ML - Machine Learning - D. Loiacono

# Machine Learning

Held by Prof. D. Loiacono at Politecnico di Milano 2023/2024

Notes by Rayan Emara

Table of contents

1. Disclaimers and preface

These notes were taken during AY 2023/2024 using older material, your mileage may vary. They're meant to accompany the lectures and in no way aim to substitute a professor yapping away at an iPad 30m away.

For any questions/mistakes you can reach me here.

All rights go to their respective owners.

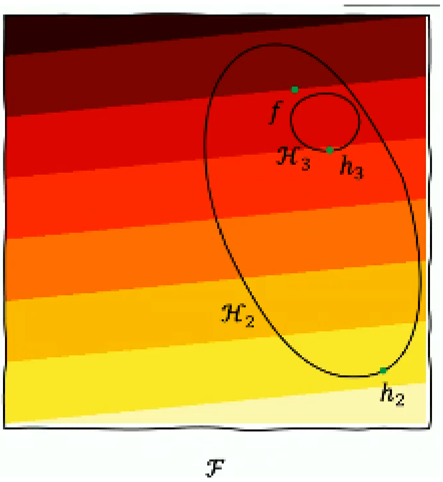
2. Overview of supervised learning

2.1. What kind of problem are we trying to solve ?

We're trying to find/approximate a function f, which is "generating" our dataset D.
The steps are:

- Define a loss function L
- Choose the hypothesis space $\mathcal{H}$
- Find in $\mathcal{H}$ an approximation h of f that minimizes L.

We might be tempted to enlarge the hypothesis space, but if we do that we can have a larger risk of approximating f very poorly as we don't know if we'd be expanding $\mathcal{H}$ in the correct direction.



2.2. The elements of a supervised learning algorithim

- The representation meaning the hypothesis space $\mathcal{H}$ and how it's designed.
- The evaluation, so how the loss function is designed.
- The optimization algorithim, how you're looking for candidate soltutions.

Some examples of representations are:

- Linear models.
- Neural networks.

Evaluation:

- Accuracy, the % of time you're correctly classifying a datapoint.
- If you have a regression problem you could use MSE.

Finally, the optimization technique really depends on your loss function:

- Gradient descent.
- Greedy search.
- Linear programming (in cases where you have some sort of constraint)

3. A supervised learning taxonomy

There are many ways you can try to classify/organize/compare, some more general paradigms other than the previous three.

3.1. Parametric vs Non-parametric

There are some learning algorithims for which the deisgn of the hypothesis space is such that when you decide to apply a learning algorithim to a problem you have to define a set of parameters that are fixed and won't change with the data these are called **Parametric**, while non parametric models scale the number of parameters with the training set.

3.2. Frequentist vs Bayesian

No meaningful definition given by the prof here.

3.3. Direct, discriminative or generative

They're different ways to see the same problem from different perspectives, in the **direct** approach you don't really care about the probabilstic interpretation, you're just optinmizing the loss function. In the **discriminative** you interpret your problem as a conditional density $p(t|x)$ you essentially try to learn the distribution and then compute the expected mean. In the **generative** tries to model the *joint* density $p(x,t)$, this allows to then infer the conditional density and generate novel samples by computing the conditional mean.

4. Linear regression

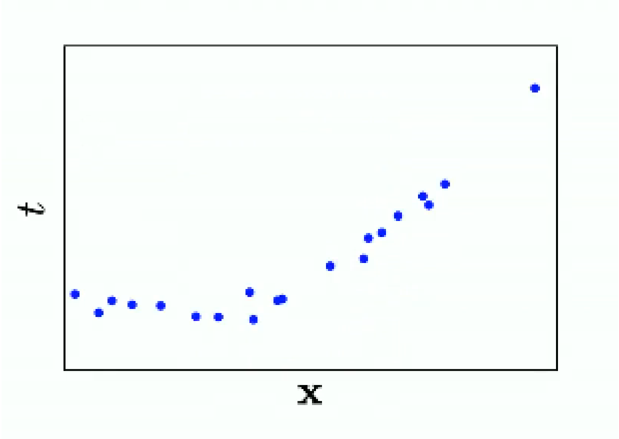
We'll focus on regression as our first type of problem, more specifically on linear models to solve.

References:

- *Pattern recognition and Machine learning*, Bishop

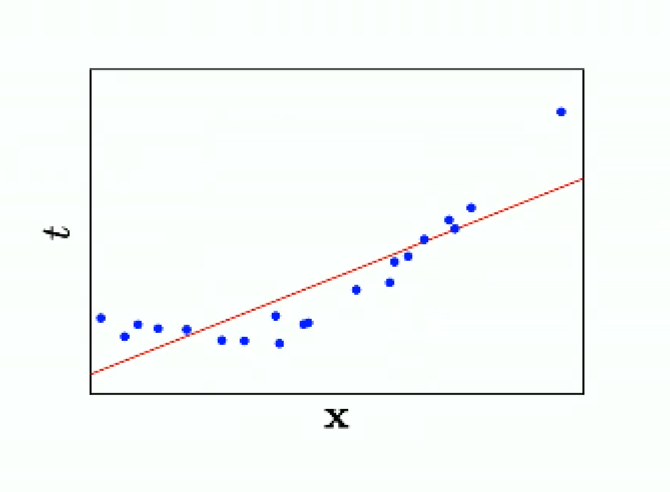
4.1. The model

We want to learn an approximation function $f(x)$:



We start from a dataset
$\mathcal{D} = < x, t > \Longrightarrow t = f(x)$
How do we model f ? How do we evaluate our approximation ? How do we optimize our approximation ?

In linear regression we model $f(x)$ with linear functions.



