

UNIVERSITÀ DEGLI STUDI
DI MODENA E REGGIO EMILIA

DIPARTIMENTO DI SCIENZE FISICHE, INFORMATICHE E
MATEMATICHE
Corso di Laurea in Matematica

Modelli di machine learning basati su statistiche delle immagini per la quantificazione della qualità visiva

Relatrice:
Prof.ssa Giorgia Franchini

Tesi di Laurea di:
Fabio Bozzoli

Correlatore:
Prof. Luca Zanni

Anno Accademico 2023/2024

Indice

1	Introduzione	3
1.1	Apprendimento Supervisionato	4
1.2	Apprendimento Non Supervisionato	4
1.3	Apprendimento Semi-Supervisionato	5
1.4	Apprendimento per Rinforzo	5
1.5	Deep Learning	6
2	Support Vector Regression	7
2.1	Support Vector Machines	7
2.2	Idea alla base delle SVR	9
2.3	Teorema di Wolfe e KKT	13
2.4	Risoluzione del problema di minimo	17
2.4.1	KKT e problema duale	17
2.4.2	Calcolo di b	19
2.5	Caso nonlineare	21
3	Random Forest	23
3.1	Decision trees	23
3.1.1	Criteri di split	24
3.1.2	Overfitting e Pruning	27
3.2	Regression trees	29
3.2.1	Caso multidimensionale	31
3.2.2	Caso con variabili indipendenti discrete	31
3.2.3	Overfitting e cross-validation	31
3.3	Bagging	33
3.4	Random Forest	35
3.4.1	Out of Bag Error	37
4	Applicazioni dell'SVR e Random Forest al BRISQUE	39
4.1	Image Quality Assessment (IQA)	39
4.2	BRISQUE	40

4.2.1	Calcolo dei parametri MSCN	41
4.2.2	Calcolo dei prodotti a coppie	43
4.2.3	Vettore delle 36 features	43
4.2.4	Calcolo del quality score	45
4.3	Applicazione con dataset Helsinki Deblur Challenge	45
4.3.1	Predisposizione ambiente di lavoro	46
4.3.2	Fase di addestramento	47
4.3.3	Fase di predizione	47
4.3.4	Risultati e considerazioni	48
5	Conclusioni	53
	Bibliografia	57

Capitolo 1

Introduzione

Il **Machine Learning (ML)** (come descritto in [3]) è una branca dell'intelligenza artificiale che permette ai computer di imparare direttamente da esempi, dati ed esperienze. Il nome si riferisce alla capacità di tali metodologie di riconoscere pattern nascosti fra i dati in modo automatico. Infatti l'aumento all'accessibilità dei dati ha permesso a tali sistemi di acquisire grandi quantità di informazioni, permettendo loro di essere addestrati su una miriade di esempi diversi. Allo stesso tempo, la crescente potenza di calcolo degli elaboratori ha sostenuto le loro capacità critiche. Dunque l'obiettivo principale del Machine Learning è ottenere previsioni molto accurate sui dati di test e spiegare come trovare automaticamente un buon predittore basato solo sulle esperienze passate.

Ogni algoritmo di Machine Learning si basa sui seguenti passi [1]:

1. Raccolta e preparazione dei dati: è necessario identificare la giusta fonte di informazione da cui estrarre i dati da fornire all'algoritmo
2. Apprendimento: un aspetto chiave è quello di essere in grado di selezionare l'algoritmo più adatto per la risoluzione del problema. Per fare la decisione corretta bisogna considerare la complessità dei dati e la scelta del *data set*.
3. Predizione: si usa il modello precedentemente creato ed allenato per effettuare previsioni.

Le tecniche di Machine Learning sono tipicamente classificate in 5 categorie: **apprendimento supervisionato**, **apprendimento non supervisionato**, **apprendimento semi-supervisionato**, **apprendimento per rinforzo** e **deep learning**. Ognuna di queste categorie presenta caratteristiche distinte e viene applicata in contesti diversi, a seconda del tipo di problema da risolvere e della natura dei dati disponibili.

1.1 Apprendimento Supervisionato

L'apprendimento supervisionato è forse la forma più comune e ben studiata di machine learning. In questo approccio, un algoritmo viene addestrato utilizzando un dataset etichettato, dove ogni esempio di input è associato a una risposta o etichetta corretta. Gli algoritmi di apprendimento supervisionato generano funzioni in grado di mappare correttamente gli input agli output desiderati.

Essi possono essere ulteriormente suddivisi in due categorie principali *classificazione* e *regressione*:

- Classificazione: l'obiettivo è predire una classe o categoria discreta per ciascun esempio di input. Un esempio classico è il riconoscimento delle immagini, dove il modello deve stabilire se un'immagine rappresenta un determinato elemento oppure no.
- Regressione: l'obiettivo è predire un valore continuo, come il prezzo di una casa basato su caratteristiche quali metratura, numero di stanze e posizione. Essa può essere lineare, polinomiale o logistica.

Alcuni degli algoritmi di apprendimento supervisionato più popolari includono le support vector machines (SVM), le reti neurali artificiali, gli alberi decisionali, i metodi ensemble come il Random Forest, i k nearest neighbors e Navie-Bayes. Questi algoritmi hanno trovato applicazione in una vasta gamma di settori, contribuendo significativamente ai progressi tecnologici e scientifici.

1.2 Apprendimento Non Supervisionato

A differenza dell'apprendimento supervisionato, l'apprendimento non supervisionato lavora con dati non etichettati. In questo caso, l'obiettivo è quello di trarre conclusioni grazie alla scoperta di pattern nascosti fra i dati senza alcuna guida esplicita. Questo tipo di apprendimento è particolarmente utile quando non si dispone di etichette o quando il costo di etichettatura è proibitivo.

Due delle tecniche principali di apprendimento non supervisionato sono il *clustering* e l'*associazione*:

- Clustering: l'obiettivo è raggruppare gli esempi di dati in cluster o gruppi omogenei, basati su caratteristiche simili. Esempi di tali algoritmi sono il k-means, il DBSCAN e le reti neurali auto-organizzanti (SOM).

- Associazione: si cerca di trovare regole interessanti e significative che descrivano grandi porzioni di dati, come nel caso delle analisi delle transazioni nei supermercati per identificare prodotti frequentemente acquistati insieme.

L'apprendimento non supervisionato è essenziale per scoprire strutture sottostanti nei dati e per ridurre la dimensionalità del problema in questione, come succede nel caso dell'analisi delle componenti principali (PCA) e del t-SNE, che aiutano a visualizzare dati complessi in spazi a dimensioni ridotte. Nel caso del PCA per esempio, partendo da molte variabili correlate fra loro si riduce la dimensionalità del dataset, mantenendo elevata la variabilità fra le informazioni. Può essere considerato un metodo di sintesi dei dati.

1.3 Apprendimento Semi-Supervisionato

L'apprendimento semi-supervisionato rappresenta un compromesso tra l'apprendimento supervisionato e non supervisionato. In questo approccio, l'algoritmo utilizza un piccolo set di dati etichettati insieme a un ampio set di dati non etichettati. E' particolarmente utile in scenari dove l'etichettatura dei dati richiede molto tempo, ma è comunque possibile ottenere un certo numero di etichette con sforzi ragionevoli, oppure quando si tratta con dataset con etichettatura non completa.

Numerosi studi hanno dimostrato che la combinazione di dati etichettati con dati non etichettati consente di ottenere ottimi risultati in termini di previsione. Inoltre, la raccolta di dati non etichettati è molto più semplice ed economica rispetto a quella dei dati etichettati, poiché quest'ultima spesso richiede interventi umani o l'esecuzione di esperimenti fisici.

Queste metodologie trovano applicazione in campi come la classificazione delle immagini, l'analisi del linguaggio naturale e il riconoscimento vocale, dove l'etichettatura manuale è particolarmente onerosa.

1.4 Apprendimento per Rinforzo

L'apprendimento per rinforzo è un paradigma di Machine Learning ispirato alla psicologia comportamentale, dove un agente interagisce con un ambiente dinamico per massimizzare una ricompensa cumulativa. E' un tipo di apprendimento automatico che permette alle macchine e agli agenti software di determinare automaticamente il comportamento ideale in un contesto specifico, al fine di massimizzare le loro prestazioni, partendo da un set di dati non etichettati. È necessario un semplice feedback di ricompensa affinché

l’agente possa apprendere il proprio comportamento. Possiamo classificare l’apprendimento per rinforzo in due categorie: l’apprendimento per rinforzo **markoviano** e l’apprendimento per rinforzo **evolutivo**.

- Processo di decisione markoviano: in questa tipologia di algoritmo il modello prende decisioni basate solo ed esclusivamente sul suo stato attuale. In questo modo si modellano delle sequenze discrete in modo iterativo, che hanno l’obiettivo di massimizzare le ricompense future.
- Algoritmi di evoluzione: essi cominciano generando codici iniziali completamente casuali. Ognuno di questi codici viene poi testato per verificare quale è in grado di raggiungere gli obiettivi che ci si aspetta. In questo modo il sistema evolve cercando sempre di premiare i codici che ottengono i risultati migliori.

Algoritmi come Q-learning, SARSA e le reti neurali profonde utilizzate nel deep reinforcement learning (come il famoso algoritmo DQN) sono alla base di molti successi recenti nel campo, tra cui il superamento di campioni umani in giochi complessi come Go e StarCraft II.

L’apprendimento per rinforzo ha applicazioni in robotica, nei sistemi autonomi e nei problemi di ottimizzazione complessi, dove l’interazione continua e l’adattamento all’ambiente sono cruciali.

1.5 Deep Learning

Il deep learning è una sottocategoria del machine learning (ben descritto in [2]) che si basa sull’uso di **reti neurali profonde**, composte da molti strati (da cui il termine “profondo”). Esso sfrutta il meccanismo di retropropagazione, un approccio che ha rivoluzionato il campo dell’apprendimento automatico grazie alla sua capacità di apprendere rappresentazioni gerarchiche dei dati, permettendo di affrontare problemi molto complessi.

Questi metodi hanno dato un apporto significativo agli algoritmi di riconoscimento di oggetti visivi, riconoscimento vocale, scoperta di farmaci ecc...

Algoritmi come le reti neurali convoluzionali (CNN), utilizzate principalmente per l’elaborazione delle immagini, e le reti neurali ricorrenti (RNN), adatte per il trattamento di dati sequenziali, hanno mostrato risultati impressionanti. Inoltre, le tecniche di apprendimento profondo come il transfer learning, che sfrutta modelli pre-addestrati per nuovi compiti, hanno ulteriormente ampliato le possibilità di applicazione del deep learning.

Capitolo 2

Support Vector Regression

Prima di addentrarci nei modelli di regressione, forse è utile fare un piccolo approfondimento sui modelli ***Support Vector Machine*** (SVM), di cui le ***Support Vector Regression*** (SVR) ne sono una generalizzazione.

2.1 Support Vector Machines

Le SVM sono delle tecniche che appartengono a quella famiglia detta di ***Apprendimento Supervisionato***. Sono state sviluppate negli anni '90 da Vladimir N. Vapnik e dal suo team e i risultati sono stati pubblicati in un paper intitolato “*Support Vector Method for Function Approximation, Regression Estimation, and Signal Processing*” nel 1995.

Le SVM sono comunemente utilizzate per problemi di classificazione binaria (ben descritti in [4]). Quindi, dato un *training set*:

$$D = \{(\mathbf{x}_i, y_i), \quad i = 1, \dots, N, \quad \mathbf{x}_i \in \mathbb{R}^n, \quad y_i \in \{-1, 1\}\}$$

l'obiettivo è quello di trovare una funzione

$$f : \mathbb{R}^n \rightarrow \{-1, 1\}$$

per classificare dei nuovi esempi \mathbf{x} dati come input alla macchina.

Nel caso di classi linearmente separabili e funzioni di predizione lineari, le SVM sono in grado di classificare i dati andando a trovare un iperpiano ottimale che massimizza in uno spazio n -dimensionale la distanza dei punti più vicini all'iperpiano di ciascuna classe. Tale iperpiano rappresenta la funzione di predizione per effettuare la classificazione di alcuni esempi.

Graficamente possiamo schematizzare il problema in questo modo:

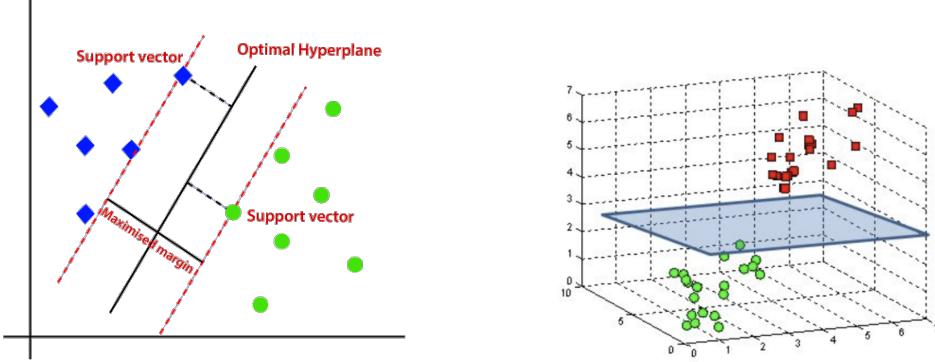


Figura 2.1: SVM per classi linearmente separabili: un caso 2D e 3D

Nel caso di dati linearmente separabili e funzione di predizione lineare, l'obiettivo è quindi quello di trovare l'iperpiano:

$$\mathbf{w} \cdot \mathbf{x} + b = 0 \quad t.c. \quad y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, \quad i = 1, \dots, N, \quad \mathbf{w} \in \mathbb{R}^n, \quad b \in \mathbb{R} \quad (2.1.1)$$

che massimizzi la distanza dall'iperpiano dei punti più vicini all'iperpiano stesso. Possiamo calcolarci la distanza d_i di un punto dall'iperpiano come segue:

$$d_i = \frac{\mathbf{w} \cdot \mathbf{x}_i + b}{\sqrt{\mathbf{w} \cdot \mathbf{w}}} \quad (2.1.2)$$

Scalandando opportunamente \mathbf{w} e b , per i punti più vicini all'iperpiano si ha $y_i d_i = (\|\mathbf{w}\|)^{-1}$. Tali punti, fondamentali per l'algoritmo, sono detti **support vectors**, da cui segue il nome della metodologia di apprendimento.

Quindi la coppia (\mathbf{w}^*, b^*) corrispondente all'iperpiano ottimale è la soluzione del problema di minimo:

$$\begin{aligned} \min & \quad \frac{1}{2} \mathbf{w} \cdot \mathbf{w} \\ \text{soggetto a} & \quad y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 \end{aligned} \quad (2.1.3)$$

La soluzione può poi essere ottenuta passando attraverso il problema duale grazie al teorema di Wolfe e applicando poi le condizioni KKT. La soluzione completa non viene riportata dal momento che le SVM non rappresentano il fulcro della nostra discussione. Si è voluto comunque riportarle per completezza, poiché le SVR, che verranno discusse in dettaglio nel seguito, risultano essere una loro generalizzazione. Possono inoltre presentarsi casi in cui gli insiemi non sono linearmente separabili o casi di separatori non lineari risolvibili attraverso particolari funzioni dette kernel, che per gli stessi motivi non verranno affrontati in questa sezione.

2.2 Idea alla base delle SVR

Il problema di regressione è una generalizzazione del problema di classificazione (come ben spiegato in [4], [5] e [6]): il modello restituisce un output continuo e non più un output da un insieme finito. Ad esempio, quando si tratta di prevedere il prezzo di una casa in base alle sue caratteristiche, il modello di regressione restituirà un valore continuo (il prezzo) invece di una classe discreta. Questo rende l'analisi di regressione adatta per problemi in cui la variabile di output è quantitativa e non categorica.

