Course – Introduction to ML

Fadel Mamar Seydou

MSc. Computational Science and Engineering

Sampled from EPFL CS-433 Machine learning course of 2020

Table of contents

- History of Machine learning
- (Linear) Regression
- Cost or loss functions
- Optimization
- Maximum likelihood estimator
- Overfitting
- Regularization
- Model selection
- Bias-variance decomposition
- Classification
- logistic regression

Last week

Table of contents

- History of Machine learning
- (Linear) Regression
- Cost or loss functions
- Optimization
- Maximum likelihood estimator
- Overfitting — Today
- Regularization
- logistic regression
- Classification
- Model selection
- Bias-variance decomposition

MLE: Least squares *demo (last week)*

$$
\begin{aligned}
\mathbf{w}_{\text{lse}} &\overset{(a)}{=} \arg\min_{\mathbf{w}} \quad -\log p(\mathbf{y}, \mathbf{X}|\mathbf{w}) \\
&\overset{(b)}{=} \arg\min_{\mathbf{w}} \quad -\log p(\mathbf{X}|\mathbf{w}) p(\mathbf{y}|\mathbf{X}, \mathbf{w}) \\
&\overset{(c)}{=} \arg\min_{\mathbf{w}} \quad -\log p(\mathbf{X}) p(\mathbf{y}|\mathbf{X}, \mathbf{w}) \\
&\overset{(d)}{=} \arg\min_{\mathbf{w}} \quad -\log p(\mathbf{y}|\mathbf{X}, \mathbf{w}) \\
&\overset{(e)}{=} \arg\min_{\mathbf{w}} \quad -\log \left[\prod_{n=1}^{N} p(y_n|\mathbf{w}_n, \mathbf{w}) \right] \\
&\overset{(f)}{=} \arg\min_{\mathbf{w}} \quad -\log \left[\prod_{n=1}^{N} \mathcal{N}(y_n \,|\, \mathbf{x}_n^\top \mathbf{w}, \sigma^2) \right] \\
&= \arg\min_{\mathbf{w}} \quad -\log \left[\prod_{n=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(y_n - \mathbf{x}_n^\top \mathbf{w})^2} \right] \\
&= \arg\min_{\mathbf{w}} \quad -N \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) + \sum_{n=1}^{N} \frac{1}{2\sigma^2}(y_n - \mathbf{x}_n^\top \mathbf{w})^2 \\
&= \arg\min_{\mathbf{w}} \quad \frac{1}{2\sigma^2} \sum_{n=1}^{N} (y_n - \mathbf{x}_n^\top \mathbf{w})^2
\end{aligned}
$$

Exercises

Let’s rewind

Generalization

To watch:

https://developers.google.com/machine-learning/crash-course/overfitting/generalization

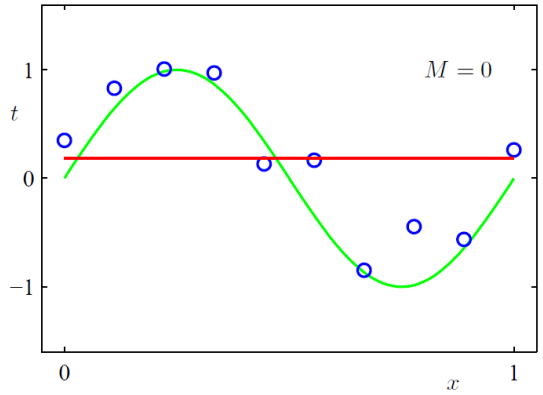
Overfitting and Underfitting

Models can be *too limited* or they can be *too rich*. In the first case we cannot find a function that is a good fit for the date in our model. We then say that we underfit. In the second case we have such a rich model family that we do not just fit the underlying function but we in fact fit the noise in the date as well. We then talk about an overfit. Both of these phenomena are undesirable. This discussion is made more difficult since all we have is data and so we do not know a priori what part is the underlying signal and what part is noise.

Underfitting

Underfitting with Linear Models

It is easy to see that linear models might underfit. Consider a scalar case as shown in the figure below.



Underfitting: remedy for simple cases

Extended/Augmented Feature Vectors From the above example it might seem that linear models are too simple to ever overfit. But in fact, linear models are highly prone to overfitting, much more so than complicated models like neural nets.

Since linear models are inherently not very rich the following is a standard "trick" to make them more powerful.

In order to increase the representational power of linear models we typically "augment" the input. E.g., if the input (feature) is one-dimensional we might add a polynomial basis (of arbitrary degree M),

before: $x_n \in \mathbb{R}$ — after: $\phi(x_n) \in \mathbb{R}^{M+1}$

$$\phi(x_n) := [1, x_n, x_n^2, x_n^3, \ldots, x_n^M]$$

alternative: $\sin(x_n)$, $\log(x_n)$

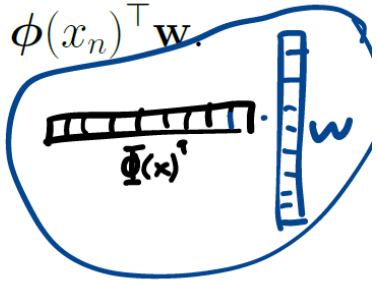
so that we end up with an extended feature vector.

We then fit a linear model to this extended feature vector $\phi(x_n)$:

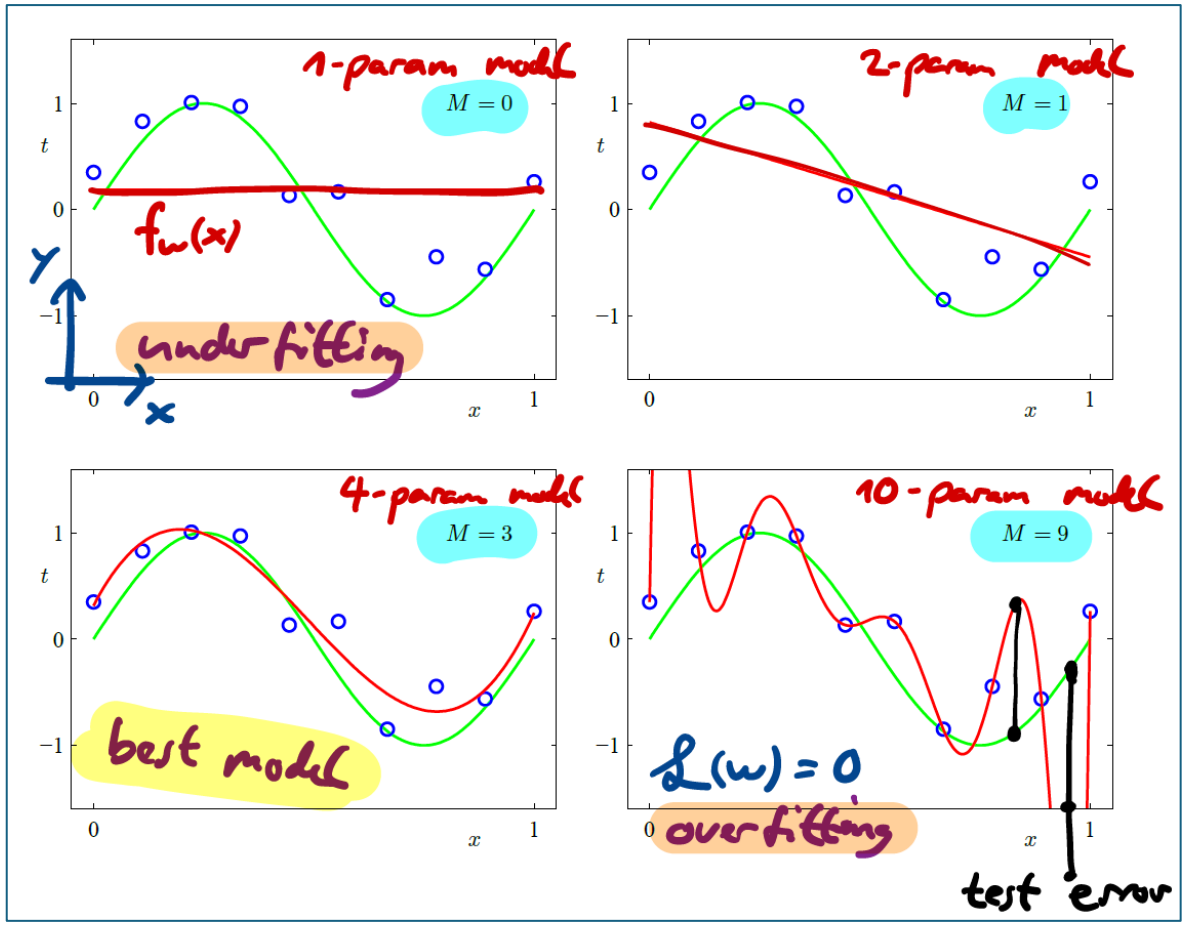
$$y_n \approx w_0 + w_1 x_n + w_2 x_n^2 + \ldots + w_M x_n^M =: \phi(x_n)^\top \mathbf{w}$$

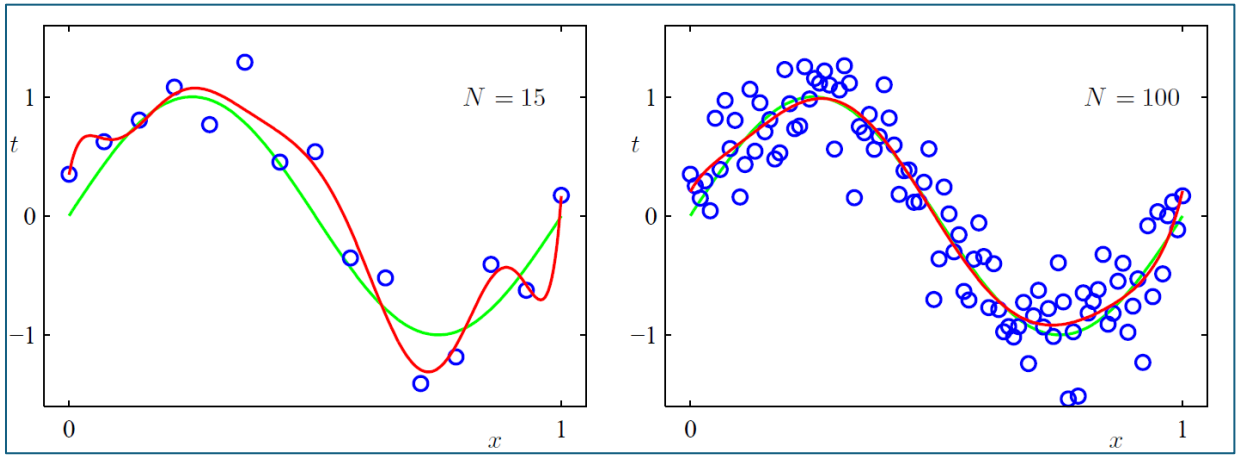
$f_w(\phi(x_n)) = f_w(x_n)$

M: model complexity



Overfitting: linear models





Overfitting reduces when data increases

Regularization

Motivation

We have seen that by augmenting the feature vector we can make linear models as powerful as we want. Unfortunately this leads to the problem of overfitting. *Regularization* is a way to mitigate this undesirable behavior.

We will discuss regularization in the context of linear models, but the same principle applies also to more complex models such as neural nets.

"Simpler models are preferred. Do not make unnecessarily more complex"

Regularization

Regularization

Through regularization, we can penalize complex models and favor simpler ones:

fit

complexity of model w

$$\min_{\mathbf{w}} \mathcal{L}(\mathbf{w}) + \Omega(\mathbf{w})$$

The second term Ω is a regularizer, measuring the complexity of the model given by $\mathbf{w}$.

L2-Regularization

L_2-Regularization: Ridge Regression

$x_n, w \in \mathbb{R}^D$

The most frequently used regularizer is the standard Euclidean norm (L_2-norm), that is

tradeoff-parameter $\lambda > 0$

$$\Omega(\mathbf{w}) = \lambda \|\mathbf{w}\|_2^2$$

where $\|\mathbf{w}\|_2^2 = \sum_i w_i^2$. Here the main effect is that large model weights w_i will be penalized (avoided), since we consider them "unlikely", while small ones are ok. When $\mathcal{L}$ is MSE, this is called ridge regression:

$\lambda \to 0$ no regularization — potential overfitting

$\lambda \to \infty$ underfitting — $\|w\| \to 0$

$\mathcal{L}(w)$ $\qquad$ $\Omega(w)$

$$\min_{\mathbf{w}} \frac{1}{2N} \sum_{n=1}^{N} \left[y_n - \mathbf{x}_n^\top \mathbf{w} \right]^2 + \lambda \|\mathbf{w}\|_2^2$$

$P = \mathcal{L} + \Omega$

L2-Regularization

regression:

$$\min_{\mathbf{w}} \underbrace{\frac{1}{2N}\sum_{n=1}^{N}\left[y_n - \mathbf{x}_n^\top \mathbf{w}\right]^2}_{\mathcal{L}(\mathbf{w})} + \underbrace{\lambda\|\mathbf{w}\|_2^2}_{\Omega(\mathbf{w})}$$

$P = \mathcal{L} + \Omega$

Least squares is a special case of this: set $\lambda := 0$.