A model like this clearly has room for improvement but we can veeery easily explain what the model is doing. Another important property is that we can solve this analitically, this is better since models like NN are hard to "debug", in this case nothing can go wrong, we have algorithims with guaranteed convergence etc...

Another thing is that these models can capture non linear interactions.

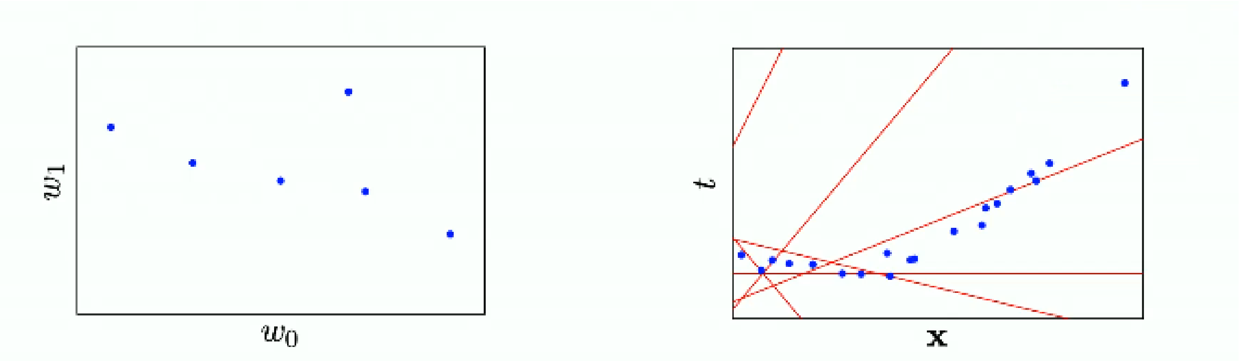
The simplest linear model can be defined as:

$$y(x, w) = w_0 + \sum_{j=1}^{D-1} w_j x_j = w^T x$$

Where:

- $x = (1, x_1, .., x_{D-1})$ is our input variable expressed as a vector to which we prepend 1.
- w_0 is called the bias parameter

My hypothesis space is 2D space with the two weights as dimensions.



Each point in the left image is a "solution" to our problem (a red line). We now need to *evaluate* these solutions, we need to define an error.

4.2. Loss function and optimization

A convenient error loss function is the sum of squared errors (SSE):

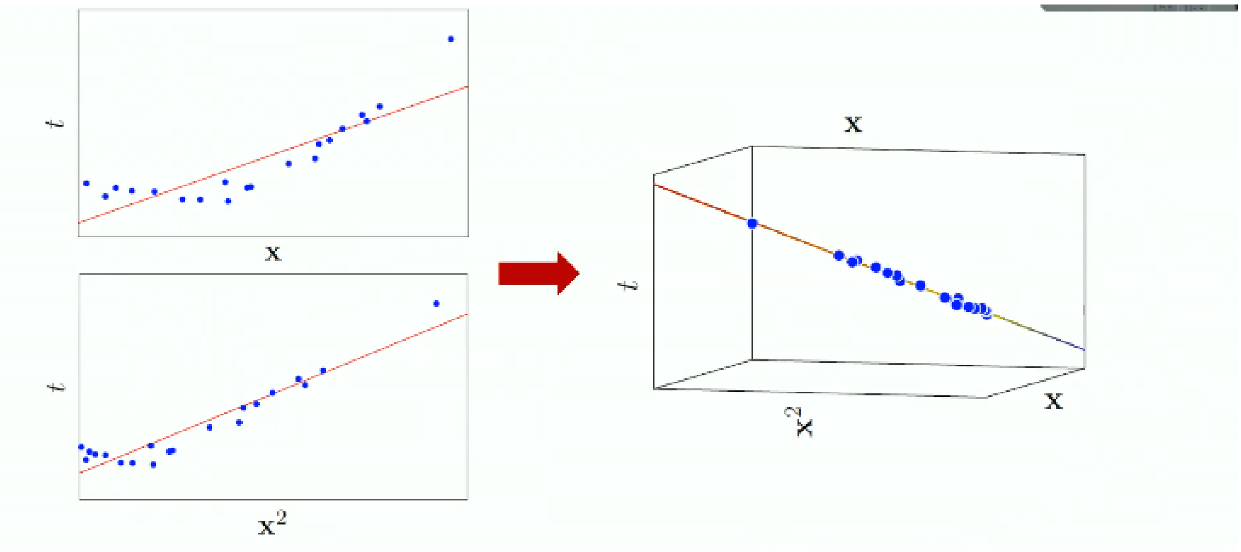
$$L(w) = \frac{1}{2}\sum_{n=1}^{N}(y(x_n, w) - t_n)^2$$

The reason we take the square instead of the absolute value is that ???

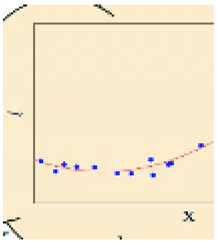
5. Linear models and basis functions

How do we capture non-linear relationships ? We can use basis functions. Instead of using $x_1, x_2, ..x_n$ we can use a set of non-linear functions ϕ_i (as many as we want), my linear model will be *linear* with respect to the basis functions (we're learning the weights).

In this example i generate 2 weights for each sample, one for x and for x^2.



If we now plot this approximation into the original space:



If we have problem specific knowledge we can pinpoint some sort of specific basis functions but generally there's no rule.
There are *families* of basis functions such as :

- Polynomial :

$$\phi_j(x) = x^j$$

- Gaussian :

$$\phi_j(x) = exp\left(-\frac{(x-\mu_j)^2}{2\sigma^2}\right)$$

- Sigmoidal:

$$\phi_j(x) = \frac{1}{1 + exp\left(\frac{\mu_j - x}{\sigma}\right)}$$

This can be hyper-parameterezide (idk how to spell it), more on this later in the course.

