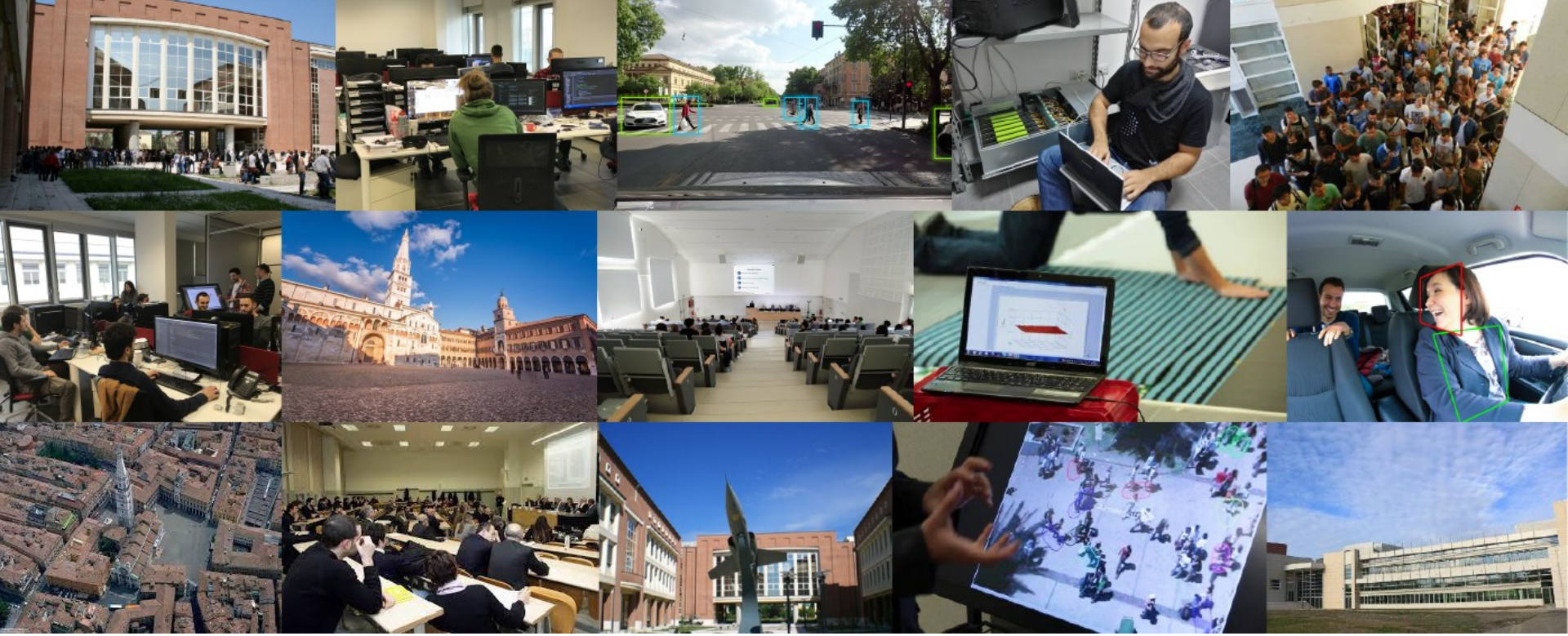


MACHINE LEARNING
- Ensemble e Model selection-

AImage Lab

UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Ensemble

AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Teorema della giuria di Condorcet

AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Confronto tra:

1. Una giuria composta da un solo giudice (“monocratica”) con una probabilità *q* di sbagliare e *p = 1 - q* di emettere la sentenza giusta
2. Una giuria composta da *n* giudici che prende decisioni a maggioranza

Ipotesi:

- Ogni giudice è “intelligente” quanto quello monocratico, ovvero ha la stessa probabilità *p* di successo (prendere la decisione corretta)
- Le decisioni dei giudici sono *indipendenti* l’una dalle altre
- $p > ½$ (*p* è maggiore dello “chance level”)

Condorcet ha dimostrato che:

1. La giuria con *n* giudici ha una probabilità di successo strettamente maggiore di quella monocratica
2. Con *n* che tende ad infinito, la probabilità d’errore della giuria con *n* giudici tende a 0

Ensemble

L'**ensemble learning** è un paradigma di machine learning in cui più modelli (detti **weak learner** o **base estimator**) sono addestrati per risolvere lo stesso task e poi combinati in un unico meta-modello (detto **strong learner** o **ensemble model**) per ottenere prestazioni (sperabilmente) migliori.