Supponiamo di avere un *training set* come segue:

$$\{(\mathbf{x}_i, y_i), \quad i = 1, \dots, N, \quad \mathbf{x}_i \in \mathbb{R}^n, \quad y_i \in \mathbb{R}\} \subset \mathbb{R}^n \times \mathbb{R}$$

Anche in questo caso la formulazione del problema nel caso di funzioni di predizione lineari può essere derivata più facilmente assumendo un punto di vista geometrico, cercando quindi un iperpiano che rappresenti la funzione di predizione:

$$y = f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b = \sum_{i=1}^n w_i x_i + b, \text{ con } \mathbf{w}, \mathbf{x} \in \mathbb{R}^n, b \in \mathbb{R} \quad (2.2.1)$$

La generalizzazione delle SVM in SVR richiede l'introduzione di una regione attorno alla funzione detta **ε -tube**. A questa regione si chiede di approssimare al meglio i dati del training set, bilanciando la complessità della funzione e l'errore di approssimazione. Più in dettaglio, l'obiettivo di questa tecnica è quello di trovare una funzione $f(x)$ più piatta (*flat*) possibile in modo che il corrispondente ε -tube approssimi al meglio i valori y_i . Nel valutare l'errore di approssimazione, la predizione del modello risulterà corretta se:

$$|y_i - \mathbf{w} \cdot \mathbf{x}_i - b| \leq \varepsilon \quad (2.2.2)$$

cioè se il valore y_i è approssimato a meno di ε , cioè (x_i, y_i) appartiene all' ε -tube. Quindi un grosso vantaggio che ci garantisce l'SVR è quello di poter decidere a priori qual è il margine di errore accettabile per approssimare i dati. Il valore di ε infatti indica lo spessore del tubo: maggiore è ε , maggiore sarà anche la tolleranza all'errore; al contrario piccoli valori di ε sono indici di una minore tolleranza all'errore con una conseguente influenza sulla scelta dei support vectors, come sarà spiegato in seguito.

Formalmente, per descrivere l' ε -tube possiamo anche utilizzare una *loss function*.

Definizione 2.2.1 (Loss function). E' detta ***loss function*** una funzione $V(x, y, f(x)) : X \times Y \times Y \rightarrow \mathbb{R}_0^+$ in cui x è un input, y è un'etichetta reale e $f(x)$ un'etichetta predetta, tale che $V(x, y, y) = 0 \ \forall x \in X, \forall y \in Y$

Poiché nella regressione i valori che si vogliono stimare sono del tipo $f(x) - y$, si avrà $V(x, y, f(x)) = \tilde{V}(f(x) - y)$. Dunque l'SVR adotta una ε -insensitive *loss function*, che può essere definita come segue:

$$V_\varepsilon(x, y, f(x)) = \max(0, |f(x) - y| - \varepsilon)$$

ossia:

$$V_\varepsilon(x, y, f(x)) = \begin{cases} 0 & \text{se } |f(x) - y| \leq \varepsilon \\ |f(x) - y| - \varepsilon & \text{se } |f(x) - y| > \varepsilon \end{cases}. \quad (2.2.3)$$

Quindi siamo riusciti a descrivere matematicamente l' ε -tube attraverso la (2.2.2) e la (2.2.3). In modo molto simile possiamo anche definire una *loss function* quadratica come segue:

$$V_\varepsilon^2(x, y, f(x)) = \begin{cases} 0 & \text{se } |f(x) - y| \leq \varepsilon \\ (|f(x) - y| - \varepsilon)^2 & \text{se } |f(x) - y| > \varepsilon \end{cases}. \quad (2.2.4)$$

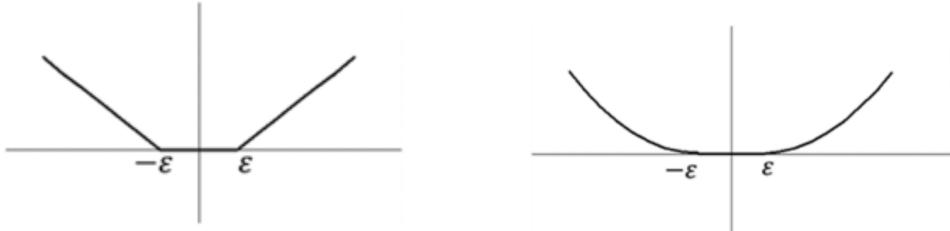


Figura 2.2: ε -insensitività della *loss function* nel caso lineare e nel caso quadratico

A questo punto ci chiediamo: come esprimere la *flatness* della funzione? L'idea è quella di minimizzare la norma di \mathbf{w} . Possiamo quindi scrivere un problema di minimizzazione come segue:

$$\min \quad \frac{1}{2} \|\mathbf{w}\|^2 \quad (2.2.5)$$

$$\text{soggetto a} \quad \begin{cases} y_i - \mathbf{w} \cdot \mathbf{x}_i - b \leq \varepsilon \\ \mathbf{w} \cdot \mathbf{x}_i + b - y_i \leq \varepsilon \end{cases} \quad (2.2.6)$$

dove ovviamente i vincoli sono stati ripresi dalla (2.2.2).

La regione ammissibile di questo problema risulta non vuota solo se l'iperpiano $\mathbf{w} \cdot \mathbf{x} + b = 0$ approssima tutti i valori y_i a meno di ε . Questo non è sempre garantito. Per questo motivo può essere utile introdurre delle **slack variables** ξ_i, ξ_i^* per far fronte alle restrizioni del problema di ottimizzazione. Esse rappresentano una misura di quanto un punto (x_i, y_i) si trova al di fuori dell' ε -tube, come possiamo anche osservare dalla seguente figura:

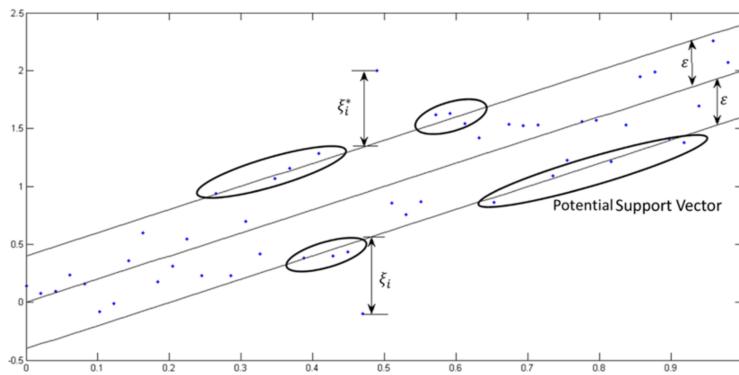


Figura 2.3: SVR 2D con slack variables

Si introduce così il concetto di **soft margin**, ossia una regione dello spazio in cui si consente al modello di fare qualche errore di predizione sui punti del training set per ottenere comunque un risultato predittivo (anche se non preciso al 100%). Esso si differenzia perciò dall'**hard margin**, in cui non sono permessi errori. Possiamo così identificare le *slack variables* anche grazie al concetto precedentemente affrontato di *loss function*:

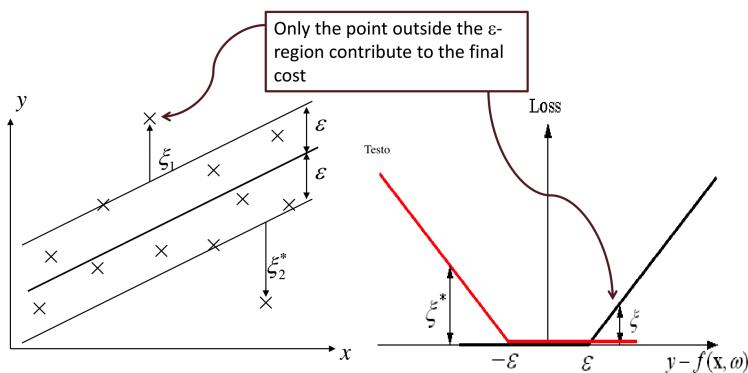


Figura 2.4: Il *soft margin* per una SVR lineare

E' ora possibile riformulare il problema nel seguente modo:

$$\min \quad \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N (\tilde{V}(\xi_i + \varepsilon) + \tilde{V}(\xi_i^* + \varepsilon)) \quad (2.2.7)$$

$$\text{soggetto a } \begin{cases} y_i - \mathbf{w} \cdot \mathbf{x}_i - b \leq \varepsilon + \xi_i \\ \mathbf{w} \cdot \mathbf{x}_i + b - y_i \leq \varepsilon + \xi_i^* \\ \xi_i, \xi_i^* \geq 0 \end{cases} \quad (2.2.8)$$

L'iperparametro C è detto **parametro di regolarizzazione** e rappresenta il compromesso tra la *flatness* di f e la quantità fino a cui sono tollerate deviazioni maggiori di ε .

Osservazione 2.2.1. Studiamo cosa succede al variare di C in \mathbb{R} :

- Se C è molto grande, allora maggiore peso verrà dato a

$$\sum_{i=1}^N (\tilde{V}(\xi_i + \varepsilon) + \tilde{V}(\xi_i^* + \varepsilon))$$

e dunque maggiore sforzo sarà impiegato nel minimizzare l'errore: rischio di *overfitting* (o sovradattamento), cioè quando il modello si adatta molto bene ai dati di training, ma non è in grado di fare previsioni future: perde cioè la capacità di generalizzare (possiamo dire che rappresenti una violazione del *rasoio di Occam*).

- Se C è molto piccolo, allora maggiore peso verrà dato a

$$\frac{1}{2} \|\mathbf{w}\|$$

e dunque maggiore sforzo sarà dedicato ad aumentare la *flatness*: rischio di consentire troppe osservazioni al di fuori dei margini.

Dal momento che

$$V_\varepsilon(\xi_i + \varepsilon) = \xi_i, \quad V_\varepsilon(\xi_i^* + \varepsilon) = \xi_i^* \quad (2.2.9)$$

possiamo scrivere il problema di minimizzazione come segue:

$$\min \quad \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N (\xi_i + \xi_i^*) \quad (2.2.10)$$

$$\text{soggetto a } \begin{cases} y_i - \mathbf{w} \cdot \mathbf{x}_i - b \leq \varepsilon + \xi_i \\ \mathbf{w} \cdot \mathbf{x}_i + b - y_i \leq \varepsilon + \xi_i^* \\ \xi_i, \xi_i^* \geq 0 \end{cases} \quad (2.2.11)$$

2.3 Teorema di Wolfe e KKT

Prima di proseguire con l'SVR è importante andare a studiare attentamente i due seguenti teoremi, che saranno fondamentali per la soluzione del problema di minimo precedentemente formulato (come riportato in [8]). Sia $f : \mathbb{R}^n \rightarrow \mathbb{R}$ una funzione differenziabile con continuità e siano $h : \mathbb{R}^n \rightarrow \mathbb{R}^m$ e $g : \mathbb{R}^n \rightarrow \mathbb{R}^p$ definite come:

$$h(x) = \begin{pmatrix} h_1(x) \\ h_2(x) \\ \vdots \\ h_n(x) \end{pmatrix} \quad g(x) = \begin{pmatrix} g_1(x) \\ g_2(x) \\ \vdots \\ g_p(x) \end{pmatrix}$$

Definizione 2.3.1 (Funzione Lagrangiana). La **funzione Lagrangiana** (o Lagrangiana) del problema

$$\begin{array}{ll} \min & f(x) \\ \text{soggetto a} & \begin{cases} h(x) = 0 \\ g(x) \leq 0 \end{cases} \end{array}$$

è definita come

$$\mathcal{L}(x, \lambda, \mu) = f(x) + \lambda^T h(x) + \mu^T g(x)$$

dove $\lambda \in \mathbb{R}^m$, $\mu \in \mathbb{R}^p$ sono i vettori dei **moltiplicatori di Lagrange**.

La Lagrangiana è quindi una funzione $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^p \rightarrow \mathbb{R}$, di classe C^1 , che tiene conto di entrambi gli elementi che definiscono il problema vincolato, ossia la funzione obiettivo e i vincoli. In forma esplicita possiamo scrivere:

$$\mathcal{L}(x, \lambda, \mu) = f(x) + \sum_{i=1}^m \lambda_i h_i(x) + \sum_{j=1}^p \mu_j g_j(x)$$

Consideriamo ora il seguente teorema che garantisce delle condizioni necessarie per la soluzione di un problema di programmazione non lineare.

Teorema 2.3.1 (Teorema di Karush-Kuhn-Tucker). Se x^* è un punto regolare dei vincoli $h(x)$ e $g(x)$ ed è un punto di minimo locale di $f(x)$ sulla regione ammissibile $\Omega = \{x \in \mathbb{R}^n : h(x) = 0, g(x) \leq 0\}$, allora esistono dei moltiplicatori lagrangiani $\lambda^* \in \mathbb{R}^m$ e $\mu^* \in \mathbb{R}^p$ che verificano le seguenti

condizioni di Karush-Kuhn-Tucker (KKT):

$$\begin{aligned}\nabla_x \mathcal{L}(x^*, \lambda^*, \mu^*) &= \nabla f(x^*) + \nabla h(x^*)\lambda^* + \nabla g(x^*)\mu^* = 0 \\ h(x^*) &= 0 \\ g(x^*) &\leq 0 \\ \mu^* &\geq 0 \\ g(x^*)^T \mu^* &= 0\end{aligned}$$

La seconda e la terza equazione stanno a indicare che il punto x^* rispetta i vincoli e vengono perciò dette **condizioni di ammissibilità**. L'ultima relazione viene invece detta **condizione di complementarietà**. Essa può anche essere scritta in forma esplicita come segue:

$$\sum_{j=1}^p g_j(x^*)\mu_j^* = 0$$

Osservazione 2.3.1. Ricordando le due disequazioni del sistema di Karush-Kuhn-Tucker, abbiamo che $g_j(x^*) \leq 0 \forall j = 1, \dots, p$ e $\mu_j^* \geq 0 \forall j = 1, \dots, p$, dunque tutti gli addendi sono non positivi. Perciò, affinché la somma sia nulla si deve avere che tutti gli addendi siano nulli, ossia:

$$g_j(x^*)\mu_j^* = 0, \quad \forall j = 1, \dots, p$$

Osserviamo inoltre che se il j -esimo vincolo è attivo, ossia $\mu_j^* > 0$, allora $g_j(x^*) = 0$. Di conseguenza possiamo riscrivere la prima condizione come segue:

$$-\nabla f(x^*) = \sum_{i=1}^m \lambda_i^* \nabla h_i(x^*) + \sum_{j \in A(x^*)} \mu_j^* \nabla g_j(x^*)$$

dove $A(x^*) = \{j \in \{1, \dots, p\} : g_j(x^*) = 0\}$ è l'insieme degli indici dei vincoli di disegualanza attivi in x^* .

Si riporta la dimostrazione nel caso di soli vincoli di uguaglianza lineari per semplicità.