Explicit solution for w: Differentiating and setting to zero:

$$\mathbf{w}^\star_{\text{ridge}} = (\mathbf{X}^\top\mathbf{X} + \lambda'\mathbf{I})^{-1}\mathbf{X}^\top\mathbf{y}$$

(here for simpler notation $\frac{\lambda'}{2N} = \lambda$)

First-order optimality

- P convex
- $\nabla P(w) \stackrel{!}{=} 0$

$\Rightarrow w$ optimal

$$\nabla\mathcal{L}(w) = -\frac{1}{N}X^T(y - Xw)$$

$$+ \quad \nabla\Omega(w) = 2\lambda \cdot w$$

$$\stackrel{!}{=} 0$$

$$\Leftrightarrow \quad (X^TX + \lambda'I)w = X^Ty$$

L2-Regularization

Another view of regularization: The ridge regression formulation we have seen above is similar to the following constrained problem (for some $\tau > 0$).

$$\min_{\mathbf{w}} \quad \frac{1}{2N}\sum_{n=1}^{N}(y_n - \mathbf{x}_n^\top \mathbf{w})^2, \qquad \text{such that } \|\mathbf{w}\|_2^2 \leq \tau$$

The following picture illustrates this.

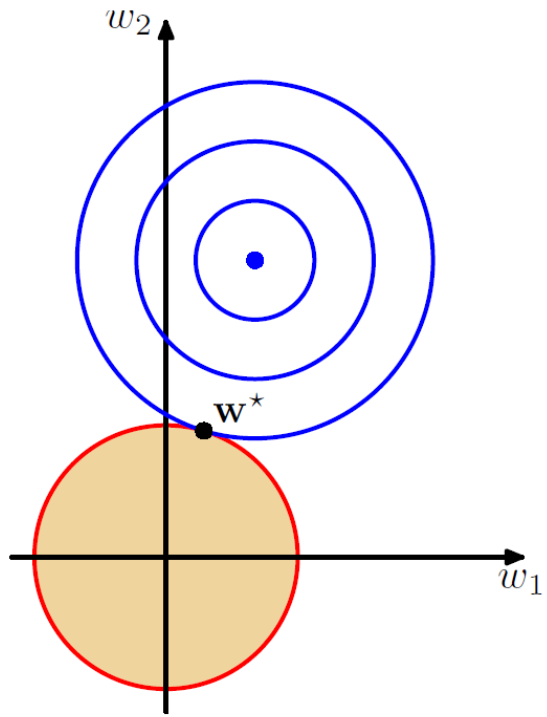


Figure 1: Geometric interpretation of Ridge Regression. Blue lines indicating the level sets of the MSE cost function.

L1-Regularization

L_1-Regularization: The Lasso

As an alternative measure of the complexity of the model, we can use a different norm. A very important case is the L_1-norm, leading to L_1-regularization. In combination with the MSE cost function, this is known as the Lasso:

$$\min_{\mathbf{w}} \quad \frac{1}{2N} \sum_{n=1}^{N} [y_n - \mathbf{x}_n^\top \mathbf{w}]^2 \; + \; \lambda \, \|\mathbf{w}\|_1$$

where

$$\|\mathbf{w}\|_1 := \sum_i |w_i|.$$

Model selection

Motivation

Assume that your friend has trained a model on some data and now claims to have found the "perfect" regression function f. How can you verify this claim and have confidence that f will have good performance? This leads us to the question of *generalization*.

As a second motivation consider the following problem. We have seen in ridge regression that the regularization parameter $\lambda > 0$ can be tuned to reduce overfitting by reducing model complexity.

$$\min_{\mathbf{w}} \quad \frac{1}{2N} \sum_{n=1}^{N} (y_n - \mathbf{x}_n^\top \mathbf{w})^2 \quad + \quad \lambda \|\mathbf{w}\|^2$$

The parameter λ is a *hyper*-parameter.

Model selection: hyperparameters

The parameter λ is a *hyper*-parameter.
In a similar manner, we can enrich the model complexity, by augmenting the feature vector $\mathbf{x}$. E.g., consider a polynomial feature expansion. Here the degree d is a hyperparameter.
To see a final example consider neural nets. Here we have tens or hundreds of hyperparameters: architechture, width, depth, type of the network etc.
In all these cases we are faced with the same problem: how do we choose these hyperparameters? This is the *model selection* problem.

Model selection: definitions

Data Model and Learning Algorithm

In order to give a meaningful answer to the above questions we first need to specify our data model.
We assume that there is an (unknown) underlying distribution $\mathcal{D}$, with range $\mathcal{X} \times \mathcal{Y}$. The data set we see, call it S, consists of independent samples from $\mathcal{D}$:

$$S = \{(\mathbf{x}_n, y_n) \text{ i.i.d. } \sim \mathcal{D}\}_{n=1}^{N}.$$

The *learning algorithm* takes the data and outputs a model within the class of models that it is given. Often this is done by optimising a cost function. We have seen that we can use e.g. (stochastic) gradient descent or least-squares for the ridge-regression model as an efficient way of implementing this learning. Write $f_S = \mathcal{A}(S)$, where $\mathcal{A}$ denotes the learning algorithm.
If we want to indicate that f_S also depends on parameters of the model, e.g., the λ in the ridge regression model, we can add a subscript to write $f_{S,\lambda}$.

Model selection:
How to assess a good model?

Given a model f, how can we assess if f is any good? We

Model selection:
Let's use the 'True Error'?

Given a model f, how can we assess if f is any good? We should compute the *expected* error over all samples chosen according to $\mathcal{D}$, i.e., we should compute

$$L_{\mathcal{D}}(f) = \mathbb{E}_{\mathcal{D}}[\ell(y, f(\mathbf{x}))],$$

where $\ell(\cdot, \cdot)$ is our loss function. E.g., for ridge regression

$$\ell(y, f(\mathbf{x})) = \tfrac{1}{2}(y - f(\mathbf{x}))^2,$$

Model selection: Empirical error, Training error

Given a model f, how can we assess if f is any good? We should compute the *expected* error over all samples chosen according to $\mathcal{D}$, i.e., we should compute

$$L_{\mathcal{D}}(f) = \mathbb{E}_{\mathcal{D}}[\ell(y, f(\mathbf{x}))],$$

where $\ell(\cdot, \cdot)$ is our loss function. E.g., for ridge regression

$$\ell(y, f(\mathbf{x})) = \tfrac{1}{2}(y - f(\mathbf{x}))^2,$$

The quantity $L_{\mathcal{D}}(f)$ has many names: *(true/expected) (risk/loss/* This is the quantity we are fundamentally interested in, but we cannot compute it since $\mathcal{D}$ is not known.
But we are given some data S. It is therefore natural to compute the equivalent *empirical* quantity

$$L_S(f) = \frac{1}{|S|} \sum_{(\mathbf{x}_n, y_n) \in S} \ell(y_n, f(\mathbf{x}_n)). \tag{1}$$

This is called the *(empirical) (risk/loss/error).*
There is one added complication. Assume that we are given the data S. If we first learn the model from S, i.e., we compute $f_S = \mathcal{A}(S)$ and then we compute the empirical risk of f_S using the *same data* S then in fact we are computing

$$L_S(f_S) = \frac{1}{|S|} \sum_{(\mathbf{x}_n, y_n) \in S} \ell(y_n, f_S(\mathbf{x}_n)).$$

This is called the *training (risk/loss/error)* and we have already discussed in previous lectures that this training error might not be representative of the error we see on "fresh" samples.
The reason that $L_S(f_S)$ might not be close to $L_{\mathcal{D}}(f_S)$ is of course overfitting.

