Machine Learning basics

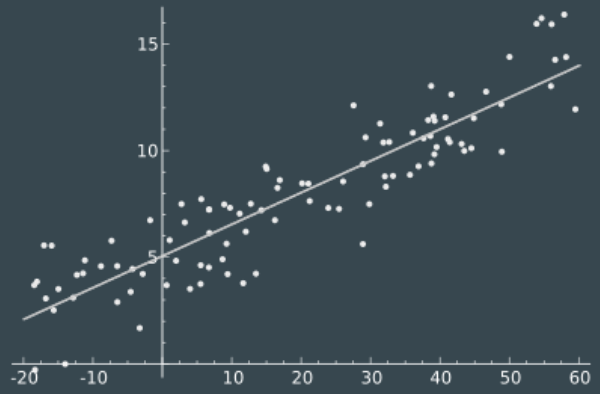
Application to electronic structure calculations

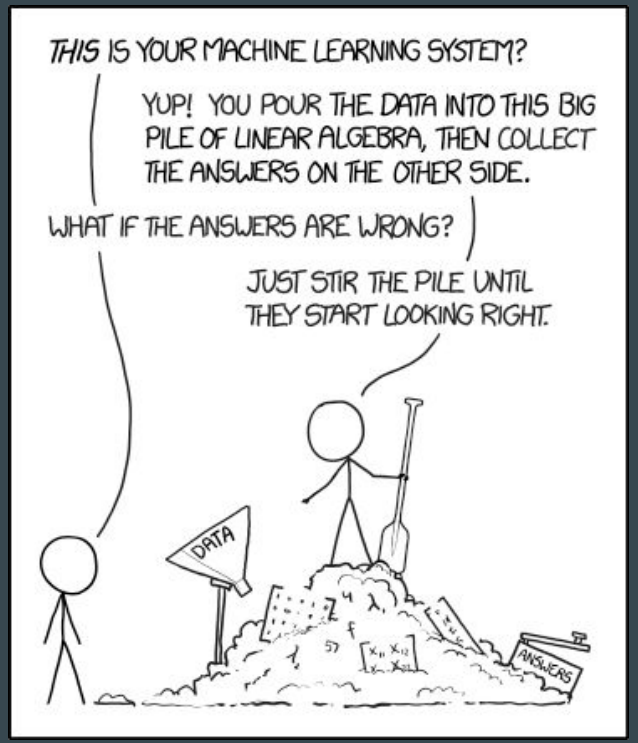
- Machine Learning
- Bayes theorem and maximum likelihood
- Linear regression
- Regularization (ridge regression)
- KRR
- Perceptron
- (Deep) Feed forward neural networks
- Training & Optimization techniques

What is Machine Learning (ML)?

“Machine learning is a field of computer science that uses statistical techniques to give computer systems the ability to "learn" (i.e., progressively improve performance on a specific task) with data, without being explicitly programmed.”