L'idea base deriva dal teorema della giuria di Condorcet e il problema principale consiste nell'ottenere *M* modelli (e.g., *M* classificatori) che siano *realmente indipendenti*, dato che normalmente sono addestrati tutti sullo stesso dataset

Bagging: training

Il Bagging è uno dei principali metodi di ensemble e consiste nell'addestrare *M* weak learner di tipo *omogeneo* (i.e., stessa classe di modelli), ognuno su un sottinsieme random dei dati, che ad inference time verranno combinati con un meccanismo di voto (classificazione) o di media (regressione). Ese.:

- Dato un dataset di training T di cardinalità n, estraggo da T un sottoinsieme T_1 in maniera random ($|T_1| = m < n$)
- Su T_1 addestro un classificatore C_1
- Da T estraggo un secondo insieme T_2 ($|T_2| = m$). Siccome l'estrazione dei sample è casuale con «reinserimento» («replacement»), l'intersezione tra T_1 e T_2 potrebbe essere non vuota
- Su T_2 addestro un secondo classificatore C_2
- …
- Continuo fino ad ottenere M classificatori
- C_1, …, C_M sono tutti dello stesso tipo (e.g., tutti Decision Tree oppure tutti k-NN, o tutti SVM, ecc.), ma i modelli addestrati differiscono tra di loro perché sono stati addestrati su sottoinsiemi diversi dei dati
- La (parziale) *indipendenza* dei weak learner, quindi, è data dalla differenza tra T_1, …, T_M

Bagging: inference

- A inference time, uso il feature vector $\boldsymbol{x}$ per calcolare $y_1 = C_1(\boldsymbol{x}), \ldots, y_M = C_M(\boldsymbol{x})$
- In output fornisco la classe y_i più votata

- In caso di regressione, il metodo è identico, con la differenza che, a inference time, fornirò in output la media tra i valori numerici $y_1, \ldots, y_M$ calcolati da $C_1(\boldsymbol{x}), \ldots, C_M(\boldsymbol{x})$

Il Bagging può portare ad un miglioramento rispetto all'uso di un singolo classificatore/regressore, *ma la cosa non è automatica*

Infatti, ogni weak learner è addestrato con un numero di sample $m < n$, e, in alcuni casi, un unico $C()$, addestrato su tutto T, potrebbe portare a risultati migliori

Bagging: Random Forest

AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Le Random Forest sono un tipo particolare di Bagging il cui modello di weak learner è il Decision Tree

Empiricamente, i Decision Tree si prestano bene ad essere usati all'interno di una Random Forest

Probabilmente il motivo è dovuto al fatto che la struttura di un Decision Tree è, in genere, molto sensibile al dataset usato per addestrarlo, ed *M* Decision Tree diversi compensano la tendenza di ognuno di loro verso l'overfitting

Stacking: training

Un altro importante metodo di Ensemble è lo Stacking, in cui si addestrano *M* weak learner *eterogenei* (classi di modelli diversi) sullo stesso dataset. Dopodicè si addestra un «meta-modello» (o «final estimator») che decide come «aggregare» (o «fondere») le decisioni degli *M* weak learner

Esempio (con $M = 3$) e un task di regressione:

- Dato $T = \{(\boldsymbol{x}^{(1)}, y^{(1)}), \ldots, (\boldsymbol{x}^{(n)}, y^{(n)})\}$, scelgo $M = 3$ classi di modelli, e.g., SVR, Decision Tree e k-NN
- Su tutto T addestro una SVR, ottenendo $C_1()$
- Su tutto T addestro un Decision Tree, ottenendo $C_2()$
- Su tutto T addestro un k-NN, ottenendo $C_3()$

Stacking: training

- A questo punto costruisco un nuovo dataset con delle feature, per così dire, «arricchite» dalle predizioni di $C_1()$, …, $C_3()$. Ad esempio:
- Dato $(\boldsymbol{x}^{(i)}, y^{(i)})$ in T, $\boldsymbol{x}^{(i)} = [x_1, x_2, x_3, x_4]$ è un vettore di $d = 4$ feature
- Ese.: $\boldsymbol{x}^{(i)} = [2, 3.5, -2, 1]$
- Do $\boldsymbol{x}^{(i)}$ in input a $C_1()$, …, $C_3()$ e ottengo 3 predizioni diverse per la variabile target:
  - $p_1 = C_1(\boldsymbol{x}^{(i)})$,
  - $p_2 = C_2(\boldsymbol{x}^{(i)})$,
  - $p_3 = C_3(\boldsymbol{x}^{(i)})$
- Ese:
  - $p_1 = 0.5 = C_1([2, 3.5, -2, 1])$,
  - $p_2 = 1.12 = C_2([2, 3.5, -2, 1])$,
  - $p_3 = -0.88 = C_3([2, 3.5, -2, 1])$
- Adesso costruisco un nuovo feature vector $\boldsymbol{z}^{(i)} = [x_1, x_2, x_3, x_4, p_1, p_2, p_3]$
- Ese.: $\boldsymbol{z}^{(i)} = [2, 3.5, -2, 1, 0.5, 1.12, -0.88]$

Stacking: training

- Ripeto il procedimento per tutti i feature vector $\boldsymbol{x}^{(i)}$ in T, ottenendo un nuovo training set $T' = \{(\boldsymbol{z}^{(1)}, y^{(1)}), \ldots, (\boldsymbol{z}^{(n)}, y^{(n)})\}$
- Notate che ora l'i-esimo training sample è $(\boldsymbol{z}^{(i)}, y^{(i)})$, dove la label di *ground truth* $y^{(i)}$ non è cambiata!
- Infine, usando T', addestro un nuovo regressore di tipologia qualsiasi (e.g., usando nuovamente un modello SVR oppure cambiando e usando, e.g., una linear regression, ecc.)
- Sia $C_f(\boldsymbol{z})$ il meta-modello addestrato su T'
- Usando i feature vector $\boldsymbol{z}$, vuol dire che il meta-modello $C_f(\boldsymbol{z})$ prende in input (anche) le predizioni dei weak learner oltre alle d feature originarie
- Ciò corrisponde ad un «giudice» che impara a prendere decisioni basandosi anche sulle risposte di un gruppo di altri M giudici
- La (parziale) indipendenza nel caso dello Stacking è data dall'eterogeneità dei weak learner