Model selection: Test error, Training error

Before we go on and explore the relationship between the true risk and the empirical risk, let us first see how we can adress the potential overfitting problem that ocurs if we train and test the model on the same data.

Problem: Validating model on the same data subset we trained it on!

Fix: Split the data into a *training* and a *test* set (a.k.a. *validation* set), call them S_{train} and S_{test}, respectively.

We apply the learning algorithm $\mathcal{A}$ to the training set S_{train} and compute the function $f_{S_{\text{train}}}$. We then compute the error on the test set, i.e.,

$$L_{S_{\text{test}}}(f_{S_{\text{train}}}) = \frac{1}{|S_{\text{test}}|} \sum_{(y_n, \mathbf{x}_n) \in S_{\text{test}}} \ell(y_n, f_{S_{\text{train}}}(\mathbf{x}_n)).$$

This is called the *(test/validation) (risk/loss/error).*

Model selection: generalization error

$$L_{S_{\text{test}}}(f) = \frac{1}{|S_{\text{test}}|} \sum_{(\mathbf{x}_n, y_n) \in S_{\text{test}}} \ell(y_n, f(\mathbf{x}_n)).$$

The true error is

$$L_{\mathcal{D}}(f) = \mathbb{E}_{(y,\mathbf{x})\sim\mathcal{D}}[\ell(y, f(\mathbf{x}))].$$

How far are these apart? This is called the *generalization error* and it is given by

$$|L_{\mathcal{D}}(f) - L_{S_{\text{test}}}(f)|.$$

First note that in expectation they are the same, i.e.,

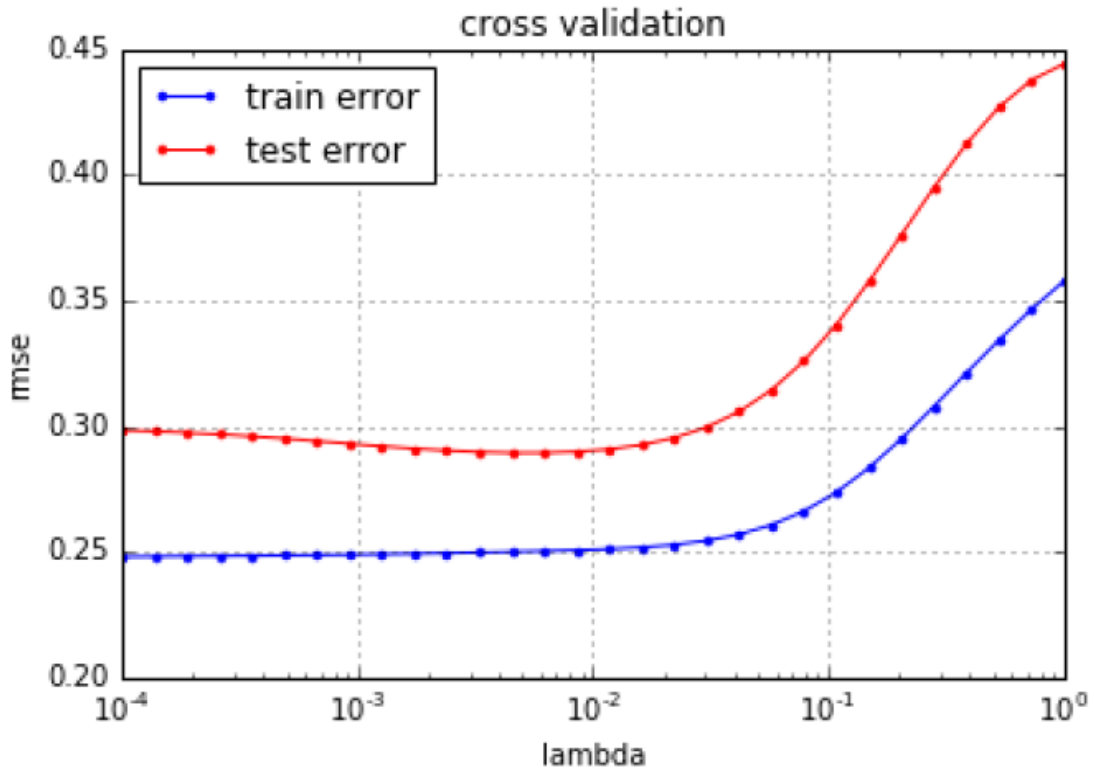
$$L_{\mathcal{D}}(f) = \mathbb{E}_{S_{\text{test}}\sim\mathcal{D}}[L_{S_{\text{test}}}(f)], \tag{2}$$

where the expectation is over the samples of the test set. But we need to worry about the variation. We claim that

$$\mathbb{P}\Big[\, |L_{\mathcal{D}}(f) - L_{S_{\text{test}}}(f)| \geq \sqrt{\frac{(b-a)^2 \ln(2/\delta)}{2|S_{\text{test}}|}} \,\Big] \leq \delta. \tag{3}$$

Insights: The error decreases as $\mathcal{O}(1/\sqrt{|S_{\text{test}}|})$ with the number test points. The more data points we have therefore, the more confident we can be that the empirical loss we measure is close to the true loss. If we want δ to be smaller we only need to increase the size of the test set slightly.

