

Multimedia Indexing and Retrieval

3. Classical machine Learning for multimedia indexing

Georges Quénot

Multimedia Information Modeling and Retrieval Group



Laboratory of Informatics of Grenoble



Learning

- Machine learning: learning from data.
- Unsupervised learning:
 - Without human intervention,
 - Simple data,
 - Automatic class extraction (clustering).
- Supervised learning:
 - With human intervention (annotation),
 - Labeled (or annotated) data
 - Classification (predefined classes),
 - Regression (continuous values).

Supervised learning

- A machine learning technique for **creating a function** from **training data**.
- The training data consist of pairs of **input objects** (typically vectors) and **desired outputs**.
- The output of the function can be a continuous value (**regression**) or a class label (**classification**) of the input object.
- The task of the supervised learner is to **predict the value** of the function for any valid input object after having seen a number of training examples (i.e. pairs of input and target output).
- To achieve this, the learner has to **generalize** from the presented data to unseen situations in a “reasonable” way.
- The parallel task in human and animal psychology is often referred to as **concept learning** (in the case of classification).
- Most commonly, supervised learning generates a **global model** that helps mapping input objects to desired outputs.

(http://en.wikipedia.org/wiki/Supervised_learning)

Learning a target function

- Target function: $f: X \rightarrow Y$
 $x \rightarrow y = f(x)$
 - x : input object, e.g., color image
 - y : desired output, e.g., class label or image tag
 - X : set of valid input objects
 - Y : set of possible output values

$$f \left(\text{img}_{\text{cat}} \right) = \text{"cat"}$$

$$f \left(\text{img}_{\text{dog}} \right) = \text{"dog"}$$

$$f \left(\text{img}_{\text{car}} \right) = \text{"car"}$$

Set of possible color images:

$$X = \bigcup_{(w,h) \in \mathbb{N}^{*2}} [0,1]^{w \times h \times 3}$$

Set of possible image tags:

$$Y = \{ \text{"cat"}, \text{"dog"} \dots \}$$

Learning a target function

- Target function: $f: X \rightarrow Y$
 $x \rightarrow y = f(x)$

- x : input object, e.g., color image
- y : desired output, e.g., class label or image tag
- X : set of valid input objects
- Y : set of possible output values

$$f \left(\begin{array}{c} \text{[Image of a cat]} \end{array} \right) = \begin{pmatrix} 0.90 \\ 0.04 \\ 0.01 \\ \dots \end{pmatrix} \begin{array}{l} \leftarrow \text{“cat”} \\ \leftarrow \text{“dog”} \\ \leftarrow \text{“car”} \\ \leftarrow \dots \end{array}$$

$$f \left(\begin{array}{c} \text{[Image of a dog]} \end{array} \right) = \begin{pmatrix} 0.07 \\ 0.88 \\ 0.02 \\ \dots \end{pmatrix}$$

$$f \left(\begin{array}{c} \text{[Image of a green car]} \end{array} \right) = \begin{pmatrix} 0.02 \\ 0.03 \\ 0.86 \\ \dots \end{pmatrix}$$

Set of possible color images:

$$X = \bigcup_{(w,h) \in \mathbb{N}^{*2}} [0,1]^{w \times h \times 3}$$

Set of possible tag scores:

$$Y = \mathbb{R}^{|\{\text{“cat”, “dog” ...}\}|} = \mathbb{R}^c$$

Learning a target function

- Target function: $f: X \rightarrow Y$
 $x \rightarrow y = f(x)$
 - x : input object, e.g., image descriptor
 - y : desired output, e.g., class label or image tag
 - X : set of valid input objects
 - Y : set of possible output values

$$f \left(D \left(\begin{array}{c} \text{Image of a cat} \end{array} \right) \right) = \begin{pmatrix} 0.90 \\ 0.04 \\ 0.01 \\ \dots \end{pmatrix}$$

$$f \left(D \left(\begin{array}{c} \text{Image of a dog} \end{array} \right) \right) = \begin{pmatrix} 0.07 \\ 0.88 \\ 0.02 \\ \dots \end{pmatrix}$$

$$f \left(D \left(\begin{array}{c} \text{Image of a green car} \end{array} \right) \right) = \begin{pmatrix} 0.02 \\ 0.03 \\ 0.86 \\ \dots \end{pmatrix}$$