Dimostrazione.

$$\min_{A^T x = b} \Rightarrow \Omega = \{x \in \mathbb{R}^n : A^T x = b\}$$

con $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, con $m < n$. Si indica con a_i la i -esima colonna di A .

$$h_i(x) = a_i^T x - b_i, \quad \nabla h_i(x) = a_i, \quad i = 1, \dots, m$$

Assumiamo che A abbia rango m , allora: $\{\nabla h_i(x), i = 1, \dots, m\}$ è un insieme di vettori linearmente indipendenti, quindi x è un *punto regolare dei vincoli*, ossia la *condizione di qualificazione dei vincoli* è soddisfatta in x . Ricordiamo:

$$\begin{aligned} \ker(A^T) &= \{w \in \mathbb{R}^n : A^T w = 0\}, \quad \text{Im}(A) = \{z \in \mathbb{R}^n : z = Av, v \in \mathbb{R}^m\} \\ \mathbb{R}^n &= \ker(A^T) \oplus \text{Im}(A) \text{ e } \text{Im}(A)^\perp = \ker(A^T) \end{aligned} \quad (2.3.1)$$

Sia $x \in \Omega$. Un vettore $d \in \mathbb{R}^n$ è una direzione ammissibile per Ω in x se $A^T(x + \alpha d) = b$, $\forall \alpha \in [0, \bar{\alpha}]$. Si ha:

$$A^T(x + \alpha d) = A^T x + \alpha A^T d = b + \alpha A^T d$$

$\Rightarrow d$ è ammissibile se e solo se $A^T d = 0 \Leftrightarrow d \in \ker(A^T)$. Adesso procediamo per assurdo, supponendo che le condizioni KKT non siano soddisfatte in $x \in \Omega$. In particolare supponiamo che non sia verificata la prima condizione, cioè quella che esprime il gradiente di $f(x)$ come combinazione lineare dei gradienti dei vincoli attivi. Ossia:

$$\nabla f(x) + A\lambda \neq 0, \quad \forall \lambda \in \mathbb{R}^m \Rightarrow \nabla f(x) \notin \text{Im}(A)$$

Per quanto detto in (2.3.1) posso scrivere il gradiente di $f(x)$ come una combinazione lineare di due vettori: uno appartenente al nucleo di A^T e uno all'immagine di A :

$$\nabla f(x) = y + z, \quad y \in \ker(A^T), \quad z \in \text{Im}(A)$$

Poiché $\nabla f(x) \notin \text{Im}(A) \Rightarrow y \neq 0$. Sia $d = -y$. Si ha:

$$d^T \nabla f(x) = -y^T(y + z) = -||y||^2 - y^T z$$

Ma $y^T z = 0$ dalla (2.3.1). Di conseguenza:

$$d^T \nabla f(x) = -||y||^2 < 0$$

Perciò d è una direzione ammissibile per Ω e di discesa per $f \Rightarrow x$ non può essere un punto di minimo vincolato. Assurdo. \square

Il seguente teorema rientra nella cosiddetta teoria della dualità. Infatti esso permette di associare ad un dato problema di ottimizzazione un suo duale che, sotto determinate ipotesi, può presentare stretti legami con il problema di partenza, ma allo stesso tempo, avere una struttura più favorevole.

Teorema 2.3.2 (Teorema di Wolfe). Si consideri il problema primale

$$\begin{aligned} \min \quad & f(x) \\ \text{soggetto a} \quad & g(x) \leq 0 \end{aligned}$$

Se $f, g_j \ j = 1, \dots, p$ sono convesse e di classe C^1 e se x^* è soluzione del problema primale e punto regolare dei vincoli, allora esiste un vettore $\lambda^* \in \mathbb{R}^p$ tale che (x^*, λ^*) è soluzione del problema duale nelle variabili x, λ

$$\begin{aligned} \max \quad & \mathcal{L}(x, \lambda) \\ \text{soggetto a} \quad & \begin{cases} \nabla_x \mathcal{L}(x, \lambda) = 0 \\ \lambda \geq 0 \end{cases} \end{aligned}$$

Inoltre il minimo del problema primale ed il massimo del problema duale coincidono, ovvero

$$f(x^*) = \mathcal{L}(x^*, \lambda^*) \tag{2.3.2}$$

Dimostrazione. Dimostriamo prima la 2.3.2. Sappiamo che dalla definizione di Lagrangiana si ha: $\mathcal{L}(x^*, \lambda^*) = f(x^*) + g(x^*)^T \lambda^*$. Ma poiché x^* è punto regolare dei vincoli, allora dalla condizione di complementarietà delle KKT si ha che $g(x^*)^T \lambda^* = 0$. Segue che: $\mathcal{L}(x^*, \lambda^*) = f(x^*)$.

Vogliamo ora dimostrare che (x^*, λ^*) è un punto di massimo per la lagrangiana, ossia che preso un qualunque punto ammissibile (x, λ) , allora si ha che $\mathcal{L}(x, \lambda) \leq \mathcal{L}(x^*, \lambda^*)$.

Infatti abbiamo che

$$\mathcal{L}(x^*, \lambda^*) = f(x^*) \geq f(x^*) + \sum_{i=1}^p \lambda_i g_i(x^*) = \mathcal{L}(x^*, \lambda)$$

dal momento che dai vincoli dati in ipotesi si sa che $\lambda \geq 0$ e $g(x) \leq 0$. Inoltre, dall'ipotesi di convessità si deve avere:

$$\mathcal{L}(x^*, \lambda) \geq \mathcal{L}(x, \lambda) + \nabla_x \mathcal{L}(x, \lambda)^T (x^* - x)$$

Sappiamo però che $\nabla_x \mathcal{L}(x, \lambda) = 0$, dunque si ottiene $\mathcal{L}(x^*, \lambda) \geq \mathcal{L}(x, \lambda)$, ossia

$$\mathcal{L}(x^*, \lambda^*) \geq \mathcal{L}(x, \lambda)$$

che è quello che volevamo dimostrare. \square

2.4 Risoluzione del problema di minimo

Riporto il problema a cui eravamo arrivati precedentemente:

$$\min \quad \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N (\xi_i + \xi_i^*) \quad (2.4.1)$$

$$\text{soggetto a} \quad \begin{cases} y_i - \mathbf{w} \cdot \mathbf{x}_i - b \leq \varepsilon + \xi_i \\ \mathbf{w} \cdot \mathbf{x}_i + b - y_i \leq \varepsilon + \xi_i^* \\ \xi_i, \xi_i^* \geq 0 \end{cases} \quad (2.4.2)$$

2.4.1 KKT e problema duale

Calcolo la Lagrangiana del problema:

$$\begin{aligned} \mathcal{L}(\mathbf{w}, b, \xi, \xi^*, \alpha, \alpha^*, \lambda, \lambda^*) := & \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N (\xi_i + \xi_i^*) + \\ & + \sum_{j=1}^N \alpha_j (y_j - \mathbf{w} \cdot \mathbf{x}_j - b - \varepsilon - \xi_j^*) + \\ & + \sum_{l=1}^N \alpha_l^* (\mathbf{w} \cdot \mathbf{x}_l + b - y_l - \varepsilon - \xi_l^*) + \\ & - \sum_{k=1}^N (\lambda_k \xi_k + \lambda_k^* \xi_k^*) \end{aligned}$$

dove $\alpha_i, \alpha_i^*, \eta_k, \eta_k^*$ sono i moltiplicatori di Lagrange.

Adesso possiamo scrivere il sistema delle KKT. Osserviamo che il prodotto tra i moltiplicatori di Lagrange e i vincoli deve essere uguale a zero. Perciò i moltiplicatori di Lagrange che sono nulli corrispondono ai dati che stanno all'interno dell' ε -tube, mentre i *support vectors* hanno moltiplicatori di Lagrange diversi da zero.

$$\left\{ \begin{array}{l} \frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^N (\alpha_i - \alpha_i^*) \mathbf{x}_i = 0 \implies \mathbf{w} = \sum_{i=1}^N (\alpha_i - \alpha_i^*) \mathbf{x}_i \\ \frac{\partial \mathcal{L}}{\partial b} = \sum_{i=1}^N (\alpha_i^* - \alpha_i) = 0 \\ \frac{\partial \mathcal{L}}{\partial \xi_i} = C - \alpha_i - \lambda_i = 0 \implies \lambda_i = C - \alpha_i \geq 0 \\ \frac{\partial \mathcal{L}}{\partial \xi_i^*} = C - \alpha_i^* - \lambda_i^* = 0 \implies \lambda_i^* = C - \alpha_i^* \geq 0 \\ y_i - \mathbf{w} \cdot \mathbf{x}_i - b - \varepsilon - \xi_i \leq 0 \\ \mathbf{w} \cdot \mathbf{x}_i + b - y_i - \varepsilon - \xi_i^* \leq 0 \\ \xi_i, \xi_i^* \geq 0 \\ \alpha_i, \alpha_i^*, \lambda_i, \lambda_i^* \geq 0 \\ \alpha_i (y_i - \mathbf{w} \cdot \mathbf{x}_i - b - \varepsilon - \xi_i) = 0 \\ \alpha_i^* (\mathbf{w} \cdot \mathbf{x}_i + b - y_i - \varepsilon - \xi_i^*) = 0 \\ \lambda_i \xi_i = 0 \\ \lambda_i^* \xi_i^* = 0 \end{array} \right. \quad \begin{array}{l} (2.4.3a) \\ (2.4.3b) \\ (2.4.3c) \\ (2.4.3d) \\ (2.4.3e) \\ (2.4.3f) \\ (2.4.3g) \\ (2.4.3h) \\ (2.4.3i) \\ (2.4.3j) \\ (2.4.3k) \\ (2.4.3l) \end{array}$$

Ricordando che $\nabla \mathcal{L} = (\frac{\partial \mathcal{L}}{\partial \mathbf{w}}, \frac{\partial \mathcal{L}}{\partial b}, \frac{\partial \mathcal{L}}{\partial \xi}, \frac{\partial \mathcal{L}}{\partial \xi^*})$, possiamo applicare il teorema di Wolfe ottenendo che il problema duale risulta essere:

$$\begin{aligned} \max \quad & \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N (\xi_i + \xi_i^*) + \sum_{i=1}^N \alpha_i (y_i - \mathbf{w} \cdot \mathbf{x}_i - b - \varepsilon - \xi_i^*) + \\ & + \sum_{i=1}^N \alpha_i^* (\mathbf{w} \cdot \mathbf{x}_i + b - y_i - \varepsilon - \xi_i^*) - \sum_{i=1}^N (\lambda_i \xi_i + \lambda_i^* \xi_i) \\ \text{soggetto a} \quad & \left\{ \begin{array}{l} \mathbf{w} = \sum_{i=1}^N (\alpha_i - \alpha_i^*) \mathbf{x}_i \\ \sum_{i=1}^N (\alpha_i^* - \alpha_i) = 0 \\ \lambda_i = C - \alpha_i \\ \lambda_i^* = C - \alpha_i^* \\ \alpha, \alpha^*, \lambda, \lambda^* \geq 0 \end{array} \right. \end{aligned}$$

A questo punto possiamo andare a sostituire nella funzione obiettivo i risultati appena ottenuti riguardo a $\mathbf{w}, \lambda_i, \lambda_i^*$, trovando così:

$$\begin{aligned}
\max_{\alpha, \alpha^*} \quad & -\frac{1}{2} \sum_{i,j=1}^N (\alpha_i - \alpha_i^*) (\alpha_j - \alpha_j^*) \mathbf{x}_i \cdot \mathbf{x}_j - \varepsilon \sum_{i=1}^N (\alpha_i + \alpha_i^*) + \\
& + \sum_{i=1}^N y_i (\alpha_i - \alpha_i^*) \\
\text{soggetto a} \quad & \begin{cases} 0 \leq \alpha_i \leq C \\ 0 \leq \alpha_i^* \leq C \\ \sum_{i=1}^n (\alpha_i^* - \alpha_i) = 0 \end{cases}
\end{aligned}$$

Osservazione 2.4.1. Dal momento che $\mathbf{w} = \sum_{i=1}^N (\alpha_i - \alpha_i^*) \mathbf{x}_i$, abbiamo che

$$f(x) = \sum_{i=1}^N (\alpha_i - \alpha_i^*) (\mathbf{x}_i \cdot \mathbf{x}) + b$$

viene detto *Support Vector expansion*, cioè \mathbf{w} può essere scritto come combinazione lineare dei modelli di addestramento \mathbf{x}_i . I vettori \mathbf{x}_i corrispondenti a moltiplicatori non nulli prendono il nome di ***support vectors***.

Questo ha due importanti conseguenze:

- Nella rappresentazione della funzione di predizione risultano significativi solo i *support vectors*.
- L'algoritmo può essere descritto in termini di prodotti scalari tra i dati di input. Anche per calcolare il valore di $f(\mathbf{x})$ non è necessario calcolare \mathbf{w} esplicitamente. Questa osservazione sarà molto utile per analizzare il caso nonlineare.

2.4.2 Calcolo di \mathbf{b}

Riprendiamo alcune delle condizioni di Karush-Kuhn-Tucker, in particolare consideriamo:

$$\alpha_i (y_i - \mathbf{w} \cdot \mathbf{x}_i - b - \varepsilon - \xi_i) = 0 \quad (2.4.4)$$

$$\alpha_i^* (\mathbf{w} \cdot \mathbf{x}_i + b - y_i - \varepsilon - \xi_i^*) = 0 \quad (2.4.5)$$

$$(C - \alpha_i) \xi_i = 0 \quad (2.4.6)$$

$$(C - \alpha_i^*) \xi_i^* = 0 \quad (2.4.7)$$

dove la (2.4.6) e la (2.4.7) sono ottenute unendo la (2.4.3k) con (2.4.3c) e la (2.4.3l) con (2.4.3d). A partire da queste quattro equazioni possiamo ottenere importanti risultati. Infatti:

- Dalla (2.4.6) e dalla (2.4.7) ottengo che solo i punti (\mathbf{x}_i, y_i) a cui corrisponde $\alpha_i = C$ (o $\alpha_i^* = C$) stanno al di fuori dell' ε -tube. Infatti se $\alpha_i \neq C$ (o $\alpha_i^* \neq C$) si ha $\xi_i = 0$ (o rispettivamente $\xi_i^* = 0$).
- Dalla (2.4.4) e (2.4.5) se $\alpha_i \neq 0$ e $\alpha_i^* \neq 0$, allora posso dividere da entrambe le parti la (2.4.4) per α_i e la (2.4.5) per α_i^* . Successivamente sommo membro a membro e trovo $\xi_i = -\xi_i^* - 2\varepsilon$. Poiché però si deve avere $\xi_i \geq 0$ e $\xi_i^* \geq 0$, allora si ha $\xi_i \neq 0$ e $\xi_i^* \neq 0$, che è assurdo. Quindi si deve avere che $\alpha_i \alpha_i^* = 0$.
- Se un qualche $\alpha_i \in (0, C)$, allora si ha dalla (2.4.6) $\xi_i = 0$. Inoltre possiamo dividere da entrambe le parti la (2.4.4) per α_i , ottenendo

$$b = y_i - \mathbf{w} \cdot \mathbf{x}_i - \varepsilon$$

Analogamente se $\alpha_i^* \in (0, C)$, si ha

$$b = y_i - \mathbf{w} \cdot \mathbf{x}_i + \varepsilon$$

Osserviamo che se $|f(\mathbf{x}_i) - y_i| < \varepsilon$, allora il secondo fattore di (2.4.4) e (2.4.5) risulta essere diverso da zero. Segue che $\alpha_i = \alpha_i^* = 0$. Detto in altre parole: se un punto è interno all' ε -tube, allora i moltiplicatori di Lagrange α_i e α_i^* si annullano. Questo significa che per trovare \mathbf{w} non abbiamo bisogno di tutti i punti \mathbf{x}_i .

Ecco spiegato anche il motivo per cui al diminuire di ε aumenta la precisione dell'algoritmo: se ε diminuisce, allora si ha un aumento del numero di *support vectors* e di conseguenza della complessità dell'algoritmo.

2.5 Caso nonlineare

Finora abbiamo supposto $f(x)$ lineare, ma è anche possibile costruire funzione di predizione di tipo nonlineare. Per fare questo, i dati possono essere mappati in uno spazio di dimensione superiore, chiamato ***feature space*** e il problema della ricerca della funzione di predizione può essere affrontato nel feature space [7].

L'idea è quindi quella di mappare i dati di input \mathbf{x}_i mediante una mappa $\Phi : \mathcal{X} \rightarrow \mathcal{F}$, dove \mathcal{X} è lo spazio dei vettori di ingresso, detto anche *input space*, mentre \mathcal{F} è lo spazio dei vettori di uscita. Nel feature space \mathcal{F} sarà poi possibile applicare l'SVR lineare come descritto precedentemente a partire solamente dai prodotti scalari delle immagini dei vari \mathbf{x}_i di partenza, utilizzando come funzione di predizione quella definita nell'osservazione 2.4.1.

