High level overview of ML:

data $\begin{cases} \text{if labelled: } (X_{N\times d}, y_{N\times 1}), \\ \text{otherwise } (X_{N\times d}): \end{cases}$

$X_{i,j} \in \mathbb{R}$ or some interval $[a,b]$

$y_i \in \mathbb{R}$ for regression, or a discrete set for classification

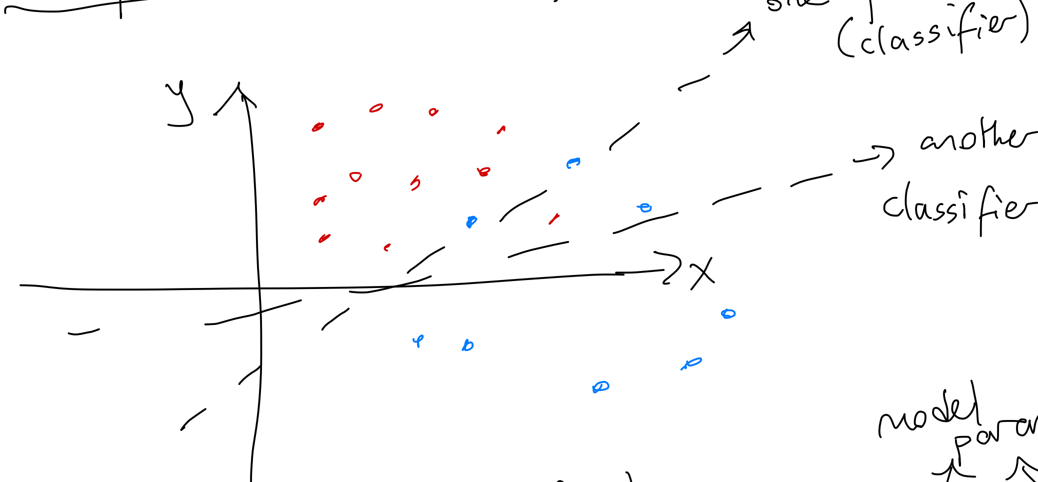
N: # of datapoints

d: # of dimensions of each datapoint

we determine a specific type of function (sometimes called estimator) that we want to use to infer something useful like the y value or class or cluster it belongs to.

This function is parameterized. These model parameters are what the "machine learns".

example: linear classifier



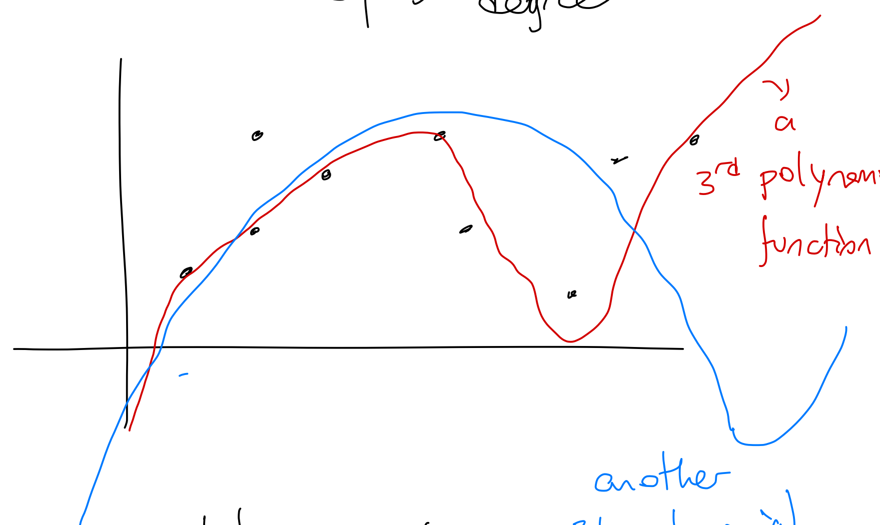
treated as a parameterized single function because these all look like

model parameters

$a x + b$

$A \cdot x + b$

Example: polynomial regression of 3rd degree



another 3dg polynomial func.

model parameters.