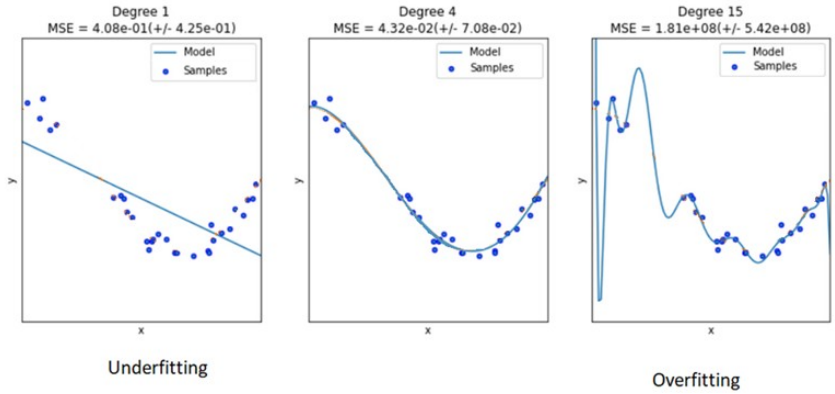
Stacking: inference

- A inference time, dato il feature vector $\boldsymbol{x} = [x_1, x_2, x_3, x_4]$ con d feature, calcolo:
  - $p_1 = C_1(\boldsymbol{x})$,
  - $p_2 = C_2(\boldsymbol{x})$,
  - $p_3 = C_3(\boldsymbol{x})$
- Infine, uso $\boldsymbol{z} = [x_1, x_2, x_3, x_4, p_1, p_2, p_3]$ e restituisco $C_f(\boldsymbol{z})$

Model Selection e hyperparameter tuning

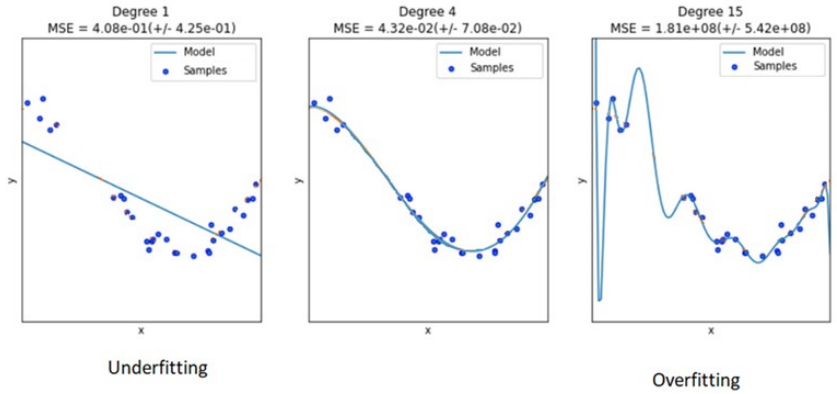
Model selection

- Finora abbiamo visto che, sia per task di regressione che di classificazione, si possono usare diversi (classi di) modelli, eventualmente combinati tra loro con metodi di Ensemble
- Ma qual è il modello migliore?
- Meglio usare una SVM o una logistic regression o un Naïve Bayes non parametrico o…?
- Usare un Ensemble può aiutarmi?
- Come faccio a capire se la classe di modelli scelta è adeguata al task/dataset che ho oppure farà underfitting o overfitting?



Model selection

- Purtroppo, non esiste una risposta generale a queste domande, in quanto la performance di una classe di modelli (*Ensemble compresi*) è fortemente dipendente dal problema e dal dataset specifico
- Questo perchè la “vera” distribuzione dei dati non è nota (cioè come i dati sono distribuiti al di là di quelli che ho in *T*), per cui non posso sapere a priori se una data classe di modelli è adeguata o farà overfitting…
- La scelta in una specifica classe di modelli, detta “model selection”, deve quindi essere fatta empiricamente, utilizzando il validation set e scegliendo il classificatore/regressore con la performance migliore sul validation set



Hyperparameter tuning

AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

- Un ragionamento analogo si applica alla scelta dei valori degli iper-parametri
- Ricordiamoci che gli iper-parametri sono, per definizione, quei parametri del modello il cui valore non può essere calcolato automaticamente
- Ad esempio, il numero di vicini (k) nel k-NN
- Oppure il peso (λ) del termine di regolarizzazione nella LASSO Regression
- O, ancora, il tipo di kernel in una SVM: quale kernel usare per un dato problema?
- E, una volta scelto il kernel, come scegliere i valori degli iper-parametri specifici di quel kernel (e.g., σ per il kernel Gaussiano, oppure a ed m per quello polinomiale, ecc.)?
- Come per la model selection, anche il “tuning” degli iper-parametri va fatto empiricamente usando il validation set, provando con valori diversi e scegliendo i valori corrispondenti alla performance migliore