Osserviamo che ad un iperpiano separatore ottimale in \mathcal{F} corrisponde una superficie di separazione nonlineare nello spazio di input. Per implementare questo approccio, sia in fase di addestramento che in fase di predizione, è necessario calcolare il prodotto scalare nello spazio \mathcal{F} delle immagini dei punti dell'input space. Di conseguenza se troviamo un'espressione per il prodotto scalare nel feature space, definita solo mediante i punti dello spazio di input, la conoscenza completa della mappa Φ non è necessaria. A tale scopo è utile richiamare la definizione di funzione kernel.

Definizione 2.5.1 (Kernel). Sia $\Phi : \mathcal{X} \rightarrow \mathcal{F}$ un'applicazione tra lo spazio di input e il feature space. Chiamiamo *kernel* una funzione $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ tale che

$$k(\mathbf{x}, \mathbf{z}) = \langle \Phi(\mathbf{x}), \Phi(\mathbf{z}) \rangle_{\mathcal{F}}$$

per ogni $\mathbf{x}, \mathbf{z} \in \mathcal{X}$, dove $\langle \cdot, \cdot \rangle_{\mathcal{F}}$ indica il prodotto scalare in \mathcal{F} .

In questo modo, la fase di addestramento dell'SVR nonlineare richiede la risoluzione del seguente problema di ottimizzazione:

$$\begin{aligned} \max_{\alpha, \alpha^*} \quad & -\frac{1}{2} \sum_{i,j=1}^N (\alpha_i - \alpha_i^*) (\alpha_j - \alpha_j^*) k(\mathbf{x}_i, \mathbf{x}_j) + \\ & - \varepsilon \sum_{i=1}^N (\alpha_i + \alpha_i^*) + \sum_{i=1}^N y_i (\alpha_i - \alpha_i^*) \\ \text{soggetto a} \quad & \begin{cases} 0 \leq \alpha_i \leq C \\ 0 \leq \alpha_i^* \leq C \\ \sum_{i=1}^n (\alpha_i^* - \alpha_i) = 0 \end{cases} \end{aligned}$$

Riprendendo l'osservazione 2.4.1 e tenendo presente l'espressione di \mathbf{w} :

$$\mathbf{w} = \sum_{i=1}^N (\alpha_i - \alpha_i^*) \Phi(\mathbf{x}_i),$$

per quanto riguarda la definizione della funzione di predizione, possiamo scrivere:

$$f(\mathbf{x}) = \sum_{i=1}^N (\alpha_i - \alpha_i^*) k(\mathbf{x}_i, \mathbf{x}) + b$$

Per completezza riportiamo alcuni esempi di kernel usati frequentemente:

- Lineare:

$$k(\mathbf{x}, \mathbf{z}) = \mathbf{x} \cdot \mathbf{z}$$

La funzione di predizione ottimale nello spazio di input è un piano.

- Polinomiale:

$$k(\mathbf{x}, \mathbf{z}) = (\mathbf{x} \cdot \mathbf{z} + 1)^d - 1, \quad d \in \mathbb{N}, \quad d > 1$$

La funzione di predizione ottimale nello spazio di input è una superficie polinomiale di grado d .

- Gaussiano:

$$k(\mathbf{x}, \mathbf{z}) = e^{-\frac{\|\mathbf{x}-\mathbf{z}\|^2}{2\sigma^2}}, \quad \sigma \in \mathbb{R}$$

La funzione di predizione ottimale nello spazio di input è una somma pesata di gaussiane centrate nei support vectors.

Ecco come possiamo visualizzare graficamente l'azione di un kernel:

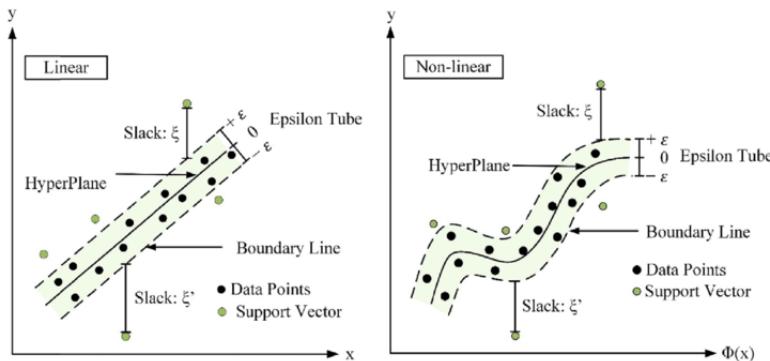


Figura 2.5: a) SVR lineare; b) SVR non lineare

Capitolo 3

Random Forest

3.1 Decision trees

Prima di parlare del Random Forest è utile spendere qualche parola per introdurre il concetto di **decision tree** (albero decisionale), su cui si basa gran parte della teoria [9].

Un albero decisionale è un algoritmo di apprendimento supervisionato non parametrico, utilizzato sia per compiti di classificazione che di regressione. Ha una struttura gerarchica ad albero, composta da un *nodo radice*, *rami*, *nodi interni* e *nodi foglia*.

Come si può vedere dal diagramma sottostante, un albero decisionale inizia con un nodo radice, che non ha rami in ingresso. I rami in uscita dal nodo radice si collegano poi ai nodi interni, noti anche come nodi decisionali. In base alle caratteristiche disponibili, entrambi i tipi di nodi effettuano valutazioni per formare sottoinsiemi omogenei, rappresentati dai nodi foglia. I nodi foglia rappresentano tutti i possibili risultati all'interno del dataset.

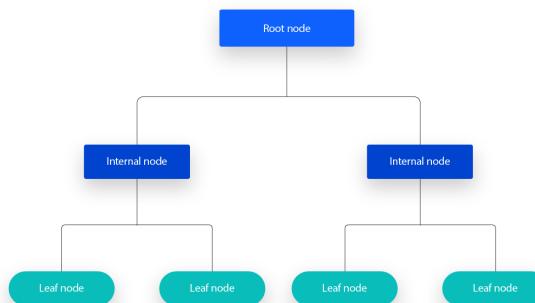


Figura 3.1: Esempio di albero decisionale

Considerando un caso di classificazione, l'apprendimento degli alberi decisionali impiega una strategia di *divide et impera* conducendo una ricerca greedy per identificare i punti di divisione ottimali all'interno di un albero. Questo processo di suddivisione viene ripetuto in modo ricorsivo dall'alto verso il basso fino a quando tutti, o la maggior parte dei record, sono stati classificati sotto etichette di classe specifiche. Gli alberi più piccoli riescono più facilmente a ottenere nodi foglia puri, ovvero punti dati appartenenti a una singola classe. Tuttavia, man mano che un albero cresce in dimensioni, diventa sempre più difficile mantenere questa purezza, il che solitamente porta ad una quantità troppo ridotta di dati all'interno di un dato sottoalbero. Quando ciò accade, si parla di frammentazione dei dati, e spesso può portare all'*overfitting*.

Di conseguenza, gli alberi decisionali preferiscono alberi piccoli, in linea con il principio del “Rasoio di Occam” [9]. In altre parole, gli alberi decisionali dovrebbero aggiungere complessità solo se necessario, poiché la spiegazione più semplice è spesso la migliore. Per ridurre la complessità e prevenire l'*overfitting*, si usa solitamente il ***Pruning*** (in italiano potatura): un processo che rimuove i rami che si dividono su caratteristiche di bassa importanza.

3.1.1 Criteri di split

Sebbene ci siano diversi modi per selezionare il miglior attributo a ogni nodo, due metodi, l'***information gain*** e il ***Gini index***, sono criteri di divisione popolari per i modelli di alberi decisionali [11]. Questi aiutano a valutare la qualità di ciascuna condizione di test e quanto bene sarà in grado di classificare i campioni in una classe.

Per parlarne consideriamo in entrambi i casi di lavorare con il seguente dataset: $\mathcal{L} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$, $y_i \in \{1, \dots, N\}$, dove le y_i stanno a rappresentare le classi a cui i vari elementi \mathbf{x}_i possono appartenere.

Information gain

Per parlare di information gain è importante introdurre il concetto di ***entropia***. Innanzitutto consideriamo $\mathcal{L}_k \subset \mathcal{L}$ dove $\mathcal{L}_k = \{(\mathbf{x}, y) \in \mathcal{L} : y = k\}$. Allora si ha certamente che $\mathcal{L} = \bigcup_{k=1}^N \mathcal{L}_k$. Definiamo la seguente probabilità: $p_k := \frac{|\mathcal{L}_k|}{|\mathcal{L}|}$. Vogliamo definire l’“*impurità*” in base alla vicinanza alla distribuzione uniforme utilizzando la divergenza di Kullback-Leibler. Essa risulta essere una misura non simmetrica (motivo per il quale non può essere definita una metrica) della differenza tra due misure di probabilità P e Q . Indicheremo quindi $KL(P||Q)$ per indicare la misura dell’informazione persa quando Q è usata per approssimare P .

Ponendo dunque $q_1 = q_2 = \dots = q_N$ la distribuzione uniforme, possiamo scrivere:

$$KL(P||Q) = \sum_{k=1}^N p_k \log_2 \left(\frac{p_k}{q_k} \right) \geq 0$$

La non negatività della divergenza di Kullback-Leibler può essere derivata dalla diseguaglianza di Jensen.

$$KL(P||Q) = \sum_k p_k \log_2(p_k) - p_k \log_2(q_k)$$

Poiché $q_k = \frac{1}{N}$, otteniamo allora:

$$\begin{aligned} KL(P||Q) &= \sum_k p_k \log_2(p_k) + p_k \log_2(N) = \\ &= \sum_k p_k \log_2(p_k) + \log_2(N) \sum_k p_k = \\ &= \sum_k p_k \log_2(p_k) + \log_2(N) \end{aligned}$$

Dove abbiamo sfruttato il fatto che $\log_2(N) = \text{cost}$ e che $\sum_k p_k = 1$. Massimizzare $KL(P||Q)$ risulta perciò equivalente a massimizzare il primo addendo, ovvero a minimizzare l'opposto di esso:

$$\max_P KL(P||Q) = \max_P \sum_k p_k \log_2(p_k) = \min_P - \sum_k p_k \log_2(p_k)$$

A questo punto possiamo porre $H(\mathcal{L}) := -\sum_k p_k \log_2(p_k)$, detta entropia dell'insieme \mathcal{L} . Possiamo osservare che $0 \leq H \leq 1$:

- Se tutti i valori nel dataset appartengono ad una stessa classe, allora si avrà $H(\mathcal{L}) = 0$;
- Se metà dei campioni sono classificati in una classe e metà in un'altra, allora si avrà $H(\mathcal{L}) = 1$.

Dunque l'attributo (o variabile indipendente) che garantisce la minore entropia è quella che permette di trovare il punto di split ottimale e perciò l'albero decisionale migliore. Quindi se definiamo il guadagno di informazione come la differenza di entropia prima e dopo una divisione su un dato attributo, allora l'attributo con il maggiore guadagno di informazione produrrà la migliore divisione, poiché svolge il miglior lavoro nel classificare categoricamente i dati di addestramento.

A questo punto possiamo definire l'*information gain* come la differenza tra l'entropia del nodo genitore con la media di quelle dei nodi figli. Ossia:

$$IG(\mathcal{L}) = H(\mathcal{L}) - \sum_{k=1}^N \frac{|\mathcal{L}_k|}{|\mathcal{L}|} H(\mathcal{L}_k)$$

Questa sarà la quantità da massimizzare.

Gini Index

Mentre l'information gain misurava la purezza e l'impurità del nodo confrontando la distribuzione presente con quella uniforme, il Gini index (detto anche *Gini Impurity*) misura la probabilità che un'istanza casuale venga classificata erroneamente quando scelta a caso. Dunque più basso è il Gini index, migliore è la suddivisione, dal momento che minore è la probabilità di classificazione errata.

Consideriamo $\mathcal{L}, \mathcal{L}_k, p_k$ definiti come prima. Allora possiamo definire il Gini index come segue:

$$G(\mathcal{L}) := \sum_{k=1}^N p_k(1 - p_k) = \sum_{k=1}^N p_k - \sum_{k=1}^N p_k^2 = 1 - \sum_{k=1}^N p_k^2$$

Nel caso binario se chiamiamo p la probabilità di una delle due classi, allora otteniamo che il Gini index è:

$$G(\mathcal{L}) = 1 - p^2 - (1 - p)^2 = 2p - p^2 = 2p(1 - p)$$

Il cui grafico è il seguente:

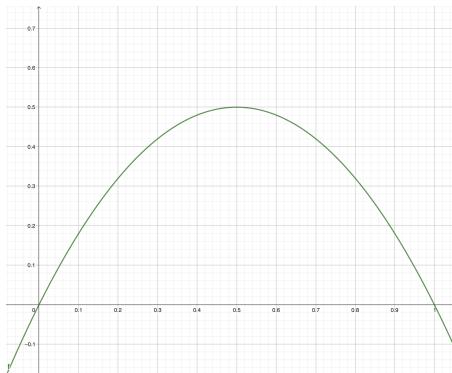


Figura 3.2: Gini index nel caso con due classi

Osserviamo che esso risulta essere una funzione non negativa per $0 \leq p \leq 1$ e raggiunge il suo massimo per $p = 0.5$. Inoltre è perfettamente ragionevole che si annulli per $p = 0$ e $p = 1$, dal momento che la probabilità di classificazione errata in tali casi risulta essere banalmente nulla.

Possiamo poi andare a definire il *Gini gain* come la differenza tra il Gini impurity dell'insieme prima della separazione e quello calcolato dopo la suddivisione. Il punto di separazione ottimale sarà quello che massimizza tale quantità.

3.1.2 Overfitting e Pruning

Come abbiamo già visto in precedenza l'overfitting è un problema da tenere in considerazione quando si costruiscono algoritmi di machine learning. Nel nostro caso in particolare (come descritto in [13]), per adattarsi anche ai dati rumorosi, l'albero crea nuovi nodi che vanno a complicare sempre di più la struttura. In questo modo quindi il modello risulterebbe perfetto sui training set, ma poco efficace nella previsione di nuovi dati.

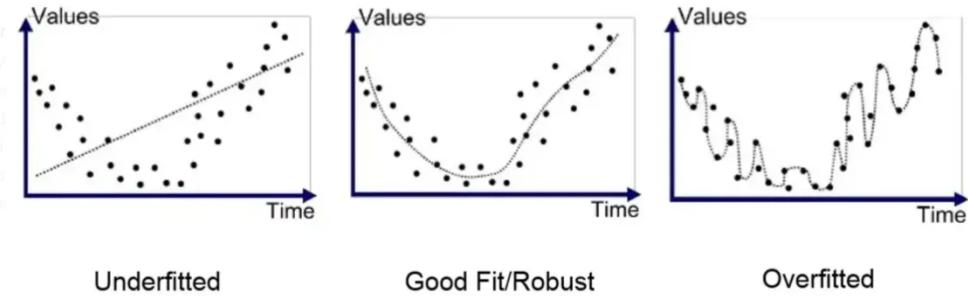


Figura 3.3: Esempi di modelli previsionali partendo da uno stesso set di dati

Una tecnica per prevenire questo spiacevole inconveniente con i decision tree è quella del ***Pruning***. Come avevamo detto precedentemente è una tecnica in grado di rimuovere parti dell'albero decisionale per impedirgli di crescere fino alla sua massima profondità. Cioè vengono rimosse quelle parti che non hanno il potere di classificare le istanze. Il pruning a sua volta può essere di due tipologie: il *pre-pruning* e il *post-pruning*.

Pre-Pruning

E' una tecnica chiamata anche *Early Stopping* o *Forward Pruning*. Ciò che si fa è fermare anticipatamente la crescita dell'albero prima che raggiunga la sua massima profondità. Si impedisce cioè che si vengano a formare dei

rami non significanti. E' prevista la regolazione di alcuni iperparametri prima dell'addestramento del modello. Essi sono parametri il cui valore (che non cambia mai) permette di controllare il processo di apprendimento.