Set of possible image descriptors:

$$X = \mathbb{R}^d \quad (\text{or subset of it})$$

Set of possible tag scores:

$$Y = \mathbb{R}^c$$

D is a predefined and fixed function

from $\bigcup_{(w,h) \in \mathbb{N}^{*2}} [0,1]^{w \times h \times 3}$ to \mathbb{R}^d

Learning from training data

- Training data: $S = (x_i, y_i)_{(1 \leq i \leq I)}$
 - I : number of training samples
- Learning algorithm: $L : (X \times Y)^* \rightarrow Y^X$
 $S \rightarrow f = L(S)$

$$((X \times Y)^* = \cup_{n \in \mathbb{N}} (X \times Y)^n)$$

Y^X : set of all applications from X to Y

- Regression or classification system:
 $y = f(x) = [L(S)](x) = g(S, x)$

Supervised learning

- Target function: $f: X \rightarrow Y$
 $x \rightarrow y = f(x)$
 - x : input object (typically vector)
 - y : desired output (continuous value or class label)
 - X : set of valid input objects
 - Y : set of possible output values
- Training data: $S = (x_i, y_i)_{(1 \leq i \leq I)}$
 - I : number of training samples
- Learning algorithm: $L : (X \times Y)^* \rightarrow Y^X$
 $S \rightarrow f = L(S)$
- Regression or classification system:
 $y = f(x) = [L(S)](x) = g(S, x)$

Two types of functions

- Target function: $f: X \rightarrow Y$
 $x \rightarrow y = f(x)$
 - maps **input objects** to **desired outputs**
 - often determined by a set of **parameters**
 - the function or its parameter are **learnt** from a **training set**
- Learning algorithm: $L: (X \times Y)^* \rightarrow Y^X$
 $S \rightarrow f = L(S)$
 - maps **training sets** to **target functions**
 - often controlled by a set of **hyper-parameters**
 - hyper-parameters may be **tuned** on a **validation set**

Model based supervised learning

- Two functions, “train” and “predict”, cooperating via a Model

- General regression or classification system:

$$y = [L(S)](x) = g(S,x)$$

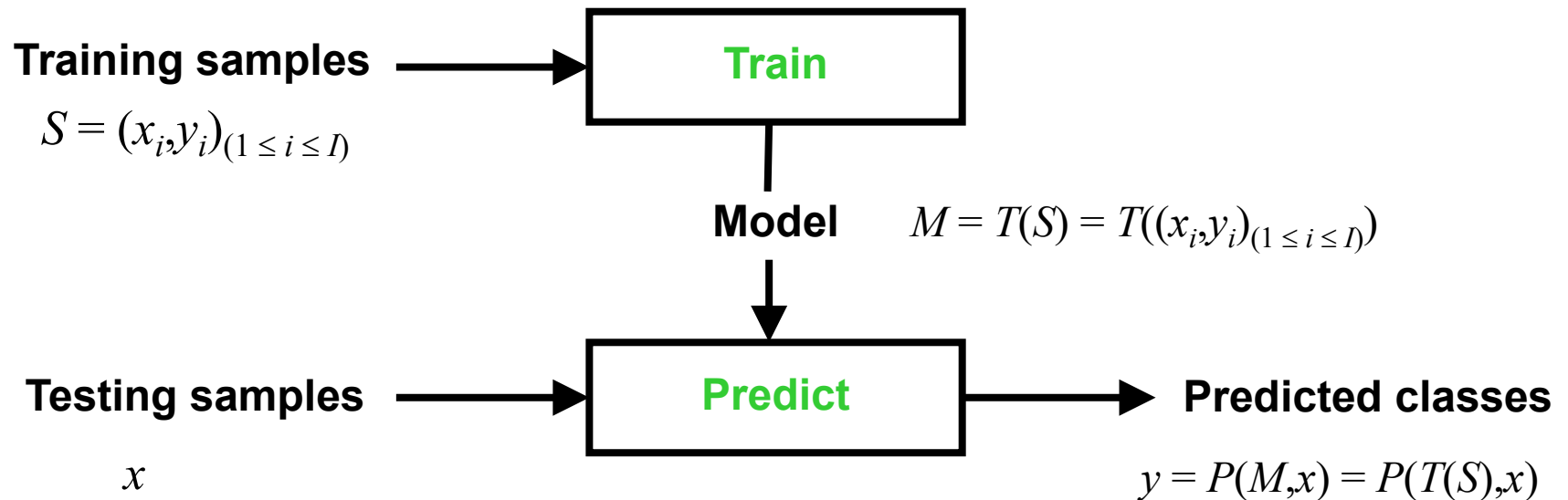
- Building of a model (train):

$$M = T(S)$$

- Prediction using a model (predict):

$$y = [L(S)](x) = g(S,x) = P(M,x) = P(T(S),x)$$

Supervised learning Classification problem



Classification methods

- Gaussian Mixture Models
- Hidden Markov Models
- Decision trees
- Genetic algorithms
- Artificial neural networks
- K-nearest neighbor
- Linear discriminant analysis
- Support vector machines
- Minimum message length
- And many more.