6. Least squares

From now on we'll assume that our problem is solved in the feature space and not in the input space (in the basis function space in the previous case).

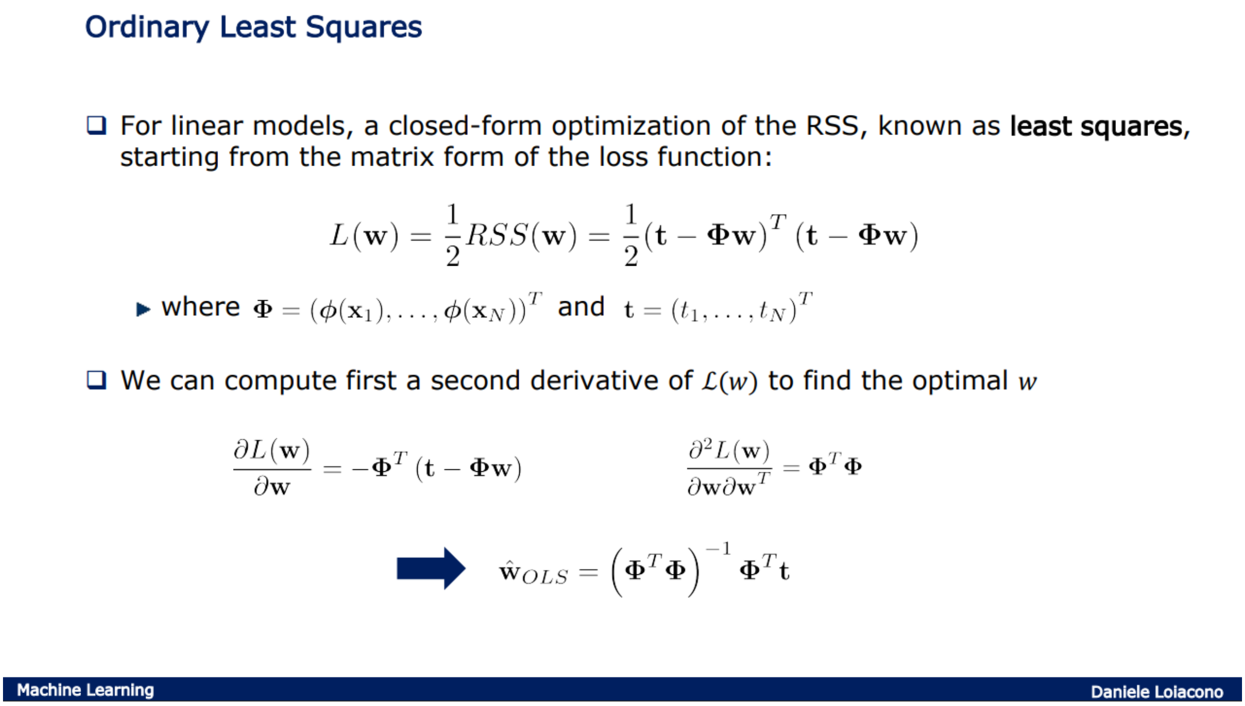
Prof emphasizes how important linear algebra is for this course...

$$L(w) = \frac{1}{2}RSS(w) - \frac{1}{2}(t - \phi w)^T(t - \phi w)$$

Where t is a vector with all the targets and ϕ is a matrix where on each row you have all the samples in the feature space (so on the cols you'd have features).

If you have to optimize this function with respect to a w. Let's assume it's a scalar value, how do we find the optimal value of w ?

Get the derivative and put the derivative equal to 0, right ?



Ordinary Least Squares

□ For linear models, a closed-form optimization of the RSS, known as **least squares**, starting from the matrix form of the loss function:

$$L(\mathbf{w}) = \frac{1}{2}RSS(\mathbf{w}) = \frac{1}{2}(\mathbf{t} - \mathbf{\Phi}\mathbf{w})^T(\mathbf{t} - \mathbf{\Phi}\mathbf{w})$$

- where $\mathbf{\Phi} = (\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_N))^T$ and $\mathbf{t} = (t_1, \ldots, t_N)^T$