Model selection: hyperparameter tuning in practice



- Select the hyperparameter with the lowest “validation” error
- The idea is that probabilistically, choosing the model with lowest test error brings us closer to the optimal model

Model selection: hyperparameter tuning in practice

- Some techniques for hyperparameter tuning
  - Grid search
  - Random search
  - Bayesian optimization
  - Read more
  - In practice we optimize in function of a performance metric (e.g., accuracy, F1-score etc.) which can be correlated to the loss. But It is also a practice of selecting hyperparameters that decrease the loss the fastest

Model selection: detecting overfitting

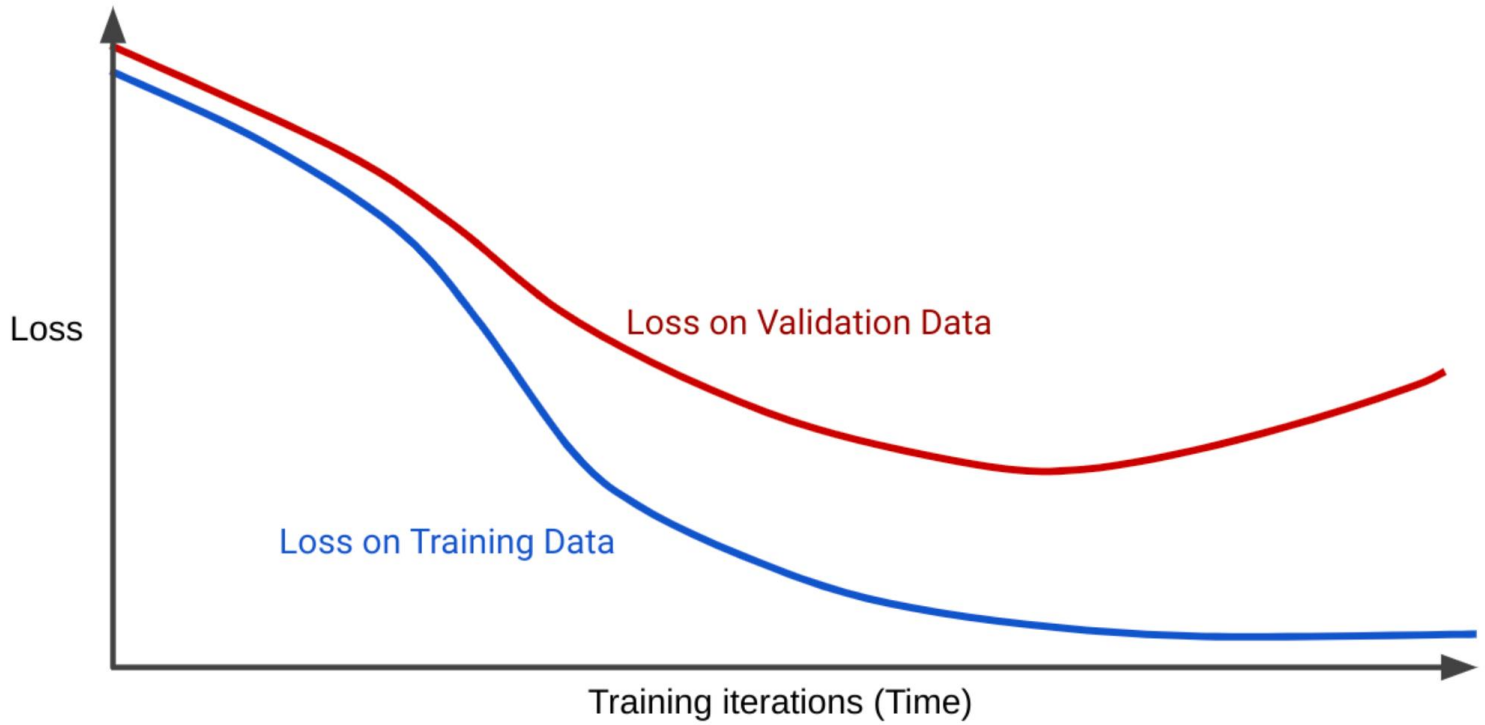


Figure 15. *A generalization curve that strongly implies overfitting.*

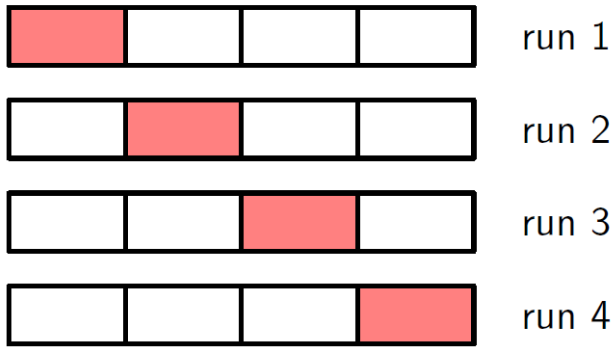
source

Model selection: cross-validation

Splitting the data once into two parts (one for training and one for testing) is not the most efficient way to use the data. Cross-validation is a better way:

K-fold cross-validation is a popular variant. Randomly partition the data into K groups. Now train K times. Each time leave out exactly one of the K groups for testing and use the remaining $K-1$ groups for training. Average the K results.

Note: Have used all data for training, and all data for testing, and used each data point the same number of times.



Cross-validation returns an unbiased estimate of the *generalization error* and its variance.

Bias-variance

Today we will focus on how the risk (true or emperical) behaves as a function of the complexity of the model class. This will lead to the important concept of the bias - variance trade-off when we perform the model selection. It will help us to decide how "complex" or "rich" we should make our model.

Let us discuss a very simple example. Consider linear regression with a one-dimensional input and using polynomial feature expansion. The maximum degree d regulates the complexity of the class. We will see that the following is typically true.

Bias-variance: definition

Video:

https://www.youtube.com/watch?v=EuBBz3bI-aA

Bias-variance

Simpler model

Assume that we only allow simple models, i.e., we restrain the degree to be small:

- We then typically will get a large bias, i.e., a bad fit.
- On the other hand the variance of $L_{\mathcal{D}}(f_S)$ as a function of the random sample S is typically small.