k nearest neighbors (k -NN)

- No training : $M = T(S) = S$ ($T = \text{identity}$)
- Compute the distances from the unknown sample x to all the training samples x_i ,
- Select the k closest x_i ,
- Compute the class of x from the classes of the closest x_i 's:
 - $k = 1$: the class of x is the class of the closest x_i ,
 - k is odd and there are only two classes : majority vote.
- k -NN is a non linear classifier and can easily model classes with very irregular shapes,

k nearest neighbors (k -NN)

- 1-NN is a simple and quite often excellent classifier, it is often chosen as a baseline for comparison between systems,
- 3-NN is more robust against isolated outliers,
- Improvement: weight class values according to the (inverse) distance to the query point
- May be slow for classification because of the need to compute the distances with all the training samples
- However a single NN search may be performed for many classifications at once (multi-label problem)
- May be used for indexing (off-line) or for search (on-line, “similarity search”)

Computation of distance for k-NN

- Euclidian distance, angle between vectors,
- Comparison between a query vector to all the vectors in the database (no pre-selection),
- “Small” number of dimensions (< 10) : clustering techniques, hierarchical search,
- “Medium” number of dimensions ($\sim 10+$) : methods based on space partitioning,
- “Large” number of dimensions ($\gg 10$) : no known method faster than a full linear scan,
- Reduction of the number of dimensions by Principal Component Analysis.
- Approximate Nearest Neighbors: LSH

Locality Sensitive Hashing (LSH)

- Hashing: store many data samples into a table of fixed length; data placed into “buckets”
- “Regular” Hashing: avoid collision for faster access, polynomial and multiples XOR functions; any type of data
- Locality preserving hashing: favor collisions of “close” samples into the same buckets; data from highly dimensional Euclidean space, multiple projection functions

LSH: Multiple projection functions

- Set of random directions
- Projection on the axes → one component per direction
- Split values on axes according to a data distribution (two, four, eight ... intervals)
- One or more bits per direction (generally one)
- Concatenation for producing the bucket index
- Multiple projections: matrix vector multiplication

LSH: Use of multiple tables

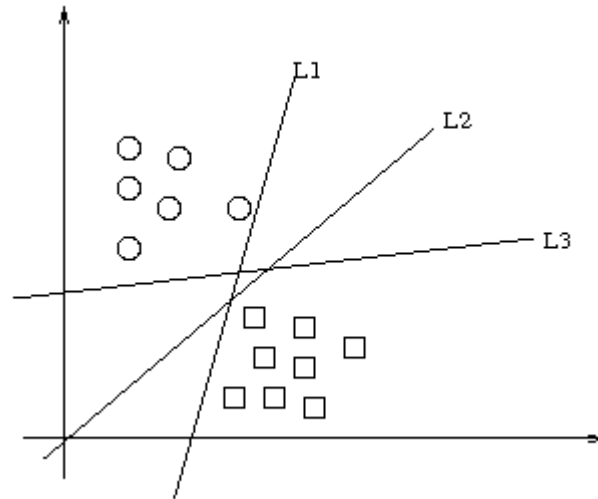
- Build many LSH tables
- For each table, select all the test samples that fall in the same bucket than the query sample
- Compute the Euclidean distance only for those samples
- Sort the test samples according to the Euclidean distances
- Euclidean distances are not approximate but some samples close to the query may not fall in the selected buckets
- The size and number of tables must be chosen so that enough and not too many samples are found for a query

LSH: Use of hamming distance

- Build binary codes (bucket index) as for one LSH table
- Hamming distance: number of bit locations in which the binary values differ: bitwise XOR followed by a count on 1 bits; modern processors have this as a single instruction
- Compute the Hamming distance between the query and all test samples: much faster than Euclidean distance
- Select samples with closest Hamming distance
- Compute the Euclidean distance only for those samples
- Similar to multiple tables from there

