Fundamentals of Machine Learning (ML)

What is ML? Setting: - Quantities that are easy to observe or measure

"features" X ("covariates")

- Quantities we would like to know but cannot easily observe or measure: "response" Y

("prediction")

$\Rightarrow$ need to define a function $\underbrace{\hat{Y}}_{\text{algorithm's response}} = f(X) \approx \underbrace{Y^*}_{\text{true value of } Y}$

· traditional science: ask a theoretician to derive a formula for $f(X)$

· machine learning: - use a "universal" function family

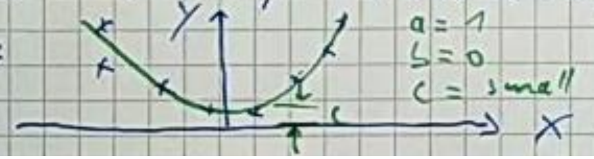
$$F(X;\ \theta\ )$$

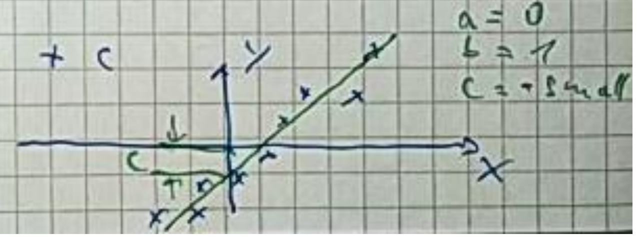
← "theta": additional parameters

by setting θ cleverly $F(X;\theta)$ can represent any $f(X)$ we want $f(X) = F(X;\hat{\theta})$

- step 2: use "training data" to find the best possible $\hat{\theta}$ such that $F(X;\hat{\theta}) \approx Y^*$

- Example: $F(X;\ \theta = (a, b, c)) = a x^2 + b x + c$

Data 1: 

Data 2: 

- probabilistic solutions: sometimes, a deterministic function $\hat{Y} = f(X)$ is not good enough: several solution Y can occur
  - the features contain only incomplete information about Y: "ambiguity"
  - — " — are noisy: "aleatoric uncertainty", "variance"
  - the mapping in reality $Y^* = f^*(X)$ may be non-deterministic ⇒ "aleatoric unc.", "variance"
  - our approximate function is not perfect, similar solutions are also plausible: "epistemic uncertainty", "bias"

⇒ solution: replace deterministic function with a <u>conditional probability</u>

$$\hat{Y} = f(X) \implies p(Y \mid X)$$

↑ several solutions/outputs for Y with different probabilities

[* we get back determinism by ~~p(Y|X)~~
$$p(Y \mid X) = \delta(Y - f(X))$$]

- example: Y is a class label; kind of fruit "apple", "nectarine"
  X: size, color
  [8 cm, red] ⇒ $\left\{ \begin{array}{l} p(Y = \text{"apple"} \mid 8\text{cm}, \text{red}) = 1/2 \\ p(Y = \text{"nectarine"} \mid 8\text{cm}, \text{red}) = 1/2 \end{array} \right.$
  [4 cm, yellow] ⇒ "apple" $\quad p(Y = \text{"apple"} \mid 4\text{cm}, \text{yellow}) = 0.9$
  [10 cm, green] ⇒ "apple"

- basic ML workflow:

(1) collect data – training data to determine $\hat{\theta}$
– test data to check if $F(x, \hat{\theta})$ is a good approximation for Y^*

$\Rightarrow$ detect "overfitting"

NEVER USE THE SAME DATA FOR TRAINING AND TEST

perfect interpolation of x but bad "generalization"

no perfect interpolation but good generalization

(2) find $\hat{\theta}$ such that $\hat{Y} = F(X; \hat{\theta}) \approx Y^*$ for the training data

(3) test on the test data if $\hat{Y} = F(X; \hat{\theta}) \approx Y^*$ (now $\hat{\theta}$ is fixed)
if not, throw away solution and start again (ex, use better function family F)

