UNIVERSITÀ DEGLI STUDI DI PERUGIA

DIPARTIMENTO DI ECONOMIA

Corso di Laurea Magistrale in
Finanza e Metodi Quantitativi per l'Economia

TESI DI LAUREA

Il metodo SHAP per l'interpretabilità delle decisioni sulla fraud detection ottenute dai modelli di machine learning

Laureando:
Gregorio Alessandretti

Relatore:
Prof. Davide Petturiti

Anno Accademico 2022/2023

Indice

1 Introduzione **5**

2 La *financial fraud detection* **7**

2.1 La frode finanziaria e i suoi costi 7

2.2 Il ruolo della *financial fraud detection* 9

3 I modelli di *machine learning* **10**

3.1 Una panoramica sul *machine learning* 10

3.2 I modelli più conosciuti 12

3.2.1 *Artificial neural networks* 12

3.2.2 *Decision trees* 13

3.2.3 *Ensemble learning* 14

3.3 Il *machine learning* nel contesto della *financial fraud detection* 15

4 L'*eXplainable Artificial Intelligence* **17**

4.1 L'importanza dell'interpretabilità 17

4.2 I metodi di interpretabilità 18

4.3 Gli *additive feature attribution method* 21

5 Le *set function* **24**

5.1 Giochi, capacità e misure 24

5.2 La derivata di una *set function* 25

5.3 La trasformata di una *set function* 26

5.4 I giochi k-additivi 28

5.5 L'integrale di *Choquet* 28

5.5.1 Il caso generale 29

5.5.2 L'integrale di *Choquet* per una *simple function* 30

6 Lo *Shapley Value* **33**

6.1 La teoria dei giochi cooperativi 33

6.2 La definizione di *Shapley Value* 36

6.3 Lo *Shapley Value* a gruppi 37

6.4 La *Shapley Interaction* 39

7 Il metodo SHAP **41**

7.1 Dalla *game theory* all'*eXplainable Artificial Intelligence* 41

7.2 Lo *SHapley Additive exPlanations value* 42

7.3 Le criticità del metodo SHAP 45

8 I metodi di approssimazione dello SHAP **47**

8.1 Il *Kernel* SHAP 47

8.1.1 *Local Interpretable Model-agnostic Explanations* 47

8.1.2 Lo SHAP come una particolare configurazione del *linear* LIME 48

8.1.3 Il vantaggio computazionale del *Kernel* SHAP 50

8.2 Il *k-additive* SHAP 51

8.2.1 La derivazione dello SHAP tramite l'integrale di *Choquet* 52

8.2.2 L'introduzione della *k*-additività 54

8.3 I metodi di approssimazione dello SHAP a gruppi 56

9 Analisi sperimentale: un'applicazione del metodo SHAP nel contesto della *financial fraud detection* **58**

9.1 Descrizione del *dataset* 58

9.2 Addestramento dei modelli di *machine learning* 59

9.3 Applicazione del metodo SHAP e delle sue estensioni 60

9.4 Applicazione dei metodi di approssimazione dello SHAP 63

9.4.1 Valutazioni sulla convergenza dei metodi di approssimazione 63

9.4.2 Valutazioni sull'efficienza computazionale dei metodi di approssimazione 66

10 Conclusioni **69**

A Codici **70**

A.1 Modelli di *machine learning* 70

A.2 Metodi di interpretabilità 71

Riferimenti bibliografici **98**

1 Introduzione

Negli ultimi anni si sta osservando una sempre maggiore diffusione, in un numero sempre maggiore di contesti, dell'utilizzo di modelli di *machine learning* come strumenti volti all'automatizzazione e/o al supporto dei processi decisionali. Questo fenomeno è favorito dalla crescente potenza di calcolo dei computer, che permette un utilizzo più efficiente di questi strumenti, nonché dallo sviluppo di modelli sempre più complessi e performanti, come ad esempio i *neural network*. Nell'ambito del *machine learning* esiste tuttavia un noto *trade-off* tra complessità ed interpretabilità e, per questo, modelli come i *neural network* vengono solitamente chiamati *black-box model*: non si è in grado di comprenderne a pieno né il funzionamento né le ragioni che li portano a produrre determinati risultati [38]. L'opacità che caratterizza questi strumenti rappresenta una notevole criticità quando il loro utilizzo viene esteso ai cosiddetti *high-stakes decision context*, quali la medicina, la finanza, la giustizia penale e così via. In questi contesti, infatti, la decisione presa dal *decision maker* può impattare in modo significativo (positivamente o negativamente) sulla società e sugli individui che la compongono. Solo una piena fiducia nei risultati prodotti da questi modelli può allora permetterne un'applicazione sicura, fiducia che manca se non si conoscono le ragioni ad essi sottostanti [44]. Per ovviare a questa criticità, la ricerca nell'ambito delle intelligenze artificiali si è iniziata a concentrare sullo sviluppo di metodi in grado di fornire spiegazioni dei risultati prodotti dai modelli di *machine learning*. Si parla in questo senso di *eXplainable Artificial Intelligence* [14]. Tra i metodi sviluppati nell'ambito di questo filone di ricerca il metodo SHAP (*SHapley Additive exPlaination*) ha riscontrato un grande successo. Questo, grazie all'estensione di un concetto molto conosciuto nel contesto della teoria dei giochi cooperativi (lo *Shapley Value*), permette di assegnare a ogni variabile utilizzata nella costruzione del modello un numero reale rappresentante il contributo (positivo o negativo) che questa ha avuto nel determinare il risultato prodotto [32]. La complessità computazionale del metodo SHAP cresce tuttavia in modo esponenziale all'aumentare del numero di variabili utilizzate e, di conseguenza, la sua implementazione diventa proibitiva per *dataset* ad alta dimensionalità. Si è per questo assistito all'introduzione di approcci che permettessero di approssimarne il metodo in questione ma garantendo una maggiore efficienza computazionale [38].

L'obbiettivo di questa tesi è quello di mostrare il funzionamento del metodo SHAP, delle sue estensioni e dei suoi metodi di approssimazione in relazione alle previsioni prodotte da alcuni modelli di *machine learning* in un particolare contesto di *high-stakes decision*, quello della *financial fraud detection*. Così facendo si vuole mostrare in che modo questo metodo permetta di incrementare l'interpretabilità e quindi l'affidabilità di questi modelli, nonché verificare i vantaggi computazionali comportati dai suoi approcci di approssimazione. Per prima cosa verrà quindi introdotto il concetto di *financial fraud detection* (Capitolo 2), con lo scopo di mostrarne la rilevanza nell'ambito dei mercati finanziari, e verrà presentata una panoramica sul *machine learning* e sui modelli più diffusi (Capitolo 3). Successivamente si affronterà il problema dell'interpretabilità e di come l'*eXplainable Artificial Intelligence* cerca di porvi una soluzione (Capitolo 4). Dopo una digressione sulla teoria delle *set function* (Capitolo 5), volta ad introdurre alcuni concetti fondamentali per la comprensione degli argomenti successivamente affrontati, e sulla teoria dei giochi cooperativi (Capitolo 6), necessaria per introdurre il concetto di *Shapley Value*, si spiegherà il funzionamento del metodo SHAP e delle sue estensioni, evidenziando anche le criticità legate al suo utilizzo (Capitolo 7). Con lo scopo di porre rimedio ad una di queste criticità verranno successivamente introdotti

due possibili approcci per l'approssimazione del metodo SHAP: il *Kernel* SHAP e il k_{ADD} - SHAP (Capitolo 8). Si concluderà presentando i risultati ottenuti da un'applicazione di questi metodi nel contesto della *financial fraud detection* (Capitolo 9). Tale applicazione è stata sviluppata tramite l'utilizzo del linguaggio di programmazione *Python* (i codici più importanti sono consultabili nell'Appendice A).

2 La *financial fraud detection*

2.1 La frode finanziaria e i suoi costi

Con il termine frode, in generale, si fa riferimento a tutti gli atti (o le dichiarazioni) commessi od omessi con lo scopo di ingannare una vittima, producendo di conseguenza un danno per quest'ultima e un guadagno illecito per l'autore [28]. Questa assume l'accezione di finanziaria quando coinvolge uno o più partecipanti dei mercati finanziari e riguarda beni, servizi o opportunità di investimento di natura finanziaria [40].

I casi di frode finanziaria possono assumere caratteristiche molto differenti in base al segmento di mercato, agli strumenti finanziari e agli attori che sono coinvolti [40], si tratta quindi di un fenomeno estremamente complesso e variegato. Tuttavia, è possibile individuare delle categorie in cui ricondurre la maggior parte delle attività fraudolente:

- *Credit card fraud*: l'autore della frode acquisisce in modo illecito i dati della carta di credito (o di debito) della vittima e li utilizza per effettuare transazioni monetarie non autorizzate [40]. Il furto dei suddetti dati può avvenire tramite l'uso di raggiri (come spesso accade nelle truffe *online*) o tramite il furto fisico della carta [2].

- *Financial statement fraud*: un attore del mercato finanziario, direttamente o tramite i propri rappresentanti, comunica informazioni totalmente o parzialmente false inerentemente il proprio stato di salute economico-finanziaria con lo scopo di ingannare i destinatari di tali informazioni [40]. Rientrano in questa categoria, ad esempio:

  - le *accounting fraud*: i dirigenti di una società quotata falsano il contenuto dei documenti di bilancio da rendere pubblici. Lo scopo può essere quello di gonfiare le performance, così da ottenere un incremento del prezzo delle azioni (e i conseguenti vantaggi legati a *bonus*, *stock option*, ecc.), o quello di nascondere una situazione di dissesto economico-finanziario [40].
  - le *mortage fraud*: il mutuatario, o altre parti interessate, comunica false informazioni al mutuante con lo scopo di vedersi accettata la richiesta di prestito che altrimenti verrebbe rifiutata [40].

- *Securities and commodities fraud*: il truffatore convince la vittima, solitamente inesperta, ad investire in un certo strumento finanziario fornendole false informazioni inerentemente le prospettive di profitto, la sicurezza e altre caratteristiche dello strumento. Lo schema Ponzi è un classico esempio di questa tipologia di frode [2].

- *Insurance fraud*: il soggetto beneficiario di un contratto di assicurazione cerca di ricavare un profitto tramite un uso improprio o illecito della polizza (ad esempio inscenando un falso incidente o utilizzando un certificato medico falso) [2].

- *Money laundering*: è la pratica con cui l'origine dei proventi derivanti da un'attività criminale viene camuffata in modo da far sembrare tali proventi leciti [2].

Va inoltre sottolineato come le forme di frode finanziaria siano in continuo mutamento con lo scopo di adattarsi alle sempre nuove forme di difesa implementate dagli operatori dei mercati finanziari e dalle autorità di vigilanza [37]. In questo senso, negli ultimi anni, un ruolo fondamentale è stato assunto dalle innovazioni tecnologiche le quali, se da un lato hanno permesso di sviluppare forme di difesa contro le attività fraudolente sempre più efficaci (si veda la Sezione 2.2), dall'altro hanno aperto nuovi orizzonti per gli artefici delle frodi [47]. La diffusione dell'*e-commerce*, ad esempio, ha reso sempre più agevole la perpetrazione di frodi on-line: si stima che all'interno della *Single Euro Payments Area* (SEPA) la percentuale di *credit card fraud* avvenute on-line sia passata dal 73% nel 2016 all'84% nel 2021 [17]. Si pensi altrimenti a come la nascita delle criptovalute abbia permesso lo sviluppo di una nuova categoria di frodi ad esse associate: nel solo 2022, a livello globale, il valore delle truffe legate alle criptovalute si attesterebbe introno ai $6 miliardi mentre si aggirerebbe intorno ai $4 miliardi il valore dei furti compiuti tramite le attività di *crypto hacking* [13].

Oltre alle perdite subite dalle vittime dirette della frode, il manifestarsi di questa variegata gamma di attività fraudolente comporta un'altrettanto variegata serie di costi per i partecipanti dei mercati finanziari. Innanzitutto, i soggetti coinvolti più o meno direttamente nella frode possono subire ingenti danni reputazionali. Si pensi, ad esempio, a quegli intermediari i cui dipendenti risultino coinvolti in operazioni di riciclaggio di denaro o a quelli che, anche involontariamente, abbiano facilitato la diffusione di uno schema Ponzi o alla società quotata i cui dirigenti risultino colpevoli di frodi contabili. Questi danni reputazionali si possono poi tramutare in danni economici sotto forma di perdita di clientela, costi legali, costi di sostituzione del management, ecc. [40]. Altri costi che possono emergere a seguito di episodi fraudolenti sono quelli sostenuti con l'intento di ripristinare la situazione precedente alla frode o prevenire il manifestarsi di altri episodi analoghi. Per esempio, la banca il cui cliente risulti vittima di *credit card fraud* sosterrà delle spese nel tentativo di recuperare il denaro rubato, per disattivare i sistemi che hanno dato origine alla truffa, per aumentare i controlli sulle future transazioni e così via [40]. Le attività fraudolente possono infine generare una serie di esternalità negative sull'intera collettività. Infatti, il manifestarsi dei costi sopra menzionati può compromettere, nel lungo termine, la stabilità degli intermediari e, di conseguenza, quella dell'intero sistema finanziario con inevitabili ricadute sull'economia reale e quindi sul benessere dei cittadini. Un esempio eclatante è quello della crisi del 2007-2008, nata nel sistema finanziario come conseguenza di una complessa rete di attività fraudolente [40]. Altre esternalità negative possono trovare origine nel riciclaggio di denaro, in quanto questo favorisce l'operato della criminalità organizzata e delle organizzazioni terroristiche [2].

Quantificare in modo esatto l'entità delle perdite associate alle frodi finanziare è molto complesso. Infatti, molti dei costi appena menzionati sono difficili da monitorare con precisione in virtù della loro particolare natura (si pensi, ad esempio, ai danni reputazionali). A questo si aggiunge la circostanza affatto inusuale di attività fraudolente che non vengono mai individuate e quindi mai quantificate. Ad esempio, accade spesso che la vittima decida di non denunciare la frode subita o che le autorità di vigilanza non riescano a rilevare una frode in atto [37]. Tuttavia, per dare un'idea generale sulla dimensione del fenomeno oggetto di studio, si ritiene utile citare alcuni dati: nel 2023, globalmente, si stima che le perdite legate a truffe finanziarie si siano attestate intorno ai $485 miliardi e che più di $3000 miliardi provenienti da fonti illecite siano circolati nel sistema finanziario tramite le attività di riciclaggio [37]. Oltre a questi dati, altri due fattori che vanno tenuti in considerazione in quanto hanno contribuito a una crescita significativa del fenomeno negli ultimi decenni sono la globalizzazione e la pressione econo-

mica. La prima ha portato gli intermediari finanziari ad espandere le loro attività a livello globale e la complessità gestionale che ne è derivata, unita a una scarsa conoscenza dei nuovi insediamenti, ha reso questi più vulnerabili nei confronti delle attività fraudolente. La seconda, comportando il peggioramento delle condizioni finanziarie di famiglie e imprese, ha spinto un maggior numero di soggetti a intraprendere attività illecite nel tentativo di risollevare tali condizioni [28].

2.2 Il ruolo della *financial fraud detection*

Quanto detto nella Sezione 2.1 evidenzia come le frodi finanziarie comportino gravi conseguenze per i mercati finanziari e i loro partecipanti in quanto, oltre a causare ingenti perdite per le vittime, possono determinare una riduzione della fiducia nel sistema finanziario e, nel lungo termine, una sua instabilità [47]. Per questa ragione nel corso degli anni due principali contromisure sono state adottate con lo scopo di prevenire il manifestarsi delle attività fraudolente o, quando questo non fosse possibile, individuarle con tempestività così da interromperle e sanzionarle: la produzione normativa delle autorità di vigilanza a l'adozione di sistemi di *financial fraud detection* [42].

Con la prima si fa riferimento a tutte le leggi, i regolamenti, le direttive o ogni altra fonte normativa introdotta dalle autorità competenti con lo scopo di impedire o comunque rendere più complicata la perpetrazione di attività fraudolente. Queste norme possono prevedere un irrobustimento dei sistemi di controllo, un inasprimento delle sanzioni, l'istituzione di apposite procedure di prevenzione, un incremento della trasparenza verso il consumatore, ecc. Ad esempio, l'introduzione della direttiva comunitaria *Payment Services Directive 2* avrebbe contributo a una riduzione delle frodi connesse a carte di credito e debito, tanto che il rapporto tra transazioni fraudolente e totali all'interno della SEPA ha raggiunto nel 2021 il minimo storico (0.028%) [17].

Con *financial fraud detection* (FFD) si fa invece riferimento a tutte quelle procedure volte a distinguere attività e comportamenti fraudolenti da quelli non fraudolenti, consentendo così l'interruzione e la sanzione delle frodi ed ottenendo quindi una riduzione dei loro impatti negativi [42]. I soggetti coinvolti nell'attuazione di tali procedure possono essere i più disparati: dalle autorità di vigilanza agli intermediari finanziari, dai revisori contabili agli analisti, fino ai dipendenti degli stessi intermediari [15]. Storicamente la FFD veniva implementata tramite metodi di tipo manuale (quali l'*auditing*) che tuttavia, nell'epoca dei *big data* e della digitalizzazione, risultano impraticabili in quanto molto costosi, lenti nel dare risposte e spesso inaccurati. Per questo motivo, negli ultimi decenni, è stato introdotto l'utilizzo di metodi di rilevamento automatico delle frodi basati sull'uso di tecniche statistiche e computazionali (presentate nel Capitolo 3) [47]. Anche grazie all'evoluzione tecnologia, che soprattutto a partire dagli anni '80 ha migliorato notevolmente le capacità di calcolo dei computer, queste tecniche si sono evolute diventando sempre più accurate (e allo stesso tempo complesse) e il loro utilizzo nel contesto della FFD è ancora oggi al centro della ricerca scientifica di settore (si veda la Sezione 3.3).

3 I modelli di *machine learning*

3.1 Una panoramica sul *machine learning*

Un algoritmo è una sequenza di istruzioni che, tramite la trasformazione di un certo *input* in un *output*, permettono la risoluzione di un problema. Esistono tuttavia dei problemi molto complessi per cui è impossibile fornire manualmente delle istruzioni in grado di risolverli: si pensi al riconoscimento facciale, alle diagnosi mediche, al *credit scoring* e così via. In questi casi una possibile soluzione è quella di istruire un computer ad estrarre autonomamente un algoritmo che risolva quel problema complesso [4]. La disciplina che si pone come obiettivo quello di sviluppare metodi e algoritmi in grado di apprendere (*learn*) autonomamente come risolvere un problema complesso prende il nome di *machine learning* (ML) [29]; un modello di *machine learning* è quindi uno dei metodi sviluppati in questo contesto.

In questo ambito un ruolo essenziale è assunto dai dati. L'attività di *learning* (a cui spesso ci si riferisce anche con il termine addestramento, *training*) si sostanzia infatti nella ricerca di *pattern* all'interno di un certo insieme di dati, *pattern* che vengono poi utilizzati nella risoluzione del problema di interesse [36]. Si procede quindi secondo un ragionamento di tipo induttivo: partendo da un caso particolare (i dati osservati) si produce una soluzione generale del problema [9]. Si distingue solitamente tra dati di *input* e dati di *output*. Entrambi, nei casi più semplici, sono rappresentati da variabili (o vettori di variabili) quantitative e qualitative ma, in generale, questi possono avere le nature più disparate (immagini, frasi, grafici, ecc.) [36]. Un altro elemento centrale all'interno di questa disciplina è rappresentato dai computer (*machine*), che rendono le ricerca dei *pattern* possibile ed efficiente grazie alla loro potenza di calcolo. Infatti, i metodi di ML sono spesso troppo complessi e la quantità di dati a disposizione troppo elevata per pensare di poter eseguire queste operazioni manualmente [4]. Questo spiega il perché l'attenzione nei confronti di questa disciplina sia esplosa solo a partire dagli anni '80, anni in cui la crescente capacità computazionale dei computer ha reso queste operazioni non più proibitive [26]. Infine, un ultimo aspetto rilevante nel contesto del ML è il concetto di incertezza: la soluzione generale ottenuta tramite il processo di *learning* non sarà mai perfetta e potrà ambire al massimo ad essere una buona e utile approssimazione del processo sottostante al problema di interesse [4]. Tutto questo permette di comprendere il perché questa disciplina si componga del contributo proveniente da tanti ambiti quali la statistica, la matematica, la *computer science* e altri.

In base al tipo di dati che vengono utilizzati per addestrare il modello, il ML viene solitamente suddiviso in:

- *Supervised learning*: i dati che si hanno a disposizione sono composti da una coppia di *input* e *output*: $D = \{(x_i, y_i)\}_{i=1}^{N}$ (dove N è il numero di osservazioni). In questo caso l'obbiettivo della fase di *training* è quello di costruire una funzione che, partendo da un certo *input* x, riesca a restituire il corretto *output* y, detto anche variabile risposta. Per questo motivo tale approccio viene anche definito come *predictive learning* [36]. Al variare della natura di y si è soliti distinguere tra problemi di:
  - *Classification*: l'*output* y è una variabile categorica o quantitativa discreta. La funzione costruita viene utilizzata per assegnare ad un certo *input* una delle categorie in cui si suddi-

vide variabile risposta. La *fraud detection* è un classico esempio di questo tipo di problema (affrontata in dettaglio nella Sezione 3.3) [9].

  - *Regression*: l'*output* y è una variabile quantitativa continua. La funzione costruita viene utilizzata per assegnare ad un certo *input* un valore appartenente all'insieme dei numeri reali. La stima del prezzo futuro di un titolo azionario è un possibile esempio di questo tipo di problema [20].

- *Unsupervised learning*: si hanno a disposizione i soli dati di *input*: $D = \{x_i\}_{i=1}^{N}$. In questo contesto l'obbiettivo della fase di *learning* non è definibile in modo chiaro [36]. La ricerca delle regolarità all'interno dei dati potrà essere infatti utile per scopi molto eterogenei come la *density estimation* (si vuole stimare la distribuzione di probabilità che ha dato origine ai dati) o il *clustering* (si vogliono suddividere i dati in *cluster* di osservazioni omogenee) [20].

Va detto che in letteratura si individuano altre categorie di ML (quali il *reinforcement learning* o il *semi-supervised learning*) che tuttavia non si ritiene utile approfondite in questa sede.

Un'altra importante distinzione che viene abitualmente fatta nel contesto del ML è quella tra modelli parametrici e non parametrici. I primi sono così definiti in quanto si assume che i dati osservati siano stati generati a partire da una certa forma funzionale. Nella fase di *learning*, quindi, si procede alla stima dei parametri che definiscono tale funzione. Maggiore è il numero di questi parametri, maggiore è la complessità del modello e maggiore è la sua flessibilità (che si può definire come la capacità di adattarsi e quindi spiegare il *data generating process*) [26]. I secondi, viceversa, non fanno alcun tipo di assunzione. Non c'è quindi la volontà di definire una funzione che globalmente descriva il *data generating process*, bensì il modello cerca di adattarsi localmente ai dati osservati [4]. Per questa ragione i modelli non parametrici hanno il vantaggio di essere più flessibili e non incorrono nel rischio di usare una forma funzionale non adeguata. Tendono tuttavia a necessitare un maggior numero di osservazioni per poter funzionare adeguatamente [4].

Uno stesso problema complesso può essere risolto tramite l'utilizzo di diversi modelli di ML che, chiaramente, forniranno differenti soluzioni a tale problema. Secondo il *no free lunch theorem* non esiste un modello che sia universalmente migliore degli altri e, per questo, al variare del problema varia il modello migliore nel risolverlo [20]. Diventa quindi fondamentale l'introduzione di misure di performance che permettano un confronto tra i modelli e, quindi, la scelta di quello ottimale in un determinato contesto. Tali misure possono variare in base alla tipologia di problema affrontato. Nel caso di una classificazione, ad esempio, si può guardare al cosiddetto *error rate*, cioè al rapporto tra le osservazioni erroneamente classificate (quando un dato di *input* viene assegnato ad un'errata categoria della variabile risposta) e le osservazioni totali. In una regressione, invece, si è soliti utilizzare il *mean squared error*, cioè la media degli scarti quadratici tra i valori della variabile risposta stimati dal modello e i corrispondenti valori osservati [26].

Si potrebbe pensare che maggiore è la flessibilità (e quindi la complessità) del modello e migliore sarà la sua performance. Tuttavia non sempre è così. Può infatti succedere che modelli troppo flessibili si adattino troppo precisamente ai dati utilizzati nella fase di *learning* modellando quindi non solo il segnale (*signal*), cioè l'informazione generalizzabile in essi contenuta, ma anche il rumore (*noise*), cioè le anomalie degli specifici dati osservati rispetto a quell'informazione generale. In questi casi si è soliti parlare di *overfitting* [4]. Per evitare questo tipo di errore è buona prassi suddividere i dati in due

gruppi: un *training set*, sul quale il modello viene addestrato, e un *test set*, sul quale il modello viene valutato. Un modello caratterizzato da *overfitting*, infatti, tenderà ad avere ottime performance quando valutato sul *training set* ma performance molto inferiori quando valutato sul *test set* [36]. Va inoltre sottolineato che più un modello è complesso e maggiore è il tempo computazionale necessario per addestrarlo. In determinati contesti si potrebbe quindi preferire un modello meno performante ma che sia computazionalmente meno esigente. Infine, bisogna tenere in considerazione che esiste un *trade-off* tra la flessibilità del modello e la sua interpretabilità. Un modello molto complesso potrebbe quindi avere una performance migliore rispetto a uno meno flessibile, tuttavia la capacità di interpretare la soluzione ottenuta potrebbe essere scadente e, in alcuni contesti, questo può essere un problema di notevole rilevanza (come verrà discusso nella Sezione 4.1) [26].

3.2 I modelli più conosciuti

In questa sezione vengono brevemente descritti alcuni dei modelli di *machine learning* più conosciuti.

3.2.1 *Artificial neural networks*

Con *artificial neural network* (ANN) si fa riferimento ad un'ampia categoria di modelli di ML che ebbero grande fama alla fine degli anni '80 e che sono tornati di grande interesse nell'ultimo decennio [26]. Il nome deriva dal fatto che la loro struttura cerca di replicare quella del cervello umano: i neuroni e le sinapsi sono sostituite da nodi e collegamenti [47].

Nello specifico i nodi si collocano su diversi livelli (*layers*): un *input layer*, in cui ogni nodo corrisponde a una delle variabili dei dati di *input*; un *output layer*, in cui sono presenti uno o più nodi corrispondenti agli *output* che si vogliono modellare; uno o più *hidden layer*, attraverso cui i dati di *input* vengono processati per ottenere l'*output* (Figura 1). Quando un ANN presenta un numero elevato di *hidden layer* si è soliti parlare di *deep neural network* (DNN) [10]. I nodi collocati su *layer* limitrofi sono collegati tra loro e, attraverso questi collegamenti, l'informazione contenuta nei dati di *input* si muove lungo la rete neurale fino a raggiungere l'*output layer*. Ogni nodo, quindi, prenderà come input l'informazione proveniente da quelli ad esso collegati appartenenti al livello precedente [47]. Quando l'informazione raggiunge un nodo, essa viene processata attraverso una funzione non lineare detta *activation function* per poi essere trasmessa a quello successivo. Questa *activation function* può essere di forme differenti e solitamente viene scelta in base al tipo di dati a disposizione e al tipo di problema che si sta modellando [9]. Infine, ad ogni collegamento è assegnato un peso e la fase di *learning* si sostanzia nella stima di questi pesi. Quindi, maggiore è il numero di *hidden layer* e di nodi in essi contenuti, maggiore sarà il numero di collegamenti e maggiore sarà la parametrizzazione (e quindi la complessità) del modello.

La stima dei pesi avviene tramite l'uso di algoritmi tra cui uno dei più diffusi è il *backpropagation algorithm*: alla prima iterazione i pesi vengono inizializzati in modo causale e, in base a questi, la rete neurale processa l'informazione fino a produrre un *output*. Questo viene confrontato con il valore osservato della variabile risposta ricavando un errore che viene utilizzato per aggiornare i pesi. Si procede in questo modo finché l'errore non diventa trascurabile o finché non si raggiunge il numero massimo di iterazioni predeterminato [47].

Gli ANN sono modelli estremamente flessibili e, per questo, sono pressoché adatti alla modellizzazione di qualsiasi tipo di problema. Soprattutto se si hanno a disposizione grandi quantità di dati tendono ad avere ottime performance, tuttavia, se troppo complessi, questi tendono facilmente a cadere nell'*overfitting* e risultano essere molto dispendiosi da un punto di vista computazionale. Inoltre, vista l'impossibilità di comprendere a pieno il modo in cui l'informazione viene processata lungo la rete, è molto difficile dare un'interpretazione ai loro *output* [10].

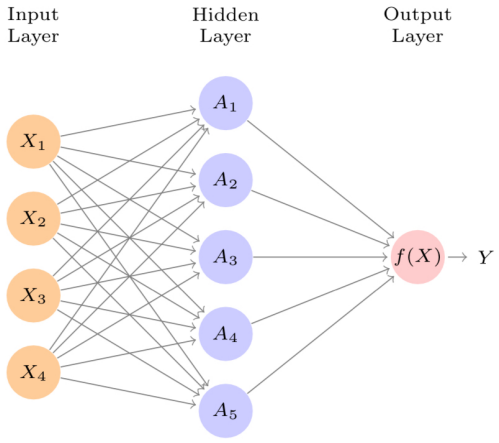

Figura 1: architettura di un semplice rete neurale [26].