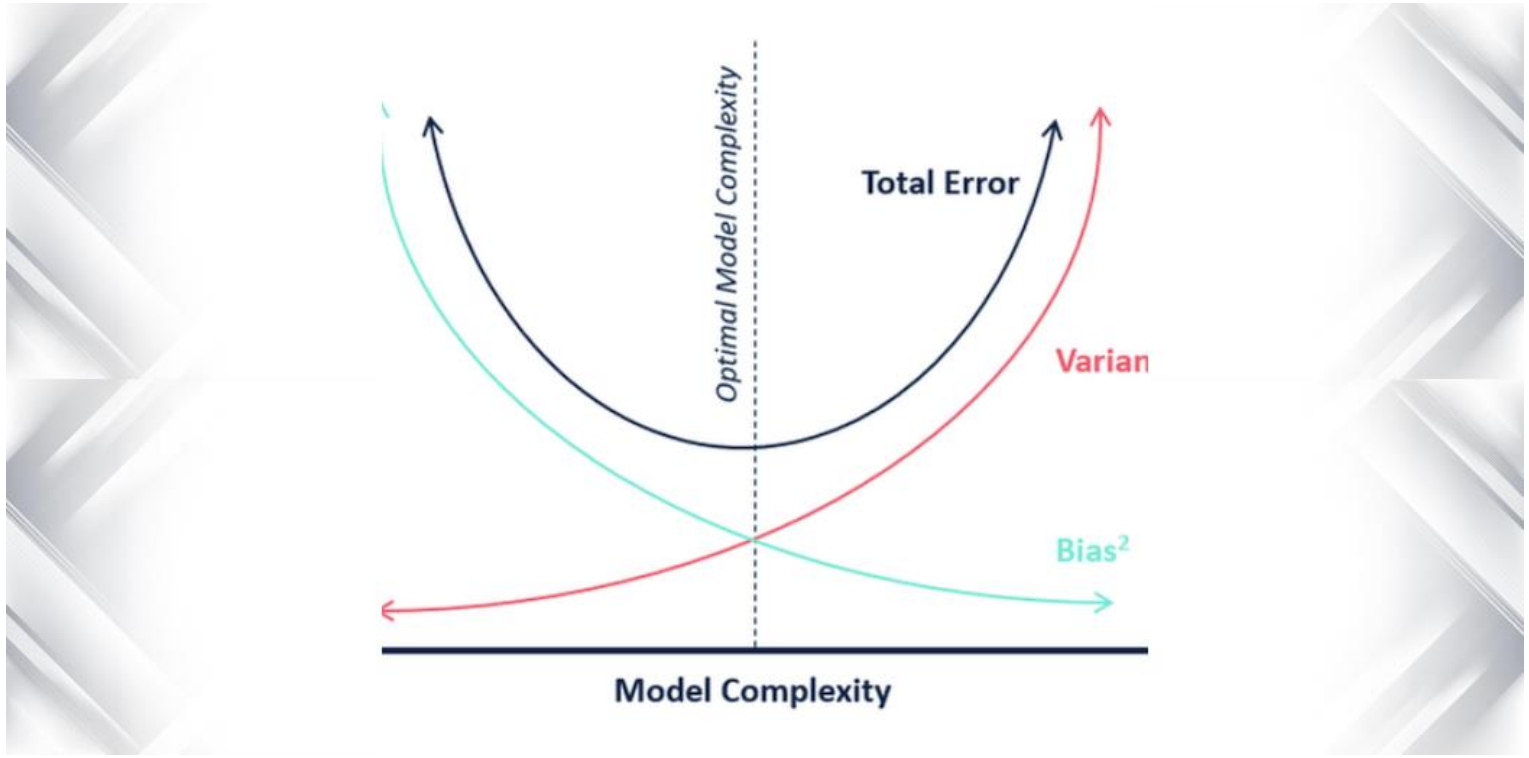
More complex model

Assume that we allow complex models, i.e., we allow large degrees:

- We then typically will find a model that fits the data very well. We will say that we have small bias.
- But we likely observe that the variance of $L_{\mathcal{D}}(f_S)$ as a function of the random sample S is large.

We say that we have low bias but high variance.

Bias-variance



Analytics vidhya

Bias-variance de composition

We are ultimately interested in how the true error

$$\mathbb{E}_{\mathbf{x}\sim\mathcal{D}}[(f(\mathbf{x})+\varepsilon-f_{S_{\text{train}}}(\mathbf{x}))^2]$$

behaves as a function of the training set S_{train} and the complexity of the model class.
But the decomposition we will discuss already applies "pointwise", i.e., for a single sample $\mathbf{x}$. It is therefore simpler if we fix $\mathbf{x}_0$, and only consider

$$(f(\mathbf{x}_0)+\varepsilon-f_{S_{\text{train}}}(\mathbf{x}_0))^2.$$

We imagine that we are running the experiment many times: we create S_{train}, we learn the model $f_{S_{\text{train}}}$, and then we evaluate the performance by computing the square loss for this fixed element $\mathbf{x}_0$.
So let us look at the expected value of this quantity:

$$\mathbb{E}_{S_{\text{train}}\sim\mathcal{D},\,\varepsilon\sim\mathcal{D}_\varepsilon}[(f(\mathbf{x}_0)+\varepsilon-f_{S_{\text{train}}}(\mathbf{x}_0))^2].$$

We will now show that we can rewrite the above quantity as a sum of *three non-negative terms* and this decomposition

$$\begin{aligned}
&\mathbb{E}_{S_{\text{train}}\sim\mathcal{D},\,\varepsilon\sim\mathcal{D}_\varepsilon}[(f(\mathbf{x}_0)+\varepsilon-f_{S_{\text{train}}}(\mathbf{x}_0))^2]\\
&\overset{(a)}{=}\mathbb{E}_{\varepsilon\sim\mathcal{D}_\varepsilon}[\varepsilon^2]+\mathbb{E}_{S_{\text{train}}\sim\mathcal{D}}[(f(\mathbf{x}_0)-f_{S_{\text{train}}}(\mathbf{x}_0))^2]\\
&\overset{(b)}{=}\mathrm{Var}_{\varepsilon\sim\mathcal{D}_\varepsilon}[\varepsilon]+\mathbb{E}_{S_{\text{train}}\sim\mathcal{D}}[(f(\mathbf{x}_0)-f_{S_{\text{train}}}(\mathbf{x}_0))^2]\\
&\overset{(c)}{=}\underbrace{\mathrm{Var}_{\varepsilon\sim\mathcal{D}_\varepsilon}[\varepsilon]}_{\text{noise variance}}\\
&\quad+\underbrace{(f(\mathbf{x}_0)-\mathbb{E}_{S'_{\text{train}}\sim\mathcal{D}}[f_{S'_{\text{train}}}(\mathbf{x}_0)])^2}_{\text{bias}}\\
&\quad+\mathbb{E}_{S_{\text{train}}\sim\mathcal{D}}\Big[\underbrace{(\mathbb{E}_{S'_{\text{train}}\sim\mathcal{D}}[f_{S'_{\text{train}}}(\mathbf{x}_0)]-f_{S_{\text{train}}}(\mathbf{x}_0))^2}_{\text{variance}}\Big].
\end{aligned}$$

Note that here S'_{train} is a second training set, also sampled from $\mathcal{D}$ that is independent of the training set S_{train}.

