IntroML - Complete Summary

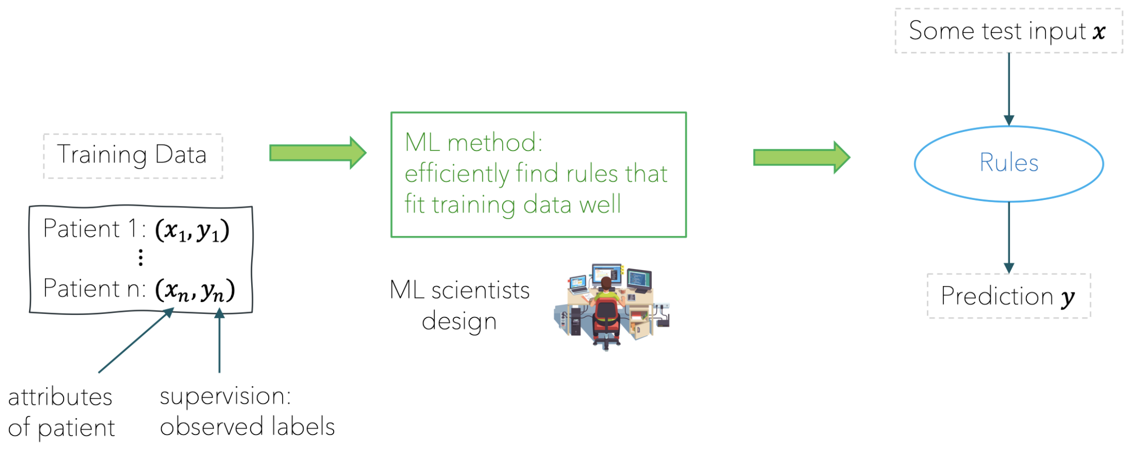
Ruben Schenk, ruben.schenk@inf.ethz.ch

March 9, 2022

1 Examples of Machine Learning Problems

The **Machine Learning (ML) approach** for supervised learning is that machines should learn rules using some example data.

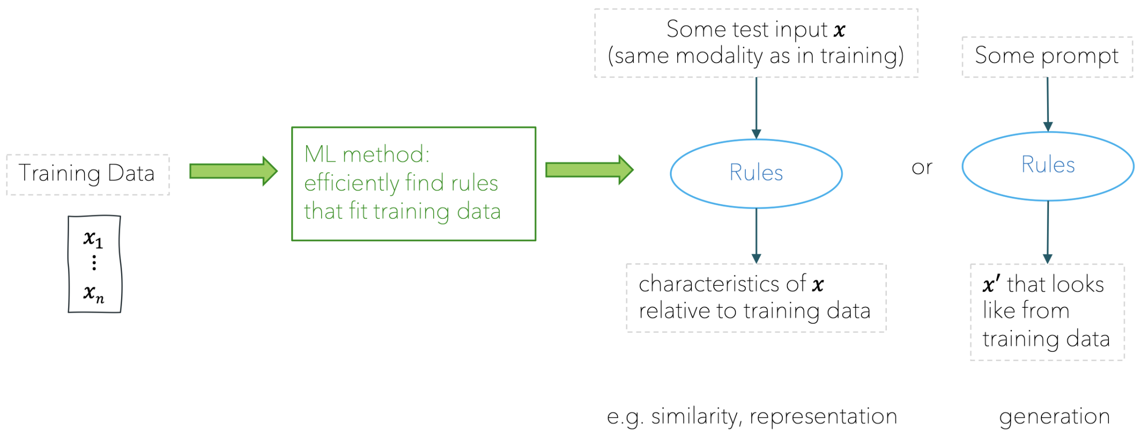
1.1 Supervised Learning


Some test input $\boldsymbol{x}$
Training Data
ML method: efficiently find rules that fit training data well
Rules
Patient 1: (x_1, y_1)
⋮
Patient n: (x_n, y_n)
ML scientists design
Prediction $\boldsymbol{y}$
attributes of patient
supervision: observed labels

The different supervised learning tasks are as follows:

- *Classification:* Predict the class (discrete scalar) of an input.
- *Regression:* Predict a value (continuous scalar) for an input.
- *Structured Prediction:* Predict an output beyond scalars.

1.2 Unsupervised Learning


Some test input $\boldsymbol{x}$ (same modality as in training)
Some prompt
Training Data
ML method: efficiently find rules that fit training data
Rules
or
Rules
x_1
⋮
x_n
characteristics of $\boldsymbol{x}$ relative to training data
$\boldsymbol{x'}$ that looks like from training data
e.g. similarity, representation
generation

Some common goals using unsupervised learning on the characteristics of some data are:

- Anomaly detection of "unusual" data points
- Identification of (relevant) unobserved variables (such as features and classes)
- Compact representation and compression of data sets
- Generation of new data

Some ML methods to learn characteristics of the data (that are covered in the class) include:

- *Clustering,* e.g. for learning similarity and anomaly of points
- *Dimensionality reduction,* e.g. for learning compact representations
- *Generative modelling,* e.g. for feature learning and data generation

1.3 Machine Learning Pipeline

We will explore the **machine learning pipeline** by considering an example where we want to list a house for sale but do not know which price is right. We therefore try to find the average market price for houses of our kind. The ML pipeline consists of the following steps:

Step 0 Find representation for the house

We have to determine how to represent houses in digital fashion, for example using a *vector of attributes.*

Step 1 Collect training data

After we found a digital model of our house, we need to collect the *training data,* i.e. attributes and sales prices from other houses and our own house.