Support Vector Machines (SVM)

- Empirical risk minimization
- Linear classifier with maximum margin



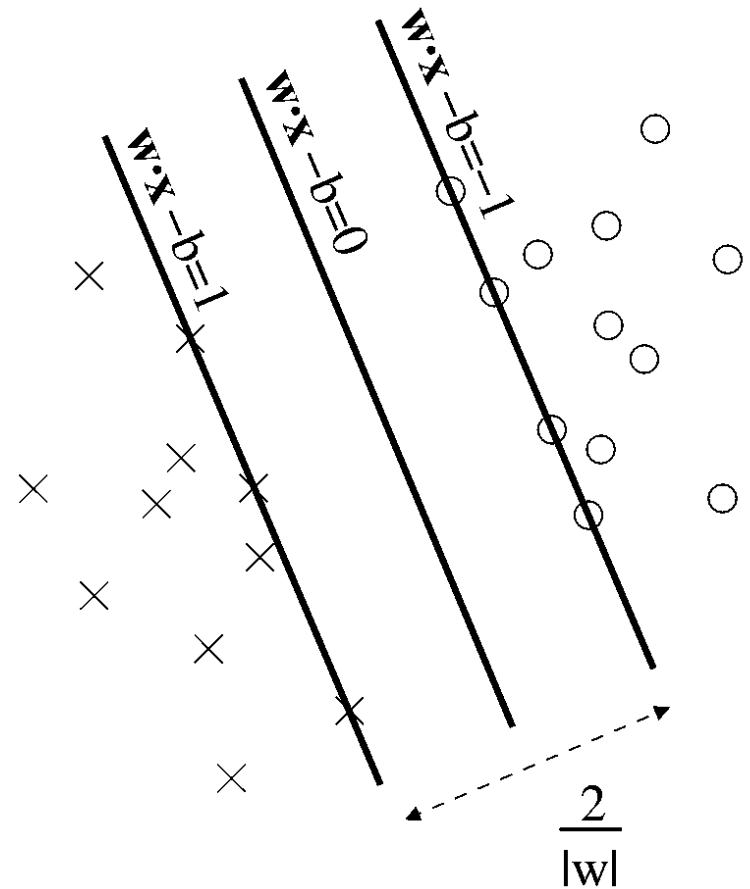
- The “kernel trick” permits non linear classification also with maximum margin and minimum empirical risk

SVM linear classification

- Maximum-margin hyperplanes for a SVM trained with samples from two classes.
- Samples along the hyperplanes are called the support vectors.
- The separated hyperplane is defined by:

$$w^T \cdot x - b = 0$$

- The margin is $2/|w|$



SVM linear classification

- If the data is linearly separable:

$$\text{if } y_i = -1 : w^T \cdot x_i - b \leq -1 \quad \text{if } y_i = +1 : w^T \cdot x_i - b \geq 1$$

- This can be rewritten as:

$$y_i \cdot (w^T \cdot x_i - b) \geq 1$$

- SVM problem primal form:

$$\text{Minimize: } \frac{1}{2} \|w\|^2 \quad \text{subject to: } y_i \cdot (w^T \cdot x_i - b) \geq 1, \quad 1 \leq i \leq n.$$

- SVM problem dual form: $w = \sum_{i=1}^n \alpha_i y_i x_i$

$$\text{maximize: } \sum_{i=1}^n \alpha_i - \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j x_i^T x_j \quad \text{subject to } \alpha_i \geq 0$$

α_i 's are non zero only for the support vectors.

SVM linear classification

- Soft margin, primal form:

$$y_i \cdot (w^T \cdot x_i - b) \geq 1 \quad \rightarrow \quad y_i \cdot (w^T \cdot x_i - b) \geq 1 - \xi_i$$

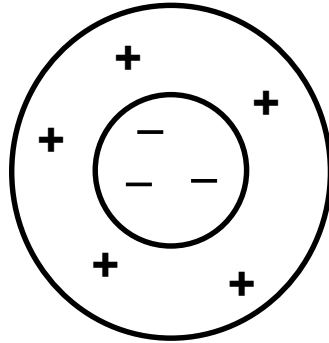
$$\min \frac{1}{2} \|w\|^2 \quad \rightarrow \quad \min \left(\frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \xi_i \right)$$