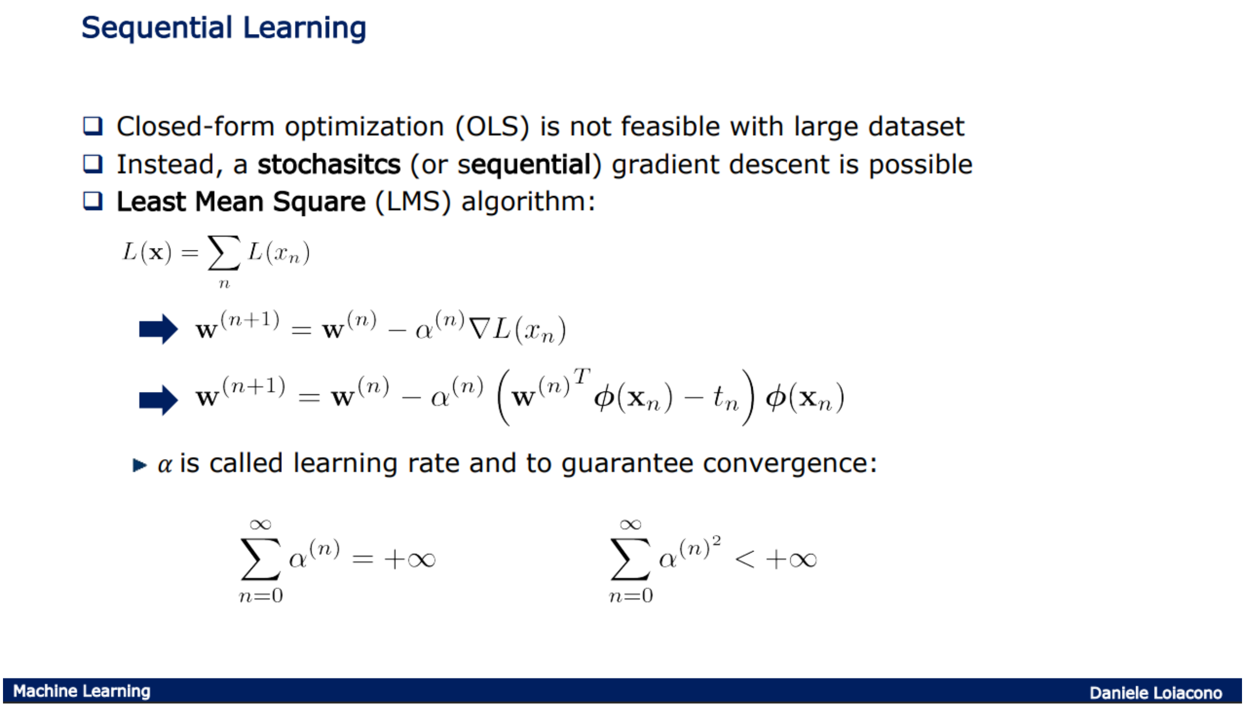
□ We can compute first a second derivative of $\mathcal{L}(w)$ to find the optimal w

$$\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}} = -\mathbf{\Phi}^T(\mathbf{t} - \mathbf{\Phi}\mathbf{w}) \qquad \frac{\partial^2 L(\mathbf{w})}{\partial \mathbf{w}\partial \mathbf{w}^T} = \mathbf{\Phi}^T\mathbf{\Phi}$$

$$\Rightarrow \quad \hat{\mathbf{w}}_{OLS} = \left(\mathbf{\Phi}^T\mathbf{\Phi}\right)^{-1}\mathbf{\Phi}^T\mathbf{t}$$

This is an analytical solution, the caveat is i need the matrix to not be singular cause then i'd have to invert it. It's also slow as fuck for large datasets.

You can instead apply **Gradient descent** (or SGD).



Sequential Learning

- ❑ Closed-form optimization (OLS) is not feasible with large dataset
- ❑ Instead, a **stochasitcs** (or **sequential**) gradient descent is possible
- ❑ **Least Mean Square** (LMS) algorithm:

$$L(\mathbf{x}) = \sum_n L(x_n)$$

$$\Rightarrow \mathbf{w}^{(n+1)} = \mathbf{w}^{(n)} - \alpha^{(n)} \nabla L(x_n)$$

$$\Rightarrow \mathbf{w}^{(n+1)} = \mathbf{w}^{(n)} - \alpha^{(n)} \left(\mathbf{w}^{(n)^T} \boldsymbol{\phi}(\mathbf{x}_n) - t_n \right) \boldsymbol{\phi}(\mathbf{x}_n)$$

- α is called learning rate and to guarantee convergence:

$$\sum_{n=0}^{\infty} \alpha^{(n)} = +\infty \qquad \sum_{n=0}^{\infty} \alpha^{(n)^2} < +\infty$$

Machine Learning Daniele Loiacono

Prof then talks about geometric interpretation of OLS (Ordinary least squares).

7. Multiple outputs

What if my target is a multitude regression targets ?

In practice it's just like running multiple regression problems in parallel, the main optimization is using the same basis functions.

8. Regularization

What happens if i use too many features ? (p.22 to 24 of 3).

We basically add a regularizing term to the loss function that takes into account how many features we added. We can extend the loss function to take into account the complexity of our model

$$L(\mathbf{w}) = L_D(\mathbf{w}) + \lambda L_W(\mathbf{w})$$

where $L_W(\mathbf{w}$ accounts for model complexity and λ is the **regularization** coefficient and w is the vector of model coefficients (or weights).

We can design $L_W(\mathbf{w}$ in many ways, we'll look at two in particular

8.1. Ridge regression (L2 regularization)

We define $L_W(w)$ as

$$L_W(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} = \frac{1}{2} \mid\mid \mathbf{w} \mid\mid_2^2$$