3.2.2 *Decision trees*

Con *decision tree* (DT) ci si riferisce a dei modelli non parametrici di *supervised learning* utilizzabili sia in contesti di classificazione che di regressione [4]. Il loro funzionamento si basa sulla segmentazione dello spazio dei dati di *input* (detto anche *predictor space*) e il loro nome deriva dalla struttura ramificata che consegue a questo tipo di operazione (Figura 2) [10].

Più nel dettaglio, il punto di partenza è il *predictor space* nella sua interezza (per questo si parla approccio *top-down*). L'obbiettivo è quello di partizionare questo spazio in modo da ottenere regioni non sovrapponibili che siano il più omogenee possibili in termini di variabile risposta. L'omogeneità di una regione viene valutata guardando alla distanza tra i valori della variabile risposta relativi alle osservazioni appartenenti a quella regione e il loro valore medio [26]. Non essendo computazionalmente fattibile la valutazione di ogni possibile partizione del *parameter space*, la segmentazione avviene attraverso un approccio di *recursive binary splitting*: a ogni iterazione si procede con una segmentazione binaria, ottenendo quindi due nuovi rami (*branches*) e due nuove regioni. Questa segmentazione viene effettuata ricercando il miglior incremento di omogeneità possibile rispetto allo stato raggiunto nell'iterazione precedente. Si continua in questo modo finché l'omogeneità non raggiunge una certa soglia predefinita o finché altri criteri non sono soddisfatti [10]. Alla fine di questo processo si saranno ottenute delle regioni sufficientemente piccole ed omogenee (definite foglie, *leaves*) che verranno utilizzate per fare previsioni: ad una futura osservazione che cadrà in una certa regione si assegnerà il valore medio della variabile risposta delle osservazioni del *training set* appartenenti a quella stessa regione oppure, nel caso di una classificazione, la categoria della variabile risposta maggiormente rappresentata in tale regione (*majority rule*) [26].

I DT sono modelli di ML molto flessibili (volendo si potrebbe continuare la segmentazione fino ad avere un'osservazione per ogni regione) e, se non eccessivamente ramificati, conservano una buona interpretabilità. Va però sottolineato che il loro livello di accuratezza tende a essere inferiore rispetto a quello di altri modelli di ML e, inoltre, si tratta di modelli poco robusti, nel senso che una piccola variazione dei dati di *input* può comportare grandi differenze nel DT ottenuto [26].

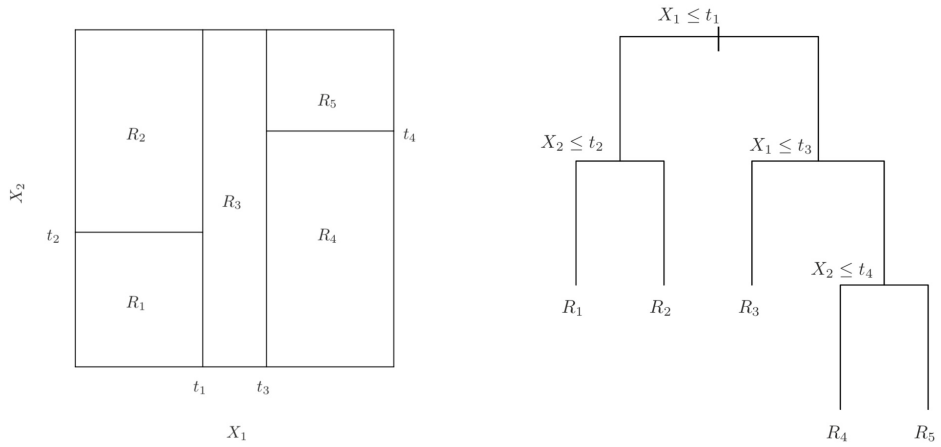

Figura 2: a sinistra un esempio di partizionamento di un *predictior space* bidimensionale utilizzando un approccio di *recursive binary splitting*. A destra il corrispondente albero decisionale [26].

3.2.3 *Ensemble learning*

L'*ensemble learning* è un approccio di *supervised learning* che si basa sulla combinazione di tanti modelli individuali con l'obbiettivo di ottenere un modello aggregato caratterizzato da performance predittive migliori (maggiore accuratezza e minore variabilità). Questo approccio si fonda sulla costatazione che la varianza della media di tante variabili casuali indipendenti e identicamente distribuite è minore della varianza delle variabili singolarmente prese [26]. C'è inoltre chi richiama il concetto di *wisdom of the crowd*, secondo cui grandi gruppi di soggetti eterogenei prendono decisioni migliori rispetto al singolo membro del gruppo [10]. Il requisito necessario per il buon funzionamento di un *ensemble model* è quindi l'indipendenza tra i modelli che lo compongono [26].

Nel corso degli anni, vari *ensemble model* sono stati proposti e, tra questi, se ne possono menzionare alcuni che hanno avuto grande diffusione:

- *Bagging*: partendo dai dati che si hanno disposizione vengono generati numerosi campioni di eguale dimensione tramite una procedura di estrazione con sostituzione definita *bootstrap* [4]. Vengono poi addestrati, uno per ogni campione *bootstrap*, tanti modelli di ML (solitamente alberi decisionali con molte ramificazioni e caratterizzati da una elevata varianza) e le previsioni da questi prodotte vengono aggregate tramite una media, se in un contesto di regressione, o tramite una *majority rule*, se in un contesto di classificazione [26]. Un algoritmo di *bagging* ha il vantaggio di poter essere parallelizzato (ogni modello individuale può essere addestrato separatamente) e questo lo rende computazionalmente più efficiente rispetto ad altri *ensemble model* [10].

- *Random Forest*: il funzionamento è analogo a quello del *bagging* tuttavia, nella fase di *training* degli alberi decisionali, la scelta della variabile rispetto a cui segmentare il *predictor space* avviene

solo rispetto a un sottoinsieme, casualmente scelto a ogni iterazione, delle variabili di input [47]. Questo generalmente garantisce una minor correlazione tra i singoli modelli e, di conseguenza, delle prestazioni migliori rispetto al più semplice *bagging* [10].

- *Boosting*: partendo da uno *week learner*, cioè un modello con capacità predittive mediocri (solitamente un DT con poche ramificazioni), si cerca di migliorarne la performance (*boost*) tramite un algoritmo di tipo sequenziale [10]. In particolare, si prendono i residui del *week learner* addestrato (cioè la distanza tra la previsione del modello e il valore osservato della variabile risposta) e si utilizzano per costruire un nuovo modello. Quest'ultimo andrà quindi a concentrarsi sugli errori compiuti dal modello precedente migliorandolo. Il processo è poi ripetuto fino al raggiungimento di una *stopping rule* [26]. Gli algoritmi di *boosting* sono oggi considerati tra i modelli di ML con le migliori prestazioni. Tra questi si possono citare l'*adaptive boosting* (AdaBoost), il *gradient boosting* e l'*extreme gradient boosting* (XGBoost) [10].

Gli *ensemble model* tendono ad essere caratterizzati da ottime performance predittive tuttavia, a causa della complessità generata dall'assemblaggio di tanti modelli, hanno solitamente tempi computazionali elevati e una scarsissima interpretabilità [26].

3.3 Il *machine learning* nel contesto della *financial fraud detection*

I modelli di *machine learning* posso essere utilizzati in una varietà incredibilmente ampia di contesti che vanno dalla medicina all'astronomia, dalle previsioni meteo alle ricerche di mercato e così via [4]. In sostanza, qualsiasi sia l'ambito, se si lavora con un insieme di dati, il ML è diventato uno strumento essenziale [26]. Tra i campi di applicazione di questa disciplina rientra quindi anche l'attività di *financial fraud detection*. Come evidenziato nella Sezione 2.2, in tale ambito si vuole stabilire se un certo evento (una transazione monetaria, un *financial statement*, ecc.) rappresenti o meno un caso di frode a partire dall'osservazione di alcune variabili (categoriche e quantitative) relative a tale evento [42].

Se ci si pone in un contesto di *learning* supervisionato, come avverrà nel Capitolo 9 dedicato alla fase sperimentale, un problema di FFD può essere visto come un problema di classificazione binaria: definendo con Y la variabile risposta binaria tale che $Y = 1$ se l'evento è fraudolento (e $Y = 0$ viceversa), con $X = (X_1, X_2, ..., X_p)$ un vettore di variabili casuali di *input* (dette *feature*) e con $\mathbf{x}$ una sua realizzazione, si vuole modellare $P(Y = 1 \mid X = \mathbf{x})$, cioè la probabilità che un evento sia fraudolento a fronte della realizzazione $\mathbf{x}$ ad esso relativa [4]. Se questa probabilità condizionata è superiore ad una certa soglia, comunemente fissata a 0.5 secondo il *maximum a posteriori principle* (MAP), l'evento viene classificato come fraudolento [36]. La stima di tale probabilità avviene sulla base di dati etichettati (*labeled*), osservando cioè sia le *feature* sia la variabile risposta di un certo campione di eventi [28].

Un problema che si incontra frequentemente nell'ambito della FFD è quello di *class imbalance*: nel campione utilizzato per addestrare il modello solo una piccola percentuale delle osservazioni sono etichettate come fraudolente. Questo accade perché solitamente i casi di frode sono un numero estremamente ridotto rispetto al totale degli eventi registrati [28]. Poiché questo disequilibrio tende a peggiorare le prestazioni dei modelli di ML è buona prassi, prima della fase di *training*, utilizzare delle tecniche volte a ribilanciare i dati [10]. Tra queste un buon successo è stato ottenuto dalla *Synthetic Minority Over-sampling Technique* (SMOTE), che riesce ad aumentare il numero di osservazioni della classe

minoritaria (*over-sampling*) tramite la generazione di osservazioni sintetiche collocate nelle vicinanze di altre osservazioni appartenenti a tale classe [6]. Tuttavia, essendo il *dataset* che verrà utilizzato nel Capitolo 9 già bilanciato, in questa sede non si ritiene utile approfondire ulteriormente questo tipo di problematica e le tecniche volte a risolverla.

La differenza tra i vari modelli di classificazione sta in come questi raggiungono la stima della probabilità condizionata sopra menzionata. Servono quindi delle misure di performance, utili nella comparazione tra più modelli. Come visto nella Sezione 3.1 una possibile metrica nei casi di classificazione è rappresentata dall'*error rate* (o dal suo complementare a 1 denominato *accuracy*). Questo tuttavia attribuisce lo stesso peso a tutti i tipi di errori mentre, nel contesto della FFD, ci sono errori definiti *false negative* (un evento viene classificato come non fraudolento quando in realtà lo è) che hanno un costo decisamente superiore rispetto ad altri, detti *false positive* (un evento viene classificato come fraudolento quando in realtà non lo è) [28]. Ad esempio, sarà meno costoso bloccare una transazione considerata erroneamente illecita rispetto a non bloccarne una che effettivamente lo è. Di conseguenza è preferibile l'uso di metriche come la *sensitivity*, il cui valore cresce al diminuire dei *false negative* (una *sensitivity* pari ad 1 implica una totale assenza di *false negative*) [10].

Sono numerosi gli studi che nel corso degli anni hanno evidenziato le ottime performance ottenute dai modelli di ML nel campo della FFD. Ad esempio, Bhattacharyya et al. (2011) in [7] hanno mostrato come una *random forest* riuscisse a raggiungere livelli di *accuracy* pari al 99.6% e di *sensitivity* pari all'81.2% nell'identificazione di transazioni fraudolente. Ravisankar et al. (2011) in [39] hanno invece mostrato che, in un contesto di *financial statement fraud*, un ANN sia riuscito a raggiungere livelli di *accuracy* e *sensitivity* vicini al 98%. Va inoltre sottolineato che in entrambi i casi menzionati i modelli di ML hanno ottenuto prestazioni superiori rispetto a un semplice modello di regressione logistica. Emerge tuttavia una criticità riguardo l'interpretabilità dei risultati ottenuti. Infatti, parallelamente alla crescita delle prestazioni dei modelli di ML, si è assistito ad una crescita della loro complessità. I metodi *ensemble*, al pari degli ANN, sono spesso etichettati come *black-box model*, non si riesce cioè a comprendere il processo che li porta ad ottenere una certa previsione e, in un contesto di FFD, questo rappresenta un problema di notevole rilevanza (su questo aspetto si tornerà nella Sezione 4.1) [30].

4 L'*eXplainable Artificial Intelligence*

4.1 L'importanza dell'interpretabilità

Nel corso degli ultimi decenni la sempre maggiore disponibilità di dati e la crescente potenza di calcolo dei computer hanno permesso lo sviluppo di complessi modelli di *machine learning*, caratterizzati da prestazioni previsionali irraggiungibili per modelli statistici più semplici quali la regressione logistica o la regressione lineare [19]. Questa sviluppo ha permesso a tali sistemi di trovare applicazione in un numero sempre maggiore di contesti e, di conseguenza, la loro influenza sulla quotidianità di ogni individuo è in continua crescita [12]. Tuttavia, come già evidenziato nella Sezione 3.1, esiste un intrinseco *trade-off* tra la complessità (e quindi la performance) di un modello e la sua interpretabilità [1].

Con interpretabilità di un modello si intende il livello di comprensione che il suo utilizzatore ha rispetto alle cause che lo portano alla produzione di un certo risultato [38] e cioè del rapporto esistente tra le variabili di *input* e gli *output* da questo prodotti (solitamente delle previsioni) [41]. Un modello, quindi, è facilmente interpretabile se è facile fornire una spiegazione del suo funzionamento in termini comprensibili per un essere umano [1]. Per questo, metodi di ML estremamente complessi (come i *deep neural network* o i metodi *ensemble*) vengono definiti *black-box model*: la logica che sta dietro le loro previsioni non è intelligibile per colui che ne fa uso [23]. A causa di questa scarsa intelligibilità si è iniziata ad osservare una crescente preoccupazione, sia da parte di privati (famiglie e imprese) sia da parte dei *policy maker*, riguardo le conseguenze che un utilizzo acritico di questi modelli può avere sulla vita di ogni individuo [12]. Come conseguenza, soprattutto in determinati contesti, sempre più frequentemente si registra una preferenza verso l'utilizzo di modelli più semplici (e spesso meno prestazionali) ma più comprensibili [32].

Per comprendere il perché si dia così tanto peso all'interpretabilità bisogna tenere presente che i metodi di ML sono degli strumenti e, in quanto tali, sono utili perché sulla base dei risultati da questi forniti verranno compiute delle azioni o prese delle decisioni. Affinché questo accada è però necessario che colui che compierà l'azione o prenderà la decisione si fidi dello strumento di cui fa uso [41]. Si potrebbe pensare che questa fiducia possa basarsi sulle misure di performance già introdotte (si vedano le Sezioni 3.1 e 3.3): se l'*accuracy* di un modello è elevata allora lo strumento è affidabile. Va però sottolineato che i modelli di ML vengono addestrati su dei *dataset* che possono contenere errori e distorsioni [23] e che i *test set* utilizzati nella valutazione della performance potrebbero essere significativamente differenti dai dati su cui il modello andrà effettivamente applicato [41]. Una previsione prodotta da questi sistemi potrebbe esse quindi condizionata dai *bias* presenti nei dati e l'impossibilità di comprendere le cause che hanno portato alla produzione di tale previsione impedisce di individuare questa distorsione [23]. Per questo, affinché non manchi la fiducia nel modello, è necessario affiancare alle misure di performance una comprensione del suo funzionamento e, quindi, la sua interpretabilità [41].

L'interpretabilità dei modelli di ML assume inevitabilmente un peso specifico superiore nei contesti di *high-stakes decision making*, cioè quelli in cui la decisione presa tramite questi strumenti può avere un impatto notevole sul benessere dei cittadini [44]. Rientrano in questa categoria settori quali la finanza, l'assistenza sanitaria, la giustizia penale ed altri [19]. Si pensi, ad esempio, al medico che faccia uso di tali modelli per effettuare una diagnosi su un paziente o al giudice che utilizzi sistemi automatizzati per

valutare la colpevolezza di un imputato. In questi casi una fiducia cieca nel modello può portare a decisioni errate, con conseguenze molto gravi per i soggetti destinatari della decisione (come il paziente a cui viene diagnosticata una patologia errata) o per l'intera collettività (come nel caso di assoluzione di un pericoloso criminale) [44]. In molti casi è stato anche evidenziato come questi modelli potessero prendere decisioni ingiustificatamente discriminatorie nei confronti di donne o di alcune minoranze etniche [23]. Va inoltre evidenziato che in questi contesti sarà lo stesso soggetto destinatario della decisione a chiedere una spiegazione dei motivi che l'hanno determinata. Ad esempio, la persona a cui viene negata una richiesta di prestito da una banca chiederà le ragioni di tale rifiuto ma, se la decisione è stata presa tramite l'uso di un *balck-box model*, la banca non sarà in grado di fornire le spiegazioni richieste [38]. Per tutti questi motivi, i *policy maker* hanno iniziato ad avanzare alcune pretese contro l'opacità di questi modelli e delle decisioni prese tramite il loro utilizzo: nel contesto comunitario si possono citare il *General Data Protection Regulation* (GDPR), approvato dal Parlamento Europeo nel 2016, e lo *White Paper on Artificial Intelligence*, pubblicato dalla Commissione Europea nel 2020. Nel primo viene introdotto il diritto per l'interessato di ricevere una spiegazione della decisione presa tramite l'uso di algoritmi [23]; nel secondo, da una parte si sottolinea l'importanza che le AI possono avere nel determinare una crescita del benessere dei cittadini europei, dall'altra si evidenziano i rischi ad esse connessi e la necessità di accrescere la loro trasparenza [16].

4.2 I metodi di interpretabilità

Quanto detto nella Sezione 4.1 rende comprensibile il perché si stia osservando una sempre maggiore attenzione nello sviluppo di sistemi di *eXplainable Artificial Intelligence* (XAI), cioè metodi in grado di interpretare i modelli di ML, anche quelli di più complessa struttura, e spiegare quindi l'origine delle loro previsioni [24]. Queste spiegazioni hanno lo scopo di migliorare la trasparenza dei modelli di ML così da accrescere la fiducia nel loro utilizzo, favorire una loro corretta e equa applicazione e permettere una più agevole individuazione di eventuali *bias* presenti nei loro *output* [14]. La Figura 3 esemplifica le modalità attraverso cui l'interpretazione può essere utilizzata all'interno di un processo decisionale: il modello di ML, sulla base delle informazioni fornite come *input*, restituisce una previsione; l'interpretazione evidenzia quali fattori abbiano maggiormente contributo (positivamente o negativamente) nel determinare quel risultato; sulla base della spiegazione fornita, il *decision maker*, che solitamente ha una conoscenza pregressa nel campo di applicazione del modello, può capire se la previsione è ragionevole (e quindi affidabile) o meno [41].

Va detto che in letteratura spesso si fa una distinzione tra il concetto di interpretazione (*interpretation*) e quello di spiegazione (*explanation*). Con il primo si indica il fornire una rappresentazione comprensibile dell'*output* di un modello intrinsecamente interpretabile. Si pensi, ad esempio, ad una regressione lineare, i cui coefficienti possono essere direttamente interpretati e quindi utilizzati per comprendere la logica retrostante una previsione. Con spiegazione di un modello si intende invece il fornire un'informazione aggiuntiva, costruita esternamente al modello stesso e che viene utilizzata per dare rappresentazione del suo funzionamento [14]. Tuttavia, considerando che in questa sede l'attenzione verrà rivolta solamente alla seconda circostanza (spiegare le previsioni prodotte dai *black-box model*), le due terminologie saranno utilizzate in modo indifferente.

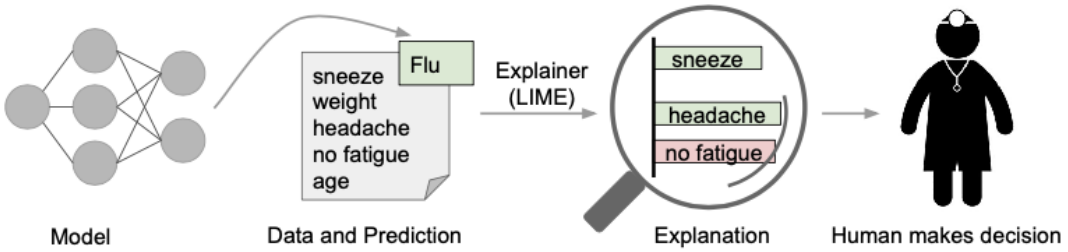

Figura 3: dalla spiegazione della previsione prodotta dal modello emerge che il raffreddore e il mal di testa sono i sintomi che hanno portato a diagnosticare l'influenza come patologia del paziente. L'assenza di affaticamento, viceversa, risulta un evidenza contro tale diagnosi. In ogni caso il medico potrà utilizzare questa spiegazione per capire se fidarsi o meno del modello [41].

Entrando più nello specifico, se si indica con f il modello di cui si vuole dare un'interpretazione, un metodo di interpretabilità si pone come scopo quello di individuare un secondo modello g, definito *explaination model*, che sia una approssimazione semplice e quindi comprensibile del primo [32]. Guardando all'*explaination model* sarà quindi possibile derivare una spiegazione approssimativa ma umanamente intelligibile della logica retrostante a f e delle previsioni da questo prodotte [41]. Ad esempio, se si riuscisse ad approssimare un DNN con un semplice modello lineare, i coefficienti di quest'ultimo potrebbero essere interpretati come il contributo delle varie *feature* nel determinare le previsioni del DNN (si rimanda alla Sezione 4.3) [32]. L'*explaination model* solitamente non è definito sullo stesso dominio del modello da interpretare. Immaginando per semplicità di notazione che tutte le variabili prese in *input* da f assumano valore in $\mathbb{R}$, allora le possibili realizzazioni $\mathbf{x}$ del vettore di *feature* X apparterranno a un *predictor space* p-dimensionale del tipo $\mathbf{x} \in \mathbb{R}^p$ (dove p è il numero di *feature*). Viceversa, l'*explaination model* g prende come valori dei *simplified input* $\mathbf{x}'$, così definiti in quanto $\mathbf{x}'$ è un vettore binario del tipo $\mathbf{x}' \in \{0,1\}^p$ [41]. Le due tipologie di *input* sono legate tra loro attraverso una *mapping function* h: $\mathbf{x} = h_{\mathbf{x}}(\mathbf{x}')$ [32]. Tale *mapping function* può ad esempio prevedere che un elemento di $\mathbf{x}'$ assuma valore 1 se il valore del corrispondete elemento di $\mathbf{x}$ è superiore ad una certa soglia. La necessità di un *input* semplificato deriva dal bisogno di rendere comprensibile l'*explaination model*. Il modello di ML è infatti addestrato su un set di variabili caratterizzate da una rappresentazione che sia processabile dal computer e adeguata per l'ottenimento di buone performance previsionali. Tuttavia, tale rappresentazione non sempre è di facile lettura per un essere umano. Per questo, l'utilizzo di una *interpretable representation* delle *feature*, cioè di un *input* semplificato, risulta spesso necessario per la costruzione di un *explaination model* che sia effettivamente comprensibile [46].

I metodi di interpretabilità sviluppati nell'ambito dell'*eXplainable Artificial Intelligence* vengono comunemente classificati in:

- metodi globali: cercano di comprendere il comportamento di un modello nel suo complesso investigando la logica dietro a tutti i possibili *output* che questo può produrre [23].
- metodi locali: cercano di spiegare le cause di una specifica istanza di interesse senza preoccuparsi del comportamento globale del modello [14].
- metodi *model-specific*: sono costruiti per l'interpretazione di una specifica classe di modelli in quanto la spiegazione viene costruita basandosi sulla struttura caratteristica di tale classe [14].

- metodi *model-agnostic*: sono utilizzabili per interpretare qualsiasi modello di ML [41]. Si concentrano unicamente sulle coppie di *input* e *output* senza investigare il funzionamento interno del modello che è trattato quindi come una *black-box* [12].

In questa sede l'attenzione verrà rivolta ai metodi *model-agnostic* locali. La prima scelta dipende dal fatto che i metodi *model-agnostic*, avendo un campo di applicazione più ampio, sono solitamente preferiti a quelli *model-specific*. La seconda deriva invece dalla costatazione che i metodi globali sono difficilmente utilizzabili nella pratica [12]. Un modello complesso, infatti, ha un comportamento piuttosto irregolare anche per combinazioni di *feature* non troppo differenti tra loro. Un interpretazione globale del suo funzionamento sarà quindi non solo difficile da ricavare ma anche di scarsa utilità [1]. I metodi locali invece, volendo interpretare una specifica istanza, si concentrano solo su una piccola regione del *predictor space*. In quella regione è plausibile che il modello risulti sufficientemente regolare da poter essere approssimato con un *explaination model* molto semplice. Chiaramente tale approssimazione (e quindi la conseguente spiegazione) avrà una valenza locale, cioè solo per quella specifica istanza o al massimo per quelle ad essa limitrofe (Figura 4) [12]. Utilizzando la notazione precedentemente introdotta, se $\mathbf{x}'^*$ rappresenta l'*input* semplificato rispetto a cui si vuole dare un'interpretazione locale del modello e se $\mathbf{x}'$ rappresenta un altro generico *input* semplificato, il metodo di interpretabilità locale cerca di assicurare che $g(\mathbf{x}') \approx f(h_{\mathbf{x}}(\mathbf{x}'))$ quando $\mathbf{x}' \approx \mathbf{x}'^*$ [32].

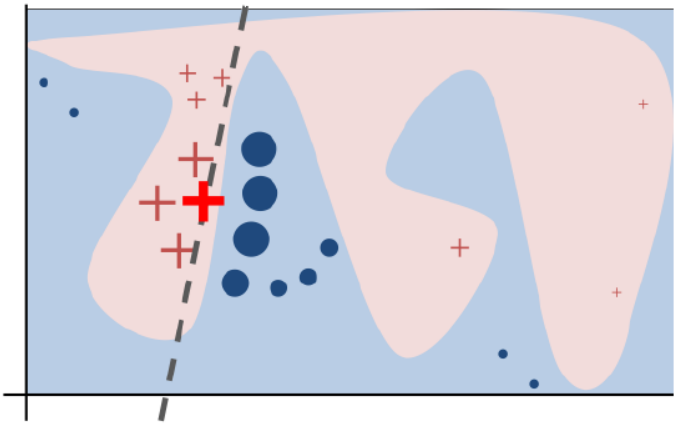

Figura 4: un esempio di modello di ML utilizzato per una classificazione binaria può essere la curva irregolare che suddivide il *predictor space* nelle due regioni di diverso colore. Nello spiegare la previsione evidenziata un *explaination model* lineare (la linea tratteggiata) riesce ad approssimare adeguatamente il modello in quella regione di piano [41].

Una previsione ottenuta da un modello di ML può essere spiegata tramite vari metodi di interpretabilità. Ognuna di queste spiegazioni sarà più o meno differente dalle altre e si riscontra quindi il problema di selezionarne una. Per questo, risulta utile introdurre alcune proprietà individuate come desiderabili per questi metodi e quindi utilizzabili per effettuare delle comparazioni [12]. Innanzitutto si evidenzia l'importanza della fedeltà (*fidelity*) dell'*explaination model*, cioè quanto bene questo riesca ad emulare il comportamento del *black-box model* di interesse [23]. Come già sottolineato, riuscire a trovare delle spiegazioni che siano globalmente fedeli è estremamente complesso e, per questo, spesso si preferisce far uso di metodi locali, i quali si accontentano di ottenere una fedeltà locale [41]. Una seconda importante proprietà è la comprensibilità (*comprehensibility*) della spiegazione, cioè quanto questa sia intelligibile e chiara per l'utente che ne fa uso [12]. Se ad esempio risultasse significativo il contributo

di centinaia di variabili nella determinazione di una previsione, non sarebbe possibile avere una piena comprensione della spiegazione fornita [41]. A tale proprietà si affianca spesso la ragionevolezza della spiegazione, cioè quanto questa sia consistente con il ragionamento umano [32]. Rispetto alla *fidelity*, che è oggettivamente quantificabile, la valutazione della comprensibilità e della ragionevolezza di un metodo di interpretabilità dipende fortemente dal soggetto che ne fa uso [41]. Ad esempio, un medico potrà agevolmente attestare la ragionevolezza della spiegazione di una diagnosi automatizzata, cosa che invece non sarà in grado di fare una soggetto non competente in campo medico. Infine va evidenziata l'importanza della semplicità computazionale del metodo di interpretabilità (*algorithmic complexity*). Questa infatti determina l'effettiva utilizzabilità del metodo in questione, soprattutto in contesti dove è richiesta una certa rapidità nel produrre la spiegazione [12]. In ogni caso va sottolineato che, nonostante si trovi ancora in una fase embrionale del suo sviluppo, la ricerca di tecniche utili per la valutazione dell'adeguatezza delle spiegazioni prodotte da questi metodi rappresenta un ambito di ricerca di notevole rilevanza [24]. Infatti, l'utilizzo di un metodo di interpretabilità non adeguato non risolve (e anzi rischia di accentuare) le criticità legate alla scarsa interpretabilità dei *black-box model*. Anche per questo alcuni sottolineano come non si debba fare un eccessivo affidamento sull'utilizzo di tali metodi e che si debba piuttosto preferire lo sviluppo di nuovi modelli di ML intrinsecamente interpretabili [44].