Alcuni di quelli che si possono avere sono:

- ***min_samples_split***: specifica il minimo numero di elementi che un nodo deve avere affinché venga suddiviso. Consideriamo il caso in cui il parametro *min_samples_split* è impostato a 50. Qualsiasi nodo con meno di 50 campioni non verrà ulteriormente suddiviso. Pertanto, tutti i nodi terminali avranno meno di 50 campioni, e tutti i nodi interni avranno più di 50 campioni (o uguali).
- ***max_depth***: specifica il massimo livello di profondità che può avere l'albero. Se non è determinato, allora l'albero cresce fino a raggiungere un livello di foglie pure o finché le foglie hanno un numero minore di dati pari a ***min_samples_split***. Ovviamente, più è alto il valore di *max_depth*, più l'albero sarà complesso. Si avrà quindi una diminuzione dell'errore ma potrebbe portare a casi di overfitting.
- ***min_samples_leaf***: specifica il numero minimo di dati che si devono avere in ogni foglia. Consideriamo uno scenario in cui il parametro *min_samples_leaf* di un modello è impostato a 5 e il nodo corrente ha 20 campioni. Il modello deve decidere se continuare a suddividere questo nodo o fermarsi qui. Per considerare questo parametro, il modello suddivide il nodo e verifica il numero di campioni nei rami appena creati. Se entrambi hanno più di 5 elementi, allora il modello lo suddividerà. Se anche solo uno due ha meno di 5 campioni, allora la suddivisione non verrà effettuata e tale nodo diventerà una foglia.

Post-Pruning

E' detta anche *Backward Pruning* ed è una tecnica che prevede l'eliminazione di rami da alberi completamente cresciuti per ridurne la complessità. In questo modo il modello potrebbe aumentare leggermente gli errori di addestramento, ma riduce drasticamente gli errori di test.

I rami da rimuovere vengono scelti con la tecnica del ***Cost Complexity Pruning (CCP)***. L'algoritmo è parametrizzato dal parametro $\alpha \geq 0$ detto ***parametro di complessità***. Esso lavora andando a calcolare un *punteggio* basato sulla *somma dei residui quadrati* (SSR) dell'albero o dei sottoalberi e su una penalità per la complessità dell'albero (T), che è una funzione del numero di foglie dell'albero o del sottoalbero. L'SSR aumenta man mano

che gli alberi diventano più corti e la penalità per la Complessità dell'Albero compensa la differenza nel numero di foglie.

Otteniamo così la seguente formula per il calcolo del punteggio:

$$tree_score = SSR + T\alpha$$

3.2 Regression trees

Adesso che abbiamo capito cos'è un albero decisionale, possiamo ormai capire cosa sia un **regression tree** (o albero di regressione) [10]. Infatti finora si è visto come utilizzare gli alberi decisionali per compiti di classificazione. Ebbene, i regression trees non sono altro che alberi decisionali usati per compiti di regressione. Dunque esso può essere utilizzato per predire output con valori continui invece di output discreti.

I criteri di divisione precedentemente affrontati - l'information gain e il gini index - non sono più evidentemente utilizzabili. Dobbiamo perciò introdurre una nuova misura che ci permetta di capire quanto le nostre previsioni divergano dal target originale, ossia il **Mean Squared Error** (o **MSE**, in italiano errore quadratico medio).

Definizione 3.2.1. (Mean Squared Error) Il MSE è una misura della media dei quadrati degli errori, cioè la differenza quadratica media tra i valori stimati e il valore effettivo, ossia:

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

dove y_i è il valore effettivo, \hat{y}_i è la previsione e n è il numero di target.

Dal momento che ciò che interessa è solo quanto la previsione varia dal target - e non in quale direzione - si eleva al quadrato la differenza e si divide l'intera somma per numero totale di record.

L'MSE è una funzione di rischio, corrispondente al valore atteso della perdita quadratica dell'errore. Poiché esso è derivato dalla distanza euclidea, allora risulta essere un valore sempre positivo che diminuisce man mano che la distanza tra il valore predetto e quello esatto si riduce, ossia che l'errore tende a zero.

A questo punto per la costruzione di un albero di regressione si consideri il seguente esempio:

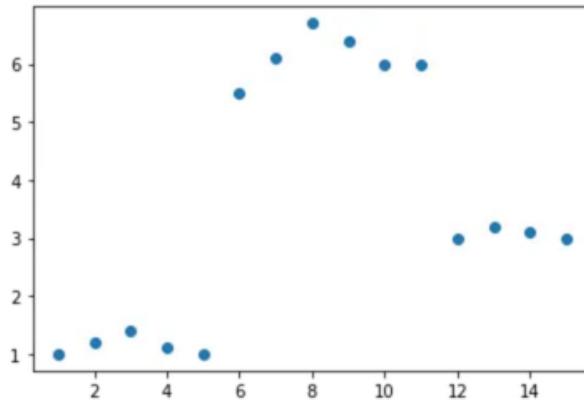


Figura 3.4: Dataset dove sia X che Y sono variabili continue

Ordiniamo i valori rappresentati in una tabella disponendoli in ordine crescente rispetto alla variabile X come segue:

X	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Y	1	1.2	1.4	1.1	1	5.5	6.1	6.7	6.4	6	6	3	3.2	3.1

Adesso calcoliamo la media tra coppie di valori consecutivi sull'asse X, come segue:

$$\frac{1+2}{2} = 1.5, \quad \frac{2+3}{2} = 2.5, \quad \dots$$

Questi rappresentano tutti i possibili punti di split. Per ognuno di questi punti si va poi a calcolare il MSE che ne risulta (dal momento che possiamo calcolare il valor medio dell'insieme $\{y_i : x_i < \hat{x}\}$, dove \hat{x} sono i vari 1.5, 2.5, ...) e si va a scegliere come punto di split quello che è in grado di minimizzarlo. In particolare in questo caso risulta che la scelta ottimale è $\hat{x} = 5.5$. Otteniamo così che il dataset è diviso in due semipiani: $x > 5.5$ e $x < 5.5$.

Dunque l'idea di base dietro all'algoritmo è quella di trovare il punto nella variabile indipendente per suddividere il set di dati in 2 parti, in modo che l'errore quadratico medio sia minimizzato in quel punto. L'algoritmo esegue questo processo in modo ripetitivo formando perciò una struttura ad albero.

Ragionando in questo modo troviamo il seguente albero di regressione per il precedente esempio:

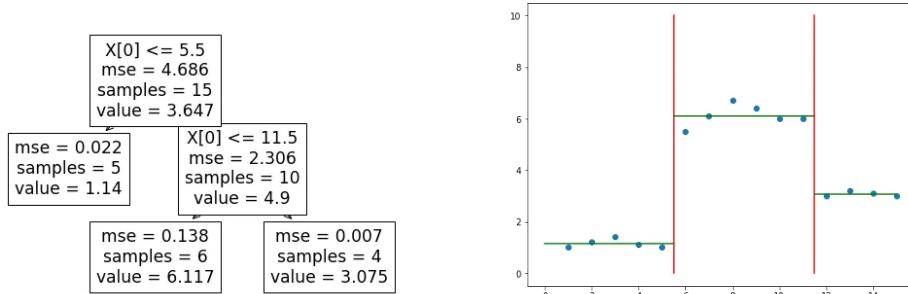


Figura 3.5: A sinistra il regression tree finale; a destra la rispettiva visualizzazione grafica

3.2.1 Caso multidimensionale

Se si sta trattando con dati multidimensionali il ragionamento non cambia. Supponendo per esempio di avere tre variabili simili alla variabile indipendente X del precedente esempio, allora quello che si fa è andare ad ordinare i dati separatamente in base alle tre variabili, e i punti che minimizzano il MSE saranno calcolate per tutte e 3 le variabili. Tra le 3 variabili e i punti di split studiati si vanno a scegliere quelli in grado di minimizzare il MSE.

3.2.2 Caso con variabili indipendenti discrete

Se la variabile indipendente non è più continua come nell'esempio precedentemente affrontato, ma assume valori discreti, allora quello che si fa è andare a suddividere i nodi in modo binario. Supponiamo ad esempio che una variabile indipendente sia la dimensione di un tumore e che esso possa appartenere a 3 diverse categorie: piccolo, medio e grande. Allora l'albero suddividerà il set di dati in base a se la dimensione del tumore è piccola, grande o media, oppure potrà anche combinare più valori basandosi sulla domanda che riduce maggiormente l'MSE.

3.2.3 Overfitting e cross-validation

Anche con gli alberi di regressione possono verificarsi casi di overfitting. In questo caso la tecnica più utilizzata per ovviare al problema è quella della ***Cross Validation***, detta anche ***k-fold validation***.

Si suddivide il training set in k parti di uguale dimensione (di solito in 5 o 10). Poi si seleziona 1 parte da usare come *validation set*, mentre le restanti $k - 1$ vengono impiegate per costruire il *training dataset*. Successivamente si avvia il processo induuttivo sul training set per la costruzione dell'albero e si verifica l'efficacia predittiva utilizzando il validation set. Si calcola in questo modo un **valore predittivo** dell'albero parziale.

Si ripete la stessa procedura k volte, selezionando ogni volta un insieme diverso come validation set, ottenendo k alberi parziali, ognuno con un relativo valore predittivo. Infine si calcola la media dei valori predittivi e viene scelto l'albero parziale con il valore predittivo più alto e lo si utilizza come modello. Esso sarà quello che risente meno del rumore causato dagli attributi irrilevanti presenti nel training set.

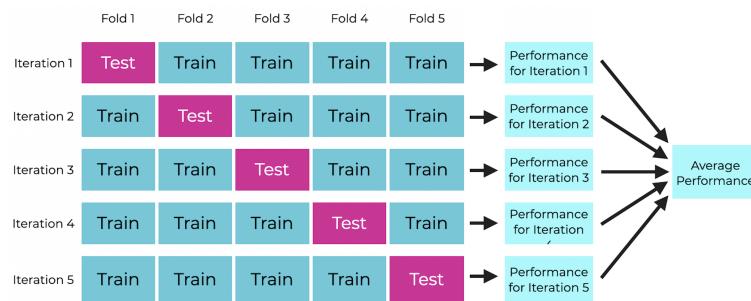


Figura 3.6: La cross validation con $k = 5$

3.3 Bagging

Gli ultimi 30 anni hanno visto gli sviluppi e le applicazioni in moltissimi aspetti del **bagging** [14] [15] e delle **Random Forests**. Invece di scegliere un singolo stimatore, il bagging utilizza un insieme di stimatori addestrati su campioni *bootstrap*, ossia campioni selezionati casualmente da un dataset iniziale con la possibilità di essere scelti più volte. Una volta ottenuti questi stimatori ne calcola la media, in modo da migliorare la robustezza dello stimatore finale.

Consideriamo un training set come segue:

$$\mathcal{L} = \{(\mathbf{x}_i, y_i), i = 1, \dots, N\}$$

dove le y_i possono essere discrete (dunque delle classi) oppure anche dei numeri. Supponiamo poi di avere una funzione che partendo da questo training set è in grado di fare previsioni: $\varphi(\mathbf{x}, \mathcal{L})$. Essa cioè partendo dalle conoscenze acquisite dal training set e dalla variabile indipendente di input \mathbf{x} è in grado di predire la y .

Ora supponiamo di avere un insieme di training set $\{\mathcal{L}_k\}_{k=1}^K$, ognuno di essi formato da N elementi indipendenti a partire dalla stessa distribuzione sottostante \mathcal{L} . L'obiettivo che ci si pone è quello di usare gli $\{\mathcal{L}_k\}_{k=1}^K$ per ottenere un migliore stimatore della singola $\varphi(\mathbf{x}, \mathcal{L})$. L'unica restrizione che abbiamo è che ci è possibile lavorare soltanto con i $\{\varphi(\mathbf{x}, \mathcal{L}_k)\}_{k=1}^K$.

Se y è una variabile numerica, allora una procedura semplice e immediata da implementare è quella di porre:

$$\varphi_A(\mathbf{x}, \mathcal{L}) = \frac{1}{K} \sum_{k=1}^K \varphi(\mathbf{x}, \mathcal{L}_k) = \mathbb{E}_{\mathcal{L}}[\varphi(\mathbf{x}, \mathcal{L}_k)]$$

dove con $\mathbb{E}_{\mathcal{L}}[\varphi(\mathbf{x}, \mathcal{L}_k)]$ si intende la media su \mathcal{L} dei valori assunti da $\varphi(\mathbf{x}, \mathcal{L}_k)$, mentre con la A al pedice di φ si intende il processo di aggregazione.

Se invece l'output che ci viene restituito dalla funzione $\varphi(\mathbf{x}, \mathcal{L})$ è una classe $j \in \{1, \dots, J\}$, allora un possibile metodo di aggregazione può essere quello implementato attraverso un sistema di votazione. Poniamo

$$N_j = |\{k : \varphi(\mathbf{x}, \mathcal{L}_k) = j\}|$$

Adesso poniamo $\varphi_A(\mathbf{x}, \mathcal{L}) = \arg \max_j N_j$, cioè il parametro j che rende massimo N_j .

Tuttavia, di solito abbiamo a che fare con un solo learning set \mathcal{L} , difficilmente si hanno dei duplicati dello stesso. Ma un processo che ci porta ad avere $\varphi_A(\mathbf{x}, \mathcal{L})$ può comunque essere ottenuto grazie al bootstrap, che come abbiamo visto precedentemente consiste nel creare nuovi training set $\{\mathcal{L}^{(B)}\}$ estraendo casualmente N elementi da \mathcal{L} con possibilità di replica. In questo modo possiamo costruirci la successione $\{\varphi(\mathbf{x}, \mathcal{L}^{(B)})\}$.

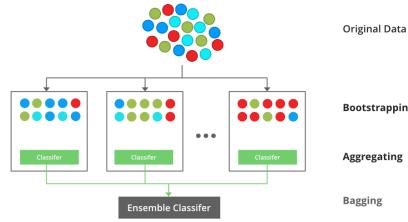


Figura 3.7: Schematizzazione del bagging

Dunque se y è un valore numerico, allora poniamo

$$\varphi_B(\mathbf{x}) = \mathbb{E}_B[\varphi(\mathbf{x}, \mathcal{L}^{(B)})]$$

mentre se y sta a indicare una classe/categoria, allora si adotta il precedente metodo di votazione.

Tale procedura è chiamata ***bootstrap aggregating*** ed è particolarmente consigliata nel caso in cui il dataset risulta instabile, cioè quando piccole variazioni di \mathcal{L} comportano variazioni non trascurabili di $\varphi(\mathbf{x}, \mathcal{L})$.

Per dimostrarlo analizziamo solo il caso in cui y assume valori numerici. Allora abbiamo:

$$\varphi_A(\mathbf{x}) = \mathbb{E}_{\mathcal{L}}[\varphi(\mathbf{x}, \mathcal{L})]$$

dove \mathbf{x} è un valore di input, mentre y è il valore di output. Allora :

$$\mathbb{E}_{\mathcal{L}}\{[y - \varphi(\mathbf{x}, \mathcal{L})]^2\} = y^2 - 2y\mathbb{E}_{\mathcal{L}}[\varphi(\mathbf{x}, \mathcal{L})] + \mathbb{E}_{\mathcal{L}}[\varphi^2(\mathbf{x}, \mathcal{L})]$$

Ricordando che $\mathbb{E}_{\mathcal{L}}[\varphi(\mathbf{x}, \mathcal{L})] = \varphi_A(\mathbf{x})$ e applicando la diseguaglianza di Jensen ($\mathbb{E}[Z^2] \geq \mathbb{E}^2[Z]$), otteniamo la seguente:

$$\mathbb{E}_{\mathcal{L}}\{[y - \varphi(\mathbf{x}, \mathcal{L})]^2\} \geq (y - \varphi_A(\mathbf{x}))^2$$

Integrando da entrambe le parti sulla distribuzione di probabilità congiunta $\mathbb{P}(y, \mathbf{x})$ si ottiene che l'errore quadratico medio di $\varphi_A(\mathbf{x})$ è minore dell'errore quadratico medio su \mathcal{L} di $\varphi(\mathbf{x}, \mathcal{L})$. Di quanto sia minore dipende dalla diseguaglianza di Jensen.