(4) ~~top~~ deployment: run $F(X, \hat{\theta})$ in a routine setting to predict $\hat{Y}$ for new data

Recap: predict response Y from observable features X

- How good can we get? How bad can it be?
- Example: cards Y: color of the card (red or black
  X: value of the card (ace, king, ...)
- case 1. the feature is perfectly informative for the response
  "best" ⇒ Y is always correctly predicted,
  but: this almost never happens in practice
- Case 2: we have no features, or features are unrelated to response
  "worst" ⇒ learning is impossible, but we can still use our prior knowledge:

  case 2a) $p(Y = \text{red}) = p(Y = \text{black}) = 0.5$

  ⇒ you can do no better than guessing

  [observation: You will be right 50% of the time, sounds not too bad!]

  case 2b) $p(Y = \text{red}) < p(Y = \text{black})$

  $\frac{1}{3} < \frac{2}{3}$

  ⇒ always go with the more probable answer

  ⇒ you will be right $\underset{66\%}{p(Y = \text{black})}$ of the time

  [if probability of majority answer is very high, you can achieve impressive accuracy by doing nothing]

• case 3: features give some information about the response

"typical" ⇒ model what happens with conditional probabilities:

3 kings (1 red / 2 black), 3 queens (2 red / 1 black)

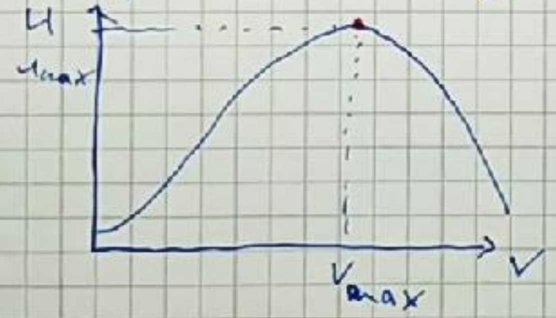
$$p(Y=r \mid X=\text{king}) = 1/3$$
$$p(Y=b \mid X=\text{king}) = 2/3$$
$$p(Y=r \mid X=\text{queen}) = 2/3$$
$$p(Y=b \mid X=\text{queen}) = 1/3$$

if you learn, that "X = king", what's the best answer

⇒ choose the most probable answer, given the evidence

mathematically: $\hat{Y} = \arg\max\limits_{Y} p(Y \mid X)$

[Notation: arg max operator: suppose $u = f(v)$



maximum: (v_{max}, u_{max})

$$u_{max} = \max_{v} f(v)$$

$$v_{max} = \arg\max_{v} f(v)$$]

if $X = \text{king}$: $\hat{Y} = \arg\max\limits_{Y \in \{r, b\}} p(Y \mid X=\text{king})$

$$= \arg\max \{ \underbrace{p(Y=r \mid X=\text{king})}_{\frac{1}{3}} , \underbrace{p(Y=b \mid X=\text{king})}_{\frac{2}{3}} \} \quad \left(\tfrac{1}{3} < \tfrac{2}{3}\right)$$

$$= b$$

recap: machine learning defines generic function families, which depend on a parameter set Θ

$$Y = F(X ; \Theta)$$

$\Rightarrow$ adjust Θ such that the predicted response is as close to the truth as possible:

$$\text{choose } \hat{\Theta} \text{ such that } \hat{Y} = F(X ; \hat{\theta}) \approx Y^*$$

(measure by a "loss function": if small, $\hat{Y}$ is close to Y^*)

- applies to deterministic respons, i.e. for each X, we return a hard decision about Y (hopefully, the best possible)
- in many situations, the uncertainty about the response should be made explicit