4.3 Gli *additive feature attribution method*

Il funzionamento di molti dei metodi di interpretabilità locale introdotti nel corso degli anni consiste nell'attribuire ad ogni *feature* utilizzata come *input* nel modello di ML un numero reale, rappresentante il contributo (positivo o negativo) che quella *feature* ha avuto nel determinare la previsione oggetto di interpretazione [14]. Per questa loro caratteristica tali metodi vengono spesso indicati con il nome di *feature attribution method* [18]. In particolare, all'interno di questa classe, un successo notevole è stato raggiunto dai metodi che assumono come *explanation model* una funzione lineare del *semplified input* binario precedentemente introdotto (4.2):

$$g(\mathbf{x}') = \phi_0 + \sum_{j=1}^{p} \phi_j \mathbf{x}'_j, \tag{1}$$

con $\mathbf{x}' \in \{0,1\}^p$, p il numero di *feature* e $\phi_j \in \mathbb{R}$ [32].

La *mapping function* che lega l'*input* originale a quello semplificato solitamente prevede che $\mathbf{x}'_j = 0$ quando il valore di $\mathbf{x}_j$ non è noto (la j-esima *feature* non è stata osservata) e che $\mathbf{x}'_j = 1$ quando, viceversa, il valore di $\mathbf{x}_j$ è conosciuto [31]. Se P rappresenta l'insieme di tutte le *feature*, si può indicare con M il sottoinsieme di P ($M \subseteq P$) composto dalle sole variabili osservate nell'*input* $\mathbf{x}^*$ (quello rispetto a cui si vuole dare un'interpretazione della previsione prodotta dal modello). Di conseguenza si avrà che $\mathbf{x}'^*_j = 1$ per $j \in M$ e, viceversa, $\mathbf{x}'^*_j = 0$ per $j \notin M$. L'Equazione 1 può essere allora riscritta nella seguente forma [38]:

$$g(\mathbf{x}'^*) = \phi_0 + \sum_{j \in M} \phi_j. \tag{2}$$

In virtù di questa particolare formulazione, i metodi caratterizzati da un *explanation model* come quello in Equazione 2 vengono comunemente definiti *additive feature attribution method* (AFAM) [31]. Questi, infatti, assegnano a ogni *feature* osservata un coefficiente ϕ e la previsione $f(\mathbf{x}^*)$ viene approssimata dalla somma di questi coefficienti più un termine ϕ_0 che, essendo pari a $g(\mathbf{x}'^*)$ quando $M = \emptyset$, rappresenta un'approssimazione della previsione prodotta dal modello quando nessuna *feature* è stata osservata [32]. Ogni coefficiente potrà essere quindi interpretato come una misura dell'importanza assunta dalla corrispondente variabile nel determinare la previsione di interesse [1].

Quando un metodo di interpretabilità appartenente alla classe degli AFAM viene utilizzato per spiegare la previsione relativa ad una specifica istanza di interesse $\mathbf{x}^*$, sarebbe opportuno che vengano rispettate le seguenti tre proprietà:

- *Local accuracy*: l'*explanation model* g, costruito per approssimare il modello f in relazione alla specifica istanza $\mathbf{x}^*$, quando valutato sull'*input* semplificato $\mathbf{x}'^*$ deve esattamente coincidere con la previsione prodotta dal modello per quella specifica istanza [32]:

$$f(\mathbf{x}^*) = g(\mathbf{x}'^*) = \phi_0 + \sum_{j \in M} \phi_j. \tag{3}$$

  In sostanza, quando valutato in $\mathbf{x}'^*$, g non deve essere solo una approssimazione di f ma deve bensì emularne in modo esatto il comportamento. Di conseguenza, la somma dei coefficienti attribuiti alle varie *feature*, aggiunta a ϕ_0, deve esattamente ricostruire la previsione $f(\mathbf{x}^*)$ oggetto di interpretazione [31].

- *Missingness*: se nell'*input* originale $\mathbf{x}^*$ una certa *feature* j-esima è assente, allora il corrispettivo coefficiente deve essere nullo [32]:

$$j \notin M \Rightarrow \phi_j = 0. \tag{4}$$

  Questa proprietà richiede quindi che alle *feature* non osservate non venga attribuita nessuna importanza nel determinare una previsione [31]. Va sottolineato che tutti gli AFAM, data la loro struttura, rispettano questa proprietà. Si è infatti già affermato che se $j \notin M$ allora $\mathbf{x}'^*_j = 0$ [32].

- *Consistency*: dati due modelli f e f' e indicando che $\mathbf{x}'_j = 0$ con la notazione $\mathbf{x}' \setminus j$, se:

$$f'(h_{\mathbf{x}}(\mathbf{x}')) - f'(h_{\mathbf{x}}(\mathbf{x}' \setminus j)) \geq f(h_{\mathbf{x}}(\mathbf{x}')) - f(h_{\mathbf{x}}(\mathbf{x}' \setminus j)) \tag{5}$$

  per ogni vettore binario $\mathbf{x}' \in \{0,1\}^p$, allora deve valere che $\phi_j(f', \mathbf{x}^*) \geq \phi_j(f, \mathbf{x}^*)$ [38]. La *consistency* richiede quindi che se passando da un modello f a un modello f' il contributo marginale della j-esima *feature* sulle previsioni non decresce qualsiasi sia l'*input* $\mathbf{x}'$, allora il coefficiente (e quindi l'importanza) assegnata a quella *feature* non deve diminuire [31].

Il perché venga ritenuto desiderabile il rispetto di tali proprietà è piuttosto evidente: queste garantiscono che il metodo di interpretabilità produca delle spiegazioni che siano ragionevoli e perciò affidabili. Ad esempio, non ci si potrebbe fidare di una spiegazione che attribuisca un'importanza non nulla a una *feature* nemmeno osservata (e che quindi non può aver influenzato la previsione) [1].

I vari AFAM si differenziano in base a come i coefficienti ϕ vengono stimati. Tra questi si possono trovare sia metodi *model-agnostic*, come il LIME (introdotto nella Sezione 8.1.1) e lo *Shapley Value* (si veda il Capitolo 6), sia metodi *model-specific*, come il *Deep*LIFT (utilizzato per l'interpretazione dei soli DNN) [32]. Lundberg e Lee (2017) in [32], basandosi sui risultati ottenuti nel contesto della teoria dei giochi cooperativi (discussa nella Sezione 6.1) da Shapley (1953) in [45] e Young (1985) in [48], hanno dimostrato che esiste un unico *additive feature attribution method* che rispetta tutte e tre le proprietà sopra menzionate: lo *Shapley Value* (SV). Questo risultato ha favorito lo sviluppo di metodi di interpretabilità *Shapley Value based*, il cui funzionamento è basato sulla formulazione dello SV, e, tra questi, una notevole diffusione è stata ottenuta dal metodo SHAP (a cui sarà dedicato il Capitolo 7) [18].

5 Le *set function*

Prima di procedere con la trattazione dello *Shapley Value* e dei metodi di interpretabilità da esso derivanti, si ritiene opportuna una digressione di carattere teorico riguardante le *set function* ed altre tematiche ad esse collegate. Queste infatti rappresentano il supporto teorico necessario per la comprensione degli argomenti che verranno successivamente affrontati. Per questo, piuttosto che illustrarle in modo disaggregato all'interno delle successive sezioni con il rischio di rendere confusionaria l'esposizione, si è preferito raccoglierle in questo Capitolo dedicato.[1]

5.1 Giochi, capacità e misure

Dato un insieme (*set*) finito X, si indica con x un suo generico elemento ($x \in X$), con $n = |X|$ la sua cardinalità, cioè il numero di elementi in esso contenuto, e con $\mathcal{P}(X)$ (o alternativamente con 2^X) il suo *power set*, cioè il *set* composto da tutti i sui sottoinsiemi (*subsets*) compreso l'insieme vuoto $\emptyset$ e l'insieme X stesso: $\mathcal{P}(X) = \{A : A \subseteq X\}$ [25]. Essendo X un insieme finito, la cardinalità del suo *power set* è data da:

$$|\mathcal{P}(X)| = \sum_{k=0}^{n} \binom{n}{k} = 2^n, \tag{6}$$

dove il coefficiente binomiale $\binom{n}{k}$ è definito come:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \tag{7}$$

e conta il numero di sottoinsiemi non ordinati composti da k elementi che si possono ottenere a partire da un *set* di n elementi [11].

Una *set function* ξ definita su X è una funzione del tipo $\xi : \mathcal{P}(X) \to \mathbb{R}$, che assegna ad ogni elemento del *power set* di X un numero reale. In base alle loro caratteristiche le *set function* vengono suddivise in:

- Giochi (*games*): sono *set function* caratterizzate dall'essere *grounded*, cioè tali che $\xi(\emptyset) = 0$, e vengono solitamente indicati con v. Il loro nome ha origine nell'ambito dalla teoria dei giochi cooperativi dove vengono utilizzati per rappresentare l'utilità ottenuta da una coalizione di giocatori (si veda la Sezione 6.1).

- Capacità (*capacities*): sono *set function* caratterizzate dall'essere *groundend* e monotone, cioè tali che se presi due generici insiemi $A, B \in \mathcal{P}(X)$ con $A \subseteq B$, allora deve valere $\xi(A) \leq \xi(B)$. Da notare che la combinazione di queste due caratteristiche implica anche la non-negatività, cioè $\xi(A) \geq 0, \forall A \in \mathcal{P}(X)$. Le capacità vengono solitamente indicate con μ.

[1] Il presente Capitolo è stata prodotto quasi interamente estraendo e rielaborando i concetti espressi all'interno del libro "Set functions, games and capacities in decision making" di Grabisch (2016) [21]. Si eviterà quindi di citare continuamente la fonte in questione sottintendendo l'origine delle informazioni riportate. Verranno invece regolarmente menzionate le altre fonti utilizzate nella stesura di tale capitolo.

- Misure (*measures*): sono *set function* caratterizzate dall'essere non-negative e additive, cioè tali che se presi due generici insiemi $A, B \in \mathcal{P}(X)$ con $A \cap B = \emptyset$ (A e B sono due insiemi disgiunti), allora deve valere $\xi(A \cup B) = \xi(A) + \xi(B)$. Da notare che l'additività, in quanto caratteristica più restrittiva, implica anche la più generica monotonia. Inoltre risulta evidente che per una *set function* additiva vale che $\xi(A) = \sum_{x \in A} \xi(\{x\})$. Una misura caratterizzata anche dall'essere normalizzata, cioè tale che $\xi(X) = 1$, viene detta *probability measure*. Le misure vengono solitamente indicate con m.

Da quanto appena detto si può notare che le capacità possono essere viste come dei giochi monotoni e, di conseguenza, le prime rappresentano un caso particolare dei secondi. Ragionando nell'ambito della teoria dei giochi cooperativi, si può pensare ad una capacità come ad un gioco che non permette collaborazioni non benefiche. Analogamente una misura è un caso particolare di capacità in cui la monotonia viene sostituita dalla più restrittiva additività. Per questo motivo le capacità vengono spesso indicate con il nome di *fuzzy measure*.

5.2 La derivata di una *set function*

Si consideri una *set function* ξ definita sull'insieme X. Presi il sottoinsieme $A \subseteq X$ e l'elemento $i \in X$, la derivata di ξ rispetto all'elemento i e valutata in A è definita come:

$$\Delta_i \xi(A) = \xi(A \cup \{i\}) - \xi(A \backslash \{i\}) \tag{8}$$

e rappresenta la variazione della *set function* ξ conseguente all'introduzione (o rimozione) dell'elemento i nel *set* A. Di conseguenza, se $\Delta_i \xi(A) > 0$ allora l'introduzione di i produce un incremento del valore di ξ e viceversa se $\Delta_i \xi(A) < 0$. Da notare che l'elemento i può appartenere o non appartenere all'insieme A. Nel primo caso si avrà che $\xi(A \cup \{i\}) = \xi(A)$ mentre nel secondo caso si avrà che $\xi(A \backslash \{i\}) = \xi(A)$.

Risulta evidente che la derivata di una *set function* definita sull'insieme X è essa stessa una *set function* definita sullo stesso insieme: $\Delta_i \xi : \mathcal{P}(X) \to \mathbb{R}$. Per questo motivo viene naturale il pensare di poter derivare $\Delta_i \xi$ rispetto a un altro elemento $j \in X$. Si può allora definire la derivata di secondo ordine di una *set function* ξ rispetto alla coppia di elementi $i, j \in X$ e valutata in A come:

$$\Delta_{ij} \xi(A) = \Delta_j(\Delta_i \xi(A)) = \xi(A \cup \{i, j\}) - \xi(A \cup \{j\} \backslash \{i\}) - \xi(A \cup \{i\} \backslash \{j\}) + \xi(A \backslash \{i, j\}) \tag{9}$$

e rappresenta la variazione subita dalla derivata prima della *set function* ξ conseguente all'introduzione (o rimozione) dell'elemento j nel *set* A. Di conseguenza, se $\Delta_{ij} \xi(A) > 0$ allora la presenza di j nell'insieme A accentua la variazione della *set function* prodotta dall'introduzione di i in A e viceversa se $\Delta_{ij} \xi(A) < 0$. Per questo è possibile vedere $\Delta_{ij} \xi(A)$ come una sorta di interazione tra i due elementi i e j. Da notare che l'orine di derivazione è ininfluente sul risultato: $\Delta_j(\Delta_i \xi(A)) = \Delta_i(\Delta_j \xi(A))$. Infine, se $i, j \notin A$, l'Equazione 9 può essere riscritta come [38]:

$$\Delta_{ij} \xi(A) = \xi(A \cup \{i, j\}) - \xi(A \cup \{j\}) - \xi(A \cup \{i\}) + \xi(A). \tag{10}$$

Quanto detto sulla derivata di secondo ordine di una *set function* può essere generalizzato attraverso l'introduzione della derivata di ordine k-esimo. Preso un sottoinsieme $K \subseteq X$, la derivata della *set function* ξ rispetto a K e valutata in A è definita come:

$$\Delta_K \xi(A) = \Delta_{K\setminus\{i\}}(\Delta_i \xi(A)) = \sum_{L\subseteq K} (-1)^{|K\setminus L|} \xi((A\setminus K) \cup L)) \tag{11}$$

deve L è un generico sottoinsieme di K e $|K\setminus L|$ è la cardinalità dell'insieme $K\setminus L$. In questo caso è possibile vedere $\Delta_K \xi(A)$ come una sorta di interazione tra tutti gli elementi dell'insieme K, tuttavia dare un'interpretazione a tale interazione risulta estremamente complicato . Chiaramente, se si pone $K = \{i, j\}$ si ottiene esattamente l'Equazione 9. Ponendo invece $K = \emptyset$ si può agevolmente verificare che $\Delta_\emptyset \xi(A) = \xi(A)$. Da notare infine che se $K \cap A = \emptyset$, allora l'Equazione 11 può essere riscritta come:

$$\Delta_K \xi(A) = \sum_{L\subseteq K} (-1)^{|K\setminus L|} \xi(A \cup L). \tag{12}$$

5.3 La trasformata di una *set function*

Indicando con $\mathbb{R}^{(2^X)}$ lo spazio di tutte le *set function* definite sull'insieme X, una trasformazione (*transformation*) T è una funzione del tipo $T : \mathbb{R}^{(2^X)} \to \mathbb{R}^{(2^X)}$, che assegna a ogni *set function* ξ la sua trasformata (*transform*) $T(\xi)$. Una trasformazione T viene detta lineare se vale:

$$T(\alpha\xi_1 + \xi_2) = \alpha T(\xi_1) + T(\xi_2), \quad \forall \xi_1, \xi_2 \in \mathbb{R}^{(2^X)}, \tag{13}$$

con α un generico numero reale. La trasformazione è invece detta invertibile se esiste la sua funzione inversa T^{-1}, che permette di passare dalla trasformata $T(\xi)$ alla *set function* originale ξ. Tra le varie possibili trasformazioni, in questa sede risulta interessante menzionarne due: la *Möbius transform* e l'*interaction transform*.

Presa una generica *set function* ξ definita sull'insieme X e preso un sottoinsieme $A \subseteq X$, la *Möbius transform* $\mathbf{m}^\xi$ valutata in A della *set function* ξ è definita come:

$$\mathbf{m}^\xi(A) = \sum_{B\subseteq A} (-1)^{|A\setminus B|} \xi(B), \tag{14}$$

con B un generico sottoinsieme di A. Si può facilmente notare che se $A = \emptyset$ allora $\mathbf{m}^\xi(\emptyset) = \xi(\emptyset)$. La *Möbius transform* è una trasformazione lineare ed invertibile, per questo è possibile recuperare la *set function* ξ attraverso l'uso della sua trasformazione inversa che si può dimostrare essere data da:

$$\xi(A) = \sum_{B\subseteq A} \mathbf{m}^\xi(B). \tag{15}$$

L'*interaction transform* I^{ξ} valutata in A della *set function* ξ è invece definita come:

$$I^{\xi}(A) = \sum_{B \subseteq X \setminus A} \frac{(n-b-a)!b!}{(n-a+1)!} \Delta_A \xi(B), \tag{16}$$

con $n = |X|$, $a = |A|$, $b = |B|$ e B che in questo caso rappresenta un generico sottoinsieme di $X \setminus A$. Notando che gli insiemi A e B sono per costruzione disgiunti, si può sostituire $\Delta_A \xi(B)$ con l'Equazione 12 ottenendo:

$$I^{\xi}(A) = \sum_{B \subseteq X \setminus A} \frac{(n-b-a)!b!}{(n-a+1)!} \sum_{L \subseteq A} (-1)^{|A \setminus L|} \xi(B \cup L)), \tag{17}$$

con L un generico sottoinsieme di A. Il nome attributo a questa trasformazione deriva dal fatto che essa può essere vista come una media pesata lungo tutti i possibili sottoinsiemi $B \subseteq X \setminus A$ delle derivate $\Delta_A \xi(B)$, le quali, nella Sezione 5.2, si è già detto poter essere interpretate come dei termini di interazione tra gli elementi dell'insieme B. In ogni caso l'interpretazione di questa trasformazione, contestualizza nella teoria dei giochi cooperativi, verrà approfondita nella Sezione 6.4. Anche l'*interaction transform* è una trasformazione lineare ed invertibile, si può quindi dimostrare che la *set function* ξ può essere recuperata attraverso la seguente trasformazione inversa:

$$\xi(A) = \sum_{D \subseteq X} \beta^{|D|}_{|A \cap D|} I^{\xi}(D), \tag{18}$$

con D un generico sottoinsieme di X. Indicando con d la cardinalità di D e con k la cardinalità di $A \cap D$, il coefficiente β^d_k viene definito come:

$$\beta^d_k = \sum_{j=0}^{k} \binom{k}{j} \mathbf{B}_{d-j}. \tag{19}$$

Da notare che, essendo $A \cap D$ un sottoinsieme di D, allora necessariamente si avrà che $0 \leq k \leq d$. Nel caso in cui $D = A$ si avrà che $A \cap D = D$ e di conseguenza $k = d$. Se invece $D = \emptyset$ allora $A \cap D = \emptyset$ e allora $k = 0$. La quantità $\mathbf{B}$ viene chiamata *Bernulli number* ed è calcolata attraverso la seguente regola:

$$\mathbf{B}_m = -\frac{1}{m+1} \sum_{l=0}^{m-1} \binom{m+1}{l} \mathbf{B}_l, \tag{20}$$

dove m è un generico numero naturale e $\mathbf{B}_0 = 1$. Si può osservare che il calcolo del *Bernulli number* avviene in maniera ricorsiva, in quanto è possibile calcolare $\mathbf{B}_m$ solo conoscendo il valore di tutti gli $\mathbf{B}_l$ (con $l = 0, ..., m-1$), e che se $k = 0$ allora $\beta^d_k = \beta^d_0 = \mathbf{B}_d$. Si può inoltre dimostrare che $\xi(X) = \sum_{i \in X} I^{\xi}(\{i\}) + \xi(\emptyset)$. Infine, risulta interessante menzionare il fatto che la *Möbius transform* e l'*interaction transform* sono legate tra loro attraverso la seguente relazione:

$$I^{\xi}(A) = \sum_{C \supseteq A} \frac{1}{c-a+1} \mathbf{m}^{\xi}(C), \tag{21}$$

dove C è un generico sovrainsieme (*superset*) di A e c è la sua cardinalità.

5.4 I giochi *k*-additivi

Un gioco v, nella sua accezione più generale, è una *set function* non additiva (si veda la Sezione 5.1). Tuttavia è possibile introdurre una particolare categoria di giochi, detti k-additivi, caratterizzati da un certo livello (anche totale) di additività. Preso un gioco v definito sull'insieme X, questo viene detto k-additivo se e solo se $\mathbf{m}^v(A) = 0$ per ogni sottoinsieme $A \subseteq X$ tale che $|A| > k$ ed esiste almeno un insieme A tale che $|A| = k$ per cui vale $\mathbf{m}^v(A) \neq 0$. Per esempio, un gioco viene detto 2-additivo se per tutti i sottoinsiemi A composti da almeno tre elementi la *Möbius transform* valutata in A vale 0, mentre per alcuni sottoinsiemi A composti da esattamente due elementi la *Möbius transform* valutata in A assume un valore diverso da 0. Da notare che se $k = 1$ il gioco è perfettamente additivo. Infatti, in base all'Equazione 15, un gioco v può essere scritto nella seguente forma:

$$v(A) = \sum_{B \subseteq A} \mathbf{m}^v(B). \tag{22}$$

Se v è 1-additivo, per tutti i sottoinsiemi $B \subseteq A$ tali che $|B| > 1$ si avrà $\mathbf{m}^v(B) = 0$ e, notando che $\mathbf{m}^v(\emptyset) = v(\emptyset) = 0$ in quanto un gioco è una *set function grounded*, l'Equazione 22 può essere riscritta come:

$$v(A) = \sum_{x \in A} \mathbf{m}^v(\{x\}). \tag{23}$$

In base alla definizione di *Möbius transform* (Equazione 14) si può notare che:

$$\mathbf{m}^v(\{x\}) = \sum_{C \subseteq \{x\}} (-1)^{|\{x\} \setminus C|} v(C) = -v(\emptyset) + v(\{x\}) = v(\{x\}). \tag{24}$$

Infine, sostituendo questo risultato nell'Equazione 23, si ottiene che:

$$v(A) = \sum_{x \in A} v(\{x\}), \tag{25}$$

che richiama la definizione di *set function* additiva data nella Sezione 5.1.

La definizione che si è data dei giochi k-additivi può essere espressa anche attraverso l'*interaction transform* I^v. Infatti, preso un generico insieme $A \subseteq X$ tale che $|A| > k$, se il gioco v è k-additivo non solo si avrà che $\mathbf{m}^v(A) = 0$ ma anche che $\mathbf{m}^v(C) = 0$ per ogni *superset* $C \supseteq A$ in quanto, per costruzione, $|C| \geq |A| > k$. Questo risultato, se inserito nell'Equazione 21, mostra che la k-additività implica $I^v(A) = 0$ per ogni $A \subseteq X$ tale che $|A| > k$, rendendo la definizione di gioco k-additivo esprimibile indifferentemente in termini di *Möbius* e di *interaction transform*. Va infine sottolineato che il concetto di k-additività può essere esteso anche a tutte le sottoclassi di giochi, come ad esempio le capacità.

5.5 L'integrale di *Choquet*

L'integrale di *Choquet* è uno dei possibili approcci utilizzabili per integrare una funzione rispetto a una *set function* non-additiva. In questa sezione verrà presentato per primo il caso generale, in cui la funzione da integrare è definita su un insieme generico, e successivamente il caso discreto, in cui tale funzione è definita su un insieme finito.

5.5.1 Il caso generale

Un'algebra $\mathcal{F}$ definita sull'insieme X è una classe di sottoinsiemi di X tale che [8]:

- $\mathcal{F}$ contiene l'insieme vuoto e l'insieme X stesso: $\emptyset, X \in \mathcal{F}$;
- se $A \in \mathcal{F}$ allora deve valere che $A^C \in \mathcal{F}$, con $A^C = X \backslash A$ l'insieme complementare di A;
- presi due insiemi $A, B \in \mathcal{F}$ allora deve valere che $A \cup B \in \mathcal{F}$ e $A \cap B \in \mathcal{F}$.

Dato un generico insieme X, un'algebra $\mathcal{F}$ definita su X e una capacità $\mu : \mathcal{F} \to \mathbb{R}^+$, una funzione $f : X \to \mathbb{R}$ viene detta $\mathcal{F}$-misurabile se gli insiemi $\{x : f(x) > t\}$ e $\{x : f(x) \geq t\}$ sono contenuti in $\mathcal{F}$, $\forall t \in \mathbb{R}$. Per una qualsiasi funzione limitata e $\mathcal{F}$-misurabile è possibile definire una *decumulative distribution function* (DDF) $G_{\mu,f} : \mathbb{R} \to \mathbb{R}$ rispetto alla capacità μ come:

$$G_{\mu,f}(t) = \mu(\{x \in X : f(x) \geq t\}). \tag{26}$$

In sostanza la DDF valutata in t equivale al valore della capacità μ in relazione all'insieme delle x tali per cui $f(x) \geq t$. Il fatto che f sia $\mathcal{F}$-misurabile è essenziale affinché la DDF sia ben definita. Infatti, per poter calcolare $G_{\mu,f}(t)$, la capacità μ deve essere definita sull'insieme $\{x \in X : f(x) \geq t\}$ che, per quanto già detto in precedenza, appartiene ad $\mathcal{F}$ solo se f è $\mathcal{F}$-misurabile. Preso un valore t' tale che $t' > t$, si può notare che:

$$\{x \in X : f(x) \geq t'\} \subseteq \{x \in X : f(x) \geq t\} \tag{27}$$

ed essendo la capacità μ per definizione una misura monotona e non-negativa allora $G_{\mu,f}(t)$ è una funzione non-crescente e non-negativa (Figura 5). Va infine evidenziato che, se la funzione f è non-negativa, allora per $t = 0$ tutte le $x \in X$ verificano $f(x) \geq t$ e, di conseguenza, $G_{\mu,f}(0) = \mu(X)$.

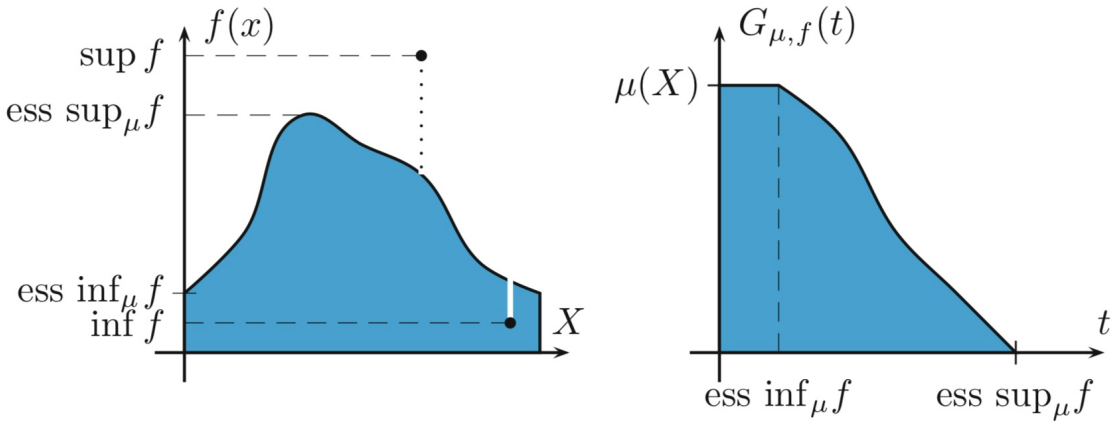

Figura 5: a sinistra una funzione limitata e non-negativa che assume valori in X. A destra la corrispondente *decumulative distribution function* costruita rispetto alla capacità μ. L'area sottostante la DDF è il valore dell'integrale di *Choquet* della funzione f rispetto alla capacità μ [21].

Data una funzione f che sia $\mathcal{F}$-misurabile, limitata e non-negativa e una capacità μ definita sull'algebra $\mathcal{F}$, l'integrale di *Choquet* rispetto a μ di f viene definito come:

$$\int f \mathrm{d}\mu = \int_0^\infty G_{\mu,f}(t)\mathrm{d}t \tag{28}$$

e rappresenta perciò l'area sottostante alla DDF di f rispetto a μ (Figura 5).