Step 2 Learn

With the attributes of both training data and our own house, we need to efficiently find a function $\hat{f} \in F$ that fits our training data.

Step 3 Predict

Finally, with our model $\hat{f}$ and the attributes of our house, we can predict the average price for our house.

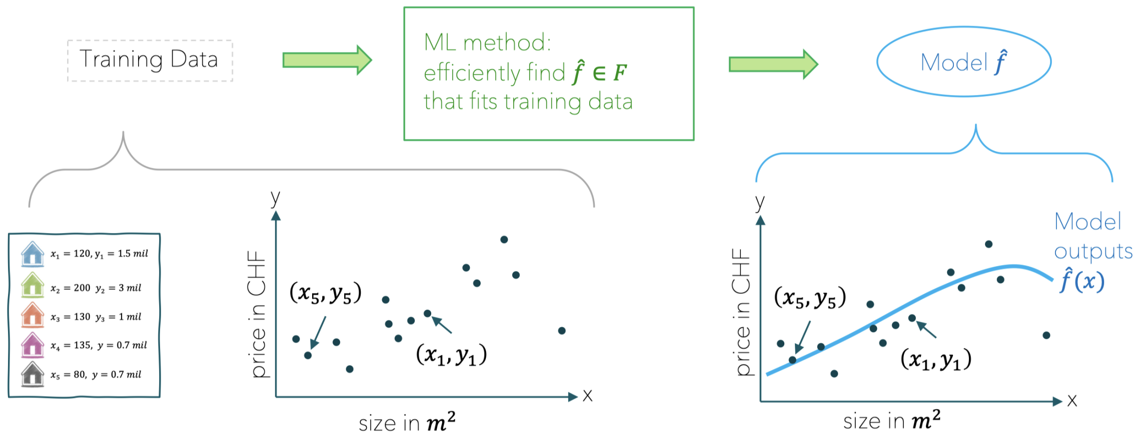
2 Linear Regression

2.1 Simple Linear Regression in 1D

2.1.1 Introduction

Coming back to our previous example of selling a house, consider the case where the only attributes are x, the size of the house in m^2, and y, the sales price in CHF.

After applying that data to out ML pipeline, we might end up with a model $\hat{f}$ as follows:


Training Data
ML method: efficiently find $\hat{f} \in F$ that fits training data
Model $\hat{f}$
$x_1 = 120, y_1 = 1.5$ mil
$x_2 = 200$ $y_2 = 3$ mil
$x_3 = 130$ $y_3 = 1$ mil
$x_4 = 135,$ $y = 0.7$ mil
$x_5 = 80,$ $y = 0.7$ mil
y
price in CHF
(x_5, y_5)
(x_1, y_1)
x
size in m^2
Model outputs $\hat{f}(x)$

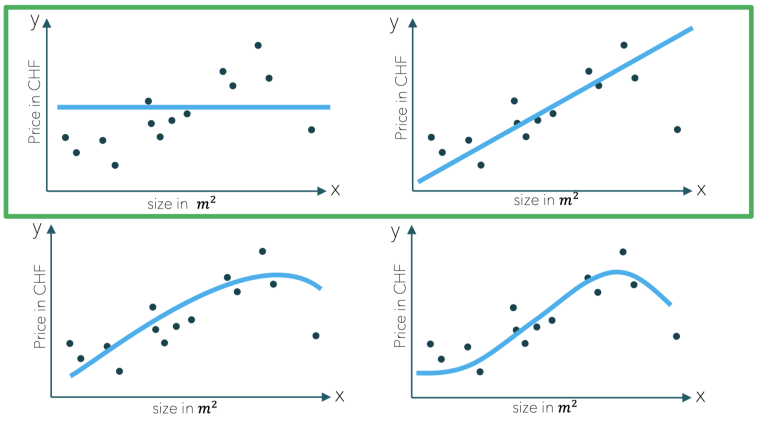
Finally, with our predict model $\hat{f}$ and the attributes of our own house, we can predict its average market price.

The main question to ask here is: How do we find the model $\hat{f}$? For this to answer we need to define the following three things:

- Function class F
- Training loss
- Optimization

2.1.2 Function Classes F

We might ask ourselves what kind of functions can we fit in 1D data? Obviously, we can fit constant and linear functions, but more generally speaking we are often looking for polynomials or non-linear functions. For this chapter, we focus on linear functions.