$\Rightarrow$ ~~learn a [illegible]~~ define a generic family of conditional probabilities

$$P_\Theta(Y \mid X) \text{ or } p(Y \mid X ; \Theta)$$

$\Rightarrow$ adjust Θ such that the predicted probabilities for each possible respons Y are as close as possible to the true probabilities ("good calibration of the probability")

$$\text{choose } \hat{\Theta} \text{ such that } \hat{p}(Y \mid X ; \hat{\theta}) \approx p^*(Y \mid X)$$

example: ~~wea~~ rain forecast: $Y = 1$: if rains tomorrow
$ = 0$: if does not rain tomorrow

if our model says $\hat{p}(Y = \text{rain} \mid X) = 0.8$, $\hat{p}(Y = \text{dry} \mid X) = 0.2$

than it should actually rain on 80% of the days with similar X

- predicting a probability is strictly more general than making a deterministic prediction, because probabilities can always be reduced to a deterministic result, but not vice versa

examples:
- choose the most probable answer as hard decision "maximum a-posteriori" response (MAP)

$$\hat{Y} = \arg\max_{Y} p_{\hat{\theta}}(Y \mid X)$$

- choose the expected value (weighted average prediction)

$$\hat{Y} = \mathbb{E}_{Y \sim p_{\hat{\theta}}(Y \mid X)}[Y] = \begin{cases} \sum_{Y} Y \cdot p_{\hat{\theta}}(Y \mid X) & \text{if } Y \text{ is discrete} \\ \int Y \cdot p_{\hat{\theta}}(Y \mid X)\, dY & \text{if } Y \text{ is real} \end{cases}$$

- choose the median of the response

$$\hat{Y} = Y \text{ such that } p(Y < \hat{Y} \mid X) = p(Y > \hat{Y} \mid X)$$

For unimodal symmetric distributions $p(Y \mid X)$, all three methods give the same answer $\hat{Y}$, e.g. Gaussian

For skewed distributions, all three are different, e.g. Poisson

For multi-modal distributions (with modes that are almost equally high), reducing to a single $\hat{Y}$ is dangerous: miss alternative outcomes, get impossible outcomes

Notation & terminology

- our observations consist of a set of <u>instances</u>

  e.g. weather forecast : each instance may be a day

  game : each round is an instance

- we assume the the training set contains N instances,

  indexed by $i \in 1, \dots, N$ (or i', i'')

  the corresponding features and response are X_i, Y_i

- we assume that X is a tuple or vector of D features

  indexed by $j \in 1, \dots, D$ (or j', j'')

  X_{ij} is feature j of instance i.

  Y is a tuple or vector of D' responses indexed by j

  <u>but</u> most of the time we consider the case $D' = 1$, only one response

- types of variables (features and response)
  - continuous $\hat{=}$ real-valued
  - discrete :
    - ordinal : values are ordered small < medium < big
    - categorial : values not ordered $\underbrace{\text{apple, orange, nectarine}}_{C=3}$

$\Rightarrow$ two fundamental types of machine learning problems

Y_i is continuous $\Rightarrow$ regression problem

Y_i is discrete $\Rightarrow$ classification problem

- if Y is discrete : C is the number of possible class labels / categories

  indexed by $k \in 1, \dots, C$

- types of training sets:
  - supervised learning: for each instance in TS, we know the features <u>and</u> the response: (X_i, Y_i)
    [only when we apply the trained algorithm in the real world is Y_i unknown]
    ⇒ training will adjust $\hat{\theta}$ such that $\hat{Y}_i = F(X_i; \hat{\theta}) \approx Y_i$ for all $i \in 1, \dots, N$
    - when we can observe Y_i in the training set, why don't we do the same in the real application? Why do we need learning at all?
      ⇒ Y_i is often only known in a "laboratory setting", not in the "wild"
      - measuring Y may require controlled conditions and/or very expensive equipment, not practical in the "wild"
      - Y can be determined by a human expert (radiologist), who may not be available all the time or too expensive
      - measuring Y may be destructive, e.g. crash test for car
      - Y may only become known in hind sight, e.g. weather, games
  - ~~super~~ unsupervised learning: no Y are known in TS
    ⇒ algorithm needs to figure out what interesting Y's might be and what their true values are, e.g. science
    existing solutions are quite limited - "data mining"
  - weakly supervised learning: getting a fully "annotated" training set (Y_i is known for every i) is <u>very expensive</u> ⇒ looks for tricks to use less perfect TS