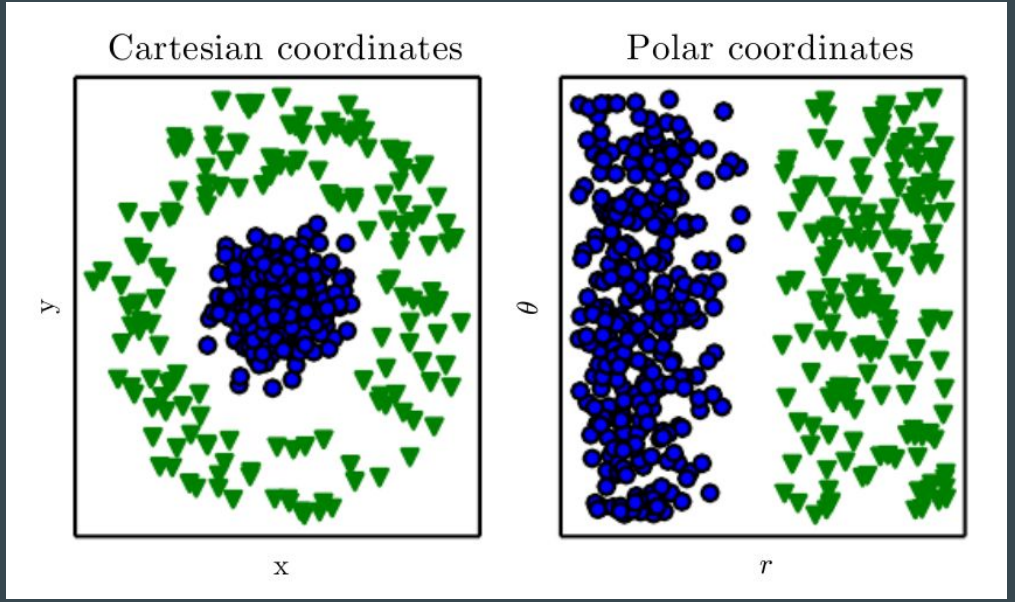
--Wikipedia





Representation

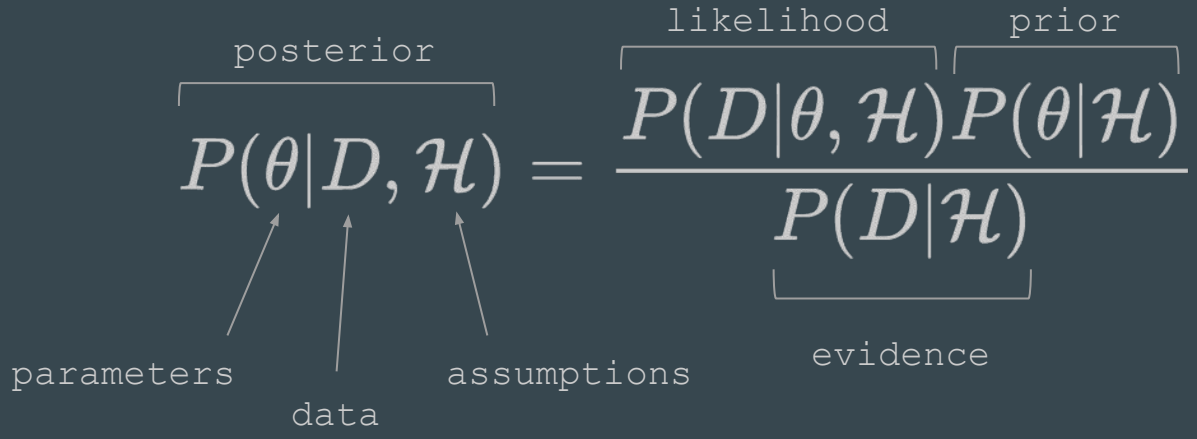
- Make a hard problem simple (and vice versa)
- Not easy to define (can we make the system define it?)
- More is not always better



Bayes' Rule

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)}$$

Typically used for inference as:



Maximum likelihood

Choosing the parameters that maximize the likelihood $P(D|\theta, \mathcal{H})$

(often easier to work with logarithms)

$$\theta_{\mathrm{ML}} = \arg\max_\theta \log P(D|\theta, \mathcal{H}) = \arg\max_\theta \sum_i \log P(x_i|\theta, \mathcal{H})$$

[cf. minimizing the cross entropy]

Calculate for a Gaussian the MLE of mean and variance.

[for proper marginalization see McKay chap. 24]

Linear regression

Given $(\vec{x}_i, y_i)$, find the best linear function describing their relation.

$$\hat{y}_i = \vec{w} \cdot \vec{x}_i$$

Least squares: minimize the estimator $C(\vec{w}) = \sum_i (y_i - \vec{w} \cdot \vec{x}_i)^2$

In matrix form: $C(w) = (Xw - y)^\top (Xw - y)$

One could do numerically or take derivative: $w_{\min} = (X^\top X)^{-1} X^\top y$

MLE: Assume a Gaussian per point, with means from linear equation.

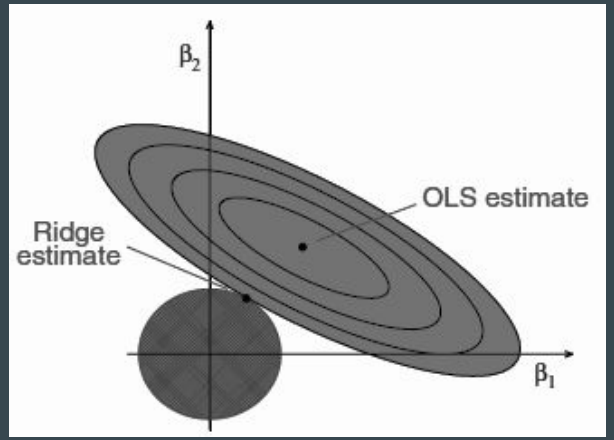
Ridge regression (Tikhonov regularization, weight decay, ...)