- Dual form:

$$\alpha_i \geq 0 \quad \rightarrow \quad 0 \leq \alpha_i \leq C$$

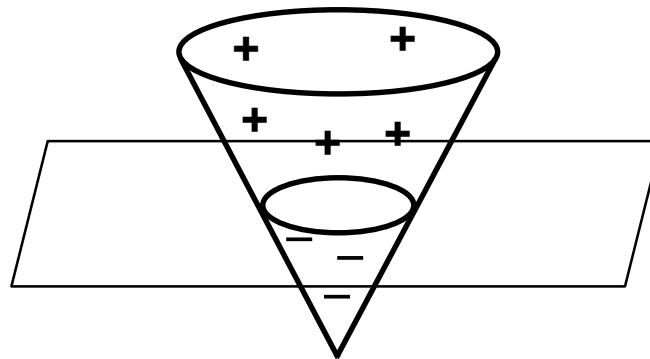
- Allows for “misclassified” samples.

SVM non-linear classification



No linear
separation in
original space

- Kernel trick: projection on a cone (2D \rightarrow 3D):
$$(x, y) \rightarrow \Phi(x, y) = \left(x, y, \sqrt{x^2 + y^2} \right)$$



Linear
separation in
 $\text{im}(\Phi)$ space

SVM non-linear classification

- Decision function:

$$f(x) = \langle w | x \rangle - b = \left\langle \sum_{i=1}^n \alpha_i y_i x_i \middle| x \right\rangle - b = \left(\sum_{i=1}^n \alpha_i y_i \langle x_i | x \rangle \right) - b$$

- Quadratic form maximization:

$$\sum_{i=1}^n \alpha_i - \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \langle x_i | x_j \rangle$$

- Kernel trick: $\langle x_i | x_j \rangle \rightarrow \langle \Phi(x_i) | \Phi(x_j) \rangle = K(x_i, x_j)$
- Φ : possibly non-linear function, does not need to be computed, implicitly defined via the kernel (K) definition, linear separation in the $\text{im}(\Phi)$ space, may be non linear in the original space.

SVM non-linear classification

- Mercer condition : $K(x_i, x_j)$ must be definite positive.
- Common kernels:
 - Polynomial (homogeneous): $K(x, y) = (x \cdot y)^d$
 - Polynomial (inhomogeneous): $K(x, y) = (x \cdot y + 1)^d$
 - Radial Basis Function: $K(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right)$
 - Sigmoid: $K(x, y) = \tanh(\kappa x \cdot y + c)$, for some (not every) $\kappa > 0$ and $c < 0$

SVM summary

- Maximization of the margin for linearly separable data
- Use of a dual form for finding support vectors and coefficients (convex optimization)
- Use of soft margin for “almost” linearly separable data
- Use of the “kernel trick” for non-linearly separable data
- Most commonly used kernel: $K(x, y) = e^{-\gamma \|x-y\|^2} \rightarrow$
 $f(x) = \sum_{i=1}^{I} \alpha_i y_i e^{-\gamma \|x-x_i\|^2} + b$: weighted sum of
Gaussians centered on the support samples (vectors)

Hyper-parameter tuning

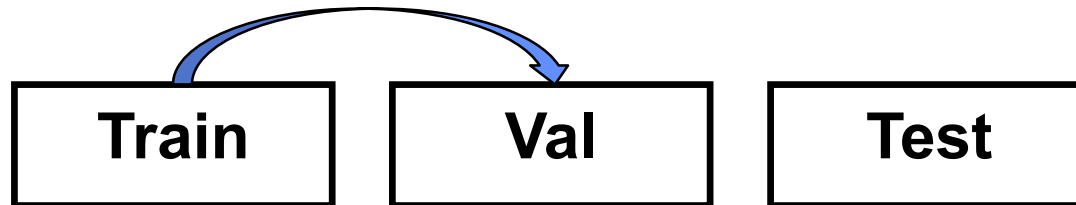
- Parameters:
 - Parameters of the model learnt from training data
 - e.g. values of the support vectors (x_i) and Lagrange coefficients (α_i) in SVMs
- “Hyper”-parameters:
 - Parameters that controls how the model (and “standard” parameters) are learnt
 - e.g. soft margin coefficient (C) in SVMs and the scale parameter in the RBF version (γ)
 - Possibly also class weights
 - Controls “how well” the classification algorithm generalizes from training data, especially the under fit versus over fit compromise

