K-Nearest Neighbors

$D = \{(x_i, y_i)\}_{i=1}^{N}$ $\quad x_i \in \mathbb{R}$ are features, $y_i \in \{1, \dots, C\}$ are class labels

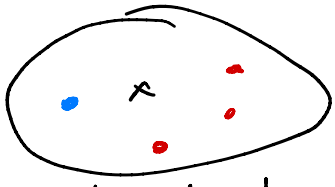
Steps: (1-NN)

1) Define a distance measure (e.g. Euclidean distance)
2) Compute the nearest neighbor for all new data points
3) Label them with the label of their nearest neighbor

} if works for both classification and regression

Results → poor generalization

K-NN classification → labels



Look at multiple nearest neighbors and pick the majority label

$N_K(x)$ → K nearest neighbors of a vector x

$$p(y = c \mid x, K) = \frac{1}{K} \sum_{i \in N_K(x)} \mathbb{I}(y_i = c)$$

$$\mathbb{I}(e) = \begin{cases} 1 & \text{if } e \text{ is true} \\ 0 & \text{if } e \text{ is false} \end{cases}$$

K: nº neighbors

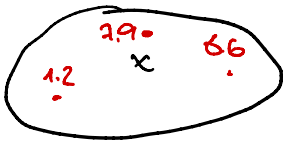
probability of the class label y being class c, given x and K

$$\hat{y} = \underset{c}{\operatorname{argmax}}\, p(y = c \mid x, K)$$ → result is a class, not a probability

$$p(y = c \mid x, k) = \frac{1}{Z} \sum_{i \in N_k(x)} \frac{1}{d(x, x_i)} \mathbb{I}(y_i = c)$$ } weighted

$$Z = \sum_{i \in N_k(x)} \frac{1}{d(x, x_i)}$$

K-NN regression → continuous

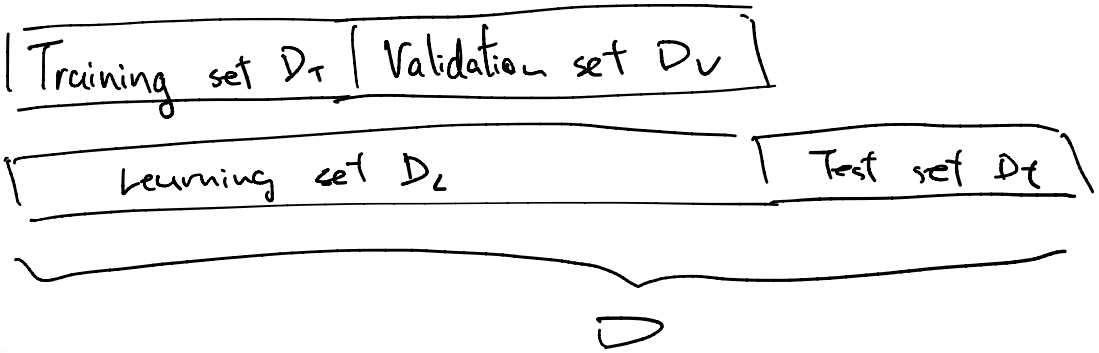


$$\hat{y} = \frac{1}{z} \sum_{i \in N_k(x)} \frac{1}{d(x, x_i)} y_i$$

How to choose k

Goal → generalization

Split the data →



	True condition	
Predicted	$y = 1$	$y = 0$
$y = 1$	TP	FP
$y = 0$	FN	TN

K-NN can be used with various distance measures

L_2 norm → $\sqrt{\sum_i (u_i - v_i)^2}$

L_1 norm → $\sum_i |u_i - v_i|$

L_∞ norm → $\max_i |u_i - v_i|$

Scaling issues → we have to normalize (Data standardization)

$$x_{i,std} = \frac{x_i - \mu_i}{\sigma_i} \quad \text{(z-score normalization)}$$

substract mean, divide variance

- Dimensionality issue → With a lot of features, the space dimensionality is big and the space is becoming very empty, so when a new data point is in the space it has no neighbors near to him. It is not good for high dimension data