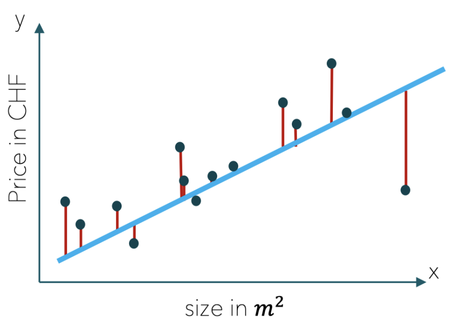


2.1.3 Training Losses

How do we define a *good fit* of the training data? The usual way is if $f(x)$ is close to y for most points, that is, if it has a **low training loss:**

$$L(f) = \frac{1}{n}\sum_{i=1}^{n} l(f(x_i), y_i),$$

where l is the **pointwise loss function,** representing the "closeness" between $f(x)$ and y for a point (x, y). For our running example, we may choose $(f(x) - y)^2$ as the loss-function (**squared loss**).



2.1.4 Minimizing Squared Loss

Remember: Our final model is the function $\hat{f}$ that minimizes the training loss (squared loss on average):

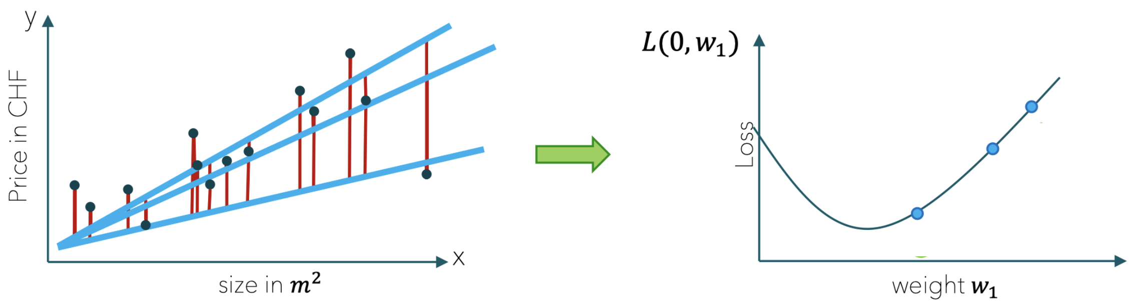
$$\hat{f} = \text{argmin}_{f \in F_{Lin}} L(f) = \text{argmin}_{f \in F_{Lin}} \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2$$

We call $\hat{f}$ the **solution of the ML method linear regression.**

If we recall that all linear functions ($f(x) = w_0 + w_1 x$) are parameterized by two scalars w_0, w_1, searching for a minimum in F_{Lin} is the same as searching for scalars w_0, w_1 that minimize:

$$\hat{w} := (\hat{w}_0, \hat{w}_1) = \text{argmin}_{w_0, w_1 \in \mathbb{R}} L(w_0, w_1) = \text{argmin}_{w_0, w_1 \in \mathbb{R}} \frac{1}{n} \sum_{i=1}^{n} (y_i - w_0 - w_1 x_i)^2$$

For simplification, we first fix $w_0 = 0$, and only optimize $L(0, w_1)$ over $w_1 \in \mathbb{R}$.



Since $L(0, w_1) = \frac{1}{n} \sum_{i=1}^{n} (y_i - w_1 x_i)^2$ is a 1D-quadratic, there's only one minimum where $L'(0, \hat{w}_1) = 0$.

Now we get back to our original problem. More generally, let w_0 again be a variable, i.e. we want to find $\hat{w} = (\hat{w}_0, \hat{w}_1) = \text{argmin}_{w_0, w_1 \in \mathbb{R}} L(w_0, w_1)$. Consider the following theorem:

Theorem 2.3 (Necessary Condition for Local Optimality): Let $\Omega \subseteq \mathbb{R}^n$ be open and $f : \Omega \to \mathbb{R}$ a continuously differentiable function. If $x_0 \in \Omega$ is a local minimizer, then

$$\nabla f(x_0) = 0.$$

Therefore, when searching for local/global minima, we only have to search within the set of *critical (or stationary) points* defined as $\{x \in \Omega \,|\, \nabla f(x) = 0\}$.

Following this we can conclude, that a global minimum $\hat{w}$ must satisfy $\nabla_w L(\hat{w}) = 0$. Hence, the minimum $\hat{w} = (\hat{w}_0, \hat{w}_1)$ satisfies:

$$\nabla_w L(\hat{w}_0, \hat{w}_1) = \begin{pmatrix} -\frac{2}{n} \sum_{i=1}^{n} (y_i - \hat{w}_0 - \hat{w}_1 x_i) \\ -\frac{2}{n} \sum_{i=1}^{n} (y_i - \hat{w}_0 - \hat{w}_1 x_i) x_i \end{pmatrix} = 0$$

How many minima $\hat{w}$ are there? It depends on $\{(x_i, y_i)\}_{i=1}^{n}$.

