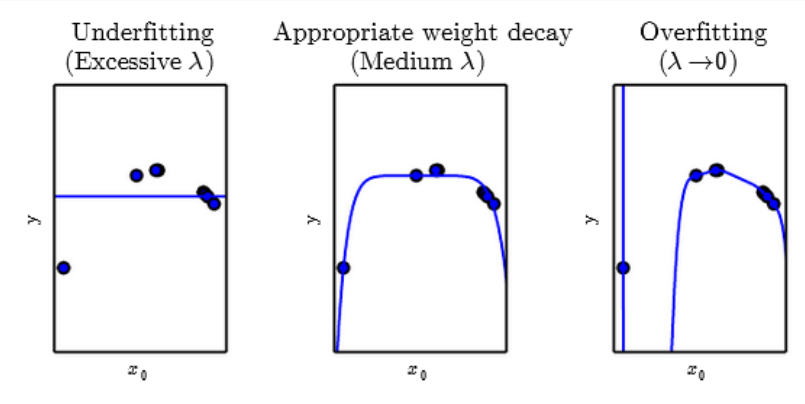
We can regularize a model by giving a learning algorithm a preference for one solution over another in its hypothesis space.

Regularization: weight decay

- We can modify the objective function (loss function) for linear regression to include *weight decay*:

$$J(\mathbf{w}) = MSE_{train} + \lambda \mathbf{w}^\top \mathbf{w}$$

- Regularizer penalizes weights that have bigger squared L^2 norm. λ controls the strength of our preference for smaller weights.



We fit a high-degree polynomial regression model and use weight decay against overfitting.

Image was taken from [GBC16]

Hyperparameters

The parameters $\mathbf{w}$ of the regression example are determined (optimized) during the training. Such parameters are also called *weights*.

Most learning algorithms have some free parameters that are not determined by the learning algorithm, but are set in advance to control the algorithm's behavior.

Such free parameters are called *hyperparameters*.

In the previous regression example the polynomial degree and λ are two hyperparameters.

Hyperparameter estimation

We cannot use our test set to make any choices about the model, including its hyperparameters.

To determine the hyperparameters, we need a *validation set* of examples that the training algorithm does not observe.

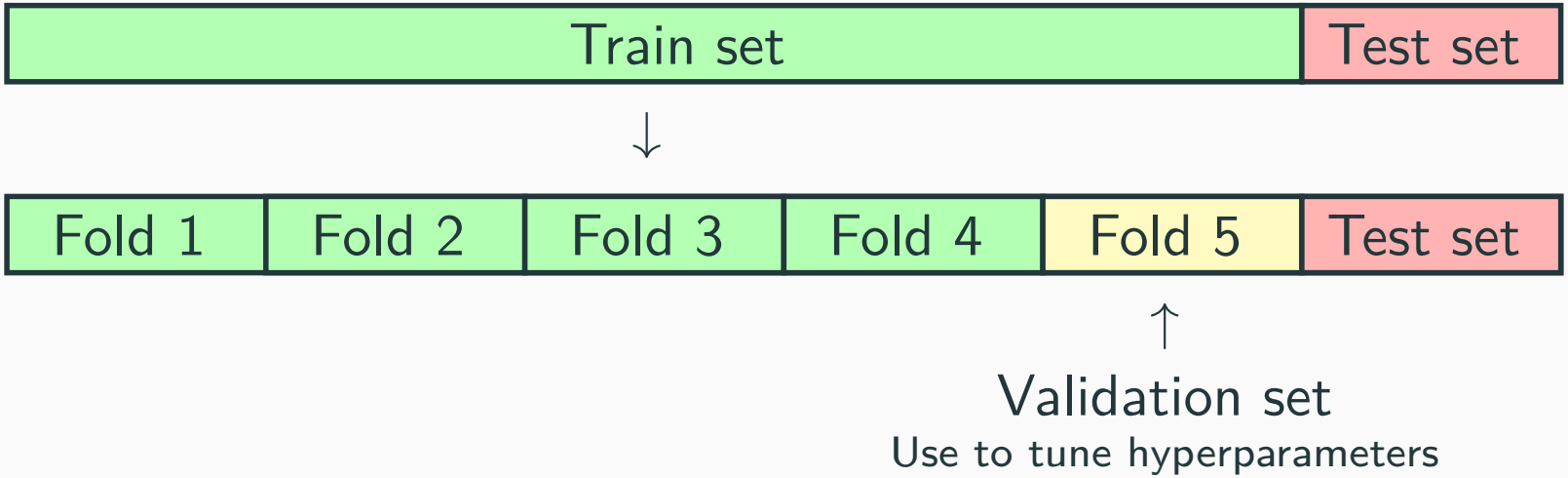
We always construct the validation set from the training data:

- Split the training data into two disjoint subsets
- One of the subsets is used to learn the parameters
- The other subset is used to estimate the generalization error during or after training and to guide the selection of the hyperparameters

After the training and all hyperparameter optimization is complete, we can estimate the generalizaiton error on the test set.

Cross-validation

If the training set is small and the split would result in the validation set being too small, cross-validation can help:



- Loop through all folds to be the validation fold
- Other folds build the train set
- Average performance values

What do we mean by learning? - revised

Learning: Based on training data, finding a model with parameters θ that generalizes well

- Empirical risk minimization view:
  - Predictor as a function
  - Loss function
  - Regularization
- Probabilistic view:
  - Predictor (estimator) as a probabilistic model
  - Likelihood
  - Prior

  $\rightarrow$ Estimating parameters based on data

Parameter Estimation

Point estimation

Point estimation is the attempt to provide the single "best" prediction of some quantity of interest:

- Single parameter
- Vector of parameters (e.g. weights in our linear regression example)
- Function

Point estimation (2)

Let $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ be a set of N independent and identically distributed (i.i.d.) data points. A *point estimator* is any function of the data:

$$\hat{\boldsymbol{\theta}}_N = g(\mathbf{x}_1, \ldots, \mathbf{x}_N).$$

- Good estimator: Function whose output is close to the true underlying $\boldsymbol{\theta}$ that generated the training data.

Properties of point estimators

How can we tell, if an estimator is any good?

$\rightarrow$ We can analyse two most common properties of point estimators:

- Bias
- Variance

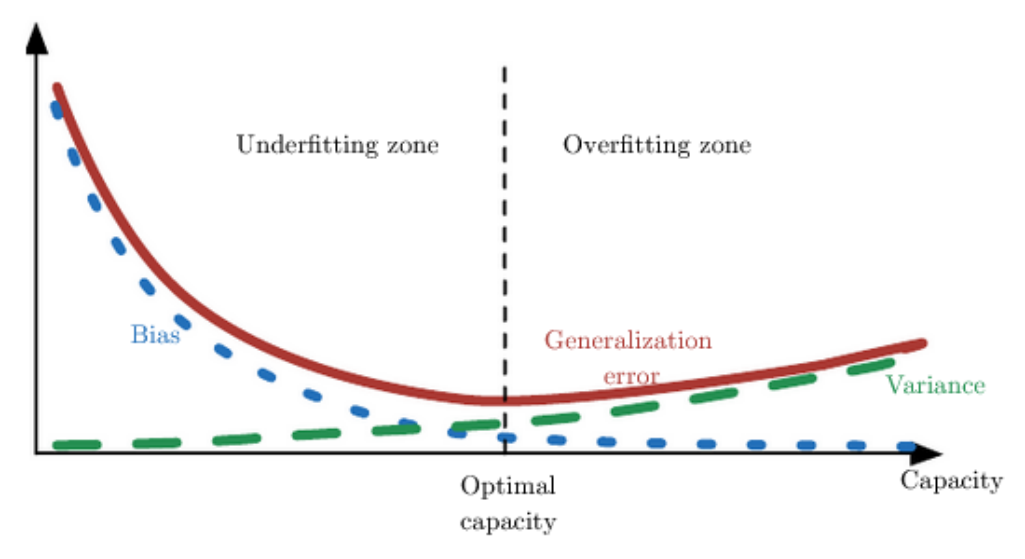
Bias

- What is the inherent error that persists in your estimator, even when trained with an unlimited amount of data?
- The error arised because the estimator leans towards a particular type of solution (e.g. linear regression)
- Bias is an inherent characteristic of the model

Variance

- Measures, how much we expect the estimator to vary as a function of the data sample, i.e. if we were to independently re-sample the dataset from the underlying data generating process
- To what extent does the estimator exhibit excessive specialization to a specific training set (overfitting)?

Bias-Variance tradeoff



The relationship between bias and variance is tightly linked to the concepts of underfitting, overfitting and model capacity.