L'effetto dell'instabilità è chiaro. Se $\varphi(\mathbf{x}, \mathcal{L})$ non varia molto con le copie di \mathcal{L} , allora i due membri saranno circa uguali. I risultati migliori si ottengono quando invece $\varphi(\mathbf{x}, \mathcal{L})$ ha delle grosse oscillazioni al variare delle copie di \mathcal{L} .

3.4 Random Forest

La tecnica del **Random Forest** può essere vista come un miglioramento del bagging, utilizzando come stimatori degli alberi decisionali [16]. Infatti il problema del bagging è la limitazione della riduzione della varianza, dal momento che iterare uno stesso algoritmo di apprendimento su differenti sottoinsiemi di un dataset potrebbe portare alla formazione di predittori altamente correlati.

Infatti se gli stimatori f_1, f_2, \dots, f_B con varianza σ^2 fossero indipendenti e identicamente distribuiti, allora:

$$Var(g) = Var\left(\frac{1}{B} \sum_{b=1}^B f_b\right) = \frac{1}{B} \sigma^2$$

Ma se invece risultano correlati, con coefficiente di correlazione ρ e stessa varianza di prima, otteniamo:

$$Var(g) = Var\left(\frac{1}{B} \sum_{b=1}^B f_b\right) = \frac{1}{B^2} \left(\sum_{b=1}^B Var(f_b) + 2 \sum_{b=c} Cov(f_b, f_c) \right)$$

Ma dal momento che tutti gli stimatori hanno stessa varianza, banalmente si ha che:

$$\sum_{b=1}^B Var(f_b) = B\sigma^2$$

Invece per quanto riguarda il secondo addendo, dobbiamo calcolarci il numero di possibili coppie di stimatori scelti tra i B possibili, ossia: $\binom{B}{2} = \frac{B(B-1)}{2}$. Perciò:

$$2 \sum_{b=c} Cov(f_b, f_c) = B(B-1)\rho\sigma^2$$

Segue che:

$$\begin{aligned} Var(g) &= \frac{1}{B^2} (B\sigma^2 + B(B-1)\rho\sigma^2) = \\ &= \frac{\sigma^2}{B} (1 + (B-1)\rho) = \\ &= \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2 \end{aligned}$$

Osserviamo dunque che la varianza di uno stimatore medio dipende dal numero di stimatori base e dalla loro correlazione. Sebbene sia possibile far tendere il secondo termine a zero semplicemente aumentando il numero degli stimatori, il primo termine rimane costante se gli stimatori non sono

indipendenti. In pratica, poiché la maggior parte dei campioni bootstrap è molto simile, gli alberi decisionali addestrati su questi set di campioni sono spesso simili e correlati fra loro. Questa è la ragione per la quale la tecnica del bagging con i decision trees non porta a ottimi risultati in termini di predizione.

L'algoritmo di Random Forest tenta di agire anche sul primo termine della precedente equazione, ossia $\rho\sigma^2$. Infatti per diminuire la correlazione tra gli stimatori, si fa uso di quello che viene chiamato **random subset projection** (ossia: proiezione di sottoinsiemi casuali) o **random feature projection** (ossia: proiezione di variabili casuali) durante il processo di formazione dell'albero. Quindi invece di considerare tutte le variabili applicate, ad ogni punto di split ogni decision tree sceglie un sottoinsieme di variabili da considerare.

Inoltre in questo modo è spesso possibile far crescere gli alberi fino alla loro massima profondità, senza la necessità di utilizzare il pruning. Grazie a questo procedimento si riesce a ridurre anche in modo significativo $\rho\sigma^2$.

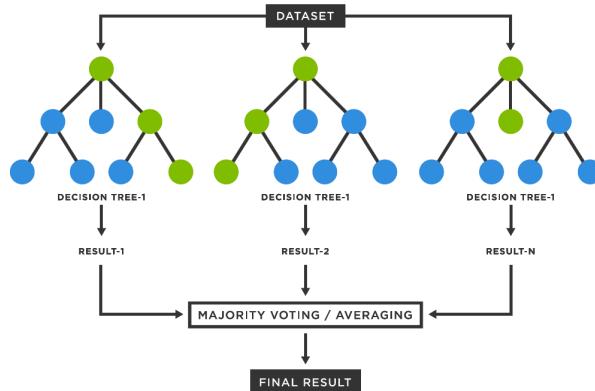


Figura 3.8: Schematizzazione del Random Forest

Tuttavia tale procedimento potrebbe portare ad un aumento del bias, dal momento che ogni singolo albero ha meno informazioni per fare previsioni non potendo utilizzare tutte le variabili disponibili ad ogni split. Pertanto è essenziale selezionare attentamente il numero di variabili da considerare ad ogni split per bilanciare bias e varianza (si parla proprio di trade off bias-varianza), tenendo conto che:

- Troppe poche variabili potrebbero portare a un aumento del bias
- Troppe variabili portano a un aumento della varianza

Dunque il processo è così schematizzabile:

1. Generare B bootstrapped training set
2. Costruire un decision tree per ogni training set e farlo crescere fino alla sua massima profondità
3. Durante la crescita dell'albero, selezionare m variabili ad ogni punto di split (*random feature projection*)
4. Combinare i B decision trees: per la regressione se ne calcoli la media, per la classificazione si proceda per votazione

Se si tratta di regressione, allora solitamente l'iperparametro m si sceglie nel seguente modo: $1 \leq m \leq \frac{p}{3}$, dove p è il numero di variabili. Se invece si tratta di classificazione si ha: $1 \leq m \leq \sqrt{p}$.

3.4.1 Out of Bag Error

L'***Out-of-Bag Error*** [18] è un metodo per stimare la performance del modello senza dover riservare un test set separato. Infatti sappiamo che grazie alla tecnica del bagging un certo numero di elementi presenti nel training set iniziale non vengono utilizzati nei bootstrapped training set. In particolare la probabilità che un punto $(x_i, y_i) \notin Boot_b$, dove $Boot_b$ rappresenta il b -esimo bootstrapped training set è:

$$\mathbb{P}((x_i, y_i) \notin Boot_b) = \left(1 - \frac{1}{N}\right)^N \longrightarrow e^{-1} \approx 37\%$$

Quindi circa il 37% dei dati iniziali non sono presenti nel b -esimo bootstrapped training set. E' una percentuale molto alta, che può essere utilizzata come test set. Chiamiamo quindi l'insieme di elementi non inclusi nel b -esimo bootstrapped training set ***Out-of-Bag sample*** (o ***OOB sample***). L'errore commesso dallo stimatore f_b è chiamato ***OOB error***.

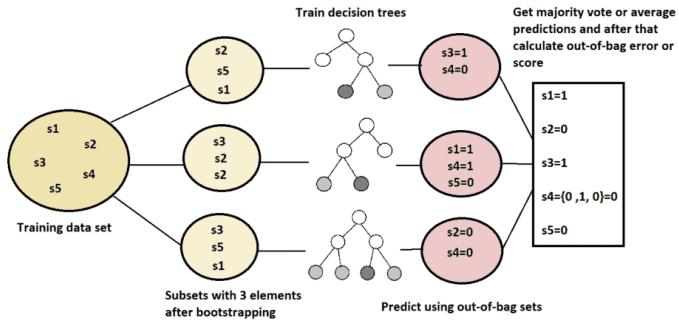


Figura 3.9: Formazione degli OOB samples e calcolo dell'OOB error

Il b -esimo OOB error è calcolato nel seguente modo:

$$err_{OOB,b} = \frac{1}{N_b} \sum_{i=1}^{N_b} Loss(y_{i,OOB}, f_b(x_{i,OOB}))$$

dove:

- N_b è il numero di dati nel b -esimo OOB sample;
- $y_{i,OOB}$ è il valore osservato per il dato $x_{i,OOB}$ nel b -esimo OOB sample;
- $f_b(x_{i,OOB})$ è la previsione del b -esimo modello;
- $Loss(y, \hat{y})$ è la funzione di perdita che misura la discrepanza tra il valore osservato y e il valore predetto \hat{y} .

Possiamo infine calcolare l'OOB error del Random Forest calcolando semplicemente la media dei vari OOB errors per ogni decision tree come segue:

$$err_{OOB} = \frac{1}{B} \sum_{b=1}^B \frac{1}{N_b} \sum_{i=1}^{N_b} Loss(y_{i,OOB}^b, f_b(x_{i,OOB}^b))$$

Capitolo 4

Applicazioni dell'SVR e Random Forest al BRISQUE

Scattare una foto è facile, ma scattarne una di alta qualità può essere un compito piuttosto difficile. E poi come si potrebbe valutare la qualità di un'immagine? Sicuramente l'occhio di un esperto ne sarebbe in grado. Tuttavia, esistono anche degli algoritmi che analizzando i pixel di un'immagine sono in grado di segnalare se essa sia sfocata oppure no. D'altra parte esistono alcuni particolari che sono impossibili da percepire per un algoritmo, in quanto non sono caratteristiche oggettive (come può essere la sfocatura), ma bensì caratteristiche soggettive.

4.1 Image Quality Assessment (IQA)

Gli algoritmi di ***Image Quality Assessment (IQA***) prendono in input una qualsiasi immagine e restituiscono in output un ***quality score***. Esistono tre diverse tipologie di questi algoritmi:

- ***Full Reference IQA***: Abbinata all'immagine di cui si deve valutare la qualità si possiede anche la stessa immagine non distorta. Questo tipo di algoritmi viene particolarmente usato quando si vuole valutare la qualità dell'immagine a seguito di un processo di compressione.
- ***Reduced-Reference IQA***: In questo caso si possiede solo l'immagine obiettivo e non quella “*pulita*”, ma solo alcune caratteristiche di essa da comparare con quelle dell'immagine distorta.
- ***Objective Blind o No Reference IQA*** L'algoritmo in questo caso prende in input solo l'immagine obiettivo e nient'altro.

In ogni caso l'output di questi algoritmi rappresenta spesso la probabilità che un occhio umano possa rilevare la differenza tra le due immagini, o un numero che quantifica la dissimilarità percettiva tra di esse. In alternativa, l'output può essere una mappa delle probabilità di rilevamento o dei valori di dissimilarità percettiva.

Le tipiche metriche di qualità dell'immagine sono il ***Mean Squared Error (MSE***, in italiano: errore quadratico medio) e il ***Peak Signal to Noise Ratio (PSNR***, in italiano: rapporto segnale rumore di picco). Queste metriche sono ancora utilizzate nonostante le loro notevoli limitazioni dovute alla loro particolare semplicità.

Siano $I(i, j)$ e $J(i, j)$ due funzioni che misurano l'intensità di un pixel dell'immagine nella posizione (i, j) (dal momento che spesso i pixel sono disposti su un grafico cartesiano). Allora definiamo l'errore quadratico medio come segue:

$$MSE[I(i, j), J(i, j)] := \frac{1}{MN} \sum_{i=1}^M \sum_{j=1}^N [I(i, j) - J(i, j)]^2$$

dove M e N sono rispettivamente l'altezza e la larghezza dell'immagine. Definiamo invece il PSNR così:

$$PSNR[I(i, j), J(i, j)] := 10 \log_{10} \frac{E^2}{MSE[I(i, j), J(i, j)]}$$

dove E è il massimo valore di intensità che un pixel può raggiungere. Dunque se l'immagine è in scala di grigi si avrà $E = 255$.

In letteratura (come nel paper [21]) si possono trovare diversi esempi di come queste misure si siano dimostrate inadeguate per la valutazione della qualità delle immagini, ed è proprio per questo motivo che ne sono state costruite delle altre.

Il grosso problema di tutte queste misure è che per essere effettuate necessitano di quello che solitamente si chiama “*ground truth*”, ossia delle etichette vere e corrette usate per addestrare i modelli di machine learning. Questo rappresenta una limitazione non indifferente dal momento che spesso non se ne dispone.

4.2 BRISQUE

Discuteremo in questo capitolo di un algoritmo No Reference IQA chiamato appunto ***Blind/Referenceless Image Spatial QUAlity Evaluator (BRI-SQUE***).

L’obiettivo che ci si pone è quello di giudicare la bontà di un’immagine in modo simile a quello che fa l’uomo, cercando di estrarre caratteristiche oggettive in questa direzione. Si è così scoperto (com’è riportato nel paper [20] pubblicato nel 2012 e nel tutorial [22]) che la distribuzione delle intensità dei pixel delle immagini naturali differisce da quella delle immagini distorte. Questa differenza nelle distribuzioni è molto più pronunciata quando si normalizzano le intensità dei pixel e se ne calcola la distribuzione su tali intensità normalizzate. In particolare, dopo la normalizzazione, le intensità dei pixel delle immagini naturali seguono una distribuzione gaussiana (curva a campana), mentre le intensità dei pixel delle immagini innaturali o distorte non lo fanno. La deviazione della distribuzione da una curva a campana ideale è quindi una misura della quantità di distorsione nell’immagine.

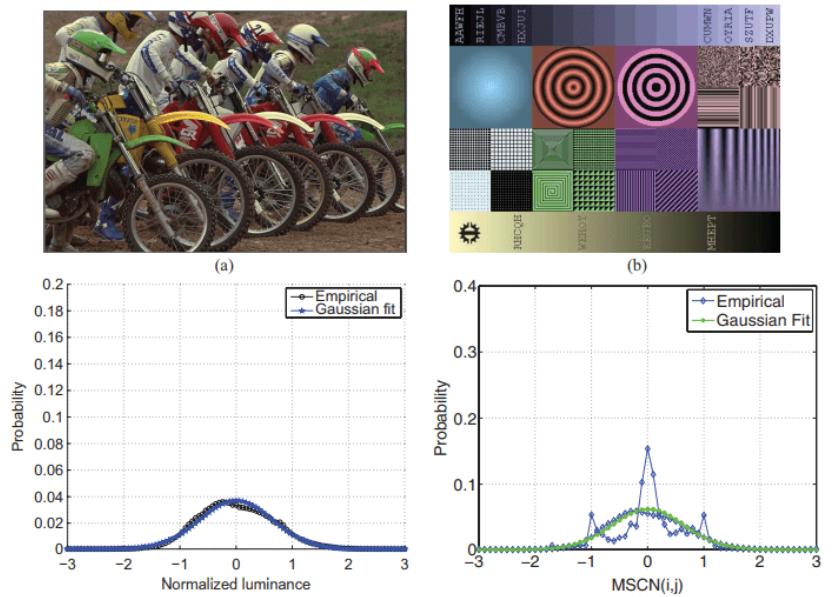


Figura 4.1: A sinistra vediamo una immagine naturale, che una volta normalizzata “*fitta*” la gaussiana. A destra un’immagine distorta che si discosta dalla curva a campana

4.2.1 Calcolo dei parametri **MSCN**

Quindi il problema si sposta alla domanda: come normalizzare un’immagine? Esistono diversi modi, quello che si utilizza per il metodo in questione è chiamato ***Mean Subtracted Contrast Normalization (MSCN)***. Per calcolare i coefficienti MSCN l’intensità dell’immagine $I(i, j)$ del pixel (i, j) è

trasformata nella “luminanza” $\hat{I}(i, j)$ come segue:

$$\hat{I}(i, j) = \frac{I(i, j) - \mu(i, j)}{\sigma(i, j) + C}$$

dove $i = 1, \dots, M, j = 1, \dots, N$ sono gli indici spaziali, M e N sono rispettivamente l'altezza e la larghezza dell'immagine, $C = 1$ è una costante usata per prevenire casi di instabilità dovuti a quando il denominatore tende a zero e invece le due funzioni μ e σ , dette rispettivamente **local mean field** (campo di media locale) e **local variance field** (campo di varianza locale) sono definite come segue:

$$\begin{aligned}\mu &= \mathbf{W} \cdot \mathbf{I} \\ \sigma &= \sqrt{\mathbf{W} \cdot (\mathbf{I} - \mu)^2}\end{aligned}$$

dove \mathbf{W} è un filtro gaussiano 2D spesso utilizzato nell'elaborazione delle immagini per applicare una sfocatura o per ridurre il rumore. Quindi μ rappresenta la sfocatura gaussiana dell'immagine, mentre σ è la sfocatura gaussiana del quadrato della differenza tra l'immagine e μ .