2.1.5 Other Losses

For now, we have only considered the **squared loss** as a loss function for our model. Squared loss weighs over- and underestimation the same, and the cost grows quadratically (*large errors hugely penalized*). Instead, one might want:

- Ignore outliers (ones with very large penalty) $\to$ *Huber loss*
- Weigh over- and underestimation differently $\to$ *Asymmetric losses*

2.2 Multiple Regression

2.2.1 Introduction

We now consider the case where we have more inputs available. We come back to our selling-a-house example and collect more data of the other houses on the market. For example:

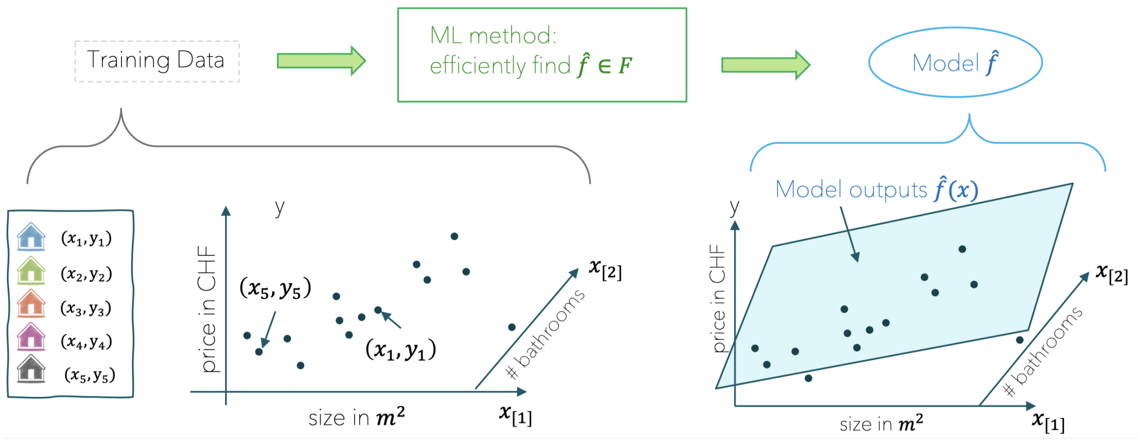
- $x_{[1]}$ = size in m^2
- $x_{[2]}$ = number of bathrooms
- $x_{[3]}$ = distance to the nearest train station
- $x_{[4]}$ = years since construction

This gives us an **attribute vector** $x = (x_{[1]}), ..., x_{[d]}$ and our model of a house is, for example, $x_1 = (120,\ 2,\ 0.5,\ 1)$, $y_1 = 1.5mio$.

Remark: It is of great importance to not confuse the following two notations:

- $x_i \to$ input attributes of sample i
- $x_{[i]} \to i$-th attribute

The following visualization shows the ML pipeline for $d = 2$, however it holds (and especially the upcoming formulas) for any $d > 1$:



2.2.2 Function Classes F

The class of linear functions with d *linear features* is given by:

$$F_{Lin} = \{f : f(x) = w_0 + \sum_{j=1}^{d} w_j x_{[j]} = w_0 + w^T x \text{ for } w = (w_1, ..., w_d) \in \mathbb{R}\}$$

For $d = 2$ this equates to $f(x) = w_0 + w_1 x_{[1]} + w_2 x_{[2]}$.

2.2.3 Training Loss

Analogous to the 1D-model, the learned model $\hat{f}$ minimizes the training loss, i.e.:

$$\hat{f} = \text{argmin}_{f \in F_{Lin}} L(f) = \text{argmin}_{f \in F_{Lin}} \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2$$

Since $\hat{f} \in F_{Lin}$ must be of the form $\hat{f}(x) = \hat{w}_0 + \hat{w}^T x$, it is equivalent to minimizing over vector w and scalar w_0:

$$\hat{w} = \text{argmin}_{w_0 \in \mathbb{R},\, w \in \mathbb{R}^d} L(w_0,\, w) = \text{argmin}_{w_0 \in \mathbb{R},\, w \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} (y_i - w_0 - w^T x_i)^2$$

We can rewrite the training loss L into **vector notation:**

$$L(w_0,\, w) = \frac{1}{n} \sum_{i=1}^{n} (y_i - w_0 - w^T x_i)^2 = \frac{1}{n} ||y - \mathbf{1} w_0 - Xw||^2,$$

where $\mathbf{1} = (1, ..., 1) \in \mathbb{R}^n$ is the all-ones vector.