$a_0 + a_1 x + a_2 x^2 + a_3 x^3 = y$

How does the learning work? Usually there is some metric (i.e. loss, cost) that is being minimized by changing the model parameters. In the case of the example 2 it can be squared error or absolute error or others (meaning that your learning changes the model parameters to minimize the squared error between the guess $\hat{y}$ and true value y). In example 1 it can be cross-entropy loss or hinge loss or squared loss between one-hot label vectors.

2nd class $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$ 1st class $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$

* One popular way of learning (or optimizing your loss) is called gradient descent. (question 2c) In this method, you take numeric gradient of the loss with respect to all model parameters: $\frac{\partial loss}{\partial a}, \frac{\partial loss}{\partial b}$

and then update your parameters: $a^{(i+1)} = a^{(i)} - \eta \cdot \frac{\partial loss}{\partial a^{(i)}}$. η is called learning rate.

then repeat until your loss converges (or maxiter)

Hyperparameters and Grid Search

Hyperparameters are <u>different from parameters</u>: they are not learnt from data.* It is you, the ML practitioner that tweaks these hyperparameters (or searches over them automatically). The set of hyperparameters determine a <u>family of functions</u>.

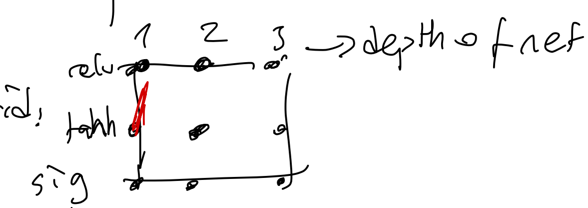
<u>Example 3</u>: <u>polynomial regression</u>: hyperparameter: degree of the polynomial
parameters: coefficients of the polynomial

<u>Example 4</u>: <u>neural network</u>: hyperparameters: depth of the network
\# of neurons in each layer
<u>activation function</u>
neuron connectivity (convolutional, dense etc)
optimizer arguments.
parameters: weights & biases

<u>Grid search</u>: Divide the space of possible hyperparameters and for each point in this space, train the model. See which parameters work the best.

grid: 

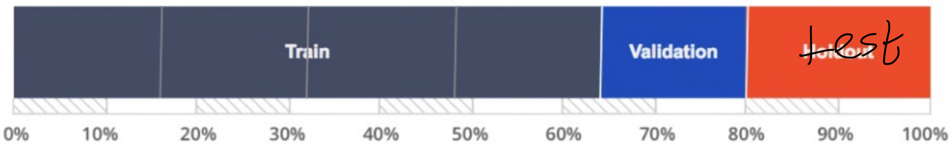
Ex3

Ex4: grid: 

* Technically you, as the programmer fit the validation data to the hyperparameters.

Validation vs test data

* Separate validation and test data to not fool yourself. "You are the easiest person to fool."
* By changing hyperparameters you <u>implicitly</u> fit the model to validation data. If validation = test it's like solving the exam questions in class.