Model Selection e hyperparameter tuning

AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

In linea generale, il procedimento da usare è il seguente:
Supponiamo di avere un dataset di training *T* e uno di validation *V* e di voler provare *q* modelli diversi (e.g., k-NN, SVM, …)
Come caso particolare, qualcuno di questi modelli potrebbe essere un Ensemble
Per ogni modello C_j ($1 <= j <= q$):

- Faccio delle prove per cercare gli iper-parametri migliori per C_j
- Ad ese., se è C_j un k-NN per la classificazione, potrei provare con tutti i valori dispari di *k* da 1 a 15
- Per ogni valore di *k*, uso *T* per addestrare C_j con *k* fissato, e poi uso *V* per calcolarne la performance
- Scelgo il valore degli iper-parametri corrispondente alla performance migliore su *V*
- Adesso posso passare a considerare gli iper-parametri di C_{j+1}

Una volta fissati gli iper-parametri per $C_1()$, …, $C_q()$, uso nuovamente *V* per calcolare la performance di ogni modello *con i suoi iper-parametri migliori*
Infine scelgo il modello con la performance migliore

Model Selection e hyperparameter tuning

AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

E' estremamente importante usare *V* per le prove e non il dataset di testing (chiamiamolo *D*), altrimenti le scelte che faccio saranno influenzate dagli esempi in *D* e non potrò usare *D* per la valutazione finale

In altri termini, non posso usare *D* per fissare gli iper-parametri e scegliere il modello per lo stesso motivo per cui non posso usare *T* per valutare un modello in fase di testing:

- Se uso *T* per il testing (i.e., *T=D*), un modello che fa overfitting dei suoi parametri su *T* avrà una performance ottima se valutato su *T*
- Se uso *D* per fissare gli iper-parametri (i.e., *V=D*), potrei avere un overfitting degli iper-parametri, e, se *V=D*, non riuscirei ad accorgermene

Quindi è bene tenere a mente che:

- *T* serve per ottimizzare i parametri (training)
- *V* serve per mettere a punto tutte le scelte implementative (iper-parametri, scelta del modello, ecc.)
- *D* serve per la valutazione finale

Scelta del range di valori per un iper-parametro

AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Come faccio a capire quali sono i valori candidati per uno specifico iper-parametro?

Tipicamente esiste un'esperienza passata che si può sfruttare e il modo più semplice per farlo è consultare la libreria di ML che si sta usando

Seguiamo un esempio utilizzando sklearn e una SVM con kernel Gaussiano

Scelta del range di valori per un iper-parametro

AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Il kernel Gaussiano è definito come:

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = exp\left(-\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||^2}{2\sigma^2}\right)$$

Nella documentazione di sklearn viene definito in maniera leggermente diversa:

1.4.6. Kernel functions

The *kernel function* can be any of the following:

- linear: $\langle \boldsymbol{x}, \boldsymbol{x}' \rangle$.
- polynomial: $(\gamma \langle \boldsymbol{x}, \boldsymbol{x}' \rangle + r)^d$, where d is specified by parameter `degree`, r by `coef0`.
- rbf: $\exp(-\gamma \|\boldsymbol{x} - \boldsymbol{x}'\|^2)$, where γ is specified by parameter `gamma`, must be greater than 0.
- sigmoid $\tanh(\gamma \langle \boldsymbol{x}, \boldsymbol{x}' \rangle + r)$, where r is specified by `coef0`.

Quindi ho che γ corrisponde a $1/2\sigma^2$

Grid Search

Sklearn suggerisce di usare una *griglia di ricerca* logaritmica con
$\gamma \in \{10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}, 10^{3}\}$

Attenzione: questi valori sono solo indicativi e in alcuni casi potrebbe essere necessario usare range e valori diversi

In generale, la *Grid Search* consiste nell'iterare training su *T* e performance evaluation su *V* con i valori dell'iper-parametro definiti in una griglia di valori candidati

Nell'esempio del k-NN visto prima, la griglia era: $k \in \{1, 3, 5, 7, \ldots, 15\}$