2.2.4 Minimizing Training Loss

Again, for simplicity, let's set $w_0 = 0$ and minimize over w. Plugging that into $L(w_0,\, w) = \frac{1}{n}||y - \mathbf{1}w_0 - Xw||^2$ yields:

$$L(0,\, w) = \frac{1}{n}||y - Xw||^2 = \frac{1}{n}||y||^2 - \frac{2}{n} y^T X w + \frac{1}{n} w^T X^T X w,$$

with gradient $\nabla_w L(0,\, w) = \frac{2}{n}(X^T X w - X^T y)$ and Hessian $D^2 L(0,\, w) = \frac{2}{n} X^T X$.

We now look at two ways to find the minimum:

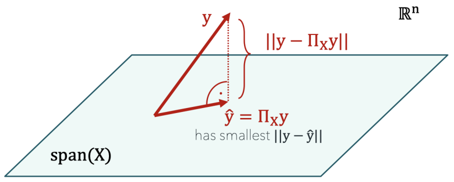
- Stationary point condition (gradient = 0)
- Geometric argument (orthogonal projection)

Stationary Point Condition Again, the minimum $\hat{w} = \text{argmin}_w L(0,\, w)$ must be a stationary point, i.e. satisfying $\nabla_w L(0,\, \hat{w}) = 0$. All stationary points of this quadratic loss $\frac{1}{n}||y - Xw||^2$ are minima, because the Hessian $\frac{2}{n} X^T X$ is positive-definite.

$\nabla_w L(0,\, w) = \frac{2}{n}(X^T X w - X^T y)$ together with stationary point condition $\nabla_w L(0,\, \hat{w}) = 0$ yields:

$$X^T y = X^T X \hat{w} \Rightarrow \hat{w} = (X^T X)^{-1} X^T y$$

Geometric Argument Since the set of all possible Xw is given by span(w), the closest point to y on span(X) is the orthogonal projection of y onto span(X). We denote this projection as $\Pi_X y$.



We now need to derive the equation for the vector $\hat{w}$ such that $X\hat{w} = \Pi_X y$. Since the residual $y - X\hat{w}$ is orthogonal to all vectors $v \in \text{span}(X)$, we have that $(y - X\hat{w})^T X w = 0$ for all $w \in \mathbb{R}^d$. Hence, we require $X^T(y - X\hat{w}) = 0 \Leftrightarrow X^T y = X^X \hat{w}$ (also called **normal equations**).

Finally, this yields again the unique solution $\hat{w} = (X^T X)^{-1} X^T y$ if $X^T X$ is invertible.

3 Optimization

3.1 1D-Case

The general idea for **optimization** in 1D is to iteratively try to minimize $L(w)$:

General iterative algorithm to minimize $L(w)$

1. Start at initial w^{start}
2. At each step calculate $w^{next} = w^{now} + \tilde{\eta} \cdot v$, where $\tilde{\eta}$ defines how far we move and v is the update direction depending on L and w^{now}
3. When we can't improve much more, stop
4. Output $\hat{w} = w^{final}$

At each step of the algorithm above, we have to ask ourselves:

- Which direction v should we choose?
- How far $\tilde{\eta}$ to go in that direction?
- Can we still improve more?

With common sense, if $L(w)$ decreases as w increases (i.e. $L'(w) < 0$), we should increase w and vice versa. In other words, we choose the direction $v = -\text{sign}(L'(w))$.

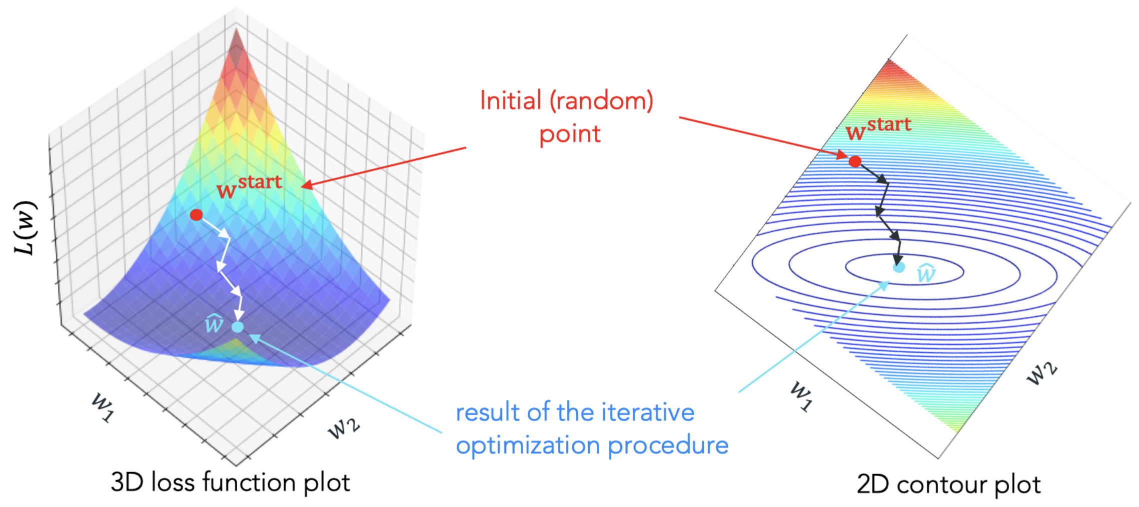
Choosing $\tilde{\eta}$ is harder. Generally speaking, we have to decrease $\tilde{\eta}$ as the slope gets smaller. Even then, η shouldn't be too large because else it can potentially begin to grow again.