Bias-variance

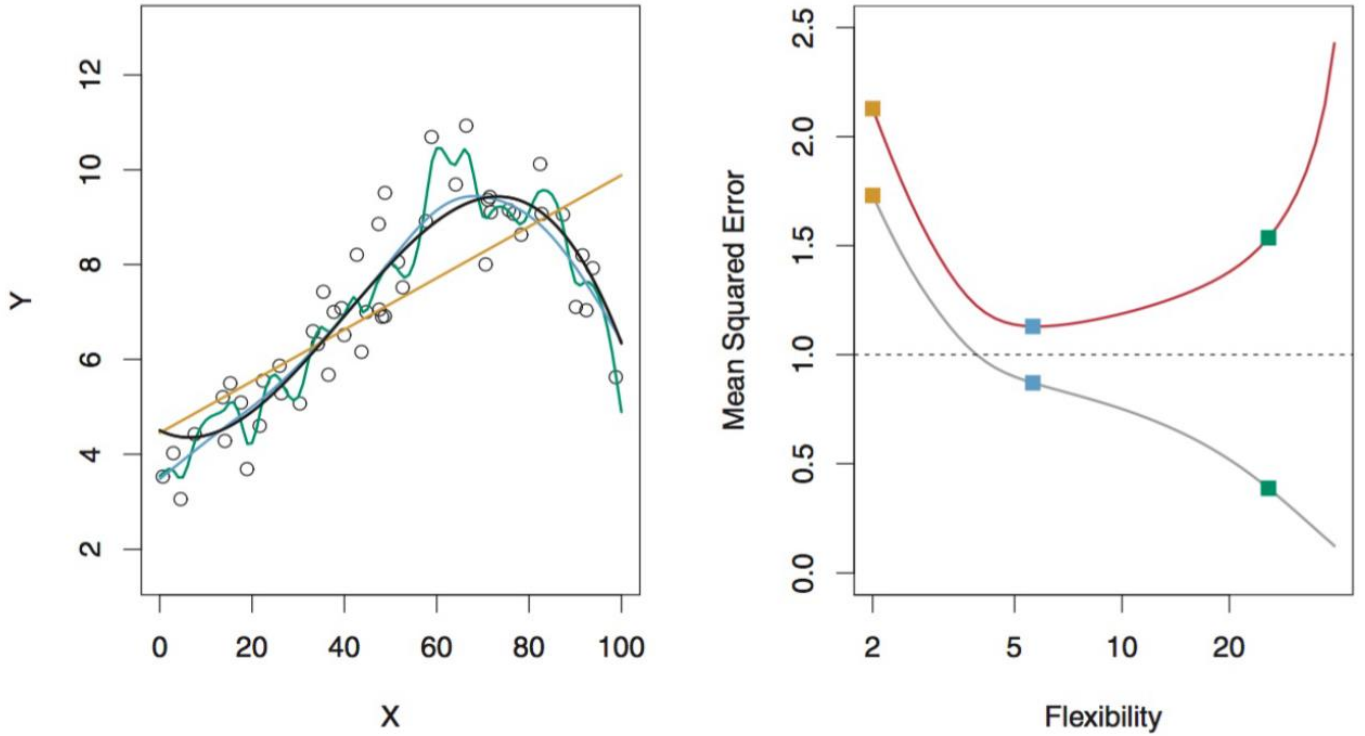


FIGURE 2.9. Left: *Data simulated from f, shown in black. Three estimates of f are shown: the linear regression line (orange curve), and two smoothing spline fits (blue and green curves).* Right: *Training MSE (grey curve), test MSE (red curve), and minimum possible test MSE over all methods (dashed line). Squares represent the training and test MSEs for the three fits shown in the left-hand panel.*

Classification

Classification

Similar to regression, classification relates the input variable $\mathbf{x}$ to the output variable y, but now y can only take on discrete values. We say that y is a *categorical* variable.

Binary classification

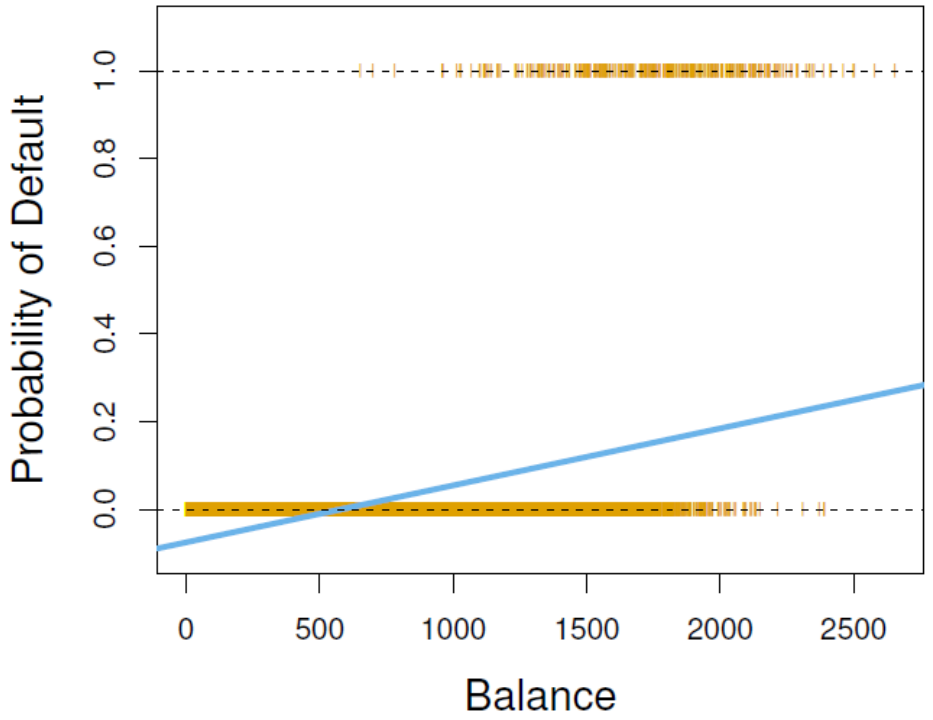
When y can only take on two values, it is called binary classification. Sometimes we refer to the two discrete values abstractly as $y \in \{\mathcal{C}_1, \mathcal{C}_2\}$. The $\mathcal{C}_i$ are called class labels or simply classes. Other times it is more conveniently to assume that $y \in \{-1, +1\}$ or $y \in \{0, 1\}$. Note that even if the class labels are real values, there is typically no ordering implied between the two classes.

Multi-class classification

In a multi-class classification, y can take on more than two values, i.e., $y \in \{\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_{K-1}\}$ for a K-class problem. Again, even though there is in general no ordering among these classes, we sometimes use the labels $y \in \{0, 1, 2, \ldots, K-1\}$.

Classification

In the figure below this approach is applied to the credit-card default problem. To keep things simple we use as feature only the *balance* (as we have seen in a previous plot the second feature (the *income*) does not contain much information).
We think of $y = 0$ as "no default" and $y = 1$ as "default". The dots we see corresponds to the various data points and all dots are either on the line $y = 0$ or $y = 1$. The horizontal axis corresponds to the input $\mathbf{x}$, the *balance*.
In the figure the output y is labeled as *probability*. This is just a convenient way of interpretation. Since the desired label y is either 0 or 1 we can think of y as the probability of a default.
So let us now run a regression on the training data S_{train}. To keep things simple we run a linear regression and learn the linear function $f_{S_{\text{train}}}(\mathbf{x})$. The result is the blue curve that is indicated.