Hyper-parameter tuning, validation set

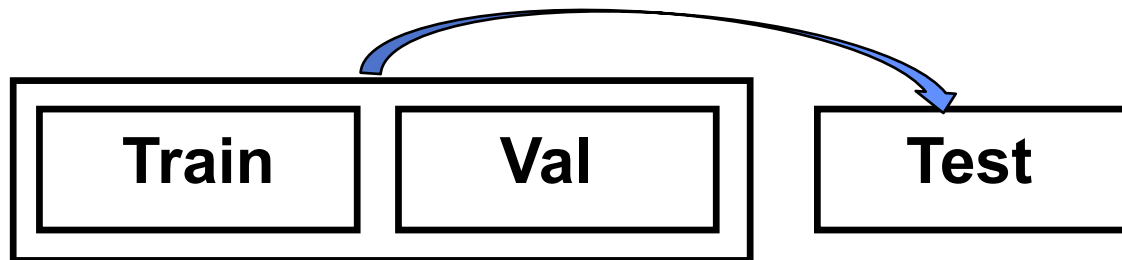
- A dataset used for training cannot be used for evaluation (over-fitting)
- Standard method: use different datasets for training and performance evaluation, each with annotated samples.
- Tuning of hyper-parameters on the test set is bad (over-fitting again)
- Good solution: use three datasets: train, val and test, all with annotated samples
- Train and evaluate several hyper-parameter values between train and val and then apply to test.

Hyper-parameter tuning, validation set

- Parameter tuning: selection of the optimal hyper-parameter combination by training on train and evaluating on val.

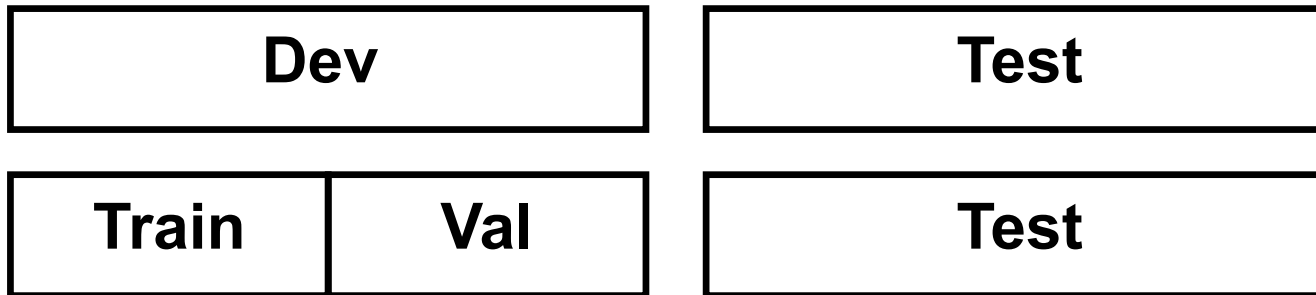


- Actual evaluation: keep the optimal hyper-parameter values, train on train+val and evaluate on test.

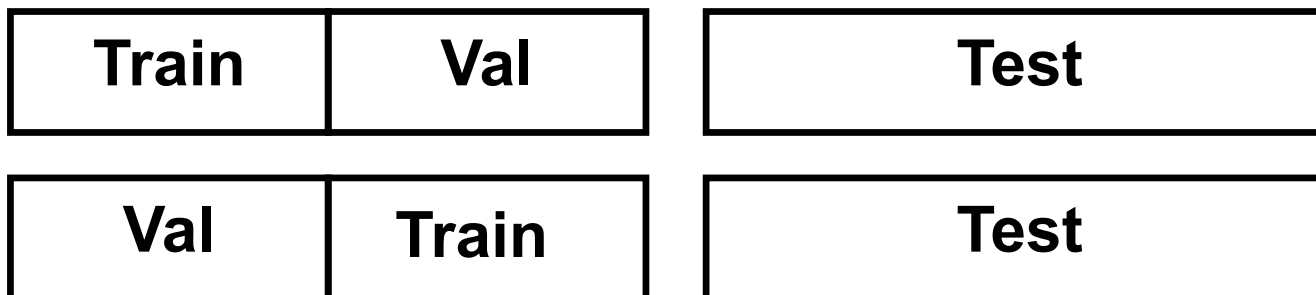


No validation set: split the training set

- Split into two equal parts, use first part as train and second part for validation (“one-fold” cross-validation)