Another important question to consider is when do we stop.

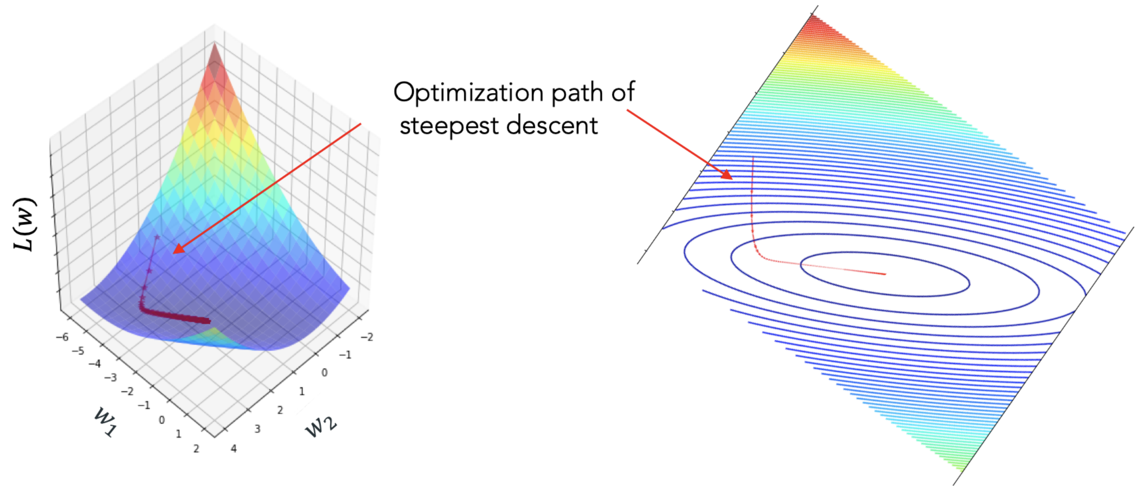
3.2 Multidimensional Case

3.2.1 Visualization of The Loss

Again, visualization beyond $d = 2$ is hard, hence our visualizations are for $w \in \mathbb{R}^2$. Example:



3.2.2 Steepest Descent



Remember, the general update formula is of the form $w^{next} = w^{now} + \tilde{\eta} v$ for some unit direction $||v|| = 1$. What is the **steepest descent direction** v?

If we assume that L is differentiable, then the linear approximation of the loss is given by:

$$L(w^{next}) \simeq L(w^{now}) + \tilde{\eta}\langle \nabla_w L(w^{now}),\, v\rangle$$

The idea is that for small $\tilde{\eta}$, the steepest direction on the tangent plane is similar to the steepest gradient on the true function. With the derivation of the steepest descent direction on the tangent plane we get $v = -\frac{\nabla L(w^{now})}{||\nabla L(w^{now}||}$. This leads to the **gradient descent update:**

$$w^{next} = w^{now} - \eta \nabla_w L(w^{now}), \quad \text{with stepsize } \eta = \frac{\tilde{\eta}}{||\nabla L(w)||}$$

This finally leads to the following algorithm:

Gradient descent algorithm to minimize $L(w)$

1. Start at initial w^0
2. Repeat $w^{t+1} \leftarrow w^t - \eta \nabla_w L(w^t)$
3. Stop when $||w^{t+1} - w^t|| \propto ||\nabla_w L(w^t)|| \leq 10^{-5}$
4. Output $\hat{w} = w^T$

The linear approximation for the loss value at step $t + 1$ is given by:

$$L(w^{t+1}) = L(w^t - \eta \nabla_w L(w^t)) \simeq L(w^t) - \eta \langle \nabla_w L(w^t),\, \nabla_w L(w^t)\rangle < L(w^t)$$

The negative gradient direction is a descent direction (for small enough stepsize η)!

3.2.3 Convergence and Stepsize for Linear Regression

Remember that for *linear regression* we have $L(w) = \frac{1}{n}||y - Xw||^2 = \frac{1}{n}[w^T X^T X w - 2w^T X^T y + y^T y]$.

If we use the gradient descent update $w^{t+1} = w^t - \eta \nabla L(w^t)$ and if we write $w_{min} = \text{argmin}_{w \in \mathbb{R}^d} L(w)$. Then indeed, $w^t \to w_{min}$ and $||w^t - w_{min}||_2 \leq \rho^t ||w^0 - w_{min}||_2$, where $\rho = ||I - \eta X^T X||_{op}$.