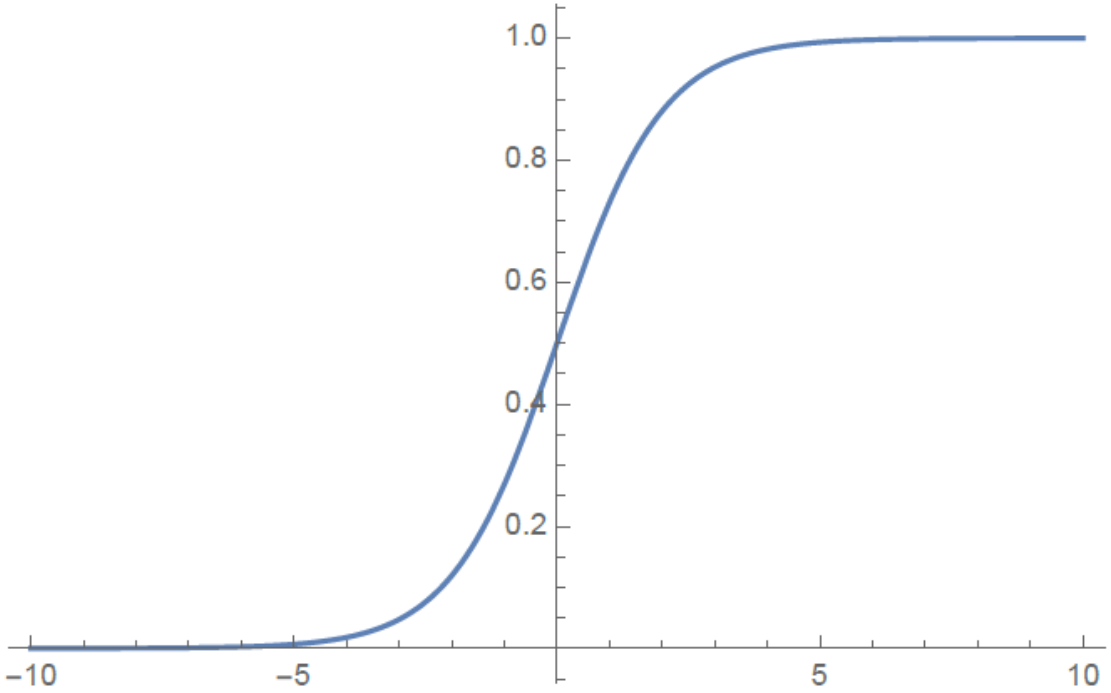


Logistic regression

We have also discussed that it is tempting to interpret the predicted value as probability.
But there are problems: (i) the predicted values are in general not in $[0, 1]$; further, (ii) very large ($y \gg 1$) or very small ($y \ll 0$) values of the prediction will contribute to the error if we use the squared loss, even though they indicate that we are very confident in the resulting classification.
It is therefore natural that we *transform* the predictions that take values in $(-\infty, \infty)$ into a true probability by applying an appropriate function. There are several possible such functions. The *logistic function*

$$\sigma(z) := \frac{e^z}{1 + e^z}$$

Logistic regression



Logistic regression

Consider the binary classification case and assume that our two class labels are $\{0, 1\}$. We proceed as follows. Given a training set S_{train} we learn a weight vector $\mathbf{w}$ (we will discuss how to do this shortly) and a "shift" (scalar) w_0. Given a "new" feature vector $\mathbf{x}$, we predict the (posterior) *probability* of the two class labels given $\mathbf{x}$ by means of

$$p(1 \mid \mathbf{x}, \mathbf{w}) = \sigma(\mathbf{x}^\top \mathbf{w} + w_0),$$
$$p(0 \mid \mathbf{x}, \mathbf{w}) = 1 - \sigma(\mathbf{x}^\top \mathbf{w} + w_0).$$

Note that we predict a real value (a probability) and not a label. This is the reason it is called logistic *regression*. But typically we use logistic regression as the first step of a classifier. In the second step we quantize the value to a binary value, typically according to whether the predicted prob-

Logistic regression

Likelihood

$$
\begin{aligned}
p(\mathbf{y} \mid \mathbf{X}, \mathbf{w}) &= \prod_{n=1}^{N} p(y_n|\mathbf{x}_n) \\
&= \prod_{n:y_n=1} p(y_n = 1|\mathbf{x}_n) \prod_{n:y_n=0} p(y_n = 0|\mathbf{x}_n) \\
&= \prod_{n=1}^{N} \sigma(\mathbf{x}_n^\top \mathbf{w})^{y_n} [1 - \sigma(\mathbf{x}_n^\top \mathbf{w})]^{1-y_n}.
\end{aligned}
$$

Cross-entropy loss

$$
\begin{aligned}
\mathcal{L}(\mathbf{w}) &= -\sum_{n=1}^{N} y_n \ln \sigma(\mathbf{x}_n^\top \mathbf{w}) + (1 - y_n) \ln[1 - \sigma(\mathbf{x}_n^\top \mathbf{w})] \\
&= \sum_{n=1}^{N} \ln[1 + \exp(\mathbf{x}_n^\top \mathbf{w})] - y_n \mathbf{x}_n^\top \mathbf{w}.
\end{aligned}
$$

Logistic regression: optimality

$$\frac{\partial \ln[1+\exp(x)]}{\partial x} = \sigma(x).$$

Therefore

$$\begin{aligned}\nabla \mathcal{L}(\mathbf{w}) &= \sum_{n=1}^{N} \mathbf{x}_n(\sigma(\mathbf{x}_n^\top \mathbf{w}) - y_n) \\ &= \mathbf{X}^\top[\sigma(\mathbf{X}\mathbf{w}) - \mathbf{y}].\end{aligned}$$

Lemma. *The cost function*

$$\mathcal{L}(\mathbf{w}) = \sum_{n=1}^{N} \ln[1+\exp(\mathbf{x}_n^\top \mathbf{w})] - y_n \mathbf{x}_n^\top \mathbf{w}$$

is convex in the weight vector $\mathbf{w}$.

$$\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} - \gamma^{(t)} \nabla \mathcal{L}(\mathbf{w}^{(t)}),$$

where $\gamma^{(t)} > 0$ is the step size and $\mathbf{w}^{(t)}$ is the sequence of weight vectors.

Logistic regression: regularization

Although the cost-function for logistic regression is lower bounded by 0 we get issues if the data is linearly separable. In this case there is no finite-weight vector $\mathbf{w}$ which gives us this minimum cost function and if we continue to run the optimization the weights will tend to infinity.

To avoid this problem, as for standard regression problems, we can add a penalty term. E.g., we consider the cost function

$$\text{argmin}_{\mathbf{w}} - \sum_{n=1}^{N} \ln p(y_n \,|\, \mathbf{x}_n^\top \mathbf{w}) + \frac{\lambda}{2}\|\mathbf{w}\|^2.$$

Exercises