We can add a penalty for large weights:

$$C(\vec{w}) = \sum_i (y_i - \vec{w} \cdot \vec{x}_i)^2 + \lambda |\vec{w}|^2$$

The minimum is now:

$$w_{\min} = (X^\top X + \lambda I)^{-1} X^\top y$$



This tends to produce small weight for less important dimensions.

Regularization is controlled with hyperparameter λ (how to tune it?).

Regularization

Regularization is a process of introducing additional information in order to solve an ill-posed problem or to prevent overfitting.

Ridge regression is L2 regularization.

In other cases we use L1 regularization (least absolute shrinkage and selection operator LASSO).

Can be seen as effect of prior distribution.

Kernel trick

We want to fit nonlinear functions, but we like linear algebra...

→ Why not to do linear algebra on a basis of nonlinear functions?

In principle it means substituting $\vec{x}_i \rightarrow \phi_\alpha(\vec{x}_i)$

but it is often fast to just compute the kernel function

$$k(\vec{x}_i, \vec{x}_j) = \langle \phi(\vec{x}_i), \phi(\vec{x}_j) \rangle$$

working in the data space instead of the original dimensions.

Kernel Ridge Regression (KRR)

We found for regression: $w_{\min} = (X^\top X + \lambda I)^{-1} X^\top y = X^\top (XX^\top + \lambda I)^{-1} y$

So our prediction for a new x' is: $y' = w^\top x' = y^\top (XX^\top + \lambda I)^{-1} (Xx')$
which is now like a projection on previous observations.

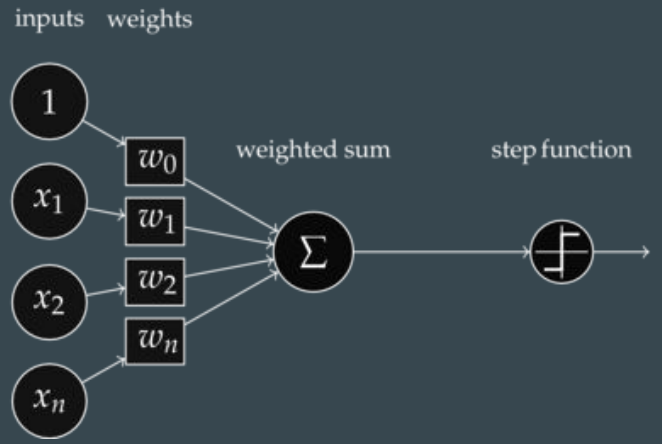
If we move to kernels $\vec{x}_i \to \phi_\alpha(\vec{x}_i)$, if we can calculate the kernel between two data points we get $y' = y^\top (K + \lambda I)^{-1} k(x_i, x')$

For example the k function could be a Gaussian.

Perceptron

The simplest neural network: one layer binary classifier, with step function

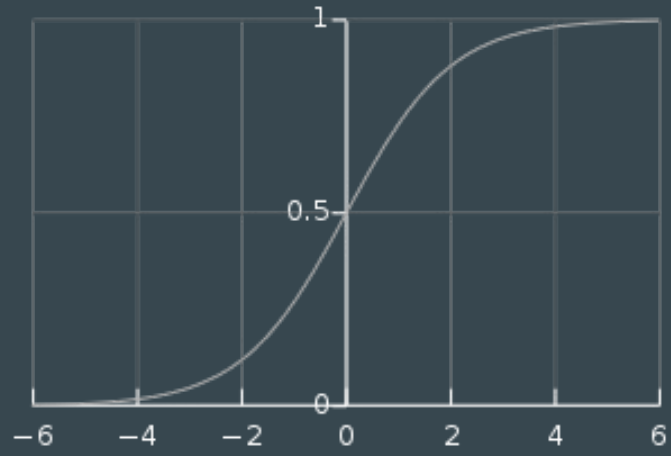
$$y = \theta\left(\sum_{\alpha} w_\alpha x_\alpha\right)$$



We update weights with $w_\alpha \leftarrow w_\alpha + \eta \sum_i (\hat{y}_i - y_i) x_\alpha^i$

(we cannot backpropagate through the step function)

Logistic regression



Regression for a binary dependent variable y=0, 1

We model with a sigmoid or logistic function:

$$\sigma(t) = \frac{1}{1 + e^{-t}}$$

Where t are the log-odds, which we take as linear combination of inputs.

Cost function (log likelihood): $C(w) = -\sum_i \log[\sigma(wx_i)^{y_i}(1 - \sigma(wx_i))^{1-y_i}]$

then optimize weights with backpropagation and SGD or similar...