Using the chain rule, the gradient at any w reads $\nabla L(w) = -X^T(y - Xw)$, and we obtain $w^{t+1} = w^t - \eta \nabla L(w^t) = w^t + \nabla X^T(y - Xw^t)$.

If we remember that the global minimum w_{min} satisfies $X^T X w_{min} = X^T y$, then:

$$w^{t+1} - w_{min} = (I - \eta X^T X)(w^t - w_{min})$$

Furthermore, remember that for any matrix A, $||A||_{op} = \sup_x \frac{||Ax||_2}{||x||_2} = \max\{|\lambda_{min}(A)|, |\lambda_{max}(A)|\}$.

For $A = X^T X$ this is just the largest eigenvalue. Furthermore, by definition, $||Ax||_2 \leq ||A||_{op}||x||_2$ and hence:

$$||w^{t+1} - w^*||_2 \leq ||I - \eta X^T X||_{op}||w^t - w_{min}||_2 \leq ||I - \eta X^T X||_{op}^{t+1}||w^0 - w_{min}||_2$$

If $\rho < 1 \Leftrightarrow \lambda_{min}(X^T X) > 0$, the error goes to 0 as t goes to ∞. We say that the **gradient descent converges linearly / exponentially.**

3.2.4 Speeding Up Gradient Descent

To *speed up the gradient descent,* we want large steps in flat areas, and small steps in high curvature ones. This leads to the momentum/accelerated method, where we combine previous direction with the negative gradient direction:

$$w^{t+1} - w^t = \alpha(w^t - w^{t-1}) - \eta \nabla_w L(w^t)$$

3.2.5 Stochastic Gradient Method (SGD)

Remember that the training loss definition for parameterized functions is $L(w) = \frac{1}{n}\sum_{i=1}^n l(y_i, f_w(x_i))$. We have already seen the following algorithm:

Gradient Descent Algorithm

1. Start at initial w^0
2. Until $w^{t+1} - w^t$ is sufficiently small, repeat:
3. (a) Update $w^{t+1} \leftarrow w^t - \eta \nabla_w L(w^t)$
4. Output $\hat{w} = w^T$

We see in line 3 of the algorithm above that we need to compute the gradients at all points. This is very costly! The memory required is $O(nd)$ and the time $O(n \times \text{cost to compute } \nabla l)$. We propose another algorithm:

Stochastic Gradient Descent (SGD) Algorithm

1. Start at initial w^0
2. Until $w^{t+1} - w^t$ is sufficiently small, repeat:
3. (a) Update $w^{t+1} \leftarrow w^t - \eta \nabla_w L_S(w^t)$
4. Output $\hat{w} = w^T$

Here, in line 3, instead of computing the gradient for all points, we only consider a random subset of points $S \subset [1, ..., n]$ (so-called *minibatch SGD*). If S is just one random point, then it's called **SGD.**

3.3 Stationary Points and Minima

Remember that the gradient descent converges to a stationary point, but we want to get to the minimum! For differentiable losses:

- w is local minimum $\rightarrow w$ is a stationary point
- w is global minimum $\rightarrow w$ is a local minimum
- $L(w) < L(v) \, \forall v \in \mathbb{R}^d \rightarrow w$ is a global minimum

Mathematically, **convexity** is the function property that guarantees a local minimum to be a global minimum. We have three different conditions:

- 0-th order condition (iff.): $L(\lambda w + (1-\lambda)v) \leq \lambda L(w) + (1-\lambda)L(v)$
- 1st order condition (iff.): $L(v) \geq L(w) + \nabla L(w)^T (v-w)$
- 2nd order condition (iff.): Hessian $\nabla^2 L(w) \geq 0$

Furthermore, we can define a **strongly convex** functions. We say that $L(w)$ is strongly convex if $L(w) - \frac{m}{2}||w||^2$ is convex. We have the following conditions:

- 1st order condition (iff. for some $m > 0$): $L(v) \geq L(w) + \nabla L(w)^T (v-w) + \frac{m}{2}||v-w||^2$
- 2nd order condition (iff. for some $m > 0$): Hessian $\nabla^2 L(w) \geq mI$

3.4 Effect of Data on Global Minimum Prediction

Before we can determine how data affects the model quality, we need to formalize what a good model is:

- Goal: Intuition for how the sample size n can change the prediction performance. What is a good prediction?
- Goal standard: Assume the "true" average price is some *ground truth linear function* $f^*(x) = x^T w^*$
- Good prediction model $\hat{f}(x)$: should be close to $f^*(x)$, i.e. have a low error.