5.5.2 L'integrale di *Choquet* per una *simple function*

Dato un generico insieme X, una funzione $f : X \to \mathbb{R}$ è detta semplice (*simple function*) se la sua immagine (*range*) $\mathbf{ran}f$ è un insieme finito: $\mathbf{ran}f = \{a_1, ..., a_r\}$. Assumendo che questa funzione sia non-negativa e che gli elementi della sua immagine siano ordinati in modo crescente (cioè tali che $a_0 = 0 \leq a_1 < a_2 < ... < a_r$), allora questa può essere decomposta nella seguente forma:

$$f(x) = \sum_{i=1}^{r} a_i \mathbf{1}_{\{x \in X : f(x) = a_i\}}, \tag{29}$$

dove $\mathbf{1}_{\{x \in X : f(x) = a_i\}}$ è un indicatore che assume valore 1 quando $f(x) = a_i$ e 0 altrimenti. Se ad esempio si avesse $f(x) = a_1$, tutti gli indicatori con $i \neq 1$ sarebbero pari a 0 ed il valore della funzione in quello specifico x (cioè a_1) sarebbe perfettamente ricostruito. Questa decomposizione viene detta verticale in quanto, graficamente, l'andamento della funzione f viene scomposto in blocchi orientati verticalmente (Figura 6). Un'altra possibile decomposizione per la *simple function* sopra definita è la seguente:

$$f(x) = \sum_{i=1}^{r} (a_i - a_{i-1}) \mathbf{1}_{\{x \in X : f(x) \geq a_i\}}, \tag{30}$$

dove $\mathbf{1}_{\{x \in X : f(x) \geq a_i\}}$ è un indicatore che assume valore 1 quando $f(x) \geq a_i$ e 0 altrimenti. Se ad esempio si avesse $f(x) = a_1$, solo l'indicatore con $i = 1$ risulterebbe pari ad 1 e, di conseguenza, si otterrebbe $f(x) = (a_1 - a_0) = a_1$, ricostruendo perfettamente il valore di f in quello specifico x. Se invece si osservasse $f(x) = a_2$, risulterebbero pari ad 1 gli indicatori per $i = 1, 2$ e si avrebbe che $f(x) = (a_1 - a_0) + (a_2 - a_1) = a_2$. Questa decomposizione viene detta orizzontale in quanto, graficamente, la *simple function* viene scomposta in blocchi orientati orizzontalmente (Figura 6).

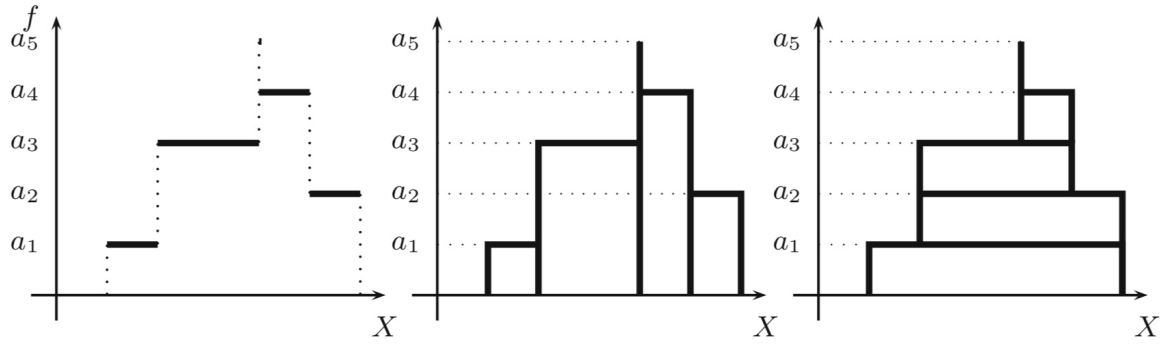

Figura 6: a sinistra l'andamento di una *simple function* la cui immagine assume $r = 5$ valori. Al centro la corrispondente decomposizione verticale. A destra la corrispondente decomposizione orizzontale [21].

Se si suppone di lavorare con un insieme finito $X = \{x_1, ..., x_r\}$ e con un'algebra $\mathcal{F} = \mathcal{P}(X)$ equivalente all'intero *power set* di X, allora qualsiasi funzione f definita su X sarà $\mathcal{F}$-misurabile (in quanto $\mathcal{F}$ contiene ogni possibile *subset* di X, compresi quelli che definiscono la $\mathcal{F}$-misurabilità) e semplice (in quanto a un dominio X finito corrisponde un'immagine $\mathbf{ran}f$ finita). In questo contesto si consideri una funzione f non-negativa e si prenda una permutazione dell'insieme X tale che l'immagine $\mathbf{ran}f$ sia ordinata in modo crescente: $0 = a_0 \leq f(x_1) = a_1 < f(x_2) = a_2 < ... < f(x_r) = a_r$.

Si può mostrare che la DDF della funzione f così definita è una funzione costante a tratti (Figura 7). Si prendano infatti gli insiemi $A_i = \{x \in X : f(x) \geq a_i\}$ per $i = 1, ..., r$. L'insieme A_1 sarà composto dagli elementi di X tali che $f(x) \geq a_1$ e, poiché a_1 è il più piccolo valore dell'immagine di f, A_1 coinciderà con X. L'insieme A_2 è invece composto dagli elementi di X tali che $f(x) \geq a_2$ e, poiché solamente a_1 è più piccolo di a_2, $A_2 = X\backslash\{x_1\}$. Si procede analogamente fino ad avere $A_r = \{x_r\}$. Riprendendo la definizione data nell'Equazione 26 risulta evidente il perché in questo caso la DDF abbia un andamento costante a tratti: per $0 \leq t \leq a_1$ si ottiene $G_{\mu,f}(t) = \mu(A_1) = \mu(X)$ in quanto, se tutte le x sono tali che $f(x) \geq a_1$, allora tutte le x verificheranno anche $f(x) \geq t$ essendo $a_1 \geq t$. Se invece si osservasse $a_1 < t \leq a_2$ si otterrebbe $G_{\mu,f}(t) = \mu(A_2) = \mu(X\backslash\{x_1\})$ in quanto, se tutte le x esclusa x_1 sono tali che $f(x) \geq a_2$, allora quelle stesse x verificheranno $f(x) \geq t$ essendo $a_2 \geq t$. Analogamente, per $a_{r-1} < t \leq a_r$ si avrà $G_{\mu,f}(t) = \mu(A_r) = \mu(\{x_r\})$ e, infine, per $t > a_r$ si avrà $G_{\mu,f}(t) = \mu(\emptyset) = 0$. Da notare inoltre che, essendo gli insiemi A_i costruiti in modo tale che $A_1 \supset A_2 \supset ... \supset A_r$ ed essendo la capacità μ una *set function* monotona, allora la DDF avrà un andamento non-crescente (Figura 7).

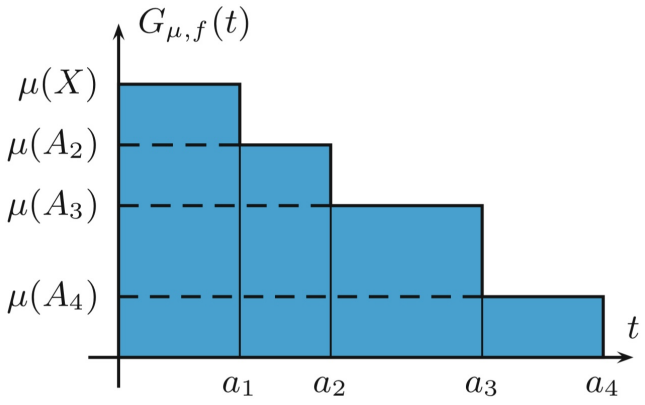

Figura 7: andamento della *decumulative distribution function* di una *simple function* f rispetto a una capacità μ. L'area in blu rappresenta l'integrale di *Choquet* di f rispetto a μ [21].

Per quanto affermato nell'Equazione 28, l'integrale di *Choquet* di una funzione f rispetto a una capacità μ rappresenta l'area sottostante la DDF di f. Nel caso della DDF di una *simple function* si può immaginare di calcolare quest'area suddividendola in rettangoli orientati verticalmente (Figura 7), come mostrato nella seguente formulazione:

$$\int f \mathrm{d}\mu = \sum_{i=1}^{r} (a_i - a_{i-1})\mu(A_i), \tag{31}$$

che richiama la decomposizione orizzontale della *simple function*.[2] Da notare che la non-crescenza della DDF non è un elemento necessario in questa formulazione dell 'integrale di *Choquet*. É infatti possibile calcolare l'area dei rettangoli orientati verticalmente anche qualora questi non fossero ordinati dal più alto al più basso. Di conseguenza, si può concludere che l'integrale di *Choquet* di una *simple function* è calcolabile anche rispetto a una *set function* non-monotona, come ad esempio un gioco. Inoltre, risulta evidente che la stessa area può essere calcolata anche suddividendola in rettangoli orientati orizzontalmente (Figura 7) tramite la seguente formulazione:

$$\int f \mathrm{d}\mu = \sum_{i=1}^{r} a_i(\mu(A_i) - \mu(A_{i+1})), \tag{32}$$

che richiama la decomposizione verticale della *simple function*. In questo caso si può notare che, qualora al posto della capacità si utilizzasse una *set function* additiva (come una misura), l'Equazione 32 potrebbe essere riscritta nella seguente forma:

$$\int f \mathrm{d}\mu = \sum_{i=1}^{r} a_i \mu(\{x_i\}), \tag{33}$$

in quanto, per costruzione, gli insiemi A_i e A_{i+1} differiscono per il solo elemento x_i. In questo caso l'integrale di *Choquet* è una media ponderata dei valori dell'immagine di f, utilizzando come pesi i valori della capacità additiva valutata sugli elementi del *set* X. Questo risultato permette di osservare che l'integrale di *Choquet* è una generalizzazione dell'integrale di *Lebesgue*, utilizzato per integrare una funzione rispetto a una misura (ad esempio per calcolare il valore atteso di una variabile casuale rispetto ad una certa *probability measure*). Infatti, nel discreto, l'integrale di *Lebesgue* assume le sembianze di una media ponderata, come quella in Equazione 33. Si può quindi considerare l'integrale di *Choquet* come una funzione di aggregazione non-additiva che generalizza la media ponderata [38]. Viene quindi introdotto con lo scopo di integrare una funzione rispetto ad una più generale *set function* non-additiva ma, in presenza di additività, diventa equivalente a quello di *Lebesgue*. Infine, si fa presente che le Equazioni 31 e 32 rimangono valide anche per funzioni f che prendono valori negativi, sempre riferendosi ad una permutazione che ordina l'immagine $\mathbf{ran}f$ in modo crescente e prendendo $a_0 = \mu(A_{r+1}) = 0$.

[2] In questo caso l'orientamento dei rettangoli è verticale nonostante la decomposizione della *simple function* sia orizzontale. Infatti nella DDF i valori dell'immagine di f sono posizionati sull'asse delle ascisse e non più su quella delle ordinate come avveniva in Figura 6.

6 Lo *Shapley Value*

Lo *Shapley Value* (SV) viene introdotto nel 1953 da Lloyd Shapley come una particolare soluzione nel contesto della teoria dei giochi cooperativi. Per questo, prima di approfondirne l'utilizzo nell'ambito dell'*eXplainable Artificial Intelligence* (oggetto della Sezione 7.1), si ritiene opportuno introdurne l'origine ed il funzionamento nel suo contesto originario.

6.1 La teoria dei giochi cooperativi

La teoria dei giochi (*game theory*) è una disciplina che studia metodi e strumenti matematici da utilizzare in contesti decisionali di tipo interattivo, cioè in situazioni in cui la decisione presa da un soggetto (detto giocatore o *player*) influenza i risultati ottenuti dagli altri e viceversa. Lo scopo è quello di fornire consigli ai giocatori riguardo i comportamenti che è più conveniente adottare, nonché quello di prevedere il comportamento che un certo giocatore potrà assumere in un determinato contesto [34]. Chiaramente, utilizzando strumenti matematici, il punto di vista di questa disciplina è razionale più che sociologico o psicologico. La *game theory*, nata agli inizi del ventesimo secolo, ha avuto negli anni una grande diffusione ed oggi trova applicazione in un vasto numero di ambiti: dalla teoria economica alle scienze politiche, dalle strategie militari alla biologia e, in generale, in qualsiasi contesto ci sia l'interesse di modellare l'interazione strategica tra più soggetti [5]. Solitamente, i giochi studiati nell'ambito di questa disciplina vengono suddivisi in due grandi gruppi: quelli strategici, in cui i giocatori agiscono individualmente, e quelli cooperativi, in cui i giocatori possono stringere degli accordi vincolanti tra loro, formando così delle coalizioni (*coalitions*) i cui membri agiranno in modo coordinato [34]. In questa sede l'attenzione viene rivolta a questo secondo gruppo di giochi.

Formalmente un gioco di tipo cooperativo può essere rappresentato attraverso la coppia (Q, v), dove $Q = \{1, ..., q\}$ è un insieme finito di giocatori e $v : \mathcal{P}(Q) \to \mathbb{R}$ è una *set function grounded* (si veda la Sezione 5.1) definita su Q, comunemente chiamata *characteristic function* [43]. Si definisce una coalizione S di giocatori ogni generico sottoinsieme di Q, compreso l'insieme Q stesso (detto *grand coalition*), l'insieme vuoto $\emptyset$ (detto *empty coalition*) e gli insiemi composti da un unico elemento (anche un singolo giocatore è considerato come una coalizione) [34]. La *characteristic function* assegna quindi ad ogni possibile coalizione S di giocatori un numero reale $v(S)$, con $v(\emptyset) = 0$. Il numero $v(S)$ viene detto valore (*worth*) della coalizione e rappresentante l'utilità che complessivamente potranno ottenere e spartirsi i membri di S attraverso la loro cooperazione [43]. Da notare che nei giochi cooperativi è spesso presente l'assunzione per cui il valore $v(S)$ non dipende in alcun modo dal comportamento dei giocatori che non partecipano alla coalizione S o dal valore ottenuto dalle altre coalizioni che si sono formate nel gioco, assunzione che tuttavia è difficile da ritrovare nella pratica e che, di conseguenza, riduce l'applicabilità di tali modelli [34]. Si sottolinea inoltre che in questa sede l'attenzione viene rivolta ad una particolare tipologia di giochi cooperativi, detti *transferable utility game*, in cui è presente l'assunzione per cui l'utilità viene incorporata in un mezzo fisico (ad esempio il denaro) ed è quindi perfettamente trasferibile tra i giocatori [43]. Ne deriva che forme di utilità come il prestigio e la reputazione non sono tenute in considerazione in questo contesto [34].

Tra le domande a cui la teoria dei giochi cooperativi cerca di rispondere, una delle più attenzionate nel contesto dei *transferable utility game* riguarda il modo in cui suddividere il valore $v(S)$ tra i membri della coalizione S. Per questo risulta di interesse dare la definizione di *solution concept*: indicando con $\mathcal{B}$ una *coalitional structure*, cioè una generica partizione[3] dell'insieme Q, un *solution concept* è una funzione φ che assegna ad ogni gioco (Q, v) e ad ogni *coalitional structure* $\mathcal{B}$ un sottoinsieme $\varphi(Q, v, \mathcal{B})$ di $\mathbb{R}^q$, con q il numero di giocatori [34]. In sostanza un *solution concept* è un insieme di punti in uno spazio q-dimensionale. Ognuno di questi punti, detti *point solution*, sarà composto da q coordinate, una per ogni giocatore, che possono quindi essere interpretate come l'utilità attribuita ad ognuno di essi. Una *point solution* può essere allora vista come uno dei possibili modi in cui suddividere l'utilità ottenuta da ogni coalizione tra i giocatori ad essa appartenenti. Chiaramente tale suddivisione dipende dal particolare gioco che si sta analizzando e dalla particolare *coalitional structure* che si è venuta a formare [34].

I giochi cooperativi su cui si concentra il lavoro di Shapley sono caratterizzati dall'ulteriore assunzione di superadditività: presa una qualsiasi coppia di coalizioni disgiunte S e T (cioè tali che $S \cap T = \emptyset$) deve valere che $v(A \cup B) \geq v(A) + v(B)$. Questa assunzione implica che l'utilità ottenuta in totale da due coalizioni separatamente non può essere maggiore di quella ottenuta dalla fusione delle due coalizioni [43]. In questo tipo di giochi esiste quindi un incentivo a formare coalizioni più numerose. La superadditivtà viene infatti introdotta con lo scopo di giustificare la formazione della *grand coalition*, cioè di una *coalitional structure* del tipo $\mathcal{B} = \{Q\}$. Così facendo si sposta l'attenzione sulla definizione di *solution concept* validi solo per questa specifica *coalitional structure* [34]. Shapley, inoltre, introduce il concetto di portante (*carrier*): indicando con Q l'universo di tutti i possibili giocatori, un portante è un qualsiasi insieme $C \subseteq Q$ tale che $v(S) = v(C \cap S)$ per ogni insieme $S \subseteq Q$. In sostanza, un giocatore i-esimo che non appartiene ad un certo portante C ($i \notin C$) non influenza il gioco, in quanto non contribuisce al valore ottenuto da qualsiasi coalizione S [45]. Per questo motivo tale giocatore viene solitamente chiamato *null player* [43].

Nel contesto appena descritto, l'obbiettivo di Shapley era quello di individuare una *point solution* che permettesse di suddividere il valore $v(Q)$, cioè l'utilità ottenuta dalla *grand coalition*, tra tutti i giocatori appartenenti a Q [34]. Indicando con $\varphi_i(v)$ il valore attribuito all'i-esimo giocatore in relazione al gioco v, questo è anche interpretabile come l'utilità che quel giocatore ricava dal partecipare a tale gioco e, di conseguenza, rappresenta una misura di quanto interesse quel giocatore può avere nel partecipare [45]. Shapley riteneva inoltre che questa *point solution* dovesse rispettare le seguenti proprietà da lui considerate come desiderabili:

- *Efficiency*: la somma dei valori attribuiti a tutti i possibili giocatori di un certo portante C deve coincidere con l'utilità complessivamente ottenuta dalla *grand coalition* Q. Formalmente:

$$\sum_{i \in C} \varphi_i(v) = v(Q). \tag{34}$$

  Questa proprietà è desiderabile in quanto è ragionevole che dei giocatori razionali si spartiscano l'intera utilità ottenuta dalla *grand coalition*, senza alcuno spreco [34].

[3] Una partizione di un insieme Q è una famiglia di sottoinsiemi non vuoti, disgiunti e la cui unione sia pari a Q [34].

- *Symmetry*: presi due giocatori i e j tali che $i, j \in Q$, se presa una qualsiasi coalizione S tale che $S \subseteq Q \backslash \{i, j\}$ vale:

$$v(S \cup \{i\}) = v(S \cup \{j\}), \tag{35}$$

  allora deve valere che $\varphi_i(v) = \varphi_j(v)$ [34]. Questa proprietà richiede quindi che l'identità del giocatore non abbia alcun peso nel determinare il valore ad esso attributo [43]. Infatti, è ragionevole pensare che se due giocatori sono identici da un punto di vista strategico, cioè contribuiscono in modo identico in qualsiasi coalizione vengano inseriti, allora a questi venga attribuito lo stesso valore [34].

- *Null player*: preso un giocatore i, se per ogni coalizione S tale che $S \subseteq Q \backslash \{i\}$ vale:

$$v(S \cup \{i\}) = v(S), \tag{36}$$

  allora deve valere che $\varphi_i(v) = 0$ [1]. Il giocatore in questione è un *null player*, in quanto non influenza in alcun modo l'utilità ottenuta da qualsiasi coalizione in cui venga inserito. Per questo è ragionevole richiedere che a tale giocatore non venga attribuito nessun valore [34].

- *Linearity*: presi due giochi v e w, la loro somma $(v + w)$ è essa stessa un gioco ed è tale per cui:

$$(v + w)(S) = v(S) + w(S), \quad \forall S \subseteq Q. \tag{37}$$

  In pratica, si può pensare alla somma di due giochi come una situazione in cui le coalizioni di giocatori partecipano simultaneamente a due giochi tra loro indipendenti (l'utilità che una coalizione ottiene in un gioco non influenza quella ottenuta dalla stessa coalizione nell'altro). Per questo, l'utilità ottenuta da una coalizione nel gioco somma si ottiene sommando le utilità ottenute nei giochi singolarmente presi [34]. La *linearity* richiede che, presi due giochi v e w, debba valere:

$$\varphi_i(v + w) = \varphi_i(v) + \varphi_i(w) \tag{38}$$

  per ogni $i \in Q$ [43]. Viene quindi richiesto che, quando due giochi sono indipendenti tra loro, il valore assegnato ad ogni giocatore partecipante alla combinazione dei due giochi sia pari alla somma dei valori assegnati a quel giocatore nei singoli giochi [45]. Questa proprietà risulta come una diretta conseguenza della definizione data nell'Equazione 37 [34].

Shapley (1953) in [45] riuscì a dimostrare che esiste un'unica *point solution* in grado di rispettare le quattro proprietà da lui stesso introdotte. Per questo tale soluzione, indicata con ϕ, è oggi conosciuta con il nome di *Shapley Value* (si veda la Sezione 6.2) [45].

La derivazione dello *Shapley Value* proposta da Shapley non convince pienamente a causa della proprietà di linearità su cui essa si basa. Infatti, a differenza di quanto avviene per le altre proprietà, non è così chiaro il perché la linearità sia un'assunzione ragionevole nell'individuazione di una *point solution* [34]. Per questo motivo Young (1985) in [48] introduce una nuova derivazione dello *Shapley Value* che sostituisce la linearità con una più ragionevole proprietà nota come monotonia (*monotonicity*): presi due qualsiasi giochi v e w definiti sullo stesso insieme Q e preso un generico giocatore $i \in Q$, se

vale:

$$v(S \cup \{i\}) - v(S) \geq w(S \cup \{i\}) - w(S) \quad \forall S \subseteq Q \backslash \{i\}, \tag{39}$$

allora deve valere che $\varphi_i(v) \geq \varphi_i(w)$. La monotonia è una proprietà ragionevole in quanto richiede che, se il contributo di un giocatore in ogni coalizione di un gioco è non inferiore al suo contributo nelle stesse coalizioni di un altro gioco, allora il valore attribuito a tale giocatore nel primo gioco non deve essere inferiore a quello attribuitogli nel secondo [34]. Young (1985) per prima cosa dimostra che ogni *solution concept* che rispetta efficienza, simmetria e monotonia rispetta anche la proprietà del *null player*, che diventa quindi ridondante e può essere rimossa. In secondo luogo dimostra che lo *Shapley Value* è l'unico *solution concept* che rispetta le tre proprietà appena menzionate, rendendo la linearità un'assunzione non più necessaria nella definizione dello SV [48].

6.2 La definizione di *Shapley Value*

Dato un *set* finito di giocatori $Q = \{1, ..., q\}$ e una *characteristic function* v, lo *Shapely Value* $\phi_i(v)$ associato all'i-esimo giocatore è definito come segue:

$$\phi_i(v) = \sum_{S \subseteq Q \backslash \{i\}} \frac{s!(q-s-1)!}{q!} [v(S \cup \{i\}) - v(S)], \tag{40}$$

con $q = |Q|$ e $s = |S|$ [21]. Lo SV si ottiene quindi come una media pesata lungo tutte le possibili coalizioni S che non contengono il giocatore i-esimo dei termini $[v(S \cup \{i\}) - v(S)]$, che rappresentano il contributo marginale di tale giocatore in quelle coalizioni [43]. Si può infatti notare che il termine $[v(S \cup \{i\}) - v(S)]$ non è nient'altro se non la derivata prima della *set function* v rispetto all'elemento i e valutata in S, cioè $\Delta_i v(S)$ (Equazione 8). In base a quanto visto nella Sezione 5.2, questa derivata si interpreta come la variazione dell'utilità ottenuta da una coalizione S a fronte dell'introduzione del giocatore i all'interno di tale coalizione. Se questa variazione è positiva il contributo marginale del giocatore è positivo e viceversa [21]. Per interpretate i pesi assegnati a questi contributi marginali si deve osservare che $\Delta_i v(S)$ dipende dal numero e dall'identità dei giocatori appartenenti alla coalizione S ma non dall'ordine in cui questi sono entrati in tale coalizione. Se $q!$ rappresenta il numero di modi in cui i giocatori appartenenti all'insieme Q possono essere ordinati[4], allora $s!$ rappresenta il numero di modi in cui i giocatori appartenenti alla coalizione S possono essere ordinati e $(q-s-1)!$ rappresenta il numero di modi in cui i giocatori non appartenenti alla coalizione S ed escluso il giocatore i-esimo possono essere ordinati. Di conseguenza, $s!(q-s-1)!$ è il numero di permutazioni degli q giocatori a cui corrisponde lo stesso contributo marginale $\Delta_i v(S)$, in quanto rappresenta tutti i possibili ordini con cui il giocatore i-esimo può entrare nella coalizione S al variare dell'ordine dei membri appartenenti e non appartenenti a tale coalizione [34]. La quantità $\frac{s!(q-s-1)!}{q!}$ rappresenta quindi la proporzione di permutazioni che produrranno lo stesso contributo marginale sul totale delle permutazioni $q!$ osservabili e, di conseguenza, è il peso che si attribuisce a tale contributo nella media ponderata. Chiaramente, la somma di questi pesi lungo tutte le coalizioni $S \subseteq Q \backslash \{i\}$ è pari ad uno [30].

[4] Si ricorda che, preso un insieme di q elementi, $q!$ rappresenta il numero totale delle permutazioni di tali elementi [34].

Per quanto detto, lo *Shapley Value* può essere interpretato come il contributo marginale che il giocatore i-esimo si aspetta mediamente di apportare partecipando al gioco [43]. Tale contributo marginale medio viene quindi valutato a prescindere dalla specifica coalizione a cui il giocatore si andrà effettivamente ad unire, in quanto tiene in considerazione tutte le possibili coalizioni di giocatori che si possono formare. Per questo, anche in virtù delle proprietà desiderabili che lo SV risulta rispettare, è ragionevole utilizzarlo come metodo per spartire l'utilità complessivamente ottenuta dalla *grand coalition* [38]. Da notare infine che, data questa sua formulazione, allo SV sono state affiancate altre possibili interpretazioni, come quella di potere o influenza del giocatore all'interno di un gioco cooperativo [33] o come quella di misura utile per valutare la convenienza a partecipare a tale gioco [45].

6.3 Lo *Shapley Value* a gruppi

Il problema principale che si riscontra nella formulazione dello *Shapely Value* presentata nell'Equazione 40 riguarda la sua complessità computazionale (problema che verrà ripreso anche nella Sezione 7.3). Si può infatti notare che tale formulazione presenta 2^{q-1} termini, uno per ogni possibile coalizione che si può formare a partire dal set $Q\backslash\{i\}$ (si veda la Sezione 5.1). Risulta quindi evidente che, al crescere del numero di giocatori q, il numero di termini da valutare aumenta esponenzialmente, rendendo il calcolo sempre più complesso. Una delle possibili soluzioni a questo problema di natura computazionale è l'utilizzo dello SV a gruppi, cioè una sua formulazione che, invece di concentrarsi su un singolo giocatore, prende in considerazione un gruppo di giocatori [27]. In letteratura è possibile individuare due differenti approcci nel calcolo dello SV a gruppi: il *Group Shapley* (GS) e lo *Shapley generalized value* (SGV).

Dato un *set* finito di giocatori $Q = \{1, ..., q\}$ e una *characteristic function* v, si consideri una partizione $\mathcal{G} = \{G_1, ..., G_g\}$ dell'insieme Q, dove ogni insieme G_k con $k = 1, ..., g$ rappresenta un gruppo di giocatori. Il *Group Shapley* $\phi_k^{GS}(v)$ relativo al k-esimo gruppo è definito come:

$$\phi_k^{GS}(v) = \sum_{\mathcal{T} \subseteq \mathcal{G}\backslash G_k} \frac{|\mathcal{T}|!(g - |\mathcal{T}| - 1)!}{g!}[v(\mathcal{T} \cup G_k) - v(\mathcal{T})], \tag{41}$$

dove $g = |\mathcal{G}|$ è il numero di gruppi, $\mathcal{T}$ è una collezione di gruppi contenuta nella partizione $\mathcal{G}$ e $v(\mathcal{T})$ rappresenta il valore del gioco v in relazione alla coalizione di tutti i giocatori che compongono i gruppi appartenenti a $\mathcal{T}$[5] [27]. Si può notare che il GS è semplicemente un'estensione dello SV in cui, invece di considerare i sottoinsiemi S di un *set* di giocatori Q, si considerano delle famiglie di insiemi $\mathcal{T}$ contenute nella partizione $\mathcal{G}$. In sostanza, è come se ogni gruppo G_k venisse considerato come un singolo giocatore e di quest'ultimo si calcolasse lo SV. Il termine $[v(\mathcal{T} \cup G_k) - v(\mathcal{T})]$ può essere interpretato come il contributo marginale che il gruppo k-esimo apporta al gioco quando inserito nella coalizione di gruppi $\mathcal{T}$ e il GS rappresenta la media ponderata di questi contributi, calcolata lungo ogni possibile coalizione di gruppi formata a partire dalla partizione $\mathcal{G}$ al netto del gruppo k-esimo. Analogamente, nel determinare il peso attributo a ogni contributo marginale, si considereranno le $g!$ possibili permutazioni dei gruppi invece che le $q!$ possibili permutazioni dei giocatori. In virtù della stretta corrispondenza

[5] Analogamente $v(\mathcal{T} \cup G_k)$ è il valore del gioco v in relazione alla coalizione dei giocatori che compongono i gruppi appartenenti a $\mathcal{T}$ uniti a quelli che compongono il gruppo G_k.

esistente con lo SV, risulta evidente che le proprietà presentate nella Sezione 6.1 sono rispettate anche dal GS e allora, per l'efficienza, vale che $\sum_{k=1}^{g} \phi_k^{GS}(v) = v(Q)$ [27]. Risulta altrettanto evidente il vantaggio computazionale conseguente all'utilizzo del GS: il numero di termini da valutare sono infatti 2^{g-1}, uno per ogni possibile coalizione di gruppi $\mathcal{T}$ che si può formare a partire dalla famiglia di gruppi $\mathcal{G}\backslash G_k$. Quindi, a meno che ogni gruppo non sia composto da un singolo giocatore, si avrà che $g < q$ e, di conseguenza, $2^{g-1} < 2^{q-1}$. Ad esempio, se si avessero $q = 20$ giocatori e $g = 5$ gruppi, il numero di termini da calcolare passerebbe da 2^{19}, cioè oltre mezzo milione, a $2^4 = 16$ [27].