Image was taken from [GBC16]

How to find a good estimator?

- We can now analyse the properties of the estimator and distinguish a good one
- Where do good estimators come from?
- Rather than guessing that some function might be a good estimator and then analyzing its bias and variance, we need some principle from which we can derive specific functions that are good estimators.

$\rightarrow$ MLE and MAP

Where does our data come from?

- Data comes from distribution $p(\mathbf{X}, \mathbf{Y})$
- If we had access to this distribution, we could solve for $P(y|\mathbf{x})$
- Can we approximate this distribution from the data?
- We start with the distribution $p(\mathbf{X})$ (unsupervised) and then generalize to $p(\mathbf{X}, \mathbf{Y})$ (supervised)

Maximum likelihood estimation (MLE)

Let $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ be a set of N (i.i.d.) data points. Then the MLE for $\boldsymbol{\theta}$ is defined as

$$\boldsymbol{\theta}_{\text{ML}} = \underset{\boldsymbol{\theta}}{\text{argmax}}\, p(\mathbf{X}; \boldsymbol{\theta})$$

MLE (2)

Since $\mathbf{x}_i$ are independent, we can rewrite it as

$$\begin{aligned}\boldsymbol{\theta}_{\mathrm{ML}} &= \operatorname*{argmax}_{\boldsymbol{\theta}} p(\mathbf{X}; \boldsymbol{\theta}) \\ &= \operatorname*{argmax}_{\boldsymbol{\theta}} \prod_{i=1}^{m} p(\mathbf{x}_i; \boldsymbol{\theta}) \\ &= \operatorname*{argmax}_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p(\mathbf{x}_i; \boldsymbol{\theta})\end{aligned}$$

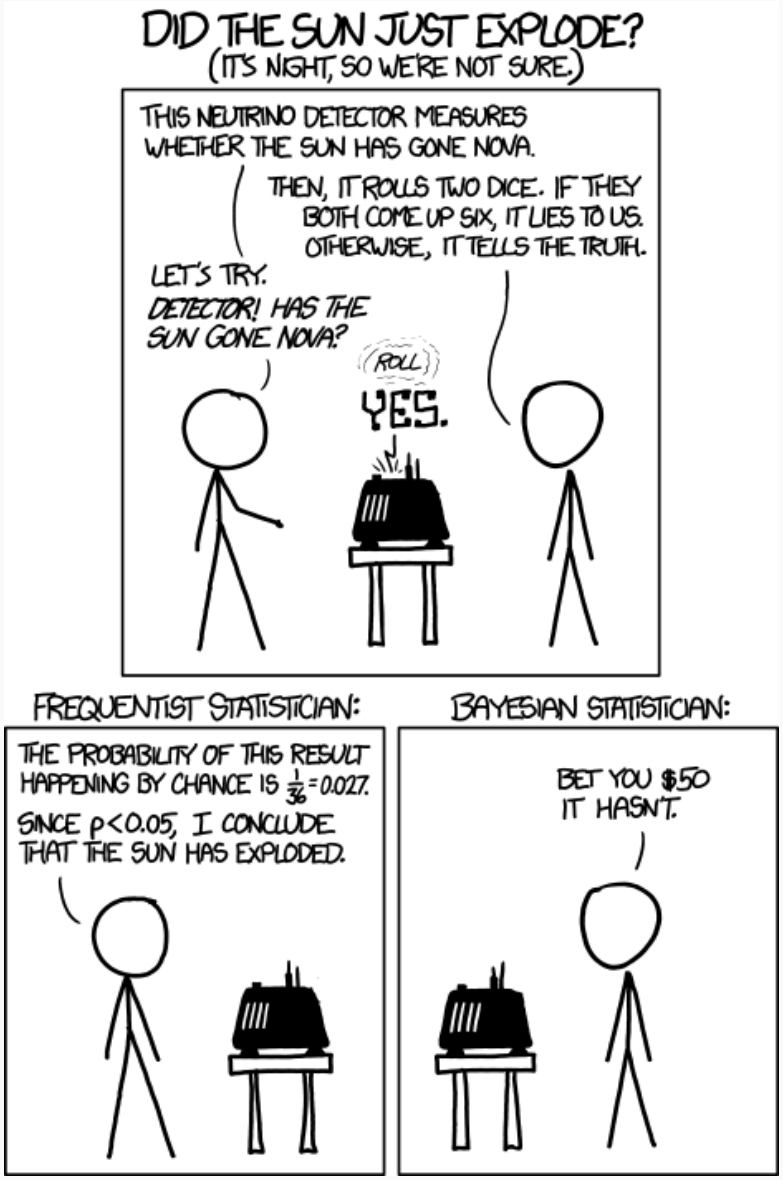
Properties of MLE

The ML estimator, as $N \to \infty$, will converge to the true value, provided that

