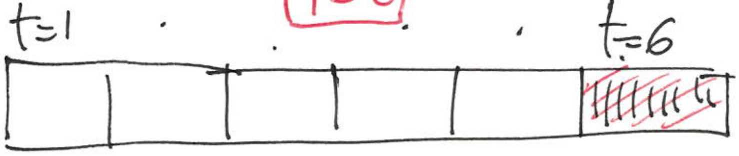
# K-NN : K-nearest neighbours as a predictor.

## T-fold cross validation

$\boxed{T=6}$




Split into T-folds

All samples $S = \{\vec{x}^{(i)}, y^{(i)}\}$ $\quad i=1,\dots,N$

$$S = \bigcup_{t=1}^{T} S_t \qquad |S_t| = \frac{|S|}{T}$$

Equal size, properly sampled

$$\bar{S}_t = S - S_t \hookrightarrow \boxed{\hat{f}_{\bar{S}_t;k}} \qquad t=1,\dots,T$$

Different model for each $\bar{S}_t$

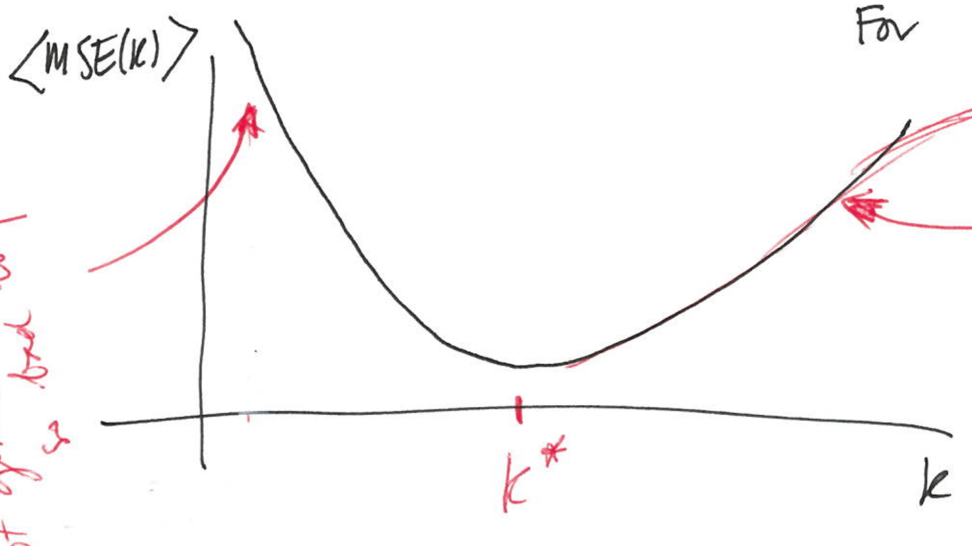
Predict $S_t$ with the model obtained from $\bar{S}_t$ $\Rightarrow$

$$\sum_{i\in S_t}\left[\hat{f}_{\bar{S}_t}(\vec{x}^{(i)}) - y^{(i)}\right]^2 = MSE_t$$

$$\langle MSE(k)\rangle = \frac{1}{T}\sum_{t=1}^{T} MSE_t$$

This is a characteristic of how well the model predicts out-of-sample

For KNN




k too small: Not generalisable to predict off-sample

k too large: Predictor too generic

Typical T ~ 5, 10 machine learning

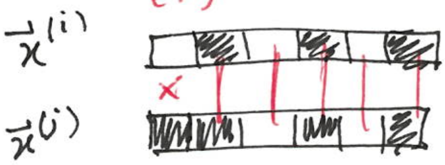
If T = N, $\boxed{\text{LOO leave-one-out}}$ — [ • Good estimate • computationally expensive • Overkill.

k-NN can also be applied to discrete descriptors

⇒ If $\vec{x}^{(i)} = \{0,1\}^P$ , $y^{(i)} \in \mathbb{R}$

Hamming distance : $d_H(\vec{x}^{(i)}, \vec{x}^{(j)})$

e.g.




$$\text{distance} = 1 \Bigg| \sum_{m=1}^{P} I(x_m^{(i)} \neq x_m^{(j)})$$

Distance is the key :
↳ Other methods use distance ⇒ kernels to have local models