Grid Search

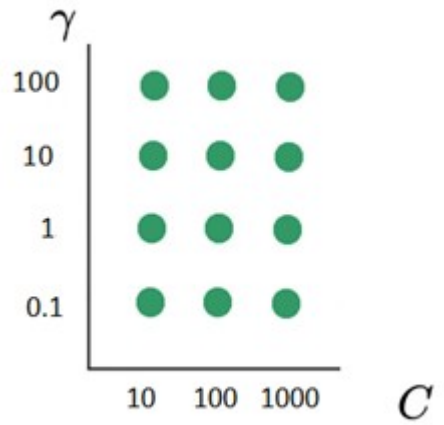
AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Cosa succede se ho più di un iper-parametro?

Ad esempio, oltre all'eventuale kernel (e ai suoi corrispondenti iper-parametri, e.g., γ), in una SVM tipicamente devo fissare anche il valore di "*C*", cioè del peso della regolarizzazione

In tal caso la griglia di ricerca deve esplorare *congiuntamente* tutti i valori candidati per γ **e** per *C*

Ad esempio, supponendo, per semplicità, di voler provare con $\gamma \in \{10^{-1}, 10^{0}, 10^{1}, 10^{2}\}$ e $C \in \{10^{1}, 10^{2}, 10^{3}\}$, la griglia risultante è mostrata nella figura a fianco



Cross-Validation

AImage Lab
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Infine, il ragionamento visto finora può essere replicato usando partizioni diverse tra *T* e *V*

Lo scopo è rendere più robusta la scelta degli iper-parametri e del tipo di modello ed evitare che queste scelte dipendano dagli specifici sample che compongono *V* (e *T*)

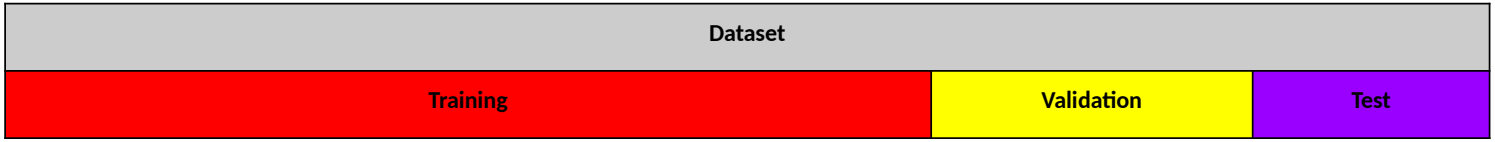
Vediamo alcune delle varianti più comuni di questo processo di partizione tra *T* e *V* e di conseguente validazione

Validation - Holdout

Nel metodo Holdout, il dataset (testing escluso) viene diviso (una tantum) in **training** set e **validation** set