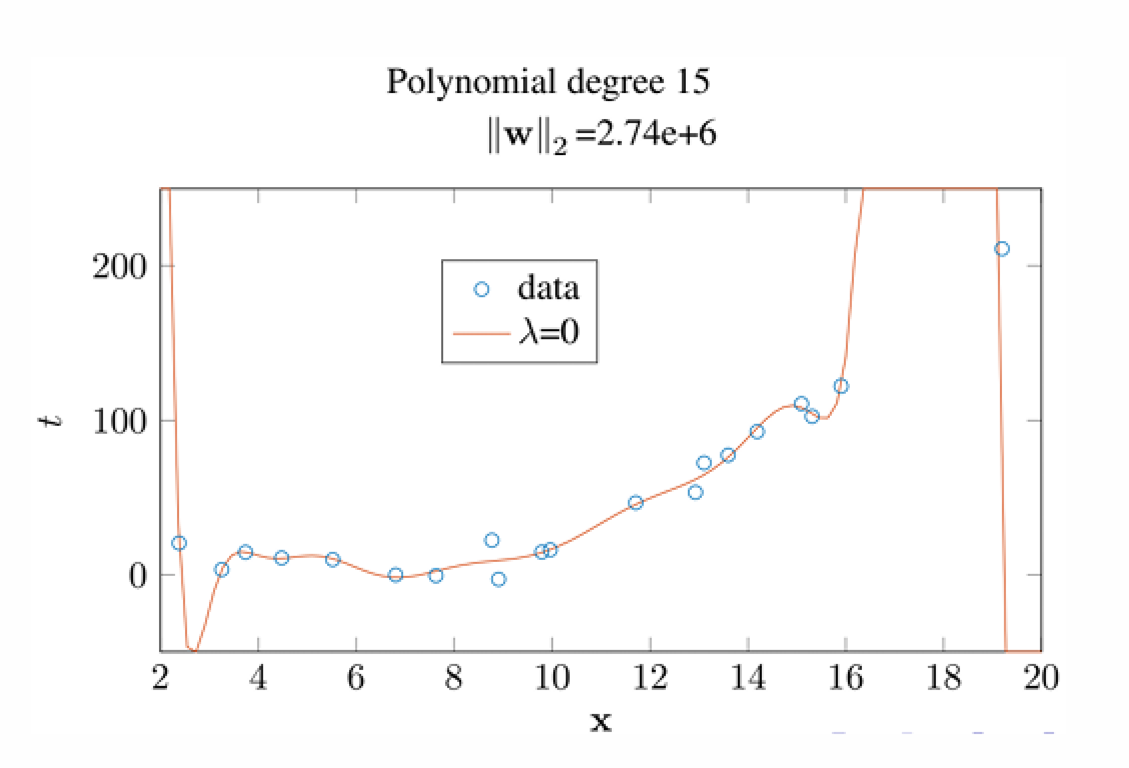
where the loss function $L(w)$ ends up being

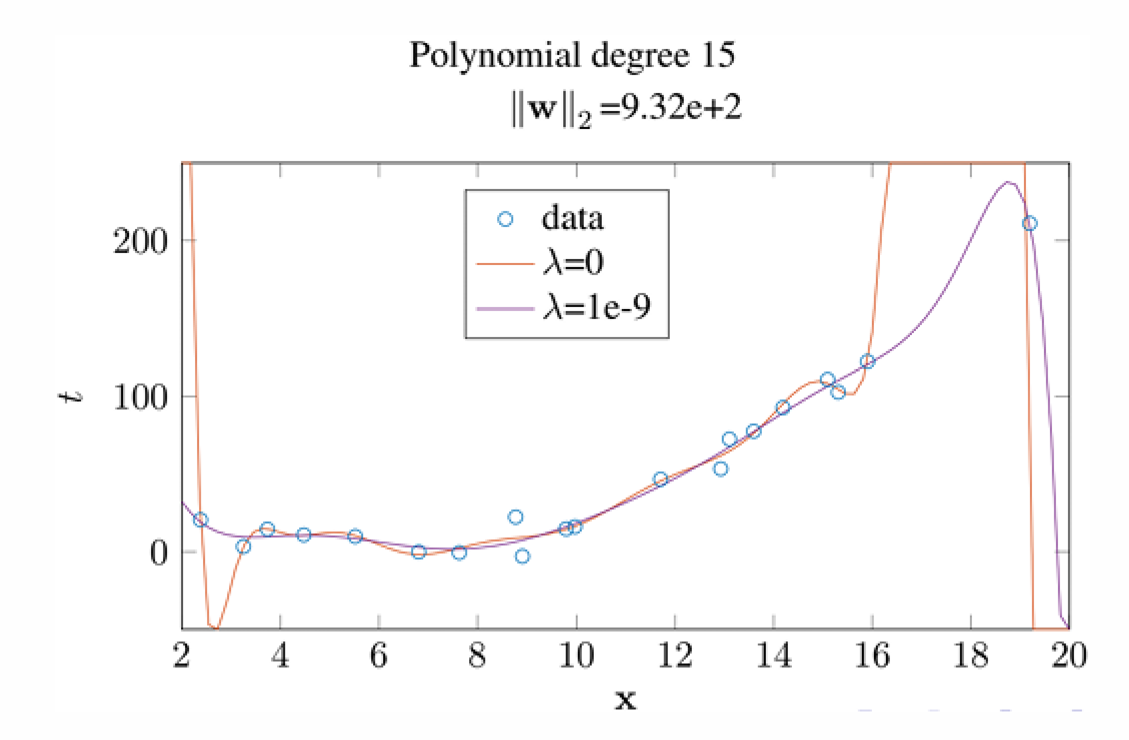
$$L\left(\mathbf{w}\right) = \frac{1}{2}\sum_{i=1}^{N}\left(t_i - \mathbf{w}^T\phi(\mathbf{x}_i)\right)^2 + \frac{\lambda}{2} \mid\mid \mathbf{w} \mid\mid_2^2$$