- the true distribution p_{data} lies within the model family $p_{\text{model}}(\cdot; \boldsymbol{\theta})$. Otherwise no estimator can recover p_{data}.
- the true distribution p_{data} corresponds to exactly one value of $\boldsymbol{\theta}$. Otherwise MLE can recover the correct p_{data} but will not be able to determine, which value of $\boldsymbol{\theta}$ was used by the data-generating process.

When would MLE fail?

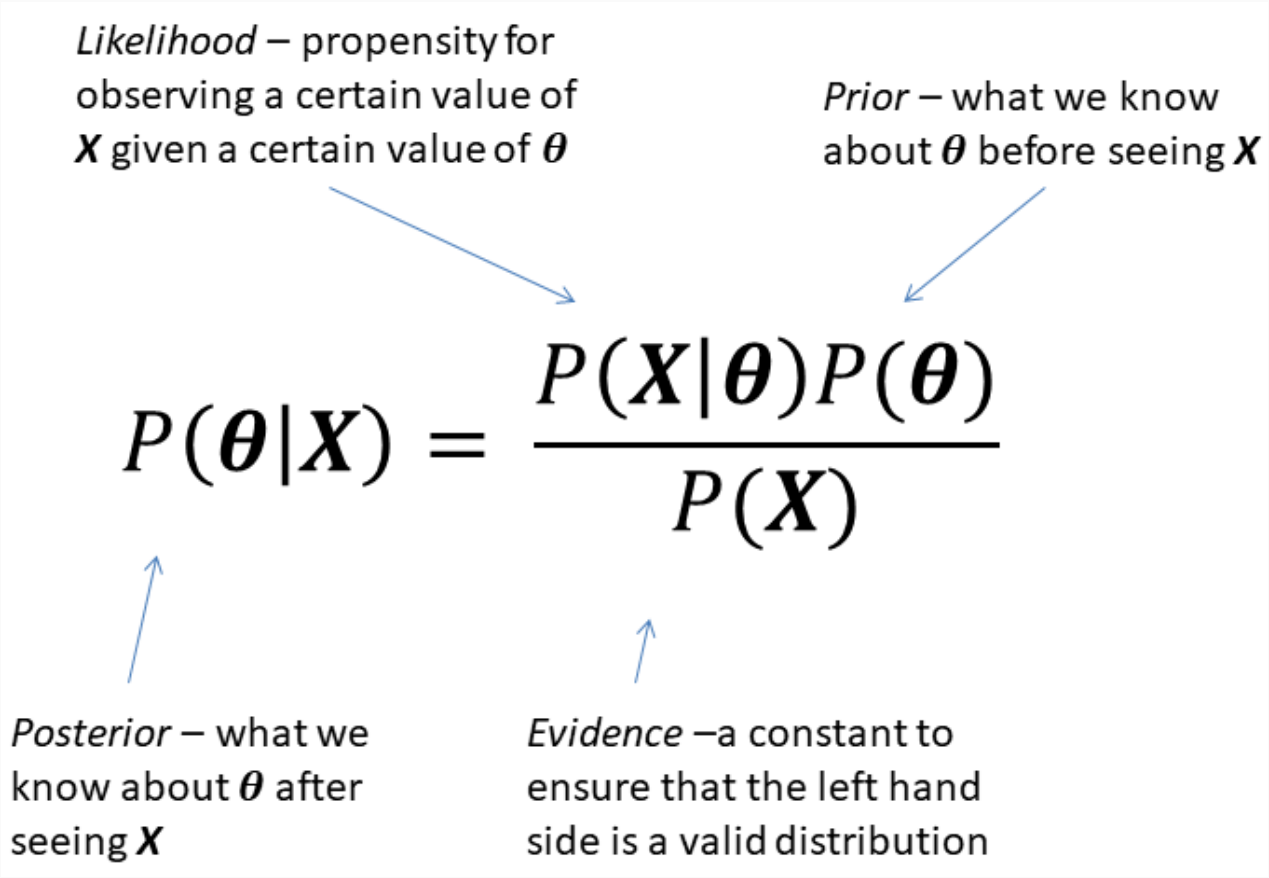
Frequentists vs Bayesians



Which parameters θ make my data most likely?