Il calcolo precedentemente mostrato può essere visualizzato graficamente dalla seguente immagine:

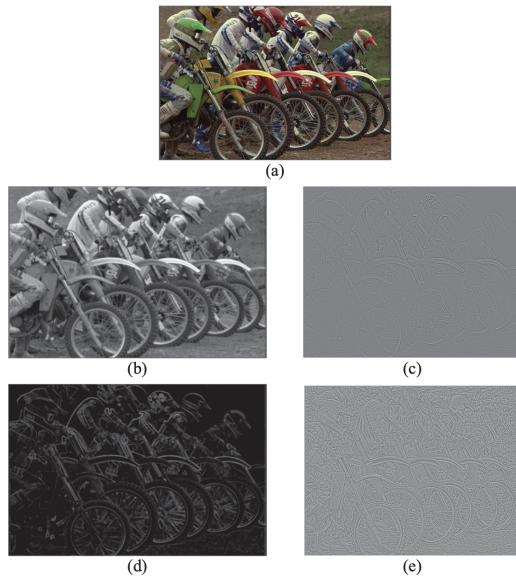


Figura 4.2: Effetto della procedura di normalizzazione. (a) Immagine originale. (b) Local mean field μ . (c) $I - \mu$. (d) Local variance field σ . (e) Coefficienti MSCN ($(I - \mu)/\sigma$).

4.2.2 Calcolo dei prodotti a coppie

MSCN fornisce una buona normalizzazione per le intensità dei pixel. Tuttavia, la differenza tra le immagini naturali e quelle distorte non è limitata alle distribuzioni delle intensità dei pixel, ma include anche la relazione tra un pixel e i suoi vicini.

Per comprendere tale relazione si utilizzano dei prodotti a coppie dell'immagine MSCN con una sua versione “shiftata”. Vengono usati quattro orientamenti per trovare i prodotti a coppie: orizzontale (H), verticale (V), diagonale principale (D_1) e diagonale secondaria (D_2). In particolare si ha:

$$\begin{aligned} H(i, j) &= \hat{I}(i, j)\hat{I}(i, j + 1) \\ V(i, j) &= \hat{I}(i, j)\hat{I}(i + 1, j) \\ D_1(i, j) &= \hat{I}(i, j)\hat{I}(i + 1, j + 1) \\ D_2(i, j) &= \hat{I}(i, j)\hat{I}(i + 1, j - 1) \end{aligned}$$

Possiamo visualizzare graficamente la disposizione di tali vicini come segue:

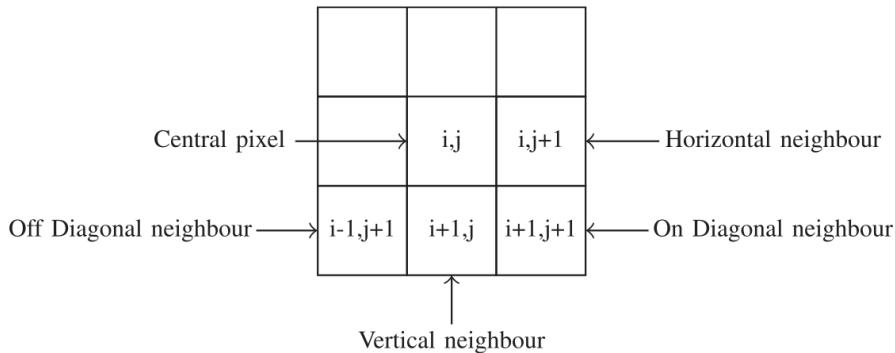


Figura 4.3: Schematizzazione degli orientamenti usati per il calcolo dei prodotti a coppie

4.2.3 Vettore delle 36 features

Fino ad ora abbiamo semplicemente derivato 5 diverse immagini partendo da una sola: un'immagine MSCN e quattro immagini dei prodotti a coppie per analizzare le relazioni con i vicini. Adesso lo scopo è quello di utilizzare queste 5 immagini per ottenere da esse un vettore composto da 36 “features”. La comodità e la potenza di questo metodo risiede proprio in queste, dal momento che esso può prendere in input un'immagine di qualsiasi dimensione,

ma il risultato sarà sempre e comunque un vettore $\mathbf{v} \in \mathbb{R}^{36}$. Da tale vettore sarà poi possibile desumere il quality score. In questo modo se si utilizzano modelli di apprendimento che necessitano di un numero fisso di features, ci si libera dalla restrizione di dover usare immagini della stessa dimensione.

I primi due elementi del vettore risultante sono calcolati adattando l'immagine MSCN a una distribuzione gaussiana generalizzata (in inglese *Generalized Gaussian Distribution*, GGD). Una GGD è caratterizzata da due parametri: uno per la forma e uno per la varianza.

Successivamente una distribuzione gaussiana generalizzata e asimmetrica (in inglese *Asymmetric Generalized Gaussian Distribution*, AGGD) viene adattata a ciascuna delle 4 immagini ottenute dai prodotti a coppie. Le AGGD sono forme asimmetriche delle GGD e hanno 4 parametri: forma, media, varianza sinistra e varianza destra. Arriviamo così ad avere 18 parametri: 2 per la GGD dall'immagine MSCN e $4 \times 4 = 16$ dalle quattro AGGD.

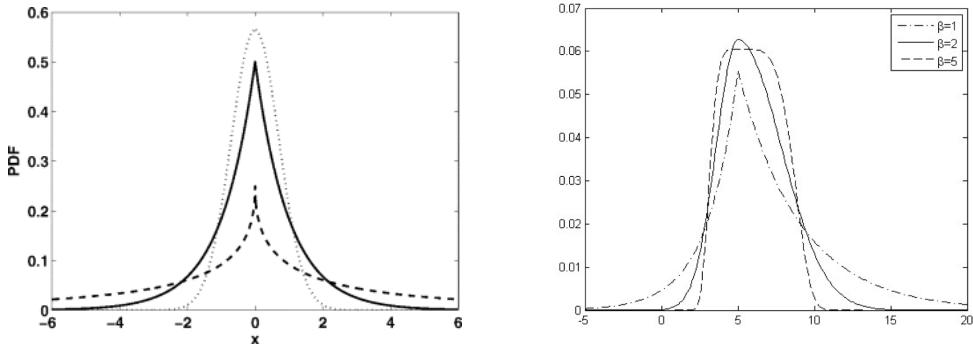


Figura 4.4: A sinistra il grafico di una distribuzione gaussiana generalizzata, mentre a sinistra il grafico di una distribuzione gaussiana generalizzata asimmetrica con parametri $\mu = 5, \alpha_1 = 2, \alpha_2 = 4, \beta \in \{1, 2, 5\}$

Dopodiché si effettua un ridimensionamento dell'immagine iniziale e se ne dimezzano le dimensioni. Su questa nuova immagine si seguono gli stessi passi precedenti per il calcolo di 18 nuovi parametri. Essi si aggiungono ai 18 trovati precedentemente, andando a completare il vettore $\mathbf{v} \in \mathbb{R}^{36}$.

In questo modo quindi partendo per esempio da un'immagine di $1080 \times 720 = 777600$ pixels e dunque da una $\mathbf{I} \in \mathbb{R}^{777600}$ siamo riusciti a spostare il problema in \mathbb{R}^{36} , semplificando il lavoro da fare per ottenere il quality score, ma soprattutto riducendo di molto le dimensioni in ottica di efficienza per la parte di training di modelli data driven.

4.2.4 Calcolo del quality score

Quello che andremo a fare è predisporre un dataset di immagini (a cui sono state associate delle etichette rappresentanti un quality score) suddiviso in una parte di train e una di test. Queste vengono convertite in vettori di 36 features. A partire da questi e dalle etichette viene addestrata un algoritmo di apprendimento (nel nostro caso una SVR e un Random Forest), in modo tale da renderlo capace di predire delle future etichette dandogli in input i vettori di features ottenuti a partire da nuove immagini.

4.3 Applicazione con dataset Helsinki Deblur Challenge

Nell'[Helsinki Deblur Challenge](#) era richiesto di predisporre un algoritmo in grado di mettere a fuoco delle immagini sfocate. E' stato dato ai partecipanti un dataset di immagini di scritte su tre righe prive di significato con qualità visive differenti (alcune di esse erano anche prive di sfocatura).

Ad ogni immagine è infatti stata associata una label, ossia un valore numerico che rappresenta la percentuale di caratteri ben identificati da un OCR (dall'inglese optical character recognition). Un OCR, detto anche riconoscimento ottico dei caratteri, è il processo che converte un'immagine di testo in un formato di testo leggibile ad una macchina.

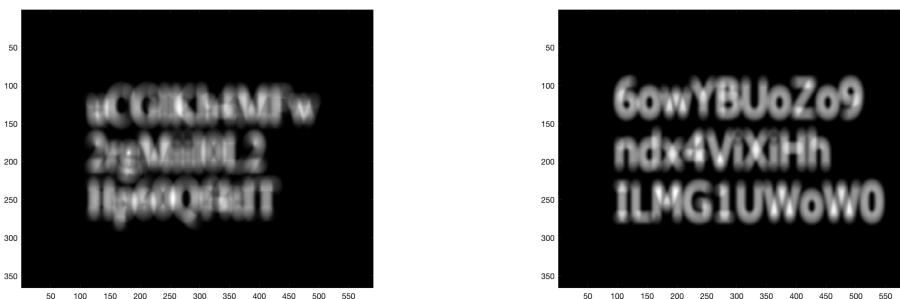


Figura 4.5: A sinistra esempio di immagine con score 11 e a destra esempio di immagine con score 42

L'obiettivo di questa tesi è addestrare una SVR e una Random Forest utilizzando come input i vettori di features associati a ciascuna immagine del dataset di addestramento e i rispettivi valori delle etichette. Successivamente,

si danno in input ai due modelli predittivi le immagini del test set e si confrontano gli output così ottenuti con le rispettive etichette di test.

Per effettuare la sperimentazione ci siamo serviti di Matlab e la macchina usata è un computer con processore Apple M1 ad 8 core, 8GB di RAM e sistema operativo MacOS Sonoma Versione 14.5.

4.3.1 Predisposizione ambiente di lavoro

All'inizio dello script è necessario rimuovere tutte le variabili presenti nell'area di lavoro e chiudere tutte le finestre di figura aperte in modo da assicurarsi di lavorare in un ambiente pulito e ordinato.

Solo successivamente si carica il dataset già suddiviso in una parte di train e una parte di test. Una volta caricato il dataset sarà possibile vedere nell'ambiente di lavoro le quattro matrici appena caricate:

- Immagini di train: una matrice di dimensione 215350×1000 che contiene le 1000 immagini di test già trasformate in scala di grigi e salvate sottoforma di vettori (originariamente erano immagini di dimensione 365×590).
- Immagini di test: una matrice di dimensione 215350×230 che contiene le 230 immagini di test, anche esse già trasformate in scala di grigi e salvate sottoforma di vettori. E' possibile visualizzarle con lo stesso comando di prima.
- Etichette di train: un vettore di dimensione 1000×1 che contiene le etichette delle immagini di train.
- Etichette di test: un vettore di dimensione 230×1 che contiene le etichette delle immagini di test.

Prima di iniziare la fase di addestramento è necessario estrapolare dalle 1000 immagini di train e dalle 230 di test le features dall'algoritmo del BRISQUE attraverso la funzione `computeBRISQUEFeatures` che è reperibile anche online ed è riportata nella sezione Appendice. Tale funzione però si applica ovviamente ad una singola immagine alla volta. Perciò è stata creata una matrice di dimensioni 36×1000 . Successivamente è stata richiamata la funzione sopracitata per ogni immagine presente nel training set attraverso un ciclo for come riportato in appendice.

4.3.2 Fase di addestramento

SVR

Per l’addestramento di una SVR è necessario il settaggio di alcuni iperparametri, come abbiamo visto anche in precedenza. Eccone alcuni:

- Il kernel che si vuole utilizzare per trasformare i dati: esso può essere lineare, polinomiale o “rbf”, ossia a base radiale. A seconda del tipo di kernel poi si possono specificare anche altri parametri. Ad esempio in caso di kernel polinomiale si può specificare il grado del polinomio di approssimazione.
- La scala del kernel che si preferisce usare, solitamente si imposta tale parametro al valore automatico che permette al compilatore di scegliere quella migliore.
- Il parametro di regolarizzazione che precedentemente avevamo chiamato C .
- La tolleranza ε .
- Il parametro di standardizzazione che permette che le caratteristiche siano riscalate per avere media zero e varianza uno prima dell’addestramento del modello.

Verranno fatte sperimentazioni al variare di questi parametri per capire quali risultano essere quelli ottimali per il nostro caso di studio. I valori dati in input ai modelli di apprendimento saranno invece i 1000 vettori di 36 features ottenuti dal preprocessamento precedente a partire dalle immagini del training set.

Random Forest

Anche per l’addestramento di una Random Forest è possibile fissare alcuni parametri come il numero di alberi che si vogliono allenare. Verranno fatte anche in questo caso delle sperimentazioni andando a variare tale parametro per osservare come cambiano le performance del modello.

4.3.3 Fase di predizione

Una volta finita la fase di addestramento ci è possibile testare i due regressori appena creati sul test set composto da 230 immagini. Si utilizzano quindi i modelli predittivi appena creati e si da loro in input la matrice di

dimensioni 36×230 composta dalle 36 features estrapolate da ogni immagine del test set.

4.3.4 Risultati e considerazioni

Per valutare i risultati considereremo qualche misura di statistica descrittiva elementare come l' R^2 , il Mean Squared Error (già incontrato anche in precedenza), il Mean Absolute Error, il tempo di addestramento e il tempo di predizione.

Definizione 4.3.1. (Coefficiente di determinazione) Chiamiamo **coefficiente di determinazione**, anche detto R^2 , un indice che misura il legame tra la variabilità dei dati e la correttezza del modello statistico utilizzato, ossia:

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$$

dove y_i sono i dati attesi, \hat{y}_i sono i dati predetti e \bar{y} è la media dei dati attesi, cioè $\bar{y}_i = \frac{1}{n} \sum_{i=1}^n y_i$. Il numeratore è detto **devianza residua** (o residual sum of squares) e il denominatore è detto **devianza totale** (o total sum of squares).

Definizione 4.3.2. (Mean Absolute Error) Chiamiamo **Mean Absolute Error** la metrica che misura la differenza assoluta media tra i valori predetti e i valori target effettivi:

$$MAE = \frac{\sum_{i=1}^n |\hat{y}_i - y_i|}{n}$$

dove \hat{y}_i sono i valori predetti dal modello, y_i sono i valori attesi e n è il numero di osservazioni effettuate.

SVR

Valutiamo le differenze che si trovano andando a variare il kernel della SVR e gli altri parametri. Il valore di ε è stato fissato a 3.3358.

Risultati con kernel lineare:

C	Tempo di train	R^2 train	R^2 test	MAE train	MAE test
10000	16.7801	0.6537	0.5370	13.3492	15.4505
100	0.5946	0.6602	0.5753	12.9834	14.1980
1	0.1024	0.6520	0.5919	13.1822	13.9351
0.1	0.0636	0.6280	0.5826	13.7519	14.4222
0.001	0.1197	0.3008	0.3073	20.4074	20.6719

Dai risultati appena ottenuti sembra che le migliori performance si raggiungano con valori di C vicini a 1. Inoltre possiamo osservare che per valori di C molto elevati il tempo di train aumenta notevolmente poichè il modello si concentra particolarmente nell'approssimare i valori assunti dai dati di train. Quando invece C è molto piccolo ricordiamo che maggior peso è dedicato ad aumentare la *flatness* della funzione.