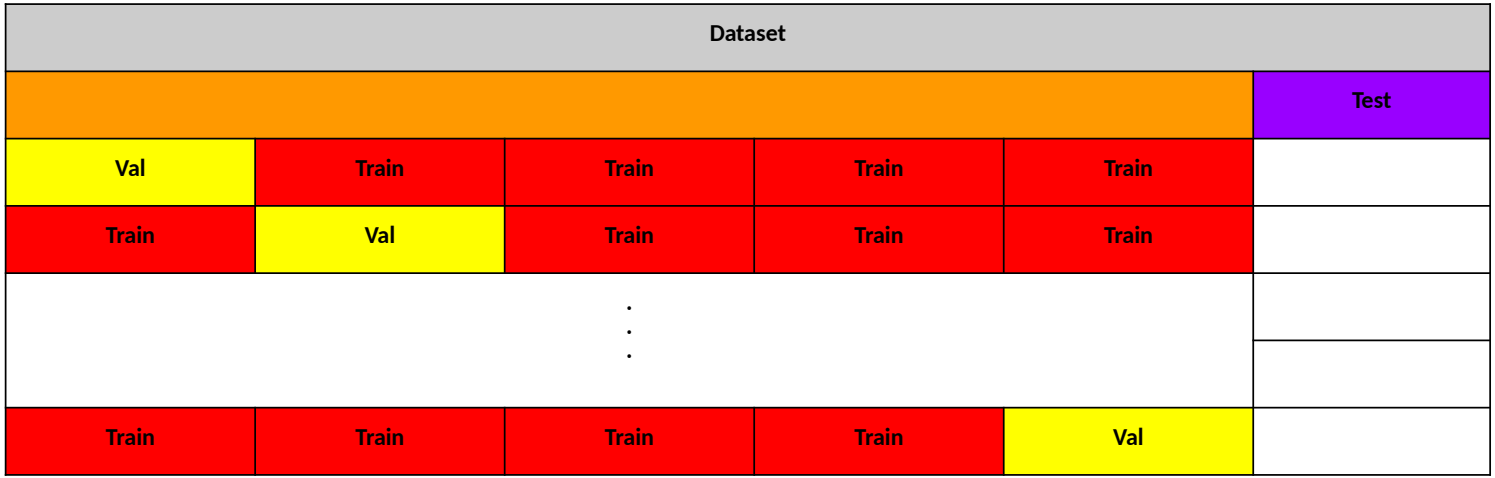
E' il metodo che abbiamo usato finora (anche se non l'abbiamo chiamato così) e corrisponde alla tecnica base di validation

Lo svantaggio di questo approccio è che le prestazioni potrebbero dipendere da quali sample sono finiti nel training set e nel validation set.



Cross-Validation - K-Fold

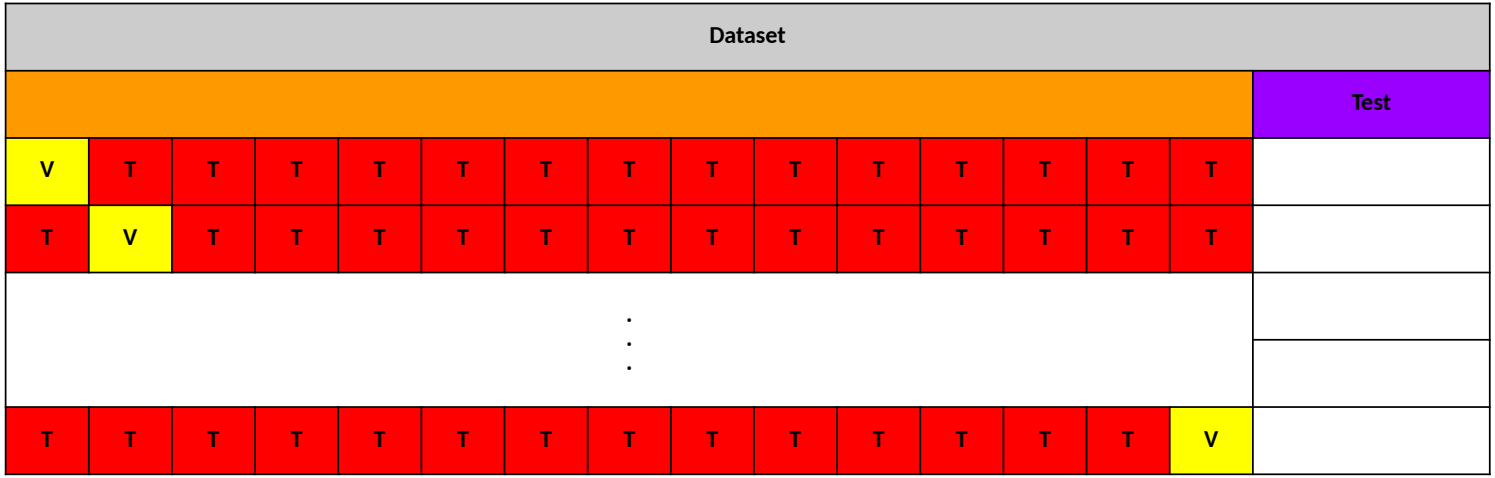
Nella k-fold cross-validation il dataset (testing escluso) viene diviso in *k* parti uguali (detti «fold»), di cui *k-1* vengono usate come training set e la restante viene usata come validation set.
Questo processo viene ripetuto *k* volte (cambiando ogni volta il validation set, i.e., il sottoinsieme su cui valutare il modello) e le prestazioni del modello sono ottenute come media delle prestazioni delle *k* iterazioni