- Frequentists write this question like this: $P(\mathbf{X}; \boldsymbol{\theta})$
- Bayesians write this question like this: $P(\mathbf{X}|\boldsymbol{\theta})$
- That means that $\boldsymbol{\theta}$ is no longer a parameter, it's a random variable and we can condition on it
- This also means there is a distribution $p(\boldsymbol{\theta})$ and we can draw data from it
- Frequentists strongly object to this: there is no event associated with $\boldsymbol{\theta}$!

Bayes


Likelihood – propensity for observing a certain value of $\boldsymbol{X}$ given a certain value of $\boldsymbol{\theta}$
Prior – what we know about $\boldsymbol{\theta}$ before seeing $\boldsymbol{X}$
$P(\boldsymbol{\theta}|\boldsymbol{X}) = \frac{P(\boldsymbol{X}|\boldsymbol{\theta})P(\boldsymbol{\theta})}{P(\boldsymbol{X})}$
Posterior – what we know about $\boldsymbol{\theta}$ after seeing $\boldsymbol{X}$
Evidence –a constant to ensure that the left hand side is a valid distribution

Maximum a posteriori estimation (MAP)

$$\boldsymbol{\theta}_{\text{MAP}} = \underset{\boldsymbol{\theta}}{\text{argmax}}\, p(\boldsymbol{\theta}|\mathbf{X}) = \underset{\boldsymbol{\theta}}{\text{argmax}}\, \underbrace{\log p(\mathbf{X}|\boldsymbol{\theta})}_{\text{log-likelihood}} + \underbrace{\log p(\boldsymbol{\theta})}_{\text{log prior}}$$

Conditional likelihood (MLE)

Let $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ be a set of N (i.i.d.) training inputs and $\mathbf{y} = \{y_1, \ldots, y_N\}$ be a set of corresponding targets. Then the MLE for $\boldsymbol{\theta}$ can be obtained as

$$\boldsymbol{\theta}_{\text{ML}} = \underset{\boldsymbol{\theta}}{\text{argmax}}\, p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta})$$

- To find the desired parameters $\boldsymbol{\theta}_{\text{ML}}$ that maximize the likelihood, we usually perform gradient ascent (or gradient descent on the negative likelihood).
- In our example of linear regression, a closed form solution exists, making iterative gradient descent unnecessary.

Log-likelihood

In practice, instead of maximizing the likelihood directly, we apply the log-transformation and minimize the negative log-likelihood:

$$\begin{aligned}
\boldsymbol{\theta}_{\text{ML}} &= \underset{\boldsymbol{\theta}}{\text{argmax}}\, p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) \\
&= \underset{\boldsymbol{\theta}}{\text{argmin}} \left(-\log \prod_{i=1}^{N} p(y_i|\mathbf{x}_i, \boldsymbol{\theta}) \right) \\
&= \underset{\boldsymbol{\theta}}{\text{argmin}} \left(-\sum_{i=1}^{N} \log p(y_i|\mathbf{x}_i, \boldsymbol{\theta}) \right)
\end{aligned}$$

Conditional likelihood with prior (MAP)

$$\boldsymbol{\theta}_{\text{MAP}} = \underset{\boldsymbol{\theta}}{\text{argmax}}\, p(\boldsymbol{\theta}|\mathbf{X}, \mathbf{y}).$$

By treating $\boldsymbol{\theta}$ as a random variable and applying Bayes' rule, we get

$$\begin{aligned}\boldsymbol{\theta}_{\text{MAP}} &= \underset{\boldsymbol{\theta}}{\text{argmax}} \underbrace{\log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta})}_{\substack{\text{conditional} \\ \text{log-likelihood}}} + \underbrace{\log p(\boldsymbol{\theta})}_{\text{log prior}} \\ &= \underset{\boldsymbol{\theta}}{\text{argmin}} \left(-\sum_{i=1}^{N} \log p(y_i|\mathbf{x}_i, \boldsymbol{\theta}) - \log p(\boldsymbol{\theta}) \right).\end{aligned}$$