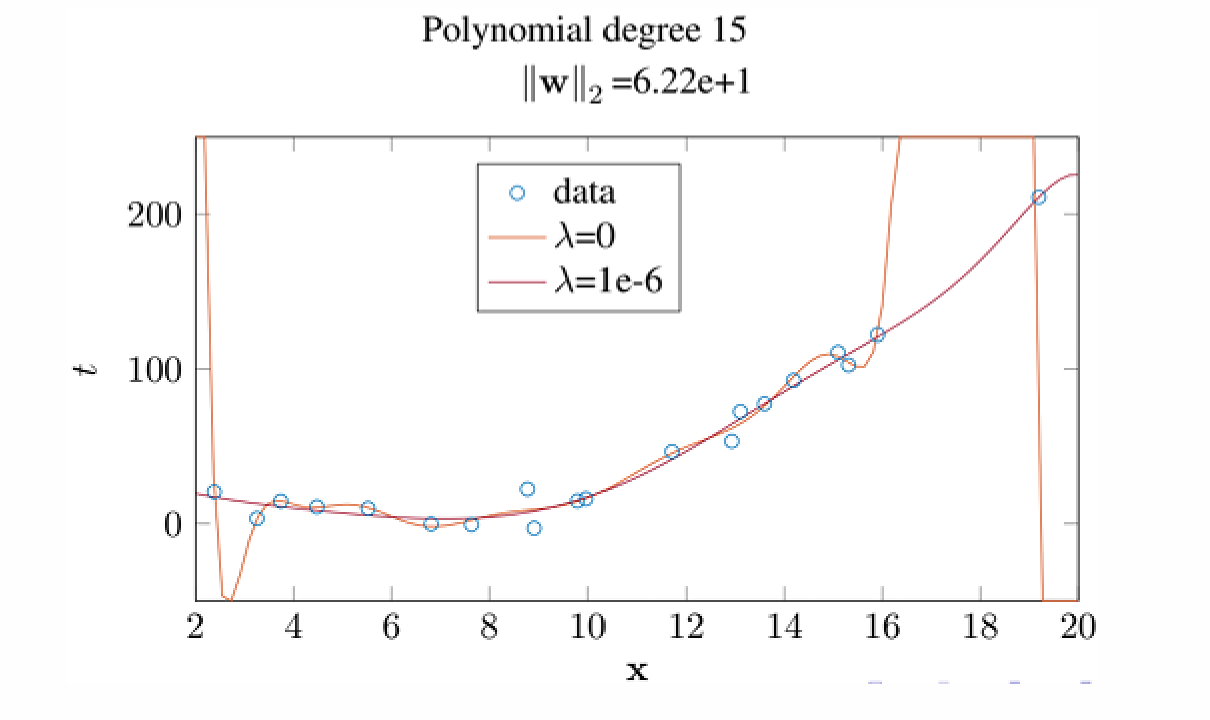
the closed form (estimator) is

$$\hat{\mathbf{w}}_{ridge} = (\lambda \mathbf{I} + \mathbf{\Phi}^T \mathbf{\Phi})^{-1} \mathbf{\Phi}^T \mathbf{t}$$

We're essentially an L^2 penalty







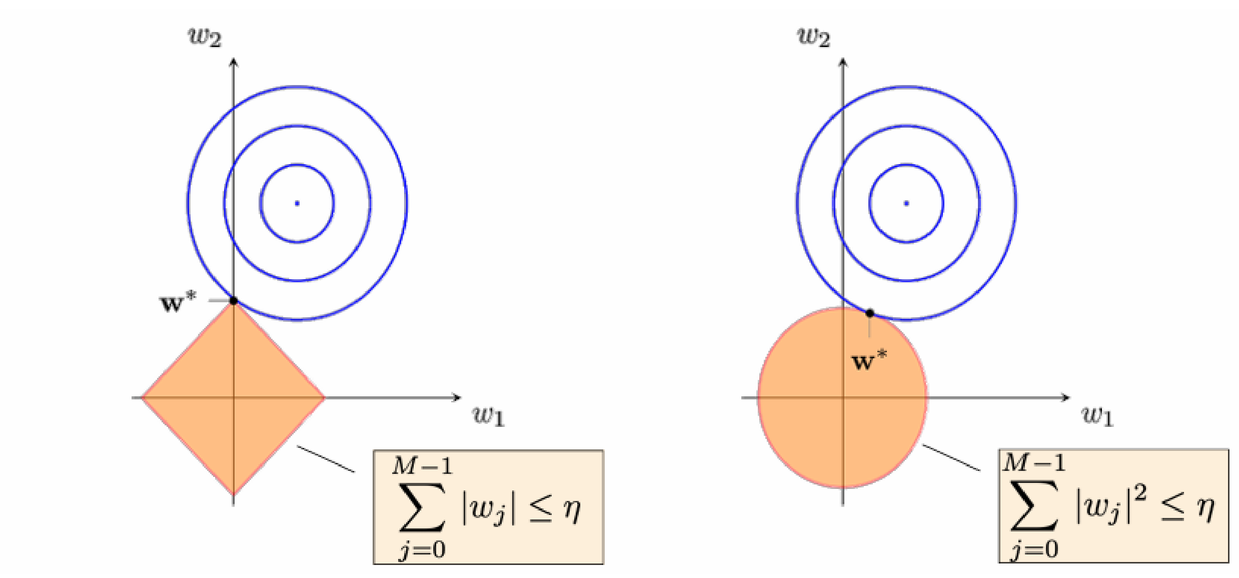
8.2. Lasso regression (L1 regularization)

Lasso stands for *least absolute shrinkage and selection operator*

$$L_W(\mathbf{w}) = \frac{1}{2} \parallel \mathbf{w} \parallel_1 = \frac{1}{2} \sum_{j=0}^{M-1} |w_j|$$

$$L(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{N} \left(t_i - \mathbf{w}^T \phi(\mathbf{x}_i)\right)^2 + \frac{\lambda}{2} \parallel \mathbf{w} \parallel_1$$

When λ is large enough some weights might be **equal to zero**. That's because we're basically creating a constraint region in the shape of a diamond (or a cross-polytope) in the weight space. When the loss function's contours (ellipsoids) are minimized under this constraint, the solution often occurs at the corners of the diamond.


w_2
$\mathbf{w}^*$
w_1
$\sum_{j=0}^{M-1} |w_j| \leq \eta$
w_2
$\mathbf{w}^*$
w_1
$\sum_{j=0}^{M-1} |w_j|^2 \leq \eta$