Decision Trees

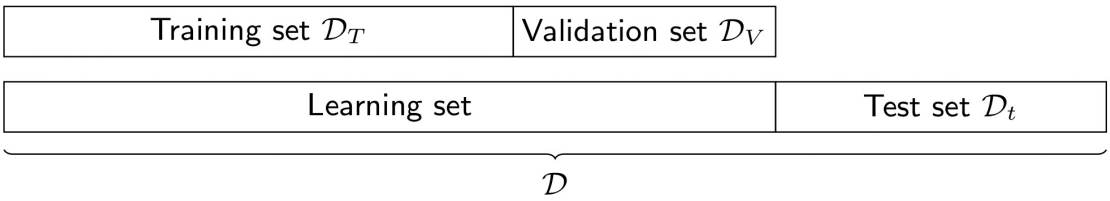
$$p(y=c \mid R) = \frac{n_{c,R}}{\sum_{c_i \in C} n_{c_i,R}}$$

$n_{c,R}$ → nº of datapoints that belongs to class C

$\sum_{c_i \in C} n_{c_i,R}$ → total datapoints

the label of a new unseen sample x is:

$$\hat{y} = \underset{c}{\operatorname{argmax}}\ p(y=c \mid x) = \underset{c}{\operatorname{argmax}}\ p(y=c \mid R) = \underset{c}{\operatorname{argmax}}\ n_{c,R}$$



- build tree from training set $\mathcal{D}_T$,
- predict *validation set* labels $\hat{y}_i$ using the tree,
- evaluate by comparing predictions $\hat{y}_i$ to true labels y_i.
- pick the tree that performs the best on the validation set
- report final performance on the test set

Misclassification rate (error) i_E at node t

$$i_E(t) = 1 - \max_c p(y=c \mid t)$$

The improvement when performing a split s of t into t_R and t_L for $i(t) = i_E(t)$ is given by: $\Delta i(s,t) = i(t) - p_L \cdot i(t_L) - p_R \cdot i(t_R)$

↓

it makes the distribution of labels more pure (mostly having instances of the same class)

issues:

1) No split performed even though combining the two sets would result in perfect classification

2) No sensitivity to changes in class probability (when one split is better than the other, it might have the same i_E)

Solution → criterion $i(t)$ measures how pure the class distribution at a node t is

- maximum → if classes are equally distributed in the node
- minimum → usually 0, if the node is pure
- symmetric

$\pi_c = p(y=c \mid t)$ Entropy → $i_H(t) = -\sum_{c_i \in C} \pi_{ci} \log_2 \pi_{ci}$

Gini Index

measures how often a randomly chosen instance would be misclassified if it was randomly classified according to the class distribution

$$i_G(t) = \sum_{c_i \in C} \pi_{ci} \cdot (1 - \pi_{ci})$$

π_{ci}: probability of picking element

$(1-\pi_{ci})$: probability is misclassified

Gini index > Entropy → no need to compute log

Overfitting → typically occurs when we try to model the training data perfectly, but it means poor generalization

- low training error (possibly 0)
- validation error is comparably high

K-fold Cross-Validation

- Split learning data into K folds
- Use K-1 folds for training and the remaining for evaluation
- Average over all folds to get an estimate

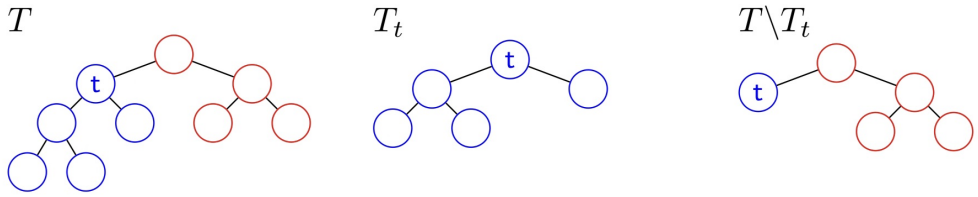
Extreme case LOOCV → train on all but one sample

Stopping criterion

Grow a tree maximally and then (post) prune it

Let T be our decision tree and t one of its inner nodes.

Pruning T w.r.t. t means deleting all descendant nodes of t (but not t itself). We denote the pruned tree $T \backslash T_t$.



- Use validation set to get an error estimate: $\mathrm{err}_{\mathcal{D}_V}(T)$.
- For each node t calculate $\mathrm{err}_{\mathcal{D}_V}(T \backslash T_t)$
- Prune tree at the node that yields the highest error reduction.
- Repeat until for all nodes t: $\mathrm{err}_{\mathcal{D}_V}(T) < \mathrm{err}_{\mathcal{D}_V}(T \backslash T_t)$.

After pruning you may use both training and validation data to update the labels at each leaf.

DT for regression

For regression (if y_i is a real value rather than a class)

- At the leaves compute the mean (instead of the mode) over the outputs
- Use the mean-squared-error as splitting heuristic