Nello stesso contesto descritto per il GS, si indichi con $g_k = |G_k|$ il numero di giocatori presenti nel gruppo k-esimo. Lo *Shapley generalized value* $\phi_k^{SGV}(v)$ relativo al gruppo in questione è definito come:

$$\phi_k^{SGV}(v) = \sum_{S \subseteq Q \backslash G_k} \frac{s!(q-s-g_k)!}{(q-g_k+1)!}[v(S \cup G_k) - v(S)], \tag{42}$$

dove $v(S \cup G_k)$ rappresenta il valore del gioco v in relazione alla coalizione composta dai giocatori appartenenti ad S uniti a quelli appartenenti al gruppo G_k [33]. Da questa formulazione si può evincere che la differenza tra i due approcci non risiede nel termine $[v(S \cup G_k) - v(S)]$, che anche in questo caso rappresenta il contributo marginale del gruppo k-esimo quando introdotto nella coalizione S, ma nelle coalizioni che vengono tenute in considerazione nel calcolo della media di tali contributi marginali. Infatti, se nel GS si valutano le sole coalizioni $\mathcal{T}$ ottenute combinando i gruppi della partizione $\mathcal{G}$ al netto del gruppo k-esimo, nello SGV, fissato un gruppo G_k, si valutano tutte le coalizioni ottenute combinando i singoli giocatori del *set* Q al netto di quelli appartenenti al gruppo fissato. In sostanza, se nel GS si valuta in che modo un gruppo contribuisca al gioco in relazione agli altri gruppi, nello SGV si valuta in che modo tale gruppo contribuisca al gioco in relazione agli altri giocatori presi individualmente [27]. Si può notare come l'ordine dei giocatori appartenenti al k-esimo gruppo non influisca sul contributo marginale che questo produce quando inserito nella coalizione S. Di conseguenza, nell'assegnare il peso a tale contributo, non è necessario tenere in considerazione le possibili permutazioni dei g_k giocatori in esso contenuti. Per questo, il numero totale di permutazioni passa dall'essere $q!$ all'essere $(q-g_k+1)!$, come se i g_k giocatori del gruppo k-esimo venissero trattati come un singolo individuo. I pesi dello SGV vengono quindi ottenuti effettuando questa sostituzione all'interno di quelli dello SV [3]. Il calcolo dello SGV relativo al gruppo k-esimo richiede la valutazione di 2^{q-g_k} termini, uno per ogni coalizione S che si può formare a partire dall'insieme $Q\backslash G_k$. Si può quindi notare che questo secondo approccio risulta meno vantaggioso da un punto di vista computazionale in quanto, generalmente, il numero $q - g_k$ di giocatori non appartenenti al gruppo k-esimo è maggiore del numero di gruppi g. Ad esempio, se i 5 gruppi del caso precedentemente descritto fossero tutti della stessa numerosità, cioè $g_k = 4$, allora per calcolare lo SGV sarebbe necessario valutare più di 60 mila termini (2^{16}), contro i 16 necessari nel GS [27]. Si può inoltre osservare che, essendo il numero di termini da valutare nello SGV dipendente dalla numerosità del gruppo k-esimo, la sua difficoltà computazionale può variare da gruppo a gruppo, mentre rimane invariata nel GS dove dipendente solamente dal numero di gruppi. Come conseguenza, se nel GS è necessario che tutti i gruppi della partizione $\mathcal{G}$ siano composti da un solo giocatore per azzerare il vantaggio computazionale da esso derivante, nello SGV è sufficiente un solo gruppo unitario per vedere il suo vantaggio azzerato. Tuttavia, fintanto che il gruppo k-esimo è composto da più di un giocatore, la complessità computazionale resta notevolmente inferiore rispetto allo SV. Va inoltre sottolineato che in

questo secondo approccio viene meno quella stretta corrispondenza che emergeva tra lo SV e il GS e, di conseguenza, la proprietà di efficienza non risulta più rispettata.

Entrambi gli approcci appena presentati rappresentano di fatto delle generalizzazioni dello SV. Si può infatti notare che, qualora la partizione $\mathcal{G}$ fosse composta da soli gruppi di singoli giocatori, allora le Equazioni 41 e 42 sarebbero equivalenti alla formulazione dello SV. Oltre che con lo scopo di ridurne la complessità computazionale, il GS e lo SGV possono essere utilizzati per ricavare informazioni che lo SV non è in grado di fornire. Infatti, in certe situazioni, si può essere più interessati a valutare il contributo di un gruppo piuttosto che di uno singolo giocatore [33]. La somma degli SV corrispondenti ai giocatori appartenenti al gruppo di interesse non sembra essere una buona soluzione al problema posto, in quanto risulta meno consistente da un punto di vista formale rispetto al GS e allo SGV ed inoltre, richiedendo il calcolo degli SV, presenta i problemi di complessità computazionale già descritti [27]. Chiaramente, essendo costruiti in maniera differente, i due approcci proposti daranno una risposta parzialmente diversa allo stesso problema. In particolare lo SGV, valutando il contributo di un gruppo in relazione a ogni possibile coalizione di giocatori, fornisce una risposta più generale rispetto a quanto fatto dal GS, che invece valuta tale contributo solo rispetto alle coalizioni di altri gruppi [27].

Il GS e lo SGV non sono tuttavia privi di criticità. Intanto emerge il problema di selezionare un criterio da utilizzare per suddividere i giocatori nei vari gruppi. Risulta evidente che, al variare del contesto in cui si sta lavorando, varieranno i criteri di suddivisione utilizzati e non è quindi possibile dare una risposta generale a questo problema. Nella Sezione 7.2 si menzioneranno alcune possibili soluzioni a questa criticità nel contesto dell'*eXplainable* AI. Un secondo limite nell'utilizzo di questi approcci sta nel fatto che non restituiscono lo stesso tipo di informazione fornita dallo SV e, di conseguenza, se l'obbiettivo è quello di valutare il contributo di uno specifico giocatore all'interno del gioco, questi non rappresentano più una strada percorribile [27].

6.4 La *Shapley Interaction*

Dato un set di giocatori $Q = \{1, ..., q\}$, una *characteristic function* v e presi due generici giocatori $i, j \in Q$ è possibile introdurre la seguente quantità:

$$I_{i,j}(v) = \sum_{S \subseteq Q \setminus \{i,j\}} \frac{s!(q-s-2)!}{(q-1)!}[v(S \cup \{i,j\}) - v(S \cup \{i\}) - v(S \cup \{j\}) + v(S)], \qquad (43)$$

nota con il nome di *Shapley Interaction* (SI) [38]. Per comprendere il perché di questo nome si deve notare che il termine $[v(S \cup \{i,j\}) - v(S \cup \{i\}) - v(S \cup \{j\}) + v(S)]$ non è nient'altro se non la derivata seconda della *set function* v rispetto agli elementi i, j e valutata in S, cioè $\Delta_{ij}v(S)$ (Equazione 10). In base a quanto visto nella Sezione 5.2, questa derivata si può interpretare come l'interazione tra i giocatori i e j nel gioco v in questione, cioè in che modo la compresenza dei due giocatori in una coalizione influenza il loro contributo marginale in tale coalizione [21]. Infatti, come visto nella Sezione 6.3 parlando di SGV, la quantità $[v(S \cup \{i,j\}) - v(S)]$, indicata per semplicità con δ, rappresenta il contributo marginale del gruppo $\{i,j\}$ quando inserito nella coalizione S. Analogamente, le quantità $[v(S \cup \{i\}) - v(S)]$ e $[v(S \cup \{j\}) - v(S)]$, per quanto già osservato nella Sezione 6.2, sono i contributi marginali dei giocatori i e j quando inseriti da soli nella coalizione S, cioè le derivate prime $\Delta_i v(S)$ e

$\Delta_j v(S)$. Si può facilmente osservare che la derivata seconda $\Delta_{ij} v(S)$ si ottiene come:

$$\Delta_{ij} v(S) = \delta - (\Delta_i v(S) + \Delta_j v(S)) \tag{44}$$

e, di conseguenza, sarà positiva quando il contributo marginale dei giocatori i e j come gruppo è maggiore della somma dei loro contributi marginali individuali, indicando un'interazione positiva tra i due giocatori. Viceversa, la derivata seconda sarà negativa quando il contributo marginale di gruppo è inferire alla somma di quelli individuali, indicando un'interazione negativa tra i due giocatori [21]. Chiaramente, al variare della coalizione S, può variare l'interazione tra i due giocatori ed è quindi utile valutare la media pesata di queste interazioni lungo tutte le possibili coalizioni $S \subseteq Q \backslash \{i, j\}$. Questa media pesata è esattamente la *Shapley Interaction* [21]. Ricordando che la derivata seconda di una *set function* non dipende dall'ordine di derivazione (si veda la Sezione 5.2), nell'assegnare i pesi alle interazioni $\Delta_{ij} v(S)$ non è rilevante l'ordine con cui i giocatori i e j entrano nella coalizione S. Per questo, nel valutare la proporzione di permutazioni che restituisce lo stesso termine $\Delta_{ij} v(S)$, è possibile trattare quella coppia di giocatori come un unico giocatore e, di conseguenza, i pesi utilizzati nella *Shapley Interaction* si ottengono lavorando con le permutazioni di $(q-1)$ giocatori [3]. Per quanto detto è facile intuire l'interpretazione che si può dare alla SI: quando $I_{i,j}(v) > 0$ (< 0), i giocatori i e j in media interagiscono positivamente (negativamente) e allora hanno un interesse maggiore a cooperare (non cooperare) tra loro. Qualora invece si osservasse $I_{i,j}(v) = 0$, i due giocatori non interagiscono in alcun modo e non ci sono quindi né incentivi né disincentivi a cooperare [22]. Si può infine notare che la SI è una generalizzazione dello SV. Se infatti si sostituisse la coppia $\{i, j\}$ con il singolo giocatore $\{i\}$ e la derivata seconda $\Delta_{ij} v(S)$ con $\Delta_i v(S)$, si tornerebbe alla formulazione presentata nell'Equazione 40 [21].

Se è possibile parlare di interazione tra due giocatori è altrettanto possibile farlo per un gruppo di giocatori. Nello specifico, presa una coalizione A di dimensione arbitraria, si può introdurre una generalizzazione della SI:

$$I_A(v) = \sum_{S \subseteq Q \backslash A} \frac{s!(q-s-a)!}{(q-a+1)!} \Delta_A v(S), \tag{45}$$

con $a = |A|$ e $l = |L|$ [38]. Questa, analogamente a quanto appena visto per il caso con due giocatori, rappresenta una media ponderata delle interazioni tra gli a giocatori appartenenti alla coalizione A. Va tuttavia detto che, essendo la derivata $\Delta_A v(S)$ di non facile interpretazione (come visto nella Sezione 5.2), questa interazione generalizzata non ha un significato chiaro e resta perciò un concetto di utilità teorica (come si vedrà nella Sezione 8.2) più che un informazione effettivamente utilizzabile nella pratica [38]. Quello che risulta interessante notare è che questa interazione generalizzata non è nient'altro se non l'*interaction transform* presentata nella Sezione 5.3, con la differenza che qui l'oggetto della trasformazione non è una *set function* generica ξ ma è bensì un gioco v. Si può quindi affermare che lo SV e la SI sono dei casi particolari di tale trasformazione, rispettivamente per $A = \{i\}$ e $A = \{i, j\}$ [21].

7 Il metodo SHAP

7.1 Dalla *game theory* all'*eXplainable Artificial Intelligence*

Dal momento della sua introduzione nel 1953, lo *Shapley Value* ha ottenuto un grande successo nell'ambito della teoria dei giochi cooperativi, tanto da ispirare una notevole produzione di articoli scientifici volti ad interpretarlo, riformularlo ed estenderlo proponendone versioni più generali [43]. Questo notevole interesse maturato nei sui confronti ha favorito la sua diffusione in numerosi ambiti di applicazione, tra cui quello dell'*eXplainable Artificial Intelligence*. Si è infatti già evidenziato (si veda la Sezione 4.3) che lo SV può essere visto come un metodo di interpretabilità locale di tipo *model-agnostic* rientrante nella categoria degli *additive feature attribution method* [32]. Per comprendere in che modo possa essere utilizzato ed interpretato in questo differente contesto, si ritiene necessario mostrare come sia possibile passare dal *framework* della teoria dei giochi cooperativi e quello dell'*eXplainable Artificial Intelligence*.

Si consideri un classico scenario di *supervised learning* (introdotto nella Sezione 3.1): un *black-box model* f viene addestrato su un *training set* $D = (\mathbf{X}, \mathbf{y})$, dove $\mathbf{X}$ e $\mathbf{y}$ rappresentano rispettivamente una matrice di dati di *input* e un vettore di dati di *output*.[6] In particolare si ha che $\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_n]$ e $\mathbf{y} = [y_1, ..., y_n]$, dove n è il numero di istanze osservate, $\mathbf{x}_i$ è un vettore di osservazioni (una per ogni *feature*) relative all'i-esima istanza e y_i è l'osservazione della variabile risposta per l'i-esima istanza (con $i = 1, ..., n$). Si indica con $\mathbf{x}^* = [x_1^*, ..., x_m^*]$ una certa istanza di interesse non appartenente al *training set*, dove x_j^* è l'osservazione della j-esima *feature* per tale istanza (con $j = 1, ..., m$). L'obbiettivo è quello di fornire una spiegazione della previsione $f(\mathbf{x}^*)$ utilizzando lo SV [38]. Riprendendo la notazione utilizzata nella Sezione 4.3, $P = \{1, ..., p\}$ rappresenta l'insieme di tutte le *feature* osservabili mentre $M = \{1, ..., m\}$ è un sottoinsieme di P contenente le sole *feature* osservate in relazione all'istanza $\mathbf{x}^*$. In questo caso, per semplicità di notazione, si assume che $M = P$, cioè che per l'istanza $\mathbf{x}^*$ sia stato possibile osservare tutte le variabili utilizzate per addestrare il modello. Il passaggio dalla teoria dei giochi cooperativi allo scenario appena descritto avviene considerando l'insieme di *feature* M come il *set* di giocatori Q e il *black-box model* f come la *characteristic function* v del gioco [1]. Lo *Shapley Value* $\phi_j(f, \mathbf{x}^*)$ associato alla j-esima *feature* in relazione alla previsione prodotta dal modello f sull'istanza $\mathbf{x}^*$ si può allora definire come:

$$\phi_j(f, \mathbf{x}^*) = \sum_{S \subseteq M \setminus \{j\}} \frac{s!(m-s-1)!}{m!} [f_{S \cup \{j\}}(\mathbf{x}^*_{S \cup \{j\}}) - f_S(\mathbf{x}^*_S)], \tag{46}$$

dove s è la cardinalità del generico sottoinsieme di *feature* S, $\mathbf{x}^*_S$ è il vettore composto dalle osservazioni relative all'istanza $\mathbf{x}^*$ per le sole *feature* appartenenti all'insieme S e f_S è il modello f addestrato sul *training set* $\mathbf{X}$ ma utilizzando solamente le *feature* appartenenti all'insieme S [32]. In analogia a quanto detto nella Sezione 6.2, questo può essere interpretato come il contributo marginale medio che la *feature* j-esima apporta nella previsione prodotta dal modello f in relazione all'istanza $\mathbf{x}^*$. Risulta perciò evidente il perché sia possibile utilizzarlo come un metodo di interpretabilità locale: esso permette di as-

[6] Si sottolinea che lo *Shapley Value* può essere indifferentemente utilizzato in contesti di *supervised* e *unsupervised learning*, in quanto, come si vedrà in questa sezione, la sua formulazione prescinde dal vettore di *output* $\mathbf{y}$. In questa sede si è scelto di presentare uno scenario supervisionato solo per garantire coerenza con la parte sperimentale che verrà presentata nel Capitolo 9.

segnare a ogni variabile un valore rappresentante l'importanza (positiva o negativa) che questa ha avuto nel determinare una certa previsione [14].

Lundberg e Lee (2017) in [32] sono riusciti a dimostrare che lo SV presentato nell'Equazione 46 è l'unico AFAM a rispettare le proprietà di *local accuracy*, *missingness* e *consistency* definite nella Sezione 4.3. Questa dimostrazione parte dalla derivazione dello SV basata sulle proprietà di *efficiency*, *symmetry* e *monotonicity* proposta da Young (1985) ed introdotta nella Sezione 6.1. Lundberg e Lee mostrano infatti che, nel contesto dell'*eXplainable* AI, la simmetria è implicita nella monotonia e può quindi essere omessa. Di conseguenza, notando che la *local accuracy* e la *consistency* sono il corrispettivo nell'ambito dell'interpretabilità dell'*efficiency* e della *monotonicity*, la dimostrazione presentata da Young secondo cui lo SV è l'unico *solution concept* a rispettare monotonia, simmetria ed efficienza può essere estesa in questo contesto. Lo SV è quindi l'unico AFAM a rispettare la *local accuracy* e la *consistency*, oltre alla *missingness* che, per costruzione, è una proprietà verificata da tutti gli AFAM [32].

Data questa sua unicità, lo SV dovrebbe essere preferito rispetto agli altri AFAM, in quanto questi potrebbero portare a risultati che violano la consistenza e/o la *local accuracy*. Va però detto che la formulazione presentata nell'Equazione 46 non risulta particolarmente attrattiva, necessitando l'addestramento di due modelli di ML per ogni termine $[f_{S\cup\{j\}}(\mathbf{x}^*_{S\cup\{j\}}) - f_S(\mathbf{x}^*_S)]$ valutato nella media ponderata [32]. Per questo lo SV ha osservato un notevole successo nell'ambito dell'*eXplainable* AI non tanto in questa sua versione ma attraverso le formulazioni dei metodi *Shapley Value based*, cioè versioni dello SV che differiscono per la scelta della funzione f utilizzata nella valutazione dei contributi marginali [18]. Questi metodi infatti, mantenendo la struttura che caratterizza lo SV, continuano a rispettare le proprietà desiderabili sopra menzionate. Chiaramente, cambiando la funzione di valutazione f, cambierà anche l'interpretazione dei valori attributi alle singole *feature* [32]. Tra i metodi SV *based*, quello che recentemente ha ottenuto il maggior successo è lo *SHapley Additive exPlanations value*, che per questo verrà approfondito nella seguente sezione.

7.2 Lo *SHapley Additive exPlanations value*

Si consideri lo stesso scenario descritto nella precedente sezione. Indicando con $\phi_j(f, \mathbf{x}^*)$ lo *SHapley Additive exPlanations value* (SHAP)[7] associato alla j-esima *feature* in relazione alla previsione $f(\mathbf{x}^*)$, questo viene calcolato nel seguente modo:

$$\phi_j(f, \mathbf{x}^*) = \sum_{S\subseteq M\setminus\{j\}} \frac{s!(m-s-1)!}{m!}[\hat{f}_{\mathbf{x}^*}(S\cup\{j\}) - \hat{f}_{\mathbf{x}^*}(S)], \tag{47}$$

dove la quantità $\hat{f}_{\mathbf{x}^*}(S)$ rappresenta la previsione attesa (o media) che ci si aspetta dal modello f data l'osservazione dell'istanza $\mathbf{x}^*$ relativamente alle sole *feature* appartenenti alla coalizione S, cioè [38]:

$$\hat{f}_{\mathbf{x}^*}(S) = \mathbb{E}[f(\mathbf{x}) \mid x_j = x^*_j, \forall j \in S]. \tag{48}$$

[7]Da qui in avanti la notazione $\phi_j(f, \mathbf{x}^*)$ verrà utilizzata indifferentemente per indicare lo SV e lo SHAP. Inoltre, per semplicità, tale notazione verrà spesso sostituita con ϕ_j, sottintendendo il riferimento al modello f e all'istanza $\mathbf{x}^*$.

Risulta quindi evidente che lo SHAP non è nient'altro se non lo SV applicato al valore atteso condizionato del modello originale f, cioè quello addestrato sulla totalità del *training set*. Di conseguenza, diversamente da quanto avviene per lo SV descritto nella Sezione 7.1, per utilizzare lo SHAP è necessario l'addestramento di un solo modello di ML, di cui poi viene valutato il valore atteso condizionatamente alla coalizione S di interesse [32].

Per capire in che modo vada interpretato lo SHAP si deve innanzitutto notare che, se la coalizione S coincide con l'intero insieme M, allora $\hat{f}_{\mathbf{x}^*}(M) = \mathbb{E}[f(\mathbf{x}) \mid \mathbf{x} = \mathbf{x}^*] = \mathbb{E}[f(\mathbf{x}^*)] = f(\mathbf{x}^*)$, cioè la previsione attesa, condizionatamente all'intero vettore $\mathbf{x}^*$, coincide esattamente con la previsione oggetto di interpretazione. Viceversa, se $S = \emptyset$ si osserva $\hat{f}_{\mathbf{x}^*}(\emptyset) = \mathbb{E}[f(\mathbf{x})]$, cioè la previsione attesa, quando non condizionata a nessun elemento del vettore $\mathbf{x}^*$, rappresenta la media delle previsioni fatte dal modello lungo tutte le possibili istanze. Lo SHAP vuole spiegare in che modo si passi dalla previsione media $\mathbb{E}[f(\mathbf{x})]$ alla specifica previsione $f(\mathbf{x}^*)$ [38]. Si può allora immaginare di inserire una alla volta le osservazioni delle varie *feature* per l'istanza di interesse. Aggiungendo x_1^* si passa da $\hat{f}_{\mathbf{x}^*}(\emptyset) = \mathbb{E}[f(\mathbf{x})]$ a $\hat{f}_{\mathbf{x}^*}(\{x_1\})$, aggiungendo x_2^* si passa da $\hat{f}_{\mathbf{x}^*}(\{x_1\})$ a $\hat{f}_{\mathbf{x}^*}(\{x_1, x_2\})$ e si continua analogamente finché, aggiungendo x_m^*, si passa da $\hat{f}_{\mathbf{x}^*}(M \backslash \{x_m\})$ a $\hat{f}_{\mathbf{x}^*}(M) = f(\mathbf{x}^*)$. Alla fine di questo processo si sarà quindi osservato il contributo marginale che ogni variabile ha avuto nel determinare la previsione di interesse $f(\mathbf{x}^*)$ a partire dalla previsione media $\mathbb{E}[f(\mathbf{x})]$ (Figura 8) [32]. Si deve però notare che il contributo apportato dalla j-esima variabile dipende dal numero e dall'identità delle variabili inserite prima di lei (non invece dall'ordine delle variabili inserite prima e dopo di lei). Per tenere conto di questo si può allora utilizzare una media ponderata dei contributi di tale variabile, al variare delle coalizioni di *feature* S inserite prima di lei nel modello. La media in questione è esattamente lo SHAP ed i pesi utilizzati, in virtù dell'evidente analogia con il ragionamento presentato nella Sezione 6.2, sono i pesi dello SV. Il valore ϕ_j può allora essere interpretato come il contributo marginale medio che la j-esima *feature* ha prodotto nel passare dalla previsione media a quella di interesse e può essere visto come una misura dell'importanza (positiva o negativa) che tale variabile ha avuto nel determinare $f(\mathbf{x}^*)$ [38].

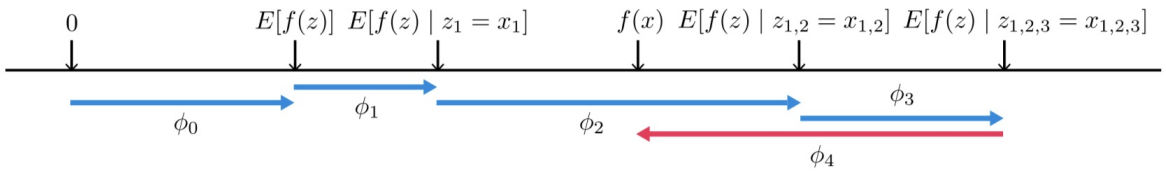

Figura 8: gli SHAP associati alle varie *feature* ricostruiscono in che modo il modello sia passato dalla previsione media ϕ_0 alla previsione oggetto di interpretazione [32].

Lo SHAP, mantenendo la struttura dello SV, rispetta tutte le proprietà desiderabili da quest'ultimo possedute, tra cui la *local accuracy*. Se per ogni $S \subseteq M$ si indica con $\mathbf{1}_S \in \{0,1\}^m$ il vettore binario tale che la sua coordinata j-esima è pari a 1 se $j \in S$ e pari a 0 se $j \notin S$, l'*input* semplificato $\mathbf{x}'^*$ può essere indicato con $\mathbf{1}_M$ [38]. Infatti, sotto l'assunzione per cui $M = P$ introdotta nella precedente sezione, ogni elemento dell'*input* semplificato sarà pari ad uno (si veda la Sezione 4.3). La proprietà di *local accuracy* presentata nell'Equazione 3 può essere allora scritta come:

$$f(\mathbf{x}^*) = g(\mathbf{1}_M) = \phi_0 + \sum_{j=1}^{m} \phi_j, \tag{49}$$

dove i coefficienti ϕ_j rappresentano gli SHAP associati alle varie *feature* [38]. In virtù dell'interpretazione data dello SHAP, risulta evidente che la sommatoria $\sum_{j=1}^{m} \phi_j$ rappresenta la somma dei contributi delle m *feature* nel passare da $\mathbb{E}[f(\mathbf{x})]$ a $f(\mathbf{x}^*)$ e, di conseguenza, il coefficiente ϕ_0 rappresenta esattamente la previsione media globale $\mathbb{E}[f(\mathbf{x})]$ [1].

Data la formulazione dello SHAP presentata nell'Equazione 47, si può notare che, per poterlo calcolare, è necessario conoscere le quantità $\hat{f}_{\mathbf{x}^*}(S) = \mathbb{E}[f(\mathbf{x}) \mid x_j = x_j^*, \forall j \in S]$ per ogni coalizione di *feature* S contenuta in M (si veda la Sezione 7.3). Per semplicità di notazione si indica con $\mathbf{x}_S = \{x_j : j \in S\}$ il vettore delle *feature* contenute nella coalizione S e con $\mathbf{x}_S^* = \{x_j^* : j \in S\}$ il vettore con le osservazioni di quelle *feature* relativamente all'istanza $\mathbf{x}^*$. Si indica inoltre con $\bar{S} = M \backslash S$ l'insieme delle *feature* non contenute nella coalizione S [35]. Per la definizione di valore atteso condizionato si ha che:

$$\hat{f}_{\mathbf{x}^*}(S) = \mathbb{E}[f(\mathbf{x}) \mid \mathbf{x}_S = \mathbf{x}_S^*] = \mathbb{E}[f(\mathbf{x}_{\bar{S}}, \mathbf{x}_S) \mid \mathbf{x}_S = \mathbf{x}_S^*] = \int f(\mathbf{x}_{\bar{S}}, \mathbf{x}_S^*) p(\mathbf{x}_{\bar{S}} \mid \mathbf{x}_S = \mathbf{x}_S^*) \mathrm{d}\mathbf{x}_{\bar{S}}, \quad (50)$$

dove il termine $p(\mathbf{x}_{\bar{S}} \mid \mathbf{x}_S = \mathbf{x}_S^*)$ rappresenta la distribuzione di probabilità del vettore $\mathbf{x}_{\bar{S}}$ condizionata a $\mathbf{x}_S = \mathbf{x}_S^*$ [1]. Tuttavia, questa distribuzione di probabilità non è quasi mai nota, rendendo impossibile il calcolo delle quantità $\hat{f}_{\mathbf{x}^*}(S)$ e, di conseguenza, dello SHAP. Per superare questa problematica un approccio molto diffuso è quello di assumere l'indipendenza tra tutte le *feature*. Infatti, sotto questa assunzione, i vettori $\mathbf{x}_S$ e $\mathbf{x}_S^*$ risultano indipendenti e la distribuzione condizionata $p(\mathbf{x}_{\bar{S}} \mid \mathbf{x}_S = \mathbf{x}_S^*)$ può essere sostituita con quella non condizionata $p(\mathbf{x}_{\bar{S}})$ [1]. L'Equazione 50 può allora essere riscritta come:

$$\hat{f}_{\mathbf{x}^*}(S) = \int f(\mathbf{x}_{\bar{S}}, \mathbf{x}_S^*) p(\mathbf{x}_{\bar{S}}) \mathrm{d}\mathbf{x}_{\bar{S}} \quad (51)$$

e, utilizzando la Monte Carlo *integration*, il suo valore può essere approssimato nel seguente modo:

$$\hat{f}_{\mathbf{x}^*}(S) \approx \frac{1}{T} \sum_{t=1}^{T} f(\mathbf{x}_{\bar{S}}^{(t)}, \mathbf{x}_S^*), \quad (52)$$

con i campioni Monte Carlo $\mathbf{x}_{\bar{S}}^{(t)}$ che vengono estratti casualmente dal *training set*, considerato come la distribuzione di probabilità empirica delle *feature* [35]. In sostanza, per calcolare la quantità $\hat{f}_{\mathbf{x}^*}(S)$ sotto l'assunzione di indipendenza, si prende dal *training set* un campione di T osservazioni delle *feature* contenute in $\bar{S}$ e si fa una media aritmetica delle previsioni prodotte dal modello fissando il vettore $\mathbf{x}_S$ pari a $\mathbf{x}_S^*$ e variando il vettore $\mathbf{x}_{\bar{S}}$ in base alla t-esima osservazione.

Data l'analogia strutturale tra lo SHAP e lo SV, le estensioni di quest'ultimo che sono state trattate nelle Sezioni 6.3 e 6.4 sono utilizzabili anche nel contesto dell'interpretabilità. Si può allora parlare di *Group SHAP* (GS), di *SHAP generalized value* (SGV) e di *SHAP interaction* (SI)[8] semplicemente sostituendo $v(S)$ con $\hat{f}_{\mathbf{x}^*}(S)$ nelle corrispettive formule (Equazioni 41, 42 e 45). In particolare, il GS e lo SGV possono essere utilizzati per investigare il contributo che un gruppo di *feature* ha avuto nel determinare una certa previsione del modello [27]. Questi metodi possono essere quindi utili per ricavare un tipo di informazione che lo SHAP non è in grado di fornire, per ridurre la complessità computazionale del metodo di interpretabilità (si veda la Sezione 6.3) e per rendere più comprensibile la spiegazione relativa

[8] Per evitare inutili complicazioni si utilizzano gli stessi acronimi già visti per le estensioni dello SV.

alla previsione di interesse. Infatti, in contesti con un numero elevato di *feature*, si avrà un altrettanto elevato numero di SHAP. L'eccessiva scomposizione della previsione da interpretare rischia quindi di compromettere l'effettiva comprensione delle cause determinanti tale risultato. L'utilizzo di gruppi di *feature* può quindi evitare l'insorgere di questo tipo di problematica. Nel contesto dell'interpretabilità, un criterio che risulta naturale nella costruzione dei gruppi è quelli di accorpare *feature* con significati affini [27]. Ad esempio, se le variabili osservate sono degli indici di bilancio si può pensare di suddividerle in indici di profittabilità, di liquidità, di indebitamento, e così via (come si vedrà nella Sezione 9.1). La SI può essere invece utilizzata per valutare in che modo due *feature* interagiscano nel determinare la previsione di interesse, ossia in che modo la loro presenza simultanea nel modello influisca sui contributi da esse apportati in tale previsione [38].

7.3 Le criticità del metodo SHAP

Il metodo SHAP descritto nella precedente sezione, utile per l'interpretazione locale delle previsioni prodotte dai *black box model*, presenta due grandi criticità che ne compromettono notevolmente le possibilità di applicazione.

La prima riguarda l'assunzione di indipendenza delle *feature*, che si è detto essere necessaria per consentire un calcolo agevole delle quantità $\hat{f}_{\mathbf{x}^*}$. Questa assunzione risulta infatti difficilmente rispettata dai *dataset* osservati nel mondo reale. Come conseguenza, in caso di elevata dipendenza/correlazione tra due o più variabili, le quantità $\hat{f}_{\mathbf{x}^*}$ saranno lontane dal loro valore reale e gli SHAP calcolati risulteranno completamente errati, producendo spiegazioni totalmente fuorvianti [1]. Alcune soluzioni a questa criticità sono state proposte da Aas et al. (2021) in [1] e consistono in diversi possibili approcci (parametrici e non) utilizzabili per stimare la distribuzione condizionata $p(\mathbf{x}_{\bar{S}} \mid \mathbf{x}_S = \mathbf{x}_S^*)$, permettendo così di rilassare l'assunzione di indipendenza. In questa sede tuttavia tali approcci non verranno approfonditi e, per questo, si continuerà a considerare come valida tale assunzione.

La seconda criticità riguarda la complessità computazionale insita nello SV (già menzionata nella Sezione 6.3) e che, per l'analogia strutturale tra i due concetti, caratterizza anche lo SHAP. Si può infatti notare che, nel calcolo del valore ϕ_j, per ogni termine della sommatoria nell'Equazione 47 risulta necessario stimare due quantità $\hat{f}_{\mathbf{x}^*}$: una relativamente alla coalizione S e l'altra relativamente alla coalizione $S \cup \{j\}$. Essendoci in totale 2^{m-1} coalizioni $S \subseteq M \backslash \{j\}$, si devono perciò calcolare $2 \times 2^{m-1} = 2^m$ valori attesi[9], uno per ogni coalizione $S \subseteq M$ [35]. Da notare inoltre che questi 2^m valori sono sufficienti per calcolare gli SHAP relativi a tutte le m *feature* dell'insieme M. Sarà infatti sufficiente ricombinarli in modo adeguato nelle rispettive sommatorie. Finché il numero m di *feature* risulta molto piccolo (per esempio minore di 10), questa criticità può non emergere. Tuttavia, se i *dataset* sono caratterizzati da decine di *feature*, cosa molto comune nel contesto del ML, il metodo SHAP diventa semplicemente inutilizzabile [27]. Una possibile soluzione a questo problema di natura computazionale è rappresentata dall'utilizzo del GS o dello SGV. Si è infatti già detto (si veda la Sezione 6.3) che questi metodi, lavorando con gruppi di variabili, permettono la valutazione di un numero inferiore di coalizioni, fintanto che i

[9] In realtà i valori da stimare sono $2^m - 2$ se si escludono $\hat{f}_{\mathbf{x}^*}(M)$ e $\hat{f}_{\mathbf{x}^*}(\emptyset)$, in quanto questi sono rispettivamente la previsione da interpretare e la media delle previsioni fatte dal modello sulle osservazioni del *training set* [35].

gruppi non sono composti da un singolo elemento. Tuttavia, se l'obbiettivo è quello di comprendere il contributo delle singole *feature* nella previsione, questi approcci non possono adeguatamente sostituire lo SHAP [27]. Per questo, una soluzione alternativa a questa criticità consiste nell'utilizzo di metodi che hanno come scopo quello di approssimare i valori ϕ utilizzando solo una parte appositamente selezionata delle coalizioni $S \subseteq M$ [38]. L'analisi di alcuni di questi metodi verrà presentata nel Capitolo 8.

8 I metodi di approssimazione dello SHAP

In questo capitolo vengono presentanti due possibili approcci per approssimare i coefficienti ϕ ottenuti con il metodo SHAP. Se si definisce con $\mathcal{P}(M)$ il *power set* di M, cioè l'insieme di tutte le possibili coalizioni di *feature* $S \subseteq M$, questi metodi considerano nel calcolo dei coefficienti ϕ solamente un *subset* $\mathcal{M}$ delle coalizioni contenute in $\mathcal{P}(M)$, rendendo così tale calcolo meno dispendioso rispetto a quello necessario nella formulazione dello SHAP esatto [38]. Gli approcci in questione sono noti come *Kernel* SHAP e *k-additive* SHAP.

8.1 Il *Kernel* SHAP

Il *Kernel* SHAP è un metodo di approssimazione dello SHAP introdotto nel 2017 da Lundberg e Lee in [32]. Questo metodo trova origine dalla costatazione che gli SHAP ϕ possono essere ricavati attraverso una particolare configurazione del problema di ottimizzazione utilizzato dal metodo di interpretabilità noto come *Local Interpretable Model-agnostic Explanations* (LIME) [32]. In questa sezione per prima cosa verrà descritto il funzionamento del LIME e del problema di ottimizzazione con cui esso ha a che fare, dopodiché si mostrerà in che modo sia possibile derivare lo SHAP in questo differente contesto ed infine si spiegherà come questa differente derivazione dello SHAP permetta una maggior efficienza computazionale.

8.1.1 *Local Interpretable Model-agnostic Explanations*

Con *Local Interpretable Model-agnostic Explanation* (LIME) si fa riferimento ad un metodo di interpretabilità locale di tipo *model-agnostic*. L'idea retrostante a questo metodo è quella di approssimare una funzione complessa f (rappresentante il *black-box model* da interpretare) intorno ad una specifica istanza di interesse $\mathbf{x}^*$ attraverso l'utilizzo di una funzione g sufficientemente semplice da poter essere interpretata (l'*explanation model*) [38]. Il *black-box model* potrebbe infatti essere troppo complesso per essere interpretato globalmente. Il LIME ha quindi l'obbiettivo di fornire una spiegazione locale di tale modello, cioè valida solo nell'intorno di una specifica istanza (si rimanda alla Sezione 4.2) [41].

Data una classe G di modelli potenzialmente interpretabili e un modello g appartenente a tale classe (l'*explanation model*), si può indicare con $\Omega(g)$ una misura della complessità di questo modello, tale che $\Omega(g)$ è tanto più piccola quanto più g è facilmente interpretabile. Si indica invece con $\mathcal{L}(f, g, \pi_{\mathbf{x}^*})$ una *loss function*, cioè una misura di quanto fedelmente g approssima il modello oggetto di interpretazione f nell'introno della specifica istanza $\mathbf{x}^*$. Questa *loss function* è tanto più piccola quanto più l'approssimazione è buona e la quantità $\pi_{\mathbf{x}^*}(x)$ rappresenta la prossimità tra $\mathbf{x}^*$ e la generica istanza $\mathbf{x}$ [41]. Poiché l'obbiettivo del LIME è quello di individuare un modello g che sia facilmente interpretabile e localmente fedele al *black-box model* f, la ricerca di g può essere formalizzata attraverso il seguente problema di ottimizzazione [41]:

$$\min_{g \in G} \mathcal{L}(f, g, \pi_{\mathbf{x}^*}) + \Omega(g). \tag{53}$$

Per poter procedere a tale minimizzazione risulta necessario individuare un campione di istanze su cui valutare la *loss function* $\mathcal{L}(f, g, \pi_{\mathbf{x}^*})$. Questo campione viene generato intorno all'*input* semplificato $\mathbf{x}'^*$

(si veda la Sezione 4.2): dato il *set* M di elementi non nulli nel vettore $\mathbf{x}'^*$ (corrispondenti alle *feature* osservate nell'istanza $\mathbf{x}^*$), si generano varie istanze $\mathbf{z}' \in \{0,1\}^m$, cioè dei vettori binari contenenti una frazione degli elementi non nulli di $\mathbf{x}'^*$ [41]. In pratica è come se si estraesse un sottoinsieme $S \subseteq M$ e si assegnasse a z'_j valore 1 se $j \in S$ e valore 0 viceversa. Dato l'insieme degli *input* semplificati $\mathbf{z}'$ così generati, utilizzando la *mapping function* $h_{\mathbf{x}}$ (che sarà differente da contesto a contesto) si ricavano i corrispondenti *input* non semplificati $\mathbf{z}$ [32].[10] A ogni *input* $\mathbf{z}$ viene infine assegnato un peso $\pi_{\mathbf{x}^*}$ che sarà tanto più grande tanto più $\mathbf{z}$ è nelle prossimità dell'istanza di interesse $\mathbf{x}^*$, assicurando così la località del modello stimato [38].

Il problema di ottimizzazione presentato nell'Equazione 53 si configura in modo differente in base alla classe G di modelli, al tipo di *loss funtion* $\mathcal{L}(f, g, \pi_{\mathbf{x}^*})$ e alla scelta delle misure di prossimità $\pi_{\mathbf{x}^*}$ e di complessità $\Omega(g)$ [41]. In questa sede l'interesse viene rivolto al cosiddetto *linear* LIME, dove G è una classe di modelli lineari e l'*explanation model* g è definito come $g(\mathbf{z}') = \beta_0 + \beta^T \mathbf{z}'$ [38]. Questa scelta permette infatti al *linear* LIME di rientrare nella categoria degli *additive feature attribution method* (presentati nella Sezione 4.3) [32]. Risulta inoltre conveniente utilizzare una *weighted square loss function* del tipo [41]:

$$\mathcal{L}(f, g, \pi_{\mathbf{x}^*}) = \sum_{\mathbf{z},\mathbf{z}' \in \mathcal{Z}} \pi_{\mathbf{x}^*}(\mathbf{z})[f(\mathbf{z}) - g(\mathbf{z}')]^2. \tag{54}$$

Con questa scelta il problema di ottimizzazione si configura infatti come un *weighted least square problem* rappresentabile nel seguente modo:

$$\min_{\beta_0, \beta^T} \sum_{\mathbf{z},\mathbf{z}' \in \mathcal{Z}} \pi_{\mathbf{x}^*}(\mathbf{z})[f(\mathbf{z}) - (\beta_0 + \beta^T \mathbf{z}')]^2 + \Omega(g) \tag{55}$$

e la cui soluzione, che consiste in una stima dai coefficienti β, β^T, è ricavabile analiticamente (come si mostrerà nella Sezione 8.1.2) [38]. Da notare inoltre che, non essendo necessario specificare una particolare forma funzionale f, questo metodo in interpretabilità è di tipo *model-agnostic*, cioè utilizzabile per interpretare le previsioni di qualsiasi *black-box model* [41].

8.1.2 Lo SHAP come una particolare configurazione del *linear* LIME

Il *linear* LIME appena presentato rientra nella categoria degli AFAM. Di conseguenza, essendo lo SV (e le sue derivazioni) l'unico tra i metodi appartenenti a questa categoria a rispettare simultaneamente le proprietà di *local accuracy* e di *consistency* (richiamate nella Sezione 7.1), i risultati prodotti dal *linear* LIME violano una di queste due proprietà (o addirittura entrambe), risultando perciò meno appetibili per un eventuale fruitore. A conferma di questo, è stata provata empiricamente l'esistenza di una concordanza molto più forte tra il ragionamento umano e lo SHAP rispetto a quella con altri AFAM [32]. Risulta tuttavia naturale chiedersi se esista una particolare configurazione del *linear* LIME che possa rispettare entrambe le proprietà sopra menzionate. Tale configurazione permetterebbe infatti di ricavare gli SHAP ϕ con un approccio differente rispetto a quello già presentato, cioè come la soluzione di un problema di ottimizzazione [32].

[10] L'insieme delle coppie di *input* $(\mathbf{z}, \mathbf{z}')$ viene indicato con $\mathcal{Z}$ [41].

Se ci si pone all'interno dello scenario descritto nelle Sezioni 7.1 e 7.2, l'*input* semplificato $\mathbf{x}'^*$ relativo all'istanza di interesse $\mathbf{x}^*$ può essere indicato con $\mathbf{1}_M$, cioè con un vettore in cui tutti gli m elementi sono pari ad 1, indicante che per l'istanza di interesse tutte le *feature* contenute in M risultano osservate. L'*input* semplificato $\mathbf{z}'$, essendo generato estraendo una coalizione S di *feature* dall'insieme M, può essere invece indicato con $\mathbf{1}_S \in \{0,1\}^m$, cioè un vettore binario pari ad 1 in corrispondenza degli elementi $j \in S$ e pari a 0 viceversa. Di conseguenza l'*explanation model* può essere scritto come $g(\mathbf{1}_S) = \beta_0 + \sum_{j \in S} \beta_j$. Infine, l'insieme degli *input* semplificati $\mathbf{z}'$, essendo un inseme di coalizioni $S \subseteq M$, può essere indicato con $\mathcal{M}$, cioè come un sottoinsieme del *power set* $\mathcal{P}(M)$ [38]. In questo contesto, si consideri la seguente configurazione del *linear* LIME:

$$\Omega(g) = 0, \tag{56}$$

$$\pi_{\mathbf{x}^*} = \pi(S) = \frac{m-1}{\binom{m}{s} s(m-s)}, \tag{57}$$

$$f(\mathbf{z}) = \hat{f}_{\mathbf{x}^*}(S), \tag{58}$$

con $s = |S|$. Il problema di ottimizzazione dell'Equazione 55 può essere allora riscritto come [38]:

$$\min_{\beta_0, \beta_1, \ldots, \beta_m} \sum_{S \in \mathcal{M}} \frac{m-1}{\binom{m}{s} s(m-s)} \left[\hat{f}_{\mathbf{x}^*}(S) - \left(\beta_0 + \sum_{j \in S} \beta_j \right) \right]^2. \tag{59}$$

Risulta interessante notare che i pesi $\pi(S)$, noti come *kernel weight*, tendono ad infinito per $S = M$. Si può infatti osservare che per $S = M$ si ha che $\hat{f}_{\mathbf{x}^*}(M) = f(\mathbf{x}^*)$ e che $g(\mathbf{1}_M) = \beta_0 + \sum_{j \in M} \beta_j$. Ricordando che la proprietà di *local accuracy* richiede $f(\mathbf{x}^*) = \beta_0 + \sum_{j \in M} \beta_j$, si può concludere che i *kernel weight* sono costruiti in modo tale da vincolare il rispetto di tale proprietà, in quanto, essendo il peso assegnato a M estremamente elevato, il processo di ottimizzazione si concentrerà maggiormente nel minimizzare la quantità $f(\mathbf{x}^*) - (\beta_0 + \sum_{j \in M} \beta_j)$, rispetto alle altre presenti nella sommatoria. Un ragionamento analogo vale per $S = \emptyset$. Infatti, anche in corrispondenza di questa coalizione si osservano pesi infiniti. Inoltre si ha che $\hat{f}_{\mathbf{x}^*}(\emptyset) = \mathbb{E}[f(\mathbf{x})]$ e che $g(\mathbf{1}_\emptyset) = \beta_0$. In questo caso i *kernel weight* servono quindi a vincolare $\beta_0 = \mathbb{E}[f(\mathbf{x})]$, che, nella Sezione 7.2, si è visto essere una caratteristica dello SHAP [38].

Quello presentato nell'Equazione 59 è un *weighted least squares problem* e, in quanto tale, presenta una soluzione chiusa ricavabile analiticamente [1]. Per mostrare tale soluzione risulta tuttavia conveniente l'introduzione di una notazione di tipo matriciale. Indicando con $n_{\mathcal{M}}$ il numero di elementi di $\mathcal{M}$ (cioè il numero di coalizioni S utilizzate nel problema di ottimizzazione) e con S_l la coalizione l-esima contenuta in $\mathcal{M}$ (con $l = 1, ..., n_{\mathcal{M}}$), si ha che:

- $\beta = [\beta_0, \beta_1, ..., \beta_m]^T$ indica il vettore di dimensione $(m+1) \times 1$ composto dai coefficienti dell'*explanation model* lineare (uno per ognuna delle m *feature*, più l'intercetta β_0) [38]. Questo rappresenta l'incognita del problema di ottimizzazione. Risolvere tale problema significa quindi trovare il vettore β tale per cui la *loss function* dell'Equazione 59 risulta minimizzata.

- $\mathbf{Z}$ indica la matrice binaria di dimensione $n_{\mathcal{M}} \times (m+1)$ contenente gli *input* semplificati $\mathbf{1}_S$. Ogni riga è quindi associata a una coalizione $S \in \mathcal{M}$. La prima colonna è composta solamente di 1,

mentre ogni elemento $j+1$ della riga l-esima è pari ad 1 se $j \in S_l$ (cioè se la j-esima *feature* è contenuta nella coalizione l-esima) e 0 viceversa [1]. Tale matrice può essere quindi rappresentata come:

$$\mathbf{Z} = \begin{bmatrix} 1 & \mathbf{1}_{S_1,1} & \mathbf{1}_{S_1,2} & \dots & \mathbf{1}_{S_1,m} \\ 1 & \mathbf{1}_{S_2,1} & \mathbf{1}_{S_2,2} & \dots & \mathbf{1}_{S_2,m} \\ \vdots & \vdots & \vdots & \ddots & \\ 1 & \mathbf{1}_{S_{n_\mathcal{M}},1} & \mathbf{1}_{S_{n_\mathcal{M}},2} & \dots & \mathbf{1}_{S_{n_\mathcal{M}},m} \end{bmatrix}, \tag{60}$$

dove $\mathbf{1}_{S_l,j}$ rappresenta l'elemento j-esimo dell'*input* semplificato relativo alla l-esima coalizione S tenuta in considerazione nel problema di ottimizzazione.

- $\mathbf{f} = [\hat{f}_{\mathbf{x}^*}(S_1), \hat{f}_{\mathbf{x}^*}(S_2), ..., \hat{f}_{\mathbf{x}^*}(S_{n_\mathcal{M}})]^T$ indica il vettore di dimensione $n_\mathcal{M} \times 1$ contenente le quantità $\hat{f}_{\mathbf{x}^*}(S)$ per ogni coalizione $S \in \mathcal{M}$ [38]. Queste quantità vengono calcolate sotto l'assunzione di indipendenza con le stesse modalità descritte nella Sezione 7.2.

- $\mathbf{W}$ indica la matrice diagonale di dimensione $n_\mathcal{M} \times n_\mathcal{M}$ contente i pesi $\pi(S)$ associati alle coalizioni $S \in \mathcal{M}$ [1]. Questa viene quindi rappresentata come:

$$\mathbf{W} = \begin{bmatrix} \pi(S_1) & 0 & \dots & 0 \\ 0 & \pi(S_2) & \dots & 0 \\ \vdots & \vdots & \ddots & \\ 0 & 0 & \dots & \pi(S_{n_\mathcal{M}}) \end{bmatrix}. \tag{61}$$

Attraverso l'utilizzo di questa notazione il problema di ottimizzazione presentato nell'Equazione 59 può essere riscritto come:

$$\min_{\beta} (\mathbf{f} - \mathbf{Z}\beta)^T \mathbf{W} (\mathbf{f} - \mathbf{Z}\beta) \tag{62}$$

e la sua soluzione in forma chiusa è data da [1]:

$$\beta = (\mathbf{Z}^T \mathbf{W} \mathbf{Z})^{-1} \mathbf{Z}^T \mathbf{W} \mathbf{f}. \tag{63}$$

Lundberg e Lee (2017) in [32] hanno dimostrato che la soluzione β appena presentata coincide esattamente con il vettore di SHAP ϕ se $\mathcal{M} = \mathcal{P}(M)$, cioè se nel processo di ottimizzazione vengono utilizzate tutte le possibili coalizioni $S \subseteq M$ [38]. Si può inoltre notare che la matrice $\mathbf{R} = (\mathbf{Z}^T \mathbf{W} \mathbf{Z})^{-1} \mathbf{Z}^T \mathbf{W}$ non dipende in alcun modo dalla specifica istanza di interesse $\mathbf{x}^*$. Di conseguenza può essere utilizzata per la valutazione degli SHAP relativamente a qualsiasi istanza, senza bisogno di doverla ricalcolare ogni volta. Ciò che cambierà da istanza a istanza è invece il vettore di valutazioni $\mathbf{f}$ [1].

8.1.3 Il vantaggio computazionale del *Kernel* SHAP

La configurazione del *linear* LIME presentata nella precedente sezione rappresenta una strada alternativa per la derivazione dello SHAP [32]. Tuttavia, si può notare che questa nuova derivazione non presenta

vantaggi computazionali rispetto a quella classica vista nella Sezione 7.2. Per ottenere lo SHAP esatto è infatti necessario utilizzare nel processo di ottimizzazione tutte le coalizioni $S \subseteq M$ e, di conseguenza, valutare la quantità $\hat{f}_{\mathbf{x}^*}(S)$ per ognuna di queste 2^m coalizioni, esattamente come nella formulazione classica dello SHAP (si veda la Sezione 7.3) [38]. Va però osservato che i *kernel weight* assegnati alle coalizioni estratte tendono ad avere dimensioni molto differenti. Nello specifico, quando la cardinalità di S è vicina a 0 o a m, i pesi assumono valori molto più elevati rispetto a quelli assegnati alle coalizioni con cardinalità intermedia. Di conseguenza, molte delle coalizioni estratte potrebbero avere un contributo estremamente piccolo nel determinare la soluzione del problema di ottimizzazione, cioè nel determinare i valori degli SHAP ϕ [1]. Si può quindi pensare di trascurare questi contributi ed ottenere una buona approssimazione dello SHAP utilizzando solamente le coalizioni con un peso maggiore. Il *Kernel* SHAP trova origine proprio a partire da questa intuizione. Questo metodo si sostanzia infatti nel risolvere il problema di ottimizzazione precedentemente presentato utilizzando un insieme $\mathcal{M}$ di coalizioni estratte in base ad una distribuzione di probabilità del tipo:

$$p_S = \frac{\pi(S)}{\sum_{S \subseteq M} \pi(S)}, \tag{64}$$

cioè tale per cui ogni coalizione ha una probabilità di essere estratta pari al corrispondente *kernel weight* normalizzato. Le coalizioni che hanno un peso maggiore nell'ottimizzazione avranno quindi una maggior probabilità di essere estratte. Di conseguenza, pur utilizzando un numero $n_{\mathcal{M}}$ di coalizioni molto inferiore rispetto a 2^m, si avrà una buona stima degli SHAP ϕ, con il vantaggio computazionale di dover valutare solamente $n_{\mathcal{M}}$ quantità $\hat{f}_{\mathbf{x}^*}(S)$ [38]. Da notare che, per garantire che ogni coalizione possa essere estratta solamente una volta, dopo ogni estrazione le probabilità p_S vanno aggiornate rimuovendo il peso corrispondente alla coalizione estratta e normalizzando i pesi rimasti. Inoltre, essendo impossibile lavorare con pesi infiniti, nella pratica si fissano $\pi(M)$ e $\pi(\emptyset)$ pari ad un numero molto grande (come ad esempio 10^6) [38].

8.2 Il *k-additive* SHAP

Il k-*additive* SHAP (k_{ADD}-SHAP) è un metodo di approssimazione dello SHAP introdotto da Pelegrina, Duarte e Grabisch nel 2023 in [38]. Questo metodo si basa sull'utilizzo di due concetti molto diffusi nell'ambito della teoria dei giochi (l'integrale di *Choquet* e i giochi k-additivi) e viene introdotto con lo scopo di far fronte ad alcune criticità che vengono riscontrate nella formulazione del *Kernel* SHAP precedentemente presentata [38]. In questa sezione[11] viene quindi mostrato in che modo sia possibile derivare gli SHAP ϕ a partire dall'utilizzo dell'integrale di *Choquet* e come l'introduzione dei giochi k-additivi permetta una maggiore efficienza computazionale nell'approssimazione di tali valori.

[11] Questa sezione è stata scritta interamente rielaborando il contenuto dell'articolo "A k-additive Choquet integral-based approach to approximate the SHAP values for local interpretability in machine learning" di Pelegrina et al. (2023) [38]. Si eviterà quindi di citare continuamente tale articolo sottintendendo l'origine delle informazioni riportate.

8.2.1 La derivazione dello SHAP tramite l'integrale di *Choquet*

Fino a questo momento la derivazione dello SV (e quindi dello SHAP) è stata ricercata nell'ambito degli AFAM, assumendo quindi un *explanation model* g di tipo lineare (o additivo) come quello dell'Equazione 1. Si può allora pensare di uscire da questa assunzione e valutare la possibilità di derivare lo SV a partire da un modello non-additivo. Si consideri quindi un *explanation model* del tipo:

$$g(\mathbf{1}_S) = \phi_0 + f_{CI}(\mathbf{1}_S), \tag{65}$$

dove ϕ_0 è l'intercetta e $f_{CI}(\mathbf{1}_S)$ è l'integrale di *Choquet* valutato sull'*input* semplificato $\mathbf{1}_S$, che si è già detto essere una funzione di aggregazione non-additiva (si veda la Sezione 5.5). Riprendendo l'Equazione 31, che descrive tale integrale nel caso discreto (che è il caso di interesse essendo M un insieme finito), e ricordando che tale formulazione è valida anche per un gioco v (come puntualizzato nella Sezione 5.5.2), si può definire l'integrale di *Choquet* valutato in una generica istanza **x** come:

$$f_{CI}(\mathbf{x}) = \sum_{j=1}^{m} [x_{(j)} - x_{(j-1)}] v(\{(j), ..., (m)\}), \tag{66}$$

dove l'indicizzazione (j) fa riferimento a una permutazione degli m elementi del vettore **x** tale che $x_{(1)} \leq x_{(2)} \leq ... \leq x_{(m)}$. Essendo l'*input* semplificato $\mathbf{1}_S$ un vettore binario, questa permutazione dei suoi elementi sarà tale per cui $\mathbf{1}_{S,(j)} = 0$ per $(j) = (1), ..., (m-s)$, mentre si avrà $\mathbf{1}_{S,(j)} = 1$ per $(j) = (m-s+1), ..., (m)$, con s il numero di elementi nella coalizione S di *feature*. Risulta quindi evidente che tra i termini della sommatoria nell'Equazione 66 solo uno sarà diverso da 0, cioè quello corrispondente alla differenza $[\mathbf{1}_{S,(m-s+1)} - \mathbf{1}_{S,(m-s)}]$. Di conseguenza, l'integrale di *Choquet* valutato nell'*input* semplificato $\mathbf{1}_S$ è dato da:

$$f_{CI}(\mathbf{1}_S) = [\mathbf{1}_{S,(m-s+1)} - \mathbf{1}_{S,(m-s)}] v(\{(m-s+1), ..., (m)\}) = v(S). \tag{67}$$

A questo punto si può ricordare che una generica *set function* ξ può essere ricostruita a partire dalle *interaction transform* I^{ξ} (Equazione 18). Si è inoltre già notato che queste *interaction transform* coincidono con le *Shapley Interaction* introdotte nella Sezione 6.4. Ne consegue che il gioco $v(S)$ può essere riscritto direttamente come funzione di tali interazioni:

$$v(S) = \sum_{D \subseteq M} \gamma_{|S \cap D|}^{|D|} I(D), \tag{68}$$

con i coefficienti γ ricavabili a partire dall'Equazione 19 e con $I(D)$ indicante la SI valutata sull'insieme $D \subseteq M$. Essendo $f_{CI}(\mathbf{1}_S) = v(S)$, si può allora concludere che l'integrale di *Choquet* può essere scritto direttamente in funzione delle SI e degli SV (che rappresentano il caso particolare di SI per $D = \{j\}$).

Dato l'*explanation model* derivante dalle osservazioni appena fatte, cioè tale che $g(\mathbf{1}_S) = \phi_0 + v(S)$, si può pensare di inserirlo all'interno del seguente problema di ottimizzazione:

$$\min_{I} \sum_{S \in \mathcal{M}} \pi'(S) \left[\bar{f}_{\mathbf{x}^*}(S) - \sum_{D \subseteq M} \gamma_{|S \cap D|}^{|D|} I(D) \right]^2, \tag{69}$$

dove $\bar{f}_{\mathbf{x}^*}(S) = \hat{f}_{\mathbf{x}^*}(S) - \phi_0$. Si può notare che questo assomiglia alla particolare configurazione di *linear* LIME a cui fa riferimento il *Kernel* SHAP. Oltre all'utilizzo di un *explanation model* non-additivo, si possono riscontrare altre due differenze rispetto a quella configurazione:

- i pesi $\pi'(S)$ sono fissati pari ad 1 per tutte le coalizioni S ad eccezione di M e $\emptyset$ a cui si associa, analogamente al *Kernel* SHAP, un peso molto elevato (come 10^6). Infatti, per $S = \emptyset$ si ha che $\hat{f}_{\mathbf{x}^*}(\emptyset) = \mathbb{E}[f(\mathbf{x})]$ e che $v(\emptyset) = 0$, essendo il gioco una *set function grounded*. Assegnare un grande peso a tale coalizione significa quindi dare maggiore attenzione alla minimizzazione della differenza $\hat{f}_{\mathbf{x}^*}(\emptyset) - \phi_0$ ed assicurare così che $\hat{f}_{\mathbf{x}^*}(\emptyset) = \phi_0$, come richiesto dalla formulazione dello SHAP. Analogamente, per $S = M$ si ha $\hat{f}_{\mathbf{x}^*}(M) = f(\mathbf{x}^*)$ e $v(M) = v(\emptyset) + \sum_j I(\{j\})$, per quanto visto nella Sezione 5.3. Notando che $I(\{j\}) = \phi_j$, assegnare un peso elevato a tale coalizione significa dare attenzione alla minimizzazione della differenza $f(\mathbf{x}^*) - \phi_0 - \sum_j \phi_j$, cioè assicurare il rispetto della *local accuracy*.

- l'incognita del problema di ottimizzazione è il vettore $\mathbf{I} = [\phi_0, \phi_1, ..., \phi_m, I(\{1,2\}), ..., I(M)]^T$ di dimensione $2^m \times 1$ contenente un termine di interazione per ogni coalizione $D \subseteq M$.

Trattandosi anche in questo caso di un *weighted least squares problem* è possibile introdurre una notazione matriciale analoga a quella vista per il *Kernel* SHAP:

- $\bar{\mathbf{f}} = [\bar{f}_{\mathbf{x}^*}(S_1), \bar{f}_{\mathbf{x}^*}(S_2), ..., \bar{f}_{\mathbf{x}^*}(S_{n_\mathcal{M}})]^T$ indica il vettore di dimensione $n_\mathcal{M} \times 1$ contenente le quantità $\bar{f}_{\mathbf{x}^*}(S)$ per ogni coalizione $S \in \mathcal{M}$. Le quantità $\hat{f}_{\mathbf{x}^*}(S)$ sono sempre calcolate sotto l'assunzione di indipendenza.

- $\mathbf{T}_\mathcal{M}$ indica la matrice di dimensione $n_\mathcal{M} \times 2^m$, detta *transformation matrix*, contenente i coefficienti $\gamma^{|D|}_{|S \cap D|}$. Ogni riga è associata a una coalizione $S \in \mathcal{M}$ mentre ogni colonna è associata ad una coalizione $D \subseteq M$. Da notare che, se il numero di righe varia a seconda della dimensione di $\mathcal{M}$, il numero di colonne è invece fisso, in quanto per ricostruire il valore $v(S)$ è necessario utilizzare tutte le 2^m coalizioni D contenute in M. Questa matrice può essere quindi rappresentata come:

$$\mathbf{T}_\mathcal{M} = \begin{bmatrix} \gamma^{|\emptyset|}_{|S_1 \cap \emptyset|} & \gamma^{|\{1\}|}_{|S_1 \cap \{1\}|} & \cdots & \gamma^{|\{m\}|}_{|S_1 \cap \{m\}|} & \gamma^{|\{1,2\}|}_{|S_1 \cap \{1,2\}|} & \cdots & \gamma^{|M|}_{|S_1 \cap M|} \\ \gamma^{|\emptyset|}_{|S_2 \cap \emptyset|} & \gamma^{|\{1\}|}_{|S_2 \cap \{1\}|} & \cdots & \gamma^{|\{m\}|}_{|S_2 \cap \{m\}|} & \gamma^{|\{1,2\}|}_{|S_2 \cap \{1,2\}|} & \cdots & \gamma^{|M|}_{|S_2 \cap M|} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \gamma^{|\emptyset|}_{|S_{n_\mathcal{M}} \cap \emptyset|} & \gamma^{|\{1\}|}_{|S_{n_\mathcal{M}} \cap \{1\}|} & \cdots & \gamma^{|\{m\}|}_{|S_{n_\mathcal{M}} \cap \{m\}|} & \gamma^{|\{1,2\}|}_{|S_{n_\mathcal{M}} \cap \{1,2\}|} & \cdots & \gamma^{|M|}_{|S_{n_\mathcal{M}} \cap M|} \end{bmatrix}. \quad (70)$$

  Si può inoltre notare che, essendo $\gamma^0_0 = \mathbf{B}_0 = 1$ (si veda la Sezione 5.3), allora la prima colonna di questa matrice è composta di soli 1.

- $\bar{\mathbf{W}}$ indica la matrice diagonale di dimensione $n_{\mathcal{M}} \times n_{\mathcal{M}}$ contente i pesi $\pi'(S)$ associati alle coalizioni $S \in \mathcal{M}$. Questa viene quindi rappresentata come:

$$\bar{\mathbf{W}} = \begin{bmatrix} \pi'(S_1) & 0 & \dots & 0 \\ 0 & \pi'(S_2) & \dots & 0 \\ \vdots & \vdots & \ddots & \\ 0 & 0 & \dots & \pi'(S_{n_{\mathcal{M}}}) \end{bmatrix}. \tag{71}$$

Attraverso l'utilizzo di questa notazione il problema di ottimizzazione presentato nell'Equazione 69 può essere riscritto come:

$$\min_{\mathbf{I}} (\bar{\mathbf{f}} - \mathbf{T}_{\mathcal{M}}\mathbf{I})^T \bar{\mathbf{W}} (\bar{\mathbf{f}} - \mathbf{T}_{\mathcal{M}}\mathbf{I}) \tag{72}$$

e la sua soluzione in forma chiusa è data da:

$$\mathbf{I} = (\mathbf{T}_{\mathcal{M}}^T \bar{\mathbf{W}} \mathbf{T}_{\mathcal{M}})^{-1} \mathbf{T}_{\mathcal{M}}^T \bar{\mathbf{W}} \bar{\mathbf{f}}. \tag{73}$$

Pelegrina, Duarte e Grabisch (2023) hanno dimostrato che se $\mathcal{M} = \mathcal{P}(M)$ allora la soluzione di questo problema restituisce esattamente le SHAP *interaction*, compresi gli SHAP ϕ.

Da quanto detto risulta evidente che l'idea sottostante a questa formulazione è analoga a quella del *Kernel* SHAP. Si è infatti mostrata una nuova possibile strada per derivare lo SHAP, sempre basata su problema di ottimizzazione ma utilizzando un *explanation model* non-additivo. Data la struttura dei pesi, anche in questo caso si può pensare di estrarre (nelle stesse modalità descritte nella Sezione 8.1.3) ed utilizzare nel problema di ottimizzazione solo un numero $n_{\mathcal{M}}$ di coalizioni. Così facendo si otterrà una buona approssimazione dello SHAP esatto ma con un dispendio computazionale di molto inferiore. Va però sottolineato che, in questo caso, la soluzione del problema di ottimizzazione non si compone solamente degli SHAP ϕ associati alle m *feature* ma anche delle SHAP *interaction* associate a ogni possibile coalizione di queste variabili. Il contenuto informativo di questa soluzione è quindi maggiore rispetto a quello fornito dal *Kernel* SHAP. Queste interazioni danno infatti una misura di come due o più *feature* interagiscono nel determinare la previsione di interesse, ossia in che modo la loro presenza simultanea nel modello influisce sui contributi da esse apportati in tale previsione. Questo tipo di informazione può essere quindi preziosa in un contesto di XAI. Risulta tuttavia evidente che questo maggior numero di incognite all'interno del problema di ottimizzazione rende l'individuazione della soluzione più complessa e di conseguenza, a parità di numero $n_{\mathcal{M}}$ di coalizioni considerate, il *Kernel* SHAP risulterà un approccio più efficiente per l'approssimazione degli SHAP.

8.2.2 L'introduzione della *k*-additività

Nella Sezione 5.4 si è introdotto il concetto di gioco k-additivo. Se con k si indica il grado di additività, un gioco k-additivo è tale per cui $I(D) = 0$ per ogni coalizione $D \subseteq M$ tale che $|D| > k$. Di

conseguenza, l'Equazione 68 può essere riscritta come:

$$v(S) = \sum_{D \in \mathcal{D}_k} \gamma_{|S \cap D|}^{|D|} I(D), \tag{74}$$

dove $\mathcal{D}_k$ rappresenta l'insieme delle coalizioni $D \subseteq M$ tali che $|D| \leq k$. Si può pensare allora di inserire questo risultato all'interno del problema di ottimizzazione presentato nell'Equazione 69, utilizzando quindi un *explanation model* che presenta qualche grado di additività. La nuova configurazione del *weighted least squares problem* così ottenuto presenta due differenze sostanziali rispetto a quella senza alcun grado di additività:

- il vettore di incognite $\mathbf{I}$ di dimensione $2^m \times 1$ viene sostituito dal vettore $\mathbf{I}_k$ di dimensione $d_k \times 1$ (con $d_k = |\mathcal{D}_k|$) contenete le sole interazioni relative alle coalizioni $D \in \mathcal{D}_k$.
- la matrice $\mathbf{T}_{\mathcal{M}}$ di dimensione $n_{\mathcal{M}} \times 2^m$ viene sostituita dalla matrice $\mathbf{T}_{\mathcal{M},k}$ di dimensione $n_{\mathcal{M}} \times d_k$ composta dalle sole colonne relative alle coalizioni $D \in \mathcal{D}_k$.

In particolare, il numero d_k di coalizioni tali che $|D| \leq k$ si ottiene come [21]:

$$d_k = \sum_{a=0}^{k} \binom{k}{a}, \tag{75}$$

essendo $\binom{k}{a}$ il numero di combinazioni non ordinate di a elementi ottenibili a partire da un *set* di k elementi. Ovviamente, per $k = m$ (cioè se non c'è alcun grado di additività) questa quantità è pari a 2^m (Equazione 6). Il valore d_k rappresenta quindi una misura della parametrizzazione del problema di ottimizzazione. Attraverso la notazione appena introdotta è possibile vedere questo problema di ottimizzazione come:

$$\min_{\mathbf{I}_k} (\bar{\mathbf{f}} - \mathbf{T}_{\mathcal{M},k}\mathbf{I}_k)^T \bar{\mathbf{W}} (\bar{\mathbf{f}} - \mathbf{T}_{\mathcal{M},k}\mathbf{I}_k) \tag{76}$$

e la sua soluzione in forma chiusa è data da:

$$\mathbf{I}_k = (\mathbf{T}_{\mathcal{M},k}^T \bar{\mathbf{W}} \mathbf{T}_{\mathcal{M},k})^{-1} \mathbf{T}_{\mathcal{M},k}^T \bar{\mathbf{W}} \bar{\mathbf{f}}. \tag{77}$$

Notando che la matrice $\mathbf{R}_{\mathcal{M},k} = (\mathbf{T}_{\mathcal{M},k}^T \bar{\mathbf{W}} \mathbf{T}_{\mathcal{M},k})^{-1} \mathbf{T}_{\mathcal{M},k}^T \bar{\mathbf{W}}$ non dipende dalla specifica istanza $\mathbf{x}^*$, questa può essere utilizzata per il calcolo della soluzione $\mathbf{I}_k$ relativamente a qualsiasi istanza di interesse.

Quella che si è appena mostrata è la configurazione del cosiddetto k_{ADD}-SHAP. Questo approccio di approssimazione dello SHAP si basa quindi sull'introduzione di alcuni gradi di additività all'interno del problema di ottimizzazione presentato nella Sezione 8.2.1. La conseguenza è che il *weighted least squares problem* da risolvere presenta una minore parametrizzazione, comportando due vantaggi computazionali. In primo luogo, poiché il problema presenta un numero inferiore di incognite, a parità di numero $n_{\mathcal{M}}$ di coalizioni considerate sarà più rapido il calcolo della soluzione. Inoltre, nonostante si stimi un numero inferiore di interazioni, non vi è una sostanziale perdita di informazione. Infatti le interazioni di orine superiore al secondo sono molto difficili da interpretare, risultando quindi la loro conoscenza scarsamente informativa. In secondo luogo, la minore parametrizzazione del problema favorisce il raggiungimento di una buona approssimazione dello SHAP esatto utilizzando un numero $n_{\mathcal{M}}$

inferiore di coalizioni. Infatti, un problema a maggior parametrizzazione (e quindi con una maggiore flessibilità) richiede solitamente un maggior numero di osservazioni (in questo caso di *input* semplificati, cioè di coalizioni) per produrre risultati di buon livello. Va però detto che questa minore flessibilità impedisce al k_{ADD}-SHAP di arrivare a convergenza con lo SHAP esatto, anche qualora si abbia $\mathcal{M} = \mathcal{P}(M)$. Gli autori che hanno introdotto questo approccio sottolineano inoltre che questo secondo vantaggio computazionale vale anche rispetto al *Kernel* SHAP. Viene infatti osservato che, nonostante in quel tipo di configurazione si lavori solamente con un numero m di parametri (non si tiene in considerazione nessun termine di interazione), la parametrizzazione sottostante al problema risulta comunque massima, essendo i giochi su cui implicitamente si fonda non-additivi. Questa maggior parametrizzazione di fondo è evidente nel fatto che, come dimostrato da Lundberg e Lee (si veda la Sezione 8.1.2), il *Kernel* SHAP riesce a convergere esattamente con lo SHAP per $\mathcal{M} = \mathcal{P}(M)$. In definitiva, il k_{ADD}-SHAP dovrebbe riuscire ad approssimare in maniera soddisfacente lo SHAP con un numero inferiore di coalizioni rispetto a quelle necessarie al *Kernel* SHAP, risultando nel complesso l'approccio più efficiente.

8.3 I metodi di approssimazione dello SHAP a gruppi

Nella Sezione 7.2 si è parlato delle estensioni dello SHAP per gruppi di *feature*: il *Group* SHAP (GS) e lo SHAP *generalized value* (SGV). Risulta quindi naturale pensare di introdurre un estensione analoga del *Kernel* e del k_{ADD} SHAP. Si potrebbe pensare che questo tipo di estensione non sia particolarmente utile, dato che il GS e lo SGV di per se già presentano dei vantaggi computazionali rispetto allo SHAP tradizionale (si rimanda alla Sezione 6.3). Va però detto che, in alcune circostanze, può risultare molto dispendiosa anche l'implementazione dei metodi a gruppi. Nel GS è possibile ad esempio che il numero g di gruppi di *feature* sia particolarmente elevato, portando il numero di valutazioni necessarie per il suo calcolo ad essere impraticabile. Si ricorda infatti che il numero di termini nella sommatoria dell'Equazione 41 è pari a 2^{g-1} (il numero di valori attesi da calcolare è 2^g) e cresce quindi in maniera esponenziale all'aumentare del numero di gruppi. Riguardo lo SGV si è invece già mostrato come il suo vantaggio computazionale risulti nettamente inferiore rispetto a quello associato al GS, senza considerare che l'eventuale presenza anche di un solo gruppo composto da una singola *feature* porta a dover effettuare lo stesso numero di valutazioni dello SHAP tradizionale (come visto nella Sezione 6.3). Si può quindi concludere che un estensione per gruppi di *feature* del *Kernel* e del k_{ADD} SHAP può avere una sua utilità pratica.

Un'estensione del *Kernel* SHAP relativamente al GS è stata presentata da Kang Lin e Yuzhuo Gao (2022) in [30]: sfruttando la stretta corrispondenza strutturale esistente tra lo SHAP e il GS (mostrata nella Sezione 6.3), questa estensione consiste semplicemente nell'applicare il *Kernel* SHAP, esattamente come descritto nella Sezione 8.1.2, trattando ogni gruppo di *feature* come una singola variabile [30]. In questa sede si è quindi pensato di introdurre un'estensione analoga anche per il k_{ADD}-SHAP, cioè utilizzando la procedura descritta nella Sezione 8.2.2 e trattando ogni gruppo come una singola variabile.

Riguardo un'estensione dei due metodi di approssimazione presentati relativamente allo SGV non si è invece trovata nessuna proposta in letteratura. In questa sede si è quindi introdotto un possibile approccio per effettuare tale estensione. Nello specifico, si è constatato che una soluzione in forma chiusa, come quella presentata nelle Equazioni 63 e 77, non è ottenibile per lo SGV. Questo metodo

infatti fa uso delle coalizioni ottenute fissando un gruppo e trattando le altre variabili individualmente (come visto nella Sezione 6.3). Di conseguenza, al variare del gruppo di interesse varia la dimensione degli *input* semplificati da utilizzare nel processo di ottimizzazione. Si immagini ad esempio uno scenario con $m = 5$ *feature* e con una partizione di gruppi del tipo $G_1 = \{1, 2\}, G_2 = \{3, 4, 5\}$. In questo caso, fissando G_1 come gruppo di interesse, si costruiscono le coalizioni ottenute combinando gli elementi del gruppo G_2. Queste coalizioni sono tutte rappresentabili attraverso vettori binari contenuti in $\{0, 1\}^3$. Se invece si fissasse il gruppo G_2 si costruirebbero le coalizioni ottenute combinando gli elementi del gruppo G_1, rappresentabili attraverso vettori binari contenuti in $\{0, 1\}^2$. Le due tipologie di coalizioni non sono compatibili e non è quindi possibile costruire una matrice $\mathbf{Z}$ comune come quella necessaria per far uso del *Kernel* SHAP. L'approccio proposto prevede quindi di lavorare con un problema di ottimizzazione differente per ogni gruppo di *feature*. In particolare, fissato un gruppo, si applica il *Kernel* SHAP (o il k_{ADD}-SHAP), esattamente come descritto nella Sezione 8.1.2, trattando quel gruppo come una singola *feature* e le *feature* non appartenenti ad esso individualmente. Questa procedura permette di ricavare un approssimazione dello SGV solamente per il gruppo fissato e dovrà quindi essere ripetuta per ogni gruppo di interesse. Chiaramente, la necessità di risolvere g differenti problemi di ottimizzazione rende il calcolo delle approssimazioni più complesso. Va però detto che l'esistenza di una soluzione in forma chiusa per questi problemi facilita il compito computazionale ed infatti, come si mostrerà nel Capitolo 9 dedicato alla fase sperimentale, il *Kernel* e il k_{ADD} SHAP rendono notevolmente più efficiente il calcolo degli SGV.

9 Analisi sperimentale: un'applicazione del metodo SHAP nel contesto della *financial fraud detection*

In questo capitolo si presenta un'applicazione del metodo SHAP, delle sue estensioni e dei suoi metodi di approssimazione relativamente all'interpretazione delle previsioni prodotte da alcuni modelli di *machine learning* nell'ambito della *financial fraud detection*. L'obbiettivo è quello di mostrare in che modo questo metodo permetta di comprendere le ragioni sottostanti a tali previsioni, aumentando la fiducia che un *decision maker* può riporre in esse, e come i metodi di approssimazione presentati riducano notevolmente la complessità computazionale intrinseca in questo metodo di interpretabilità. Questa applicazione è stata prodotta tramite l'utilizzo del linguaggio di programmazione *Python* (i codici più importanti sono riportati nell'Appendice A).

9.1 Descrizione del *dataset*

Per questa applicazione si è fatto uso del *dataset* costruito ed utilizzato da Kang Lin e Yuzhuo Gao nell'articolo "Model interpretability of financial fraud detection by group SHAP" (2022).[12] Questo si compone dell'osservazione per $n = 286$ società cinesi quotate nello *Shanghai Stock Exchange* o nello *Shenzhen Stock Exchange* (i due principali mercati azionari cinesi) di $m = 10$ variabili, rappresentanti degli indici di bilancio relativamente all'anno fiscale 2020. Queste variabili sono inoltre suddivise in $g = 5$ gruppi in base all'affinità delle informazioni da esse apportate (Tabella 1) [30].

Tabella 1: nomi, codici di riferimento e gruppi di appartenenza delle variabili osservate.

Gruppo	**Codice**	**Variabile**
Operation	Finanomalies142	Standalone to Consolidated Ratio
	Leverage9	AP to Liability Ratio
Profitability	Finanomalies122	Other Payables Ratio
	Finanomalies101	Working Capital Turnover
	Finanomalies100	Cash to Debt Ratio
Solvency	Finanomalies5	AR Ratio
	Finanomalies1	CF to Revenue Ratio
Leverage	Finanomalies32	Tax to Revenue
	Finanomalies87	Profit to CF Ratio
Growth	Finanomalies61	Average Gross Margin Index

[12] Il *dataset* è disponibile al seguente indirizzo web: `https://codeocean.com/capsule/5373825/tree/v1`.

Per le società nel campione è stato osservato se abbiano o meno subito qualche forma di sanzione (più o meno grave) da parte della *China Securities Regulatory Commission* per anomalie individuate nei bilanci da queste pubblicati [30]. In sostanza, si conosce quali società siano state incriminate di *Financial statement fraud* (si veda la Sezione 2.1) e quali no. La variabile risposta è quindi binaria e assume valore 1 se la società è risultata fraudolenta e 0 viceversa. Va infine sottolineato che il *dataset* risulta già perfettamente bilanciato, cioè il numero di società fraudolente e non fraudolente nel campione è lo stesso [30].

9.2 Addestramento dei modelli di *machine learning*

Per quanto appena detto in relazione al *dataset* di interesse, è evidente che ci si trova in un contesto di *learning* supervisionato (essendo in possesso delle osservazioni della variabile risposta) e, in particolare, di classificazione binaria (essendo la variabile risposta di natura binaria). L'obbiettivo è quindi quello di addestrare dei modelli di *machine learning* che, data l'osservazione dei dati di bilancio, permettano di distinguere tra società fraudolente e non fraudolente. In particolare, come già mostrato nella Sezione 3.3, lo scopo è quello di modellare la probabilità $P(Y = 1 \mid X = \mathbf{x})$, cioè la probabilità che una società sia fraudolenta data l'osservazione dei suoi dati di bilancio $\mathbf{x}$. Se questa probabilità è superiore a 0.5 la società viene classificata come fraudolenta e viceversa.

Prima di procedere con l'addestramento dei modelli di ML, si è deciso di suddividere il *dataset* in un *training set*, contenente l'80% delle osservazioni ($n_{tr} = 228$), e in un *test set* ($n_{te} = 58$) (si veda la Sezione 3.1). Il *training set* è stato utilizzato per addestrare due differenti *black-box model*: un *neural network* (NN) e un *extreme gradient boosting* (XGB) (introdotti nella Sezione 3.2).[13] Si è inoltre costruito un semplice modello di regressione logistica con lo scopo di avere un *benchmark* per la valutazione delle performance dei due *black-box model*. I modelli così ottenuti sono stati utilizzati per produrre delle previsioni inerentemente alle osservazioni contenute nel *test set*. Confrontando queste previsioni con i valori della variabile risposta effettivamente osservati si sono calcolate due misure di performance: l'*accuracy* e la *sensitivity* (descritte nella Sezione 3.3) (Tabella 2).

Tabella 2: performance dei modelli addestrati.

Modello	**Accuracy**	**Sensitivity**
Logistic	0.603	0.516
Neural Network	0.655	0.709
Extreme Gradient Boosting	0.672	0.677

[13] I parametri dei modelli addestrati sono stati individuati tramite una procedura di 10-*fold cross validation* e sono:

- Neural Network: *hidden_ layer_ sizes* = $(50, 100)$, *learning_ rate_ init* = 0.001, *solver* = lbfgs.
- Extreme Gradient Boosting: *max_ depth* = 100, *n_ estimators* = 100, *learning_ rate* = 0.05, *colsample_ bytree* = 0.7, *subsample* = 0.7.

I codici utilizzati per addestrare tali modelli sono presentati nell'Appendice A.1.

Si può notare che i due *black-box model* risultano migliori rispetto al semplice modello di regressione logistica relativamente ad entrambe le metriche di performance utilizzate. Il loro utilizzo risulterebbe quindi preferibile, se non fosse per l'impossibilità di spiegare le ragioni che li hanno portati a produrre certe previsioni. Il contesto di interesse (quello della *financial fraud detection*) è infatti un *high-stakes decision making context* (si rimanda alla Sezione 4.1) e, di conseguenza, il *decision maker* (in questo caso la *China Securities Regulatory Commission*) ha bisogno di maggiori garanzie riguardo l'affidabilità delle previsione ottenute. L'introduzione di metodi per l'interpretabilità può risultare quindi di grande utilità. Va infine sottolineato che in questa sede lo scopo principale è quello di mostrare il funzionamento dei metodi di interpretabilità, cioè spiegare i risultati dei modelli di ML indipendentemente dalle loro performance. Per questo motivo non si è posta molta attenzione nel ricercare un *dataset* più numeroso o nel perfezionare i modelli addestrati così da poter migliore le loro performance previsionali.

9.3 Applicazione del metodo SHAP e delle sue estensioni

Dati i *black-box model* addestrati, si è deciso di far uso del metodo SHAP per provare a comprendere le ragioni sottostanti alle previsioni da questi prodotte. Inoltre, essendo le variabili del *dataset* già suddivise in gruppi, si è deciso di far uso anche dei due differenti approcci per il calcolo dello SHAP a gruppi: il *Group* SHAP e lo SHAP *generalized value*. Come già detto, in questo contesto i modelli di ML restituiscono come *output* una probabilità. L'obbiettivo è quindi quello di individuare il contributo che le varie *feature* (o i vari gruppi di *feature*) hanno avuto nel determinare tale probabilità.[14]

Per prima cosa si mostra un esempio di spiegazione fornita dal metodo SHAP (Figura 9). Il modello oggetto di interpretazione è il NN e la previsione spiegata è relativa ad un'istanza correttamente classificata come fraudolenta dal modello. Partendo da $\mathbb{E}[f(\mathbf{x})]$, che rappresenta la previsione mediamente prodotta dal NN sul *test set*, lo SHAP assegna ad ogni *feature* un valore rappresentante il contributo (positivo o negativo) che questa ha avuto nella previsione. La somma dei contributi, aggiunta alla previsione media, restituisce la previsione prodotta dal modello $f(\mathbf{x}^*)$ (proprietà di *local accuracy*, come già discusso nella Sezione 7.2). Da questa spiegazione emerge che due variabili su tutte (*Finanomalies*122 e *Finanomalies*142) hanno portato il NN a classificare l'istanza come fraudolenta. Questa informazione è di fondamentale importanza per il *decision maker*. Questi potrà infatti valutare, anche grazie alla sua esperienza nel settore, se tale spiegazione è o meno ragionevole, incrementando o riducendo la fiducia che egli ripone nella previsione. La Figura 10 mostra invece i valori dello SHAP relativamente alla previsione prodotta sulla stessa istanza dall'XGB. Si può notare come la spiegazione fornita dai due modelli sia piuttosto simile. Questa somiglianza tuttavia non è sempre garantita in quanto, avendo i modelli addestrati dei funzionamenti molto differenti tra loro, le ragioni retrostanti alle loro previsioni potrebbero essere notevolmente diverse. A conferma di questo, si è calcolato lo scarto quadratico medio tra gli SHAP relativi ai due modelli lungo tutte le istanze del *test set*. Questo si attesta intorno a 0.055 e, tenuto conto della dimensione dei singoli contributi, risulta una scarto piuttosto significativo, dimostrando che la somiglianza tra la Figura 9 e 10 è un caso abbastanza fortuito. I metodi di interpretabilità possono

[14] I codici utilizzati per il calcolo dello SHAP e delle sue estensioni sono presentati nell'Appendice A.2. Quelli utilizzati per la costruzione delle figure sono invece stati presi dal seguente indirizzo web: `https://codeocean.com/capsule/5373825/tree/v1`.

essere quindi utili anche sotto questo punto di vista: dati due modelli che producono previsioni simili, si può preferire quello le cui spiegazioni risultino più ragionevoli.

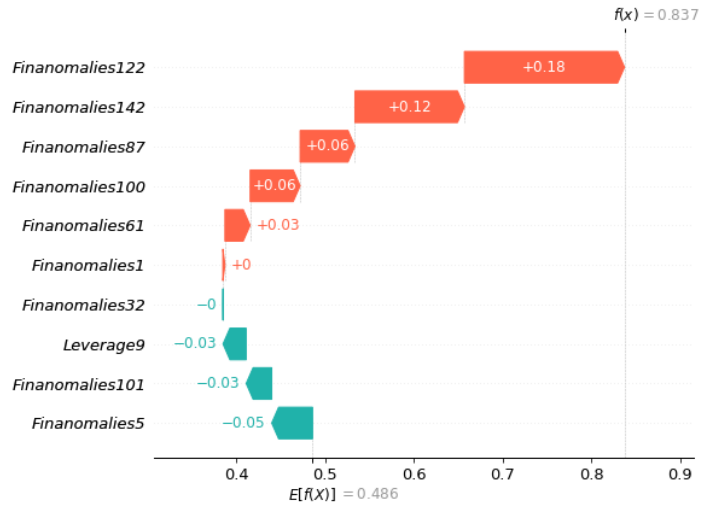

Figura 9: spiegazione fornita dal metodo SHAP relativamente ad una previsione prodotta dal NN.

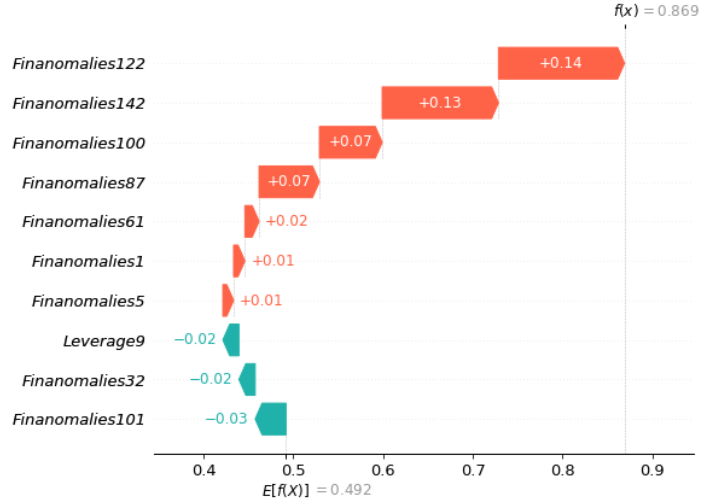

Figura 10: spiegazione fornita dal metodo SHAP relativamente ad una previsione prodotta dall'XGB.

Un'ulteriore informazione di interesse per il *decision maker* può riguardare il contributo dei gruppi di *feature* nel determinare una certa previsione. Si presenta quindi un esempio di spiegazione fornita dal GS e dallo SGV, sempre in relazione alla solita istanza (Figura 11). Dall'uso di questi metodi emerge che gli indici di profitto sono la categoria di variabili che ha maggiormente portato il modello (il NN in questo caso) a classificare l'istanza come fraudolenta. Da notare che i due differenti approcci, pur presentando delle sostanziali differenze da un punto di vista teorico (mostrate nella Sezione 6.3), hanno prodotto una spiegazione molto simile. Per valutare globalmente la differenza tra questi, si è quindi calcolato lo scarto quadratico medio tra i valori del GS e dello SGV lungo tutte le istanze del *test set*. Questo, in relazione ad entrambi i modelli addestrati, si attesta intorno a 0.004, mostrando che tra i due metodi non esiste una differenza significativa, quantomeno non in questo caso. Si potrebbe quindi pensare di utilizzarne solo uno, magari quello più efficiente da un punto di vista computazionale. Si può inoltre notare che

il valore complessivamente ricostruito dal GS (pari a 0.837) coincide esattamente con quello ricostruito dallo SHAP (Figura 9), dimostrando che il GS rispetta la proprietà di *local accuracy* (come visto nella Sezione 6.3), diversamente dallo SGV che restituisce un totale leggermente differente.

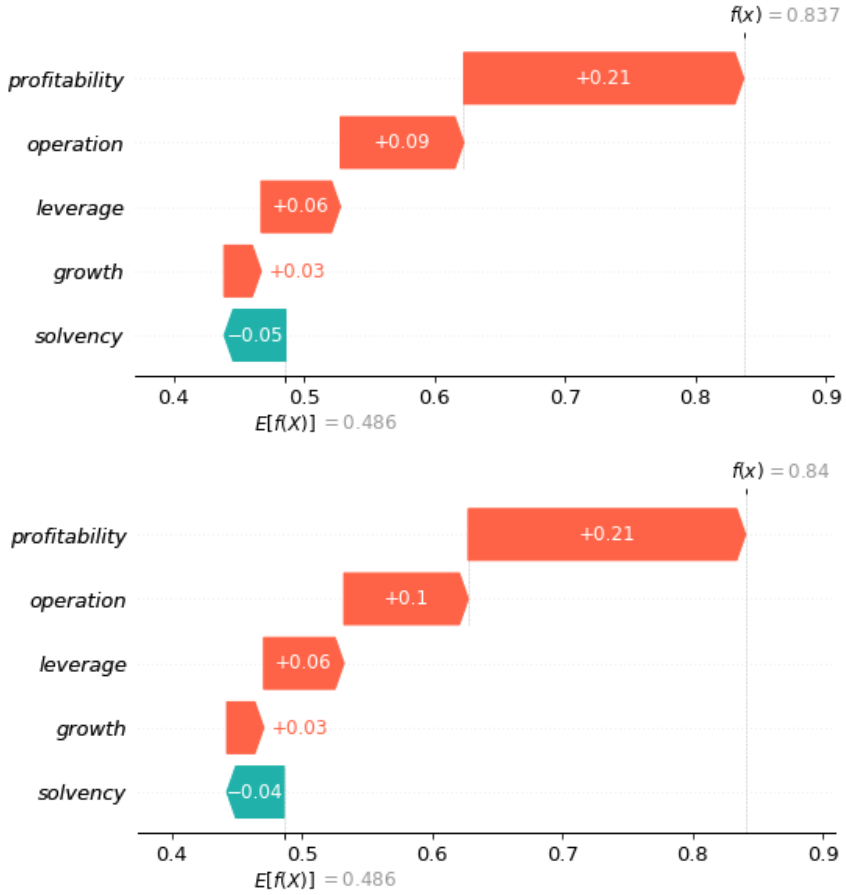

Figura 11: in alto la spiegazione fornita dal *Group* SHAP relativamente ad una previsione prodotta dal NN. In basso la spiegazione della stessa previsione ma fornita dallo SHAP *generalized value*.

Risulta inoltre di interesse confrontare i tempi computazionali necessari ai vari metodi di interpretabilità per produrre la spiegazione.[15] La Tabella 3 mostra, per ogni modello addestrato, il tempo mediamente necessario per il calcolo delle varie configurazioni del metodo SHAP relativamente ad una singola istanza. Innanzitutto si può notare come l'efficienza computazionale vari al variare del modello da interpretare. Infatti, il tempo necessario per il calcolo delle quantità $\hat{f}_{\mathbf{x}^*}$ dipende da quanto lo specifico modello impiega a produrre una previsione (si veda la Sezione 7.2). In questo caso il NN risulta più efficiente rispetto all'XGB. In secondo luogo si osserva come il GS risulti nettamente più rapido in confronto agli altri due metodi. Come già detto nella Sezione 6.3, ciò dipende dal numero inferiore di coalizioni tenute in considerazione per il suo calcolo: in questo caso si hanno infatti $2^m = 2^{10} = 1024$ coalizioni da valutare nello SHAP, contro le $2^g = 2^5 = 32$ necessarie nel GS. Infine, si nota che lo SGV non comporta nessun vantaggio computazionale di rilievo rispetto allo SHAP. Questo dipende dal

[15] Si sottolinea che il tempo computazionale per l'applicazione dei metodi proposti dipende anche dal computer utilizzato per eseguire il calcolo. I tempi riportati in questa e nelle successive sezioni potrebbero quindi variare se si ripetessero gli esperimenti proposti su un differente supporto.

fatto che uno dei gruppi considerati è composto da una sola *feature*, portando lo SGV a dover valutare lo stesso numero di coalizioni dello SHAP (come sottolineato nella Sezione 6.3).

Tabella 3: Tempo medio (in secondi) necessario per il calcolo delle varie forme di SHAP relativamente ad una singola istanza.

Modello	**Metodo**	**Tempo**
Neural Network	SHAP	1.125
	Group SHAP	0.035
	SHAP Generalzied	1.171
Extreme Gradient Boosting	SHAP	1.825
	Group SHAP	0.056
	SHAP Generalized	1.741

9.4 Applicazione dei metodi di approssimazione dello SHAP

Dopo aver mostrato alcune applicazioni del metodo SHAP e delle sue estensioni, risulta interessante far uso dei metodi di approssimazione dello SHAP con lo scopo di valutare la bontà di queste approssimazioni, nonché la loro efficienza computazionale.

9.4.1 Valutazioni sulla convergenza dei metodi di approssimazione

In questa fase della sperimentazione si vuole verificare quanto valide siano le approssimazioni prodotte dal *Kernel* e dal k_{ADD}-SHAP. Per fare questo si riprende l'esperimento condotto da Pelegrina, Duarte e Grabisch nell'articolo "A k-additive Choquet integral-based approach to approximate the SHAP values for local interpretability in machine learning" (2023): scelto un certo modello, un certo metodo di approssimazione e un certo numero $n_{\mathcal{M}}$ di coalizioni da utilizzare nel processo di ottimizzazione, per ogni istanza del *test set* viene calcolata la seguente quantità:

$$\varepsilon_i = \sum_{j=1}^{m} (\phi_j^{exact,i} - \phi_j^{proxi,i})^2, \tag{78}$$

dove $\phi_j^{exact,i}$ rappresenta il valore dello SHAP esatto relativo alla j-esima *feature* e all'i-esima istanza del *test set* e $\phi_j^{proxi,i}$ ne rappresenta la *proxi* prodotta dal metodo di approssimazione scelto. La media di questa quantità lungo tutte le *feature* del *test set* viene indicata con $\bar{\varepsilon}$ e rappresenta una misura della bontà delle *proxi* prodotte dal metodo di approssimazione. Poiché tale metodo produce risultati differenti al variare delle $n_{\mathcal{M}}$ coalizioni estratte, si ripete questa procedura per un numero R di estrazioni differenti (ma della stessa numerosità). A questo punto si avrà a disposizione una serie di valori $\bar{\varepsilon}^1, ..., \bar{\varepsilon}^R$ e da questa distribuzione empirica si estraggono la mediana, il primo e il nono decile. Tutto l'esperimento descritto viene poi ripetuto variando il numero $n_{\mathcal{M}}$ di coalizioni estratte, il metodo di approssimazione scelto e il modello da interpretare [38].

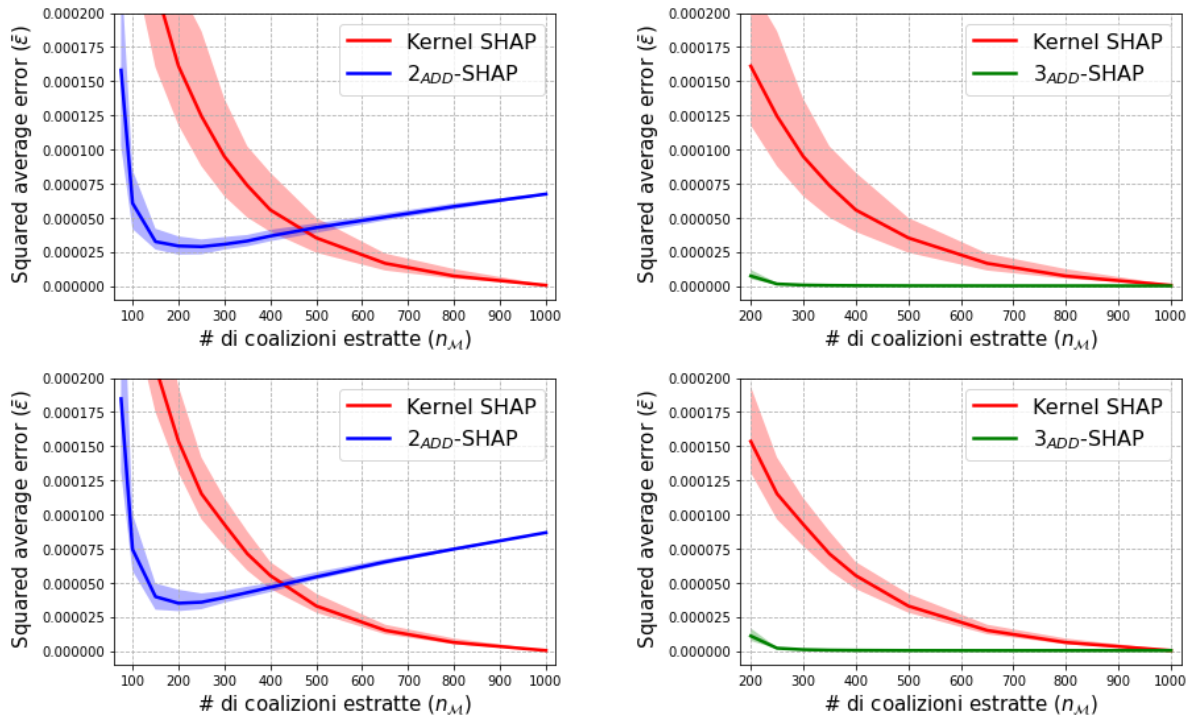

Figura 12: I due grafici in alto mostrano la convergenza al crescere di $n_{\mathcal{M}}$ del *Kernel*, del 2_{ADD} e del 3_{ADD} SHAP verso lo SHAP esatto quando i metodi in questione sono applicati sul NN. I due grafici in basso sono invece riferiti al caso dell'XGB.

La Figura 12 mostra i risultati dell'esperimento appena presentato relativamente al metodo SHAP e alle sue approssimazioni.[16] Nello specifico si è scelto di fissare $R = 100$ e $n_{\mathcal{M}}$ pari a 13 differenti valori compresi tra 50 e 1000 (ricordando che in questo caso il numero massimo di coalizioni è $2^{10} = 1024$). Le curve rappresentano l'andamento della mediana della distribuzione degli $\bar{\varepsilon}$ al variare del numero $n_{\mathcal{M}}$ di coalizioni considerate. Le aree colorate attorno alle curve mostrano invece l'andamento del primo e del nono decile e danno quindi una misura di quanto la bontà delle approssimazioni vari al variare delle specifiche coalizioni estratte. Si può innanzitutto notare che, se il *Kernel* e il 3_{ADD}-SHAP[17] convergono verso lo SHAP esatto al crescere di $n_{\mathcal{M}}$, viceversa il 2_{ADD}-SHAP tende a divergere. Questo si può spiegare tenendo in considerazione che il 2_{ADD}-SHAP è il modello meno flessibile tra quelli considerati, cioè quello a minor parametrizzazione (si veda la Sezione 8.2.2). Se si considera 0.000025 (cioè 0.005 se riportato sulla scala originale) un livello di errore quadratico sufficientemente piccolo da considerare come ottima l'approssimazione, si può notare che il 2_{ADD} e il 3_{ADD}-SHAP raggiungono questo livello per un numero di coalizioni (circa 200) inferiore rispetto a quelle necessarie al *Kernel* SHAP (circa 600). In sostanza, per quanto tutti e tre i metodi permettano di raggiungere una buona approssimazione dello SHAP con un numero nettamente inferiore di valutazioni, i due metodi che presentano dei gradi di additività (ed in particolare il 3_{ADD}-SHAP) risultano essere più parsimoniosi sotto questo punto di vista. Questo risultato si spiega tenendo in considerazione che l'alta parametrizzazione del *Kernel* SHAP lo porta ad aver bisogno di un maggior numero di osservazioni per produrre buoni risultati (come discusso

[16] I codici per le figure sono stati presi all'indirizzo `https://github.com/GuilhermePelegrina/k_addSHAP`.

[17] I codici utilizzati per calcolare il *Kernel* e il k_{ADD}-SHAP, sia in relazione allo SHAP sia in relazione alle sue estensioni, sono presentati nell'Appendice A.2.

nella Sezione 8.2.2). Si può infine notare che, al crescere del numero $n_{\mathcal{M}}$, la volatilità delle approssimazioni si riduce in quanto, estraendo un numero maggiore di coalizioni, le R differenti estrazioni risulteranno sempre più simili tra loro.

La Figura 13 mostra invece i risultati dello stesso esperimento ma applicato allo SGV e alle sue approssimazioni. I valori di R e di $n_{\mathcal{M}}$ utilizzati in questo caso sono gli stessi dell'esperimento descritto in Figura 12. Infatti si è già detto che, essendo uno dei gruppi composti da una sola *feature*, il numero di coalizioni per lo SGV è lo stesso dello SHAP. I risultati che emergono risultano analoghi a quelli già descritti nel caso dello SHAP. Anche in questo caso si può infatti notare una non-convergenza del 2_{ADD}-SHAP e una convergenza più rapida per il 3_{ADD}-SHAP rispetto agli altri metodi. Va però detto che in questo caso la convergenza dei modelli risulta in generale più lenta rispetto a quella mostrata in Figura 12. Si raggiunge infatti un buon livello di approssimazione con $n_{\mathcal{M}} \approx 350$ per il 3_{ADD}-SHAP, con $n_{\mathcal{M}} \approx 750$ per il *Kernel* SHAP mentre per il 2_{ADD}-SHAP non si raggiunge mai un livello ottimo di approssimazione. Si può inoltre notare che la volatilità degli errori quadratici risulta in questo caso molto più elevata, soprattutto per il *Kernel* SHAP. In ogni caso l'utilizzo di questi metodi risulta essere una valida soluzione per ridurre il numero di valutazioni necessarie al calcolo dello SGV.

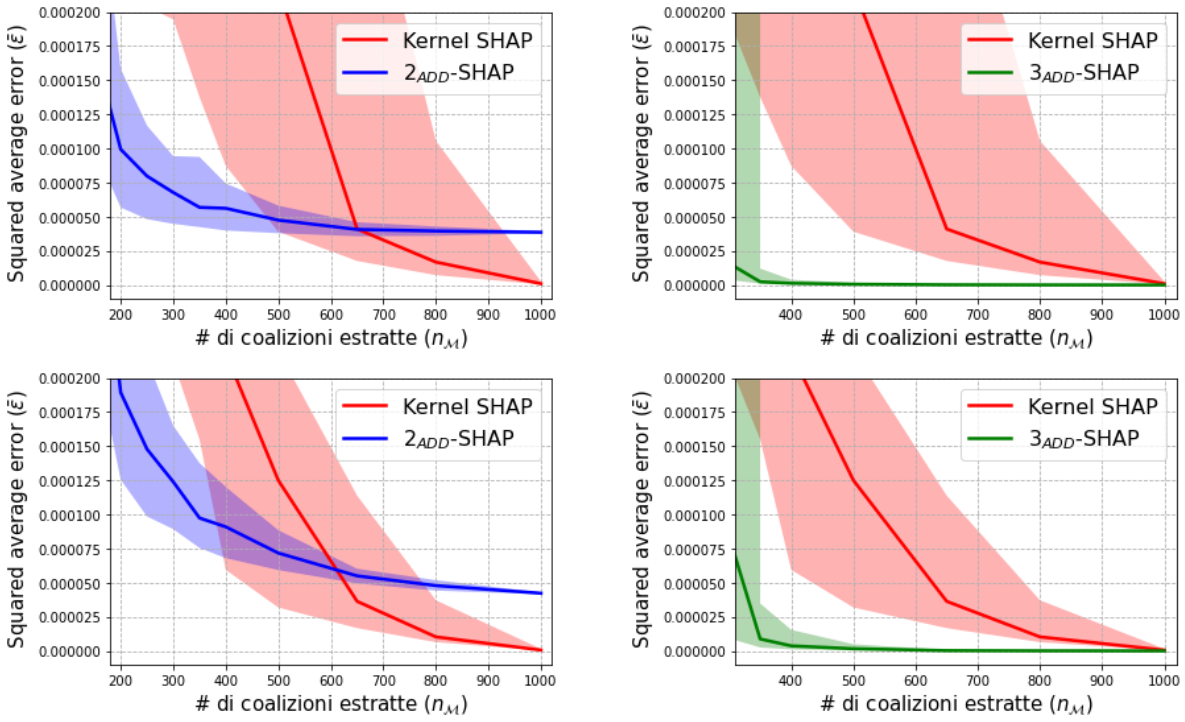

Figura 13: I due grafici in alto mostrano la convergenza al crescere di $n_{\mathcal{M}}$ del *Kernel*, del 2_{ADD} e del 3_{ADD} SHAP verso lo SHAP *generalized value* quando i metodi in questione sono applicati sul NN. I due grafici in basso sono invece riferiti al caso dell'XGB.

Si è infine ripetuto l'esperimento anche per il *Group* SHAP. Va detto che in questo caso il vantaggio computazionale dei metodi di approssimazione risulta marginale, in quanto il numero di coalizioni necessarie per il suo calcolo è pari a $2^5 = 32$. Tuttavia, in un eventuale contesto con un numero elevato di gruppi, anche l'approssimazione del *Group* SHAP può avere una sua utilità. In questo è stato fissato $R = 100$ mentre $n_{\mathcal{M}}$ è stato fissato a 10 diversi valori compresi tra 15 e 32. In Figura 14 vengono presen-

tati i risultati dell'esperimento solo in relazione al *Kernel* e al 2_{ADD}-SHAP, in quanto per il 3_{ADD}-SHAP si osserva un errore quadratico inferiore a 0.0002 (cioè il limite superiore della figura) solo per $n_{\mathcal{M}} = 30$, rendendo il suo grafico di difficile rappresentazione. Anche in questo caso è possibile notare una lieve non-convergenza del 2_{ADD}-SHAP, che risulta tuttavia raggiungere un ottimo livello di approssimazione per un numero di coalizioni (circa 25) inferiore a quelle necessarie al *Kernel* SHAP (circa 31).

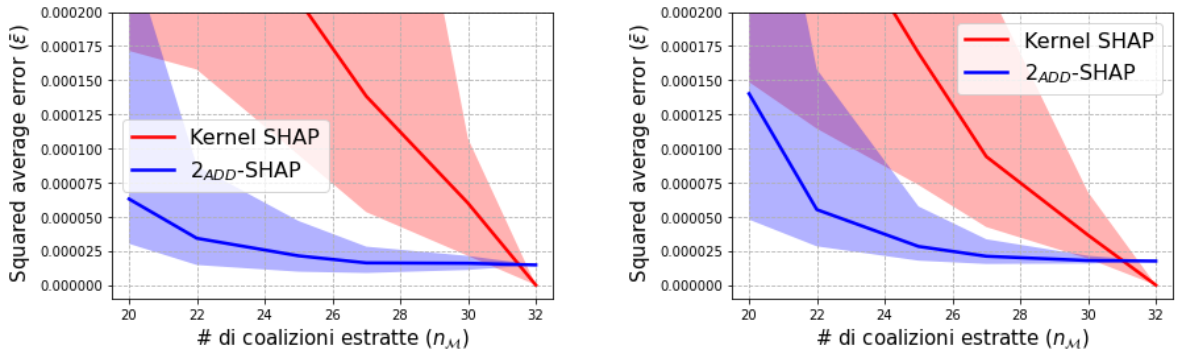

Figura 14: I due grafici mostrano la convergenza al crescere di $n_{\mathcal{M}}$ del *Kernel* e del 2_{ADD}-SHAP verso il *Group* SHAP. A sinistra i metodi sono riferiti al NN, a destra sono riferiti all'XGB.

9.4.2 Valutazioni sull'efficienza computazionale dei metodi di approssimazione

Dopo aver valutato la convergenza dei vari metodi di approssimazione verso lo SHAP esatto, si ritiene di interesse confrontare l'efficienza computazionale di queste approssimazioni. Per farlo si è sfruttato un procedimento analogo a quello descritto nella precedente sezione: fissato un modello, un metodo di approssimazione e un numero $n_{\mathcal{M}}$ di coalizioni estratte, si è misurato il tempo necessario per il calcolo dell'approssimazione dello SHAP relativamente ad ogni istanza nel *test set* e per ogni ripetizione dell'estrazione. Si è poi fatta una media dei valori così ottenuti ricavando il tempo mediamente necessario per il calcolo di una certa approssimazione dello SHAP per una singola istanza. La misurazione è stata poi ripetuta per differenti valori di $n_{\mathcal{M}}$, per ogni metodo di approssimazione e per ogni modello che è stato addestrato. Il numero R di ripetizioni delle estrazioni e i valori di $n_{\mathcal{M}}$ sono stati fissati nello stesso modo descritto per gli esperimenti della precedente sezione.

La Figura 15 mostra i risultati delle misurazioni appena descritte in relazione ai metodi di approssimazione dello SHAP. Si può innanzitutto notare che l'andamento del tempo computazionale medio al variare del numero di coalizioni estratte cresce in maniera pressoché lineare. Questo risultato sembra piuttosto ragionevole, in quanto il tempo necessario per valutare le singole coalizioni risulta grossomodo identico. Se si aumenta il numero di coalizioni da valutare, aumenterà allora in modo proporzionale il tempo necessario per valutarle e, quindi, per calcolare l'approssimazione dello SHAP. Si può inoltre notare che non ci sono differenze computazionali significative tra i metodi di approssimazione presentati. Questo dipende dal fatto che la principale difficoltà di calcolo non riguarda la risoluzione dei differenti problemi di ottimizzazione, poiché questi presentano una soluzione in forma chiusa, ma bensì il calcolo delle quantità $\hat{f}_{\mathbf{x}^*}$, che avviene nelle stesse modalità per tutti i metodi di approssimazione. La differente soluzione del problema di ottimizzazione comporta quindi solo differenze minime di efficienza: ad esempio, il 3_{ADD}-SHAP risulta leggermente più dispendioso, in quanto la soluzione da esso calcolata si

compone di un maggior numero di incognite (come discusso nella Sezione 8.2.2). Se è quindi vero che i vari metodi necessitano grossomodo dello stesso tempo per produrre la soluzione (a parità di $n_{\mathcal{M}}$), allora è evidente che quello più efficiente è quello che necessita del minor numero di coalizioni per approssimare adeguatamente lo SHAP. Per quanto visto nella sezione precedente il metodo in questione sembra essere il 3_{ADD}-SHAP. Si può infine notare che tutti e tre i metodi comportano notevoli vantaggi rispetto allo SHAP esatto, rendendo la sua approssimazione computazionalmente molto efficiente.

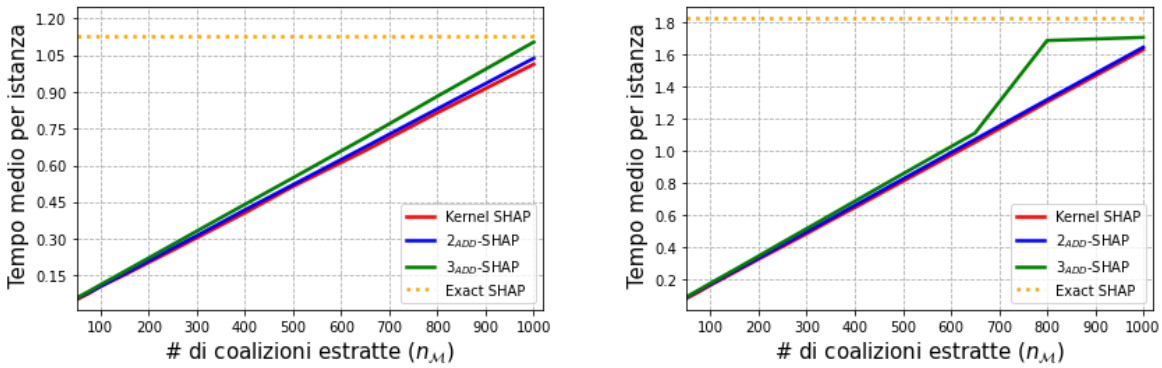

Figura 15: andamento del tempo medio (in secondi) necessario per l'utilizzo dei vari metodi di approssimazione dello SHAP su una singola istanza al variare del numero $n_{\mathcal{M}}$ di coalizioni estratte. Il grafico a sinistra si riferisce al NN, quello a destra all'XGB.

Le Figure 16 e 17 riportano le misurazioni sopra descritte in relazione ai metodi di approssimazione dello SGV e del GS. Riguardo lo SGV i risultati ottenuti sono pressoché identici a quelli della Figura 15 e si possono quindi raggiungere le stesse conclusioni. Si può inoltre notare che il tempo computazionale necessario ad implementare le approssimazioni dello SGV non è così elevato, nonostante venga risolto un problema di ottimizzazione per ogni gruppo di *feature* (come detto nella Sezione 8.3). Riguardo il GS si può invece notare che il *Kernel* SHAP risulta discretamente più efficiente rispetto agli altri due approcci. Va però tenuto in considerazione il fatto che in questo caso i tempi computazionali sono estremamente ridotti (essendo ridotto il numero di coalizioni da valutare) e, di conseguenza, questa maggiore efficienza rientra nell'ordine dei millesimi di secondo, risultando quindi poco rilevante.