Cross-Validation - Leave-One-Out

La leave-one-out è un caso particolare di k-fold cross-validation in cui ogni fold contiene un solo sample

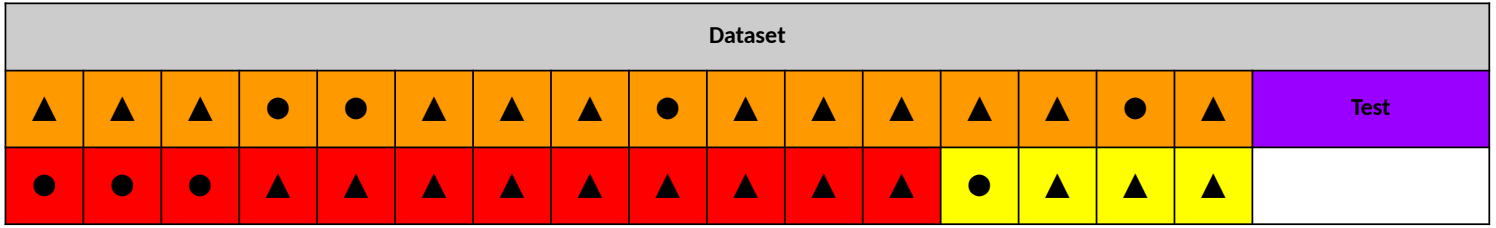
E’ una tecnica ormai poco usata e veniva utilizzata quando i dataset erano piccoli e bisognava sforzarsi per avere *T* abbastanza grande



Validation - Stratificazione

La stratificazione è una tecnica in cui si dividono i dati in modo tale che ogni subset (training/validation nella Holdout, fold nella k-fold) contenga sample di ciascuna classe in quantità **proporzionale** alla frequenza delle classi nel dataset di training (i.e., la prior di *y*)

Questo approccio è utile soprattutto per dataset sbilanciati



Considerazioni computazionali

Se *T* e *V* sono grandi, fare tutte le prove di cui abbiamo parlato in questa lezione può essere computazionalmente (molto) oneroso

Ad ese., pensate ad un modello con *h* iper-parametri, ognuno con una griglia di ricerca di *m* elementi e ad una grid search fatta con cross-validation con *k* fold

Si tratterebbe di addestrare e valutare *h X m X k* modelli, cosa che diventa proibitiva se *T* e *V* sono grandi

Tuttavia, se *V* è grande, il bisogno di usare cross-validation diminuisce (perché *V* ha una rilevanza statistica più significativa) e posso usare Holdout

Considerazioni computazionali

Inoltre, se non riesco (per motivi computazionali) a fare una grid search congiunta degli iper-parametri, posso accontentarmi di farla sequenziale, ovvero di trovare il valore migliore di ogni iper-parametro indipendentemente dagli altri. Ad ese.:

- Uso la Grid Search per trovare il valore migliore del primo iper-parametro, utilizzando il valore mediano della griglia per tutti gli altri
- Una volta trovato tale valore, lo tengo fisso senza cambiarlo più
- Dopodichè uso la Grid Search per trovare il valore migliore del secondo iper-parametro
- Fisso il secondo iper-parametro e passo al terzo…

Riferimenti

- **A tu per tu col Machine Learning. L'incredibile viaggio di un developer nel favoloso mondo della Data Science**, Alessandro Cucci, The Dot Company, 2017 [cap.5]

- **Pattern Classification, second edition,** R. O. Duda, P. E. Hart, D. G. Stork, Wiley-Interscience, 2000