9. Least squares and Maximum likelihood

We can also approach regression in a probabilistic way by defining a probabilistic model that maps input x to outputs t using some unknown parameters w

$$y(x, w)$$

we then model the **likelihood** i.e. the probability that observed data $\mathcal{D}$ is generated by a given set of parameters w

$$p(\mathcal{D}|w)$$

we then estimate those parameters by maximizing the likelihood that those w generated our observed sample

$$w_{ML} = \arg\max_{w} p(\mathcal{D}|w)$$

In the case of linear regression our model can be defined as:

$$t = y(x, w) + \epsilon = w^T \phi(x) + \epsilon$$

where $y(x, w)$ is taken to be a linear model disrupted by Gaussian noise $\epsilon \sim \mathcal{N}(0, \sigma^2)$
Given a dataset $\mathcal{D}$ of N samples with inputs $X = \{x_1, \dots, x_N\}$ and outputs $T = \{t_1, \dots, T_N\}^T$

$$p(\mathcal{D}|w) = p(\mathbf{t}|\mathbf{X}, w, \sigma^2) = \prod_{n=1}^{N} \mathcal{N}\left(t_n|w^T \phi(\mathbf{x}_n), \sigma^2\right)$$

Since we're assuming the datapoints to be i.i.d. the likelihood function is the product of individual Gaussian distributions for each data points.
To find w_{ML} we usually prefer to deal with the **log**-likelihood since it is a strictly increasing map

$$\ell(w) = \ln p(\mathbf{t}|\mathbf{X}, w, \sigma^2) = \sum_{n=1}^{N} \ln p(t_n|\mathbf{x}_n, w, \sigma^2) = -\frac{N}{2} \ln\left(2\pi\sigma^2\right) - \frac{1}{2\sigma^2} RSS(w)$$

we then set the gradient of this object to zero and find the solution to that equation

$$\nabla \ell(w) = \sum_{n=1}^{N} t_n \phi(\mathbf{x}_n)^T - w^T \left(\sum_{n=1}^{N} \phi(\mathbf{x}_n)\phi(\mathbf{x}_n)^T \right) = 0$$

$$w_{ML} = \left(\mathbf{\Phi}^T \mathbf{\Phi}\right)^{-1} \mathbf{\Phi}^T \mathrm{t}$$

10. Bayesian Linear Regression

We first define the model given our current knowledge, we capture our assumptions by wrapping them into our a priori distribution (in teal) over some unknown parameters before seeing the data, we then observe the data and compute the posterior probability distribution (in red) for the parameters given the observed data.

$$p(parameters|data) = \frac{p(data|parameters)p(parameters)}{p(data)}$$

we then use the posterior distribution to

- Make predictions
- Account for uncertainty over the parameters

Example

The way this works is basically the concept of Bayesian learning, it allows us to update our beliefs after observing data.

Say we have a prior belief that a certain coins bias p follows a β distribution with parameters $\alpha = 2$ and $\beta = 2$, this means we think the coin is fairly balanced with a bias centered around $p = 0.5$.
Now let's say we flip the coin 10 times and observe that it lands heads 6 times and tails 4 times, using Bayes' theorem we compute the posterior distribution of the coin bias p given the observed data which will also be a β distribution (prove this hehe) with different parameters.

We've successfully employed Bayes' theorem to update our prior belief system, we can now use this posterior distribution to

- Predict the outcome of future coin flips by averaging over the posterior distribution of p
- Quantify our uncertainty about the coin bias p using summary statistics of the posterior distribution, such as the mean, median, or credible intervals
- U se the posterior distribution to calculate the expected loss under different decisions and choose the decision with the lowest expected loss.

The general formula is:

$$p(w|\mathcal{D}) = \frac{p(\mathcal{D}|w)p(w)}{p(\mathcal{D})}$$

where $p(w)$ is the prior probability over the parameter or what we know before observing the data, $p(\mathcal{D}|w)$ is the **likelihood** or the probability of observing the data ($\mathcal{D}$) given some value of the parameters (w).
We're essentially computing a probabilistic mean so we have to normalize the product on the numerator, that's where $p(\mathcal{D})$ comes

$$p(\mathcal{D}) = \int p(\mathcal{D}|w)p(w)\mathrm{d}w$$

We usually model the prior distribution using a Gaussian likelihood

$$p(w) = \mathcal{N}\left(w|w_0, \mathbf{S}_0\right)$$

this means that the posterior distribution result from the Bayes theorem is still a Gaussian distribution (known property of Gaussian distributions)

$$p(\mathbf{w}|\mathbf{t}, \mathbf{\Phi}, \sigma^2) \propto \mathcal{N}\left(\mathbf{w}|\mathbf{w}_0, \mathbf{S}_0\right)\mathcal{N}\left(\mathbf{t}|\mathbf{\Phi}\mathbf{w}, \sigma^2\mathbf{I}\right)$$

$$\begin{aligned}
p(\mathbf{w}|\mathbf{t}, \mathbf{\Phi}, \sigma^2) &= \mathcal{N}\left(\mathbf{w}|\mathbf{w}_N, \mathbf{S}_N\right) \\
\mathbf{w}_N &= \mathbf{S}_N\left(\mathbf{S}_0^{-1}\mathbf{w}_0 + \frac{\mathbf{\Phi}^T\mathbf{t}}{\sigma^2}\right) \\
\mathbf{S}_N^{-1} &= \mathbf{S}_0^{-1} + \frac{\mathbf{\Phi}^T\mathbf{\Phi}}{\sigma^2}
\end{aligned}$$

Note that:

- The posterior acts as prior for the next iteration in the case of sequential data.
- If the prior has infinite variance w_N converges to the **maximum likelihood** estimator
- If $\mathcal{S} \to \infty$ it converges to **ordinary least squares** instead.