The rest is the same
$N_k(\vec{x}^{in})$ and $\hat{y} = \frac{1}{k} \sum_{i \in N_k(\vec{x}^{in})} y^{(i)}$

# Classification problem with KNN

$y^{(i)} \in C_i$

$\vec{x}^{(i)}$ , $\boxed{y^{(i)} \text{ discrete.}}$ $\quad i = 1, \dots, N$

$C_q = \{c_1, \dots, c_q\}$

$$\left\{ \begin{array}{l} \vec{x}^{(i)} \in \mathbb{R}^p \\ \vec{x}^{(i)} \in \{0,1\}^p \end{array} \right\} \quad d(\vec{x}^{(i)}, \vec{x}^{(j)})$$

As long as we have the distance:

same ✓ (1) Choose $K$ (number of neighbors)

same ✓ (2) For $\vec{x}^{in}$ find $N_K(\vec{x}^{in})$

Predictor is different for classification: new → (3) $\hat{y} \in C_q$ e.g. by majority rule.


$k=4 \quad \hat{y}(\vec{x}^{in}) \rightarrow \triangle$

$k=7 \quad \hat{y}(\vec{x}^{in}) \rightarrow \square$

Other predictors are used, including those that give a probabilistic output:

$\forall\, i \in N_K(\vec{x}^{in})$ compute $\vec{Q}(\vec{x}^{in}) = \frac{1}{K} \sum_{i \in N_K(\vec{x}^{in})} I(y^{(i)} \in \vec{C}_q)$

$$I(\cdot) = \begin{cases} 1 & \text{if true} \\ 0 & \text{otherwise} \end{cases}$$

$$\vec{Q}(\vec{x}^{in}) = \begin{pmatrix} \text{\# of points in } C_1 \\ \vdots \\ \text{\# of points in } C_q \end{pmatrix}_{q \times 1} \frac{1}{K}$$

$\Downarrow$

Each component $Q_i$ : Probability of $\vec{x}^{in}$ belonging in class $C_i$

---

Standard KNN applies an argmax to this probability:

$$\hat{y} = \arg\max_q \vec{Q}(\vec{x}^{in})$$

$\Uparrow$

Chooses the class with the maximum probability.

# Global models for classification.

$\vec{x}^{(i)} \in \mathbb{R}^p$ ; $y^{(i)} \in \{0, 1\}$ $\quad i = 1, \dots, N$

Binary.

Classic model: Logistic ~~"regression"~~ $\hookrightarrow$ classification.

Motivation: log odds are linear combination of # descriptors

$$\beta_0 + \vec{x}^T \cdot \vec{\beta} = \log \frac{P(y=1)}{P(y=0)} = \log \frac{\mathbb{P}(y=1)}{1 - \mathbb{P}(y=1)}$$

$$\vec{x}^T \cdot \vec{\beta}_{(p+1)\times 1}$$

$$\frac{\mathbb{P}(y=1)}{1 - \mathbb{P}(y=1)} = e^{\vec{x}^T \cdot \vec{\beta}} \Rightarrow \mathbb{P}(y=1) = \underbrace{\frac{1}{1 + e^{-\vec{x}^T \cdot \vec{\beta}}}} = h_{\vec{\beta}}(\vec{x})$$

Bernoulli variable with probability of success $\mathbb{P}(y=1)$

$$\hookrightarrow P(Y = y \mid \vec{x}, \vec{\beta}) = h_{\vec{\beta}}(\vec{x})^y \left(1 - h_{\vec{\beta}}(\vec{x})\right)^{1-y}$$

---

If we assume independence in our samples; we can factorize probabilities to get:

$$\mathbb{P}\left(\{y^{(i)}\} \mid \{\vec{x}^{(i)}\}, \vec{\beta}\right) = \prod_{i=1}^{N} h_{\vec{\beta}}\left(\vec{x}^{(i)}\right)^{y^{(i)}} \left(1 - h_{\vec{\beta}}\left(\vec{x}^{(i)}\right)\right)^{1-y^{(i)}}$$

log-likelihood:

$$\mathcal{L} = \sum_{i=1}^{N} y^{(i)} \log h_{\vec{\beta}}\left(\vec{x}^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - h_{\vec{\beta}}\left(\vec{x}^{(i)}\right)\right)$$

$\nabla_{\vec{\beta}} \mathcal{L}$ to be maximized.