Cross validation

<u>5-fold:</u>

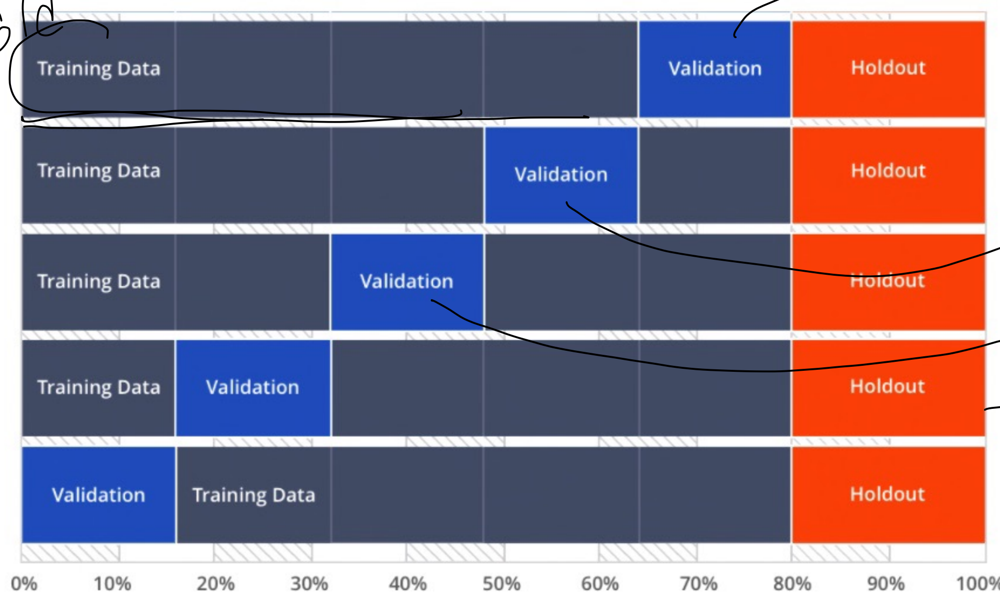
10

K-fold

K times

train

1st fold



→ 95%

→ 94%

→ 93

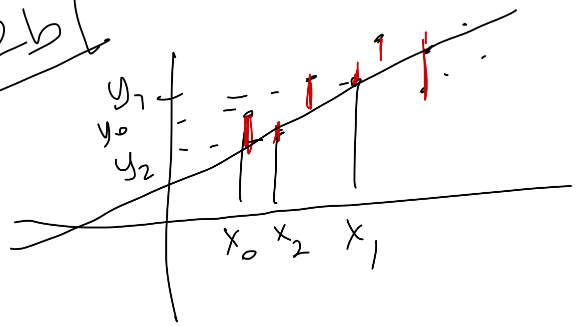
→ 96%

avg $\overline{94.5\%}$

* Divide training data into 5 "folds", keep 1 as validation set. Train model on the remaining 4/5's.
* Repeat using other folds.

<u>Importance:</u> Decrease the bias in accuracy.

Q2b



N: number of data points

x_0, y_0
x_1, y_1
$\vdots$
x_{N-1}, y_{N-1}

input outputs

find $a, b \in \mathbb{R}$

$$\underbrace{a\,\underline{x_i} + b = \underline{y_i}}\ \Big\}\ N \text{ eqn}$$

can be rewritten

$$\underline{\underline{X}}_{N\times 2} \cdot \underbrace{\begin{bmatrix} a \\ b \end{bmatrix}}_{\underline{\gamma}} = \underline{y}_{N\times 1}$$

$$\rightarrow \underline{\underline{X}} \cdot \underline{\gamma} = \underline{y}$$

$$\underline{\underline{X}} = \begin{pmatrix} x_0 & 1 \\ x_1 & 1 \\ x_2 & 1 \\ \vdots & \end{pmatrix}_{N\times 2} \qquad \underline{y} = \begin{pmatrix} y_0 \\ \vdots \\ y_{N-1} \end{pmatrix}$$

Least squares solution to $\begin{bmatrix} a \\ b \end{bmatrix}$:

$$\text{"squared error"} = \sum_{i=0}^{N-1} \Big(\underbrace{\underline{\underline{X}} \cdot \underline{\gamma}}_{\underline{\hat{y}}} - \underline{y}\Big)_{i\ N\times 1}^{2} = (\underline{\underline{X}} \cdot \underline{\gamma} - \underline{y})^T_{1\times N} (\underline{\underline{X}} \cdot \underline{\gamma} - \underline{y})_{N\times 1}$$

Convex in $\underline{\gamma}$ so $\dfrac{\partial \text{ "squared error"}}{\partial \gamma_i} = 0$ gives min or max.

$$\frac{\partial}{\partial \gamma} \underline{\gamma}^T \underline{\underline{X}}\,\underline{\underline{X}}\,\underline{\gamma} - \underline{\gamma}^T \underline{\underline{X}}^T \underline{y} - \underline{y}^T \underline{\underline{X}} \cdot \underline{\gamma} = 0 \Rightarrow \underbrace{-2\underline{\underline{X}}^T \underline{y} - 2\underline{\underline{X}}^T \underline{\underline{X}}\,\underline{\gamma} = 0} \Rightarrow$$

same

$$\underline{\underline{X}}^T \underline{\underline{X}}\,\underline{\gamma} = \underline{\underline{X}}^T \underline{y}$$

$\Rightarrow \underline{\underline{X}}^T \underline{\underline{X}}$ invertible

$$\underline{\gamma}^* = (\underline{\underline{X}}^T \underline{\underline{X}})^{-1} \underline{\underline{X}}^T \underline{y}$$

solution