Consideriamo ora il caso con kernel polinomiale di terzo grado:

C	Tempo di train	R^2 train	R^2 test	MAE train	MAE test
1	14.3041	0.7197	-0.7003	10.8526	20.4008
0.1	12.4545	0.7251	0.1765	10.8329	16.6794
0.001	11.6303	0.5488	0.3675	15.5129	17.9518
0.0001	0.2179	0.6332	0.5613	13.6861	15.2197

Deduciamo che il kernel polinomiale risulta essere quello adatto per il caso in questione solo se i valori di C sono vicini a 0.0001 (basti osservare l' R^2). Per valori di C vicini a 1 possiamo osservare invece un tipico caso di overfitting: il modello approssima molto bene i dati di addestramento ma non presenta alcuna capacità di predizione.

Osserviamo ora cosa succede applicando un kernel di tipo rbf:

C	Tempo di train	R^2 train	R^2 test	MAE train	MAE test
10000	0.3628	0.9881	0.3120	3.1108	19.3263
100	0.1298	0.8629	0.5019	6.8734	15.9813
50	0.1260	0.8428	0.5240	7.5546	15.3721
1	0.1641	0.3221	0.2407	19.6335	21.2206
0.1	0.1007	0.0496	0.0402	24.2222	24.6533

Sembra che in questo caso la scelta ottimale sia per valori di C vicini a 50. Osserviamo però che le performance migliori (sia in termini di tempo di addestramento, che di R^2 ecc...) le abbiamo ottenute utilizzando un kernel lineare e con valori di C prossimi a 1.

Possiamo provare che per valori di C molto grandi vi è overfitting andando a vedere come si comporta il modello dandogli in input i dati di training. Riporto l'esempio di una SVR con kernel rbf e $C = 10000$:

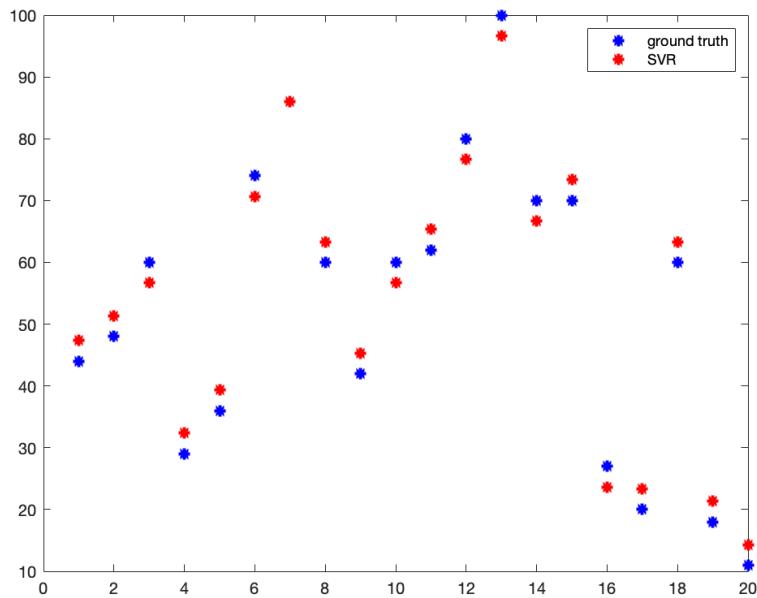


Figura 4.6: Predizione delle label delle prime 20 immagini del training set usando SVR con kernel rbf, $C = 10000$ e $\varepsilon = 3.3358$

Si vede molto bene come le predizioni effettuate con la SVR ricalchino quasi esattamente i risultati attesi.

Random Forest

Studiamo adesso cosa succede con un modello di addestramento di tipo Random Forest, andando a capire come cambiano le sue performance variando il numero di alberi.

Numero alberi	Tempo di train	R^2 train	R^2 test	MAE train	MAE test
1000	6.1555	0.8822	0.5923	7.1307	13.8395
100	0.6009	0.8813	0.5826	7.1690	13.9122
10	0.1124	0.8624	0.5529	7.8004	14.5500
1	0.1554	0.6919	0.2922	10.8699	18.4487

Osserviamo che si riescono ad ottenere risultati in termini di previsione paragonabili a quelli ottenuti con il modello di previsione SVR a kernel lineare e con parametri $C = 1, \varepsilon = 3.3358$.

E' possibile anche andare ad osservare dal punto di vista grafico come si comportano tali due modelli sulle prime 20 immagini del dataset:

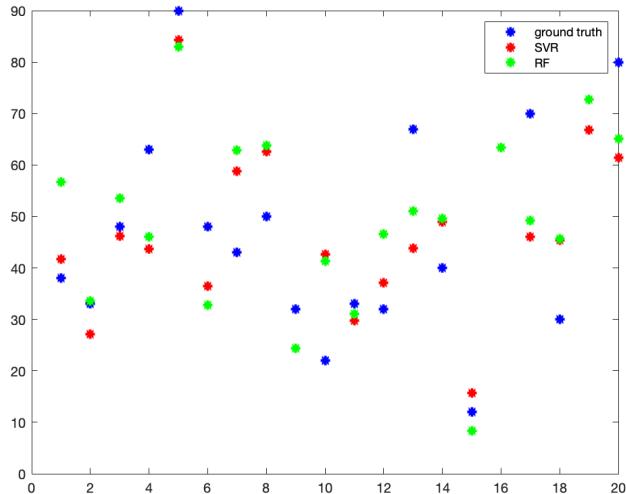


Figura 4.7: Predizione delle label delle prime 20 immagini usando SVR con kernel lineare, $C = 1$ e $\varepsilon = 3.3358$ e Random Forest con 100 alberi

Possiamo anche riportarle sottoforma di tabella:

Numero immagine	Previsione SVR	Previsione RF	Etichetta reale
1	41.656	56.708	38
2	27.066	33.621	33
3	46.133	53.521	48
4	43.646	46.061	63
5	84.294	82.976	90
6	36.501	32.742	48
7	58.734	62.821	43
8	62.645	63.775	50
9	24.309	24.378	32
10	42.601	41.314	22
11	29.752	31.127	33
12	37.107	46.599	32
13	43.853	51.049	67
14	48.931	49.581	40
15	15.757	8.374	12



Figura 4.8: A sinistra è riportata l’immagine n. 3 del test set, con etichetta 48 ma risultati previsti da SVR di 46.133 e da RF di 53.521; a destra l’immagine n. 5 con etichetta 90, ma risultati previsti da SVR di 84.294 e da RF di 82.976; in basso l’immagine n. 15 con etichetta 12, ma risultati previsti da SVR di 15.757 e da RF di 8.374

Siamo riusciti ad ottenere in questo modo risultati soddisfacenti sia con modelli di previsione di tipo SVR, sia con modelli di Random Forest. Dato che in termini di previsione le performance ottenute si assomigliano molto, probabilmente risulta essere preferibile una SVR, dal momento che i tempi di train sono notevolmente inferiori rispetto al metodo ensemble.

Capitolo 5

Conclusioni

Abbiamo visto come i problemi di imaging risultino essere particolarmente importanti e difficili da risolvere in un contesto di decisioni data driven. Infatti abbiamo studiato alcune misure di qualità delle immagini, ma nessuna di esse risulta essere in grado di valutarle come farebbe un occhio umano. E' per questo motivo che è stato studiato il metodo BRISQUE. Partendo da una immagine di qualsiasi dimensione è possibile estrarre 36 features in grado di descrivere sinteticamente e dal punto di vista matematico e statistico l'immagine.

Successivamente si sono studiate due tecniche di apprendimento supervisionato: la SVR e il Random Forest.

Dopo aver compreso teoricamente come tali modelli predittivi vengono costruiti si è deciso di applicarli a un dataset preso da una [challenge](#). Esso era già stato suddiviso in una parte di train e una di test. Ad ogni immagine del dataset è stata associata una etichetta che rappresentava il grado di chiarezza della stessa.

Dopo aver estratto le 36 features da ciascuna immagine sono state addestrate le SVR e le Random Forest andando a variare alcuni parametri. Siamo così riusciti ad osservare che le SVR risultano essere più sensibili alle variazioni dei parametri. Invece le Random Forest riescono ad avere risultati molto più stabili, a patto che gli alberi utilizzati siano maggiori di 1.

In conclusione i migliori risultati sono stati raggiunti con SVR lineare con $C \approx 1$ e Random Forest con 1000 alberi, ottenendo rispettivamente un R^2 di test uguale a 59,19% e 59,23%.

Appendice

computeBRISQUEFeatures

```
1 function full_NSSfeatures = computeBRISQUEFeatures( 
2     im)
3
4 num_features = 18;
5 num_scales = 2;
6
7 full_NSSfeatures = zeros(num_features*num_scales,1);
8
9 for i = 1:num_scales
10
11     % Normalize image to zero mean and ~unit std
12     immean = imgaussfilt(im,7/6,'FilterSize',7,'
13         Padding','replicate');
14     imstd = sqrt(abs(imgaussfilt(im.*im,7/6,'
15         FilterSize',7,'Padding','replicate') - immean
16             .*immean));
17     imnorm = (im-immean)./(imstd+1);
18
19     % Compute the AGGD parameters
20     NSSfeats = images.internal.computeNSSFeatures(
21         imnorm);
22     full_NSSfeatures((i-1)*num_features+1:i*
23         num_features) = NSSfeats';
24
25     im = imresize(im,0.5);
26 end
27 end
```

Quella appena riportata è la funzione Matlab utilizzata per il calcolo delle 36 features a partire da una qualsiasi immagine data come input. Essa restituisce perciò un vettore chiamato `full.NSSfeatures` di dimensioni 36×1 che le contiene tutte. E' coperta dal Copyright: Copyright 2016 The MathWorks, Inc.

Calcolo R^2

```

1 SS_tot = sum((LABEL_TEST - mean(LABEL_TEST)).^2);
2 SS_res_svr = sum((LABEL_TEST - y_predict_svr).^2);
3 SS_res_rf = sum((LABEL_TEST - y_predict_rf).^2);
4 R2_svr = 1-(SS_res_svr/SS_tot);
5 R2_rf = 1-(SS_res_rf/SS_tot);

```

Nell'algoritmo appena riportato `SS_tot` rappresenta la devianza totale (total sum of squares), mentre `SS_res_svr` e `SS_res_rf` rappresentano le devianze residue per il calcolo dell' R^2 del modello SVR e Random Forest rispettivamente.

Calcolo Mean Squared Error

```

1 mse = sum((LABEL_TEST - y_predict_svr).^2) / 230;
2 mse_rf = sum((LABEL_TEST - y_predict_rf).^2) / 230;

```

Si è semplicemente tradotto da linguaggio matematico a linguaggio Matlab il procedimento per il calcolo del MSE riportato nella definizione 3.2.

Calcolo del Mean Absolute Error

```

1 MAE_svr = mean(abs(LABEL_TEST - y_predict_svr));
2 MAE_rf = mean(abs(LABEL_TEST - y_predict_rf));

```

Anche qui si è semplicemente tradotto in linguaggio Matlab la formula riportata nella definizione 4.3.4.

Plot dei risultati

```
1 figure;
2 plot(1:20, LABEL_TEST(1:20), '-b', 'LineWidth', 2);
3 hold on;
4 plot(1:20, y_predict_svr(1:20), '-r', 'LineWidth',
      2);
5 hold on;
6 plot(1:20, y_predict_rf(1:20), '-g', 'LineWidth', 2)
    ;
7 hold off;
8 legend({'ground truth', 'SVR', 'RF'});
```

Con queste righe di codice si mostra il grafico cartesiano a linee dove sono riportati i valori attesi delle etichette (chiamati in legenda “*ground truth*”), le predizioni effettuate con il modello SVR e memorizzate nel vettore `y_predict_svr` e le predizioni effettuate con il modello Random Forest memorizzate in `y_predict_rf`.

Calcolo delle features

Algoritmo per il calcolo delle features nella parte di train:

```
1 for i = 1:1000
2     f(:,i) = computeBRISQUEFeatures(reshape(
3         IM_TRAIN(:,i),365,590));
4 end
```

Algoritmo per il calcolo delle features nella parte di test:

```
1 g = zeros(36,230)
2 for i = 1:230
3     g(:,i) = computeBRISQUEFeatures(reshape(
4         IM_TEST(:,i),365,590));
5 end
```

Bibliografia

- [1] Ra, A., Souza, Rio D., (2019) *A Review on Machine Learning Algorithms*. In *International Journal for Research in applied science & engineering technology* 7(VI), 792-796.
- [2] Vargas, R., Mosavi, A., Ruiz, R., *Deep Learning: A Review* Preprints (2018), 2018100218.
- [3] LeCun, Y., Bengio, Y., Hinton, G., (2015) *Deep Learning*. In *Nature*, 521(7553), 436-444.
- [4] Awad, M., Khanna, R., (2015) *Support Vector Regression*. In *Efficient Learning Machines*, Apress, Berkeley, CA.
- [5] Cristianini, N., Shawe-Taylor, J., (2000) *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods* Cambridge University Press.
- [6] Smola, A.J., Schölkopf, B. (2004) *A tutorial on support vector regression*, Statistics and Computing 14, 199-222.
- [7] Valentini, G., *Introduzione ai metodi kernel*, DSI - Università di Milano.
- [8] Zanni, L., Bonettini, S., (2020) *Dispense di ottimizzazione numerica*, Publisher.
- [9] IBM: <https://www.ibm.com/topics/decision-trees>
- [10] Prasad Ashwin, *Regression Trees — Decision Tree for Regression — Machine Learning*, “Medium”, 8 agosto 2021, <https://medium.com/analytics-vidhya/regression-trees-decision-tree-for-regression-machine-learning-e4d7525d8047>
- [11] Cornell CS4780: lezione di Kilian Weinberger da [CS4780 - Machine Learning for Intelligent Systems](#), Cornell University.

- [12] Steorts, R. C., *Tree Based Methods: Regression Trees*. STA 325, Chapter 8 ISL, Duke University.
- [13] Ravindran Rishika, *Overfitting and Pruning in Decision Trees — Improving Model's Accuracy*, “Medium”, 18 gennaio 2023, <https://medium.com/nerd-for-tech/overfitting-and-pruning-in-decision-trees-improving-models-accuracy-fdbe9ecd1160>.
- [14] Lee, Tae-Hwy, Ullah, A., Wang, R., (2020) *Bootstrap Aggregatin and Random Forest*. In *Macroeconomic Forecasting in the Era of Big Data*, 389-429.
- [15] Breiman, L., (1996) *Bagging predictors*. In *Machine Learning*, 24, 123-140.
- [16] Breiman, L., (2001) *Random Forests*. In *Machine Learning*, 45, 5-32.
- [17] Murphy, K. P., (2012) *Machine Learning: a probabilistic perspective*, The MIT Press.
- [18] Bhatia Navnina, *What is Out of Bag (OOB) score in Random Forest?*, “Medium”, 26 giugno 2019, <https://towardsdatascience.com/what-is-out-of-bag-oob-score-in-random-forest-a7fa23d710>.
- [19] Helsinki Deblur Challenge: <https://fips.fi/HDCdata.php#anchor1>.
- [20] Mittal, A., Moorthy, A. K., Bovik, A. C., (2012), *No-Reference Image Quality Assessment in the Spatial Domain*. In *IEEE transactions on image processing* Vol. 21, No. 12.
- [21] Seshadrinathan, K., Pappas, T. N., Safranek, R. J., Chen, J., Wang, Z., Sheikh, H. R., Bovik, A. C., (2009) *Image Quality Assessment*. In *Essential Guide of Image Processing*, Elsevier.
- [22] Shrimali, K. R., *Image Quality Assessment: BRISQUE*, “LearnOpenCV”, 20 giugno 2018, <https://learnopencv.com/image-quality-assessment-brisque/>.
- [23] MathWorks: https://it.mathworks.com/help/stats/index.html? s_tid=CRUX_lftnav