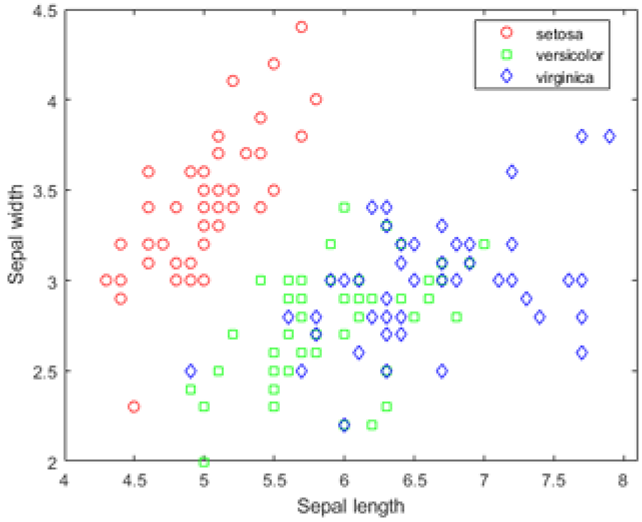
MISSING SOME STUFF HERE p.6 of Salvatore Buono study the part where OLS and ML

11. Linear classification

Given a dataset $\mathcal{D}$ we want to learn an approximation function $f(x)$ that maps input x to a discrete class C_k where $K = 1, \ldots, k$.

$$\mathcal{D} = \langle x, C_k \rangle \quad \longrightarrow \quad C_k = f(x)$$

We need to predict discrete class labels, more specifically the (linear) decision boundaries that divide the different class labels.
This type of classification is linear despite employing non-linear activation functions, the name comes from the linear decision boundaries



There are 3 main approaches:

- **Discriminant function:** we model a parametric *function* that directly maps the input to a class.
- **Probabilistic discriminative approach:** where we model $p(C_k|x)$ with respect to certain parameters and learn them from the training dataset (logistic regression).
- **Probabilistic generative approach (Bayesian):** where we model $p(C_k|x)$ and the prior $p(C_k)$ and infer the posterior distribution using Bayes' theorem

In linear classification we'll use **generalized linear models**:

$$f(\mathbf{x}, \mathbf{w}) = f\left(w_0 + \sum_{j=1}^{D-1} w_j x_j\right) = f(\mathbf{x}^T\mathbf{w} + w_0)$$

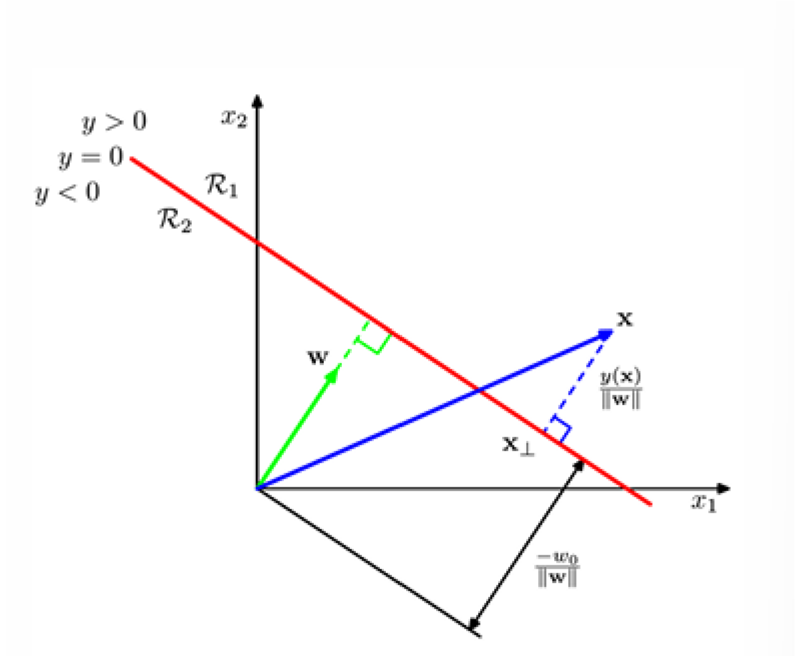
where $f(\cdot)$ is **not** linear in $\mathbf{w}$ because its output is a discrete value or alternatively a probability value. The $f(\cdot)$ effectively partitions the input space into **decision boundaries** (**decision surfaces** in higher dimensions) which are linear functions of $\mathbf{x}$ and $\mathbf{w}$ as they satisfy

$$\mathbf{x}^T\mathbf{w} + w_0 = \text{const}$$

The most common choice for encoding for the general $k-$classes problem is **1-of-k** (or 1 hot encoding) encoding where $t \in B^k$ with B representing the set of 0 and 1.
With this encoding t and $f(\cdot)$ represent the density probability over the classes.
In the case of two-class problems we might opt for the natural $t \in B^0$, $t \in \{-1, 1\}$ is sometimes used for some algorithms.

The discriminant linear function for a two-class problem would be

$$f(\mathbf{x}, \mathbf{w}) = \begin{cases} C_1, & \text{if } \mathbf{x}^T\mathbf{w} + w_0 \geq 0 \\ C_2, & \text{otherwise} \end{cases}$$


$y > 0$
x_2
$y = 0$
$y < 0$
$\mathcal{R}_1$
$\mathcal{R}_2$
$\mathbf{w}$
$\mathbf{x}$
$\frac{y(\mathbf{x})}{\|\mathbf{w}\|}$
$\mathbf{x}_\perp$
x_1
$\frac{-w_0}{\|\mathbf{w}\|}$