Can be used as a classification model.

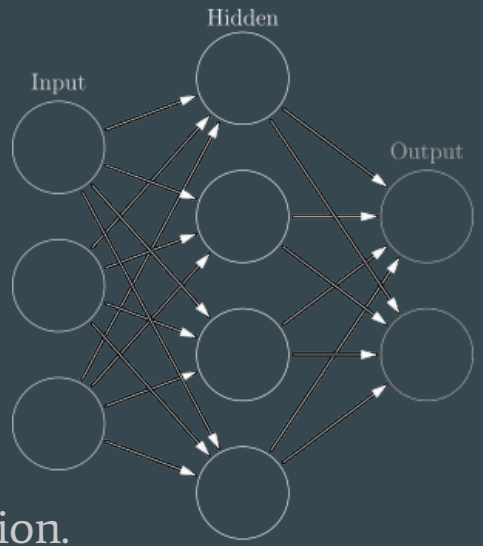
Feedforward neural networks

Parallel/serial stack of single neuron units (cf. perceptron):

$$h_i = f(w_{ij}x_j + b_i)$$

Activation function f can be: step, sigmoid, ReLU, gaussian, ...

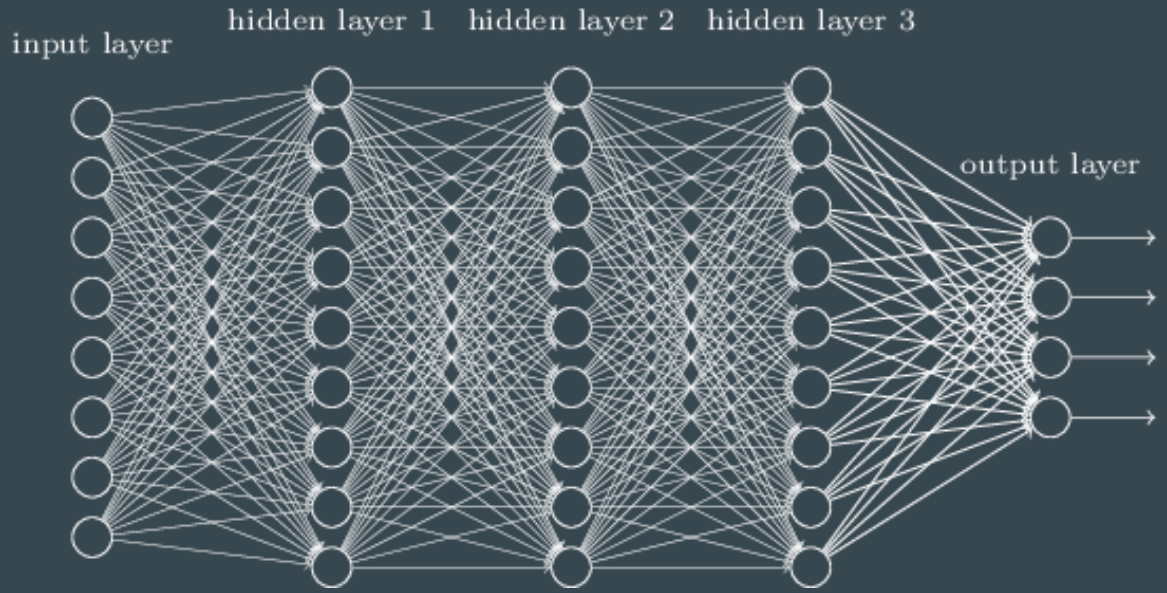
Define cost function C(w), then optimize through backpropagation.



Simplest deep neural network

Stack multiple hidden units.

Why does this work?



Gradient descent

For linear regression we saw that

$$\nabla_w C(w) = 2X^\top (Xw - y)$$

We can find the minimum by updating

$$w \leftarrow w - \alpha \nabla_w C(w)$$

But how hard is this to compute?

Also: what do we really want to achieve?

Stochastic gradient descent

What if we calculated the cost only on a subset of B data points? → (mini)batch

$$w \leftarrow w - \frac{\alpha}{B} \nabla_w \sum_{i=1}^{B} C(w, \vec{x}_i)$$

This is an estimator for the mean, plus a stochastic term of width $\propto \frac{1}{\sqrt{B}}$

[cf. overdamped Langevin dynamics]

Can we improve this?

Backpropagation

Given a network and some new example:

- Calculate current predictions
- Calculate cost of current prediction (supervised/unsupervised)
- Calculate gradients with chain rule

Can have problems with vanishing gradients.