When we say that the prediction model should have a *low error,* we mean that:

$$l(\hat{f}(x), \, f^*(x)) = (\hat{f}(x) - f^*(x))^2 = ((\hat{w} - w^*)^T x)^2,$$

should be low. We often measure:

- Average error $\mathbb{E}(\hat{f}(x) - f^*(x))^2$ over all x, or
- for linear functions $||\hat{w} - w^*||_2$ since $((\hat{w} - w^*)^T x)^2 \leq (||\hat{w} - w^*||_2 ||x||_2)^2$

We often model the observed data as $y_i = f^*(x_i) + \epsilon_i = \langle w^*, \, x_i \rangle + \epsilon_i$ and call ϵ_i **noise.**

3.5 Different Dimensionalities

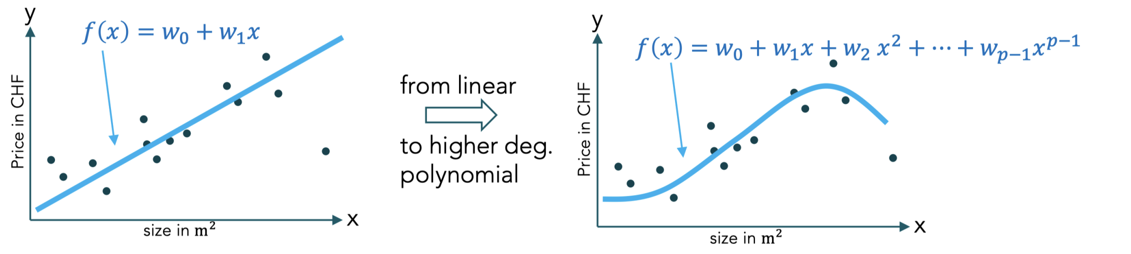
Let us assume $n = 2$ points and w_0 to be unknown. There are infinitely many w that satisfy $y = \mathbf{1}w_0 + Xw$, i.e. that interpolates the training data. If we now assume $n = 3$ points, for $d = 2$ there is now only one interpolating solution that satisfies $y = \mathbf{1}w_0 + Xw$.

In general: *the dimensionality of interpolating solutions decreases.*

Again, assuming $n > d$, we still have infinitely many w interpolating the training data. Which solution does the gradient descent actually find? If initialized at 0, the gradient descent converges to the minimum norm solution $\hat{w}_{GD} = \text{argmin}_w ||w||_2$ such that $y = \mathbf{1}w_0 + Xw$ that interpolates the data.

3.6 Going Beyond Linear Functions

Instead of linear functions, we might also consider **polynomial functions.** Both can be written as $f(x) = \sum_{j=1}^{p} w_j \phi_j(x)$ with feature vector $\phi(x) = (\phi_1(x), ..., \phi_p(x)) \in \mathbb{R}^p$:



Linear function with $\phi_1(x) = 1, \phi_2(x) = x$ uses additional monomials $\phi_3(x) = x^2 \ldots \phi_p(x) = x^{p-1}$

For the loss function we again consider the squared loss. We seek the minimizer of the square loss

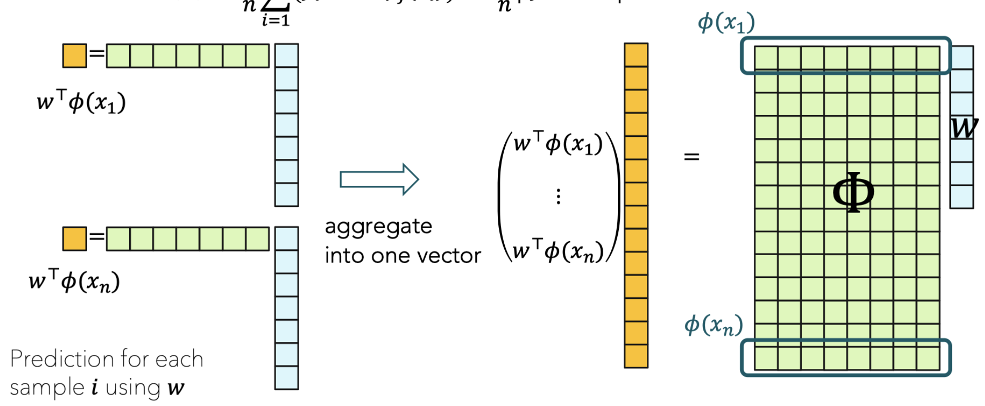
$$\hat{f} = \text{argmin}_{f \in F_\phi} \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2,$$

this time in some fixed feature function space F_ϕ. This is equivalent to minimizing over vector w:

$$\hat{w} = \text{argmin}_{w \in \mathbb{R}^p} L(w) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \sum_{j=1}^{p} w_j \phi_j(x))^2$$

We can rewrite the training loss L in matrix vector notation as follows:

$$L(w) = \frac{1}{n} \sum_{i=1}^{n} \left(y_i - w^\top \phi_j(x_i)\right)^2 = \frac{1}{n} ||y - \Phi w||^2$$



$w^\top \phi(x_1)$
$w^\top \phi(x_n)$
Prediction for each sample i using w
aggregate into one vector
$\begin{pmatrix} w^\top \phi(x_1) \\ \vdots \\ w^\top \phi(x_n) \end{pmatrix}$
=
$\phi(x_1)$
$\phi(x_n)$
Φ
w