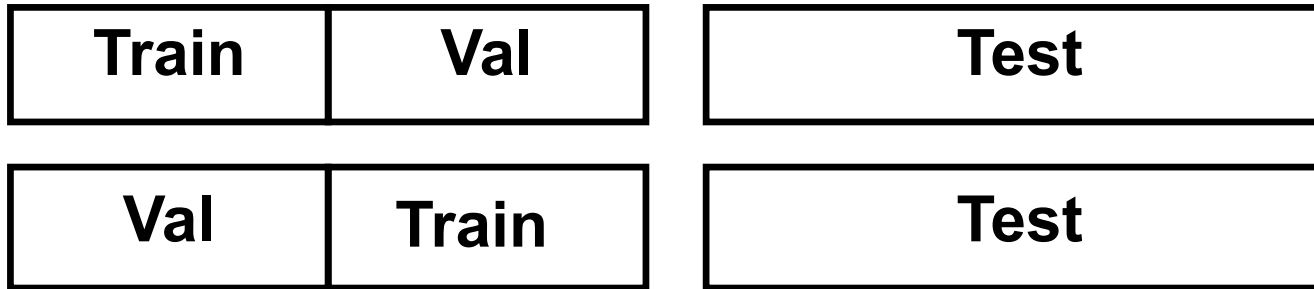


- Two-fold cross-validation



Two-fold cross-validation

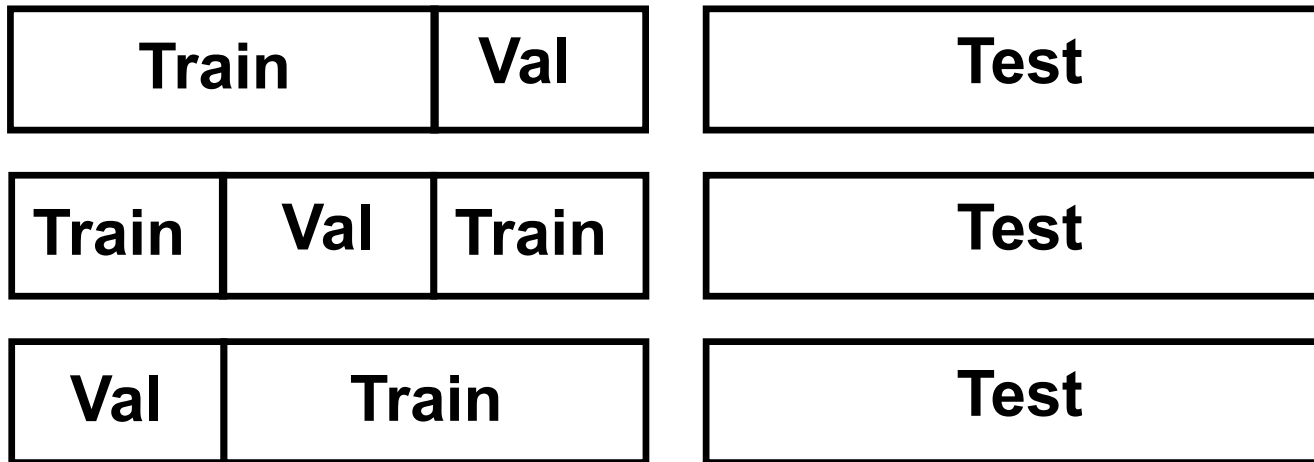
- Use two parts alternatively for training and validation



- The whole development set is used both for training and for evaluation during hyper-parameter tuning
- Tuning is done on MAP (hyper-parameters)
 - Either average the MAP on the two validations
 - Or compute a global MAP on the concatenated scores
- Training is done on half of the development set each time

N-fold cross-validation

- Use N parts of $1/N$ of the development set alternatively for validation and the complement $((N-1)/N)$ for training



- The whole development set is used both for training and for evaluation during hyper-parameter tuning
- Training is done on $(N-1)/N$ of the development set each time, the greater N, the better.

Probabilized output

- SVM scores possibly ranges from $-\infty$ to $+\infty$
- Probabilities are expected to range from 0 to 1
- Sigmoid transform: $p(\text{score}) = 1/(1+e^{(A*\text{score}+B)})$
- Additional hint: among the samples within a small interval around p , a fraction of about p would have positive labels
- Platt's (1999) method: learn A and B by cross-validation to optimally satisfy the above hint
- Probability normalized outputs better for late fusion
- Linear SVM + sigmoid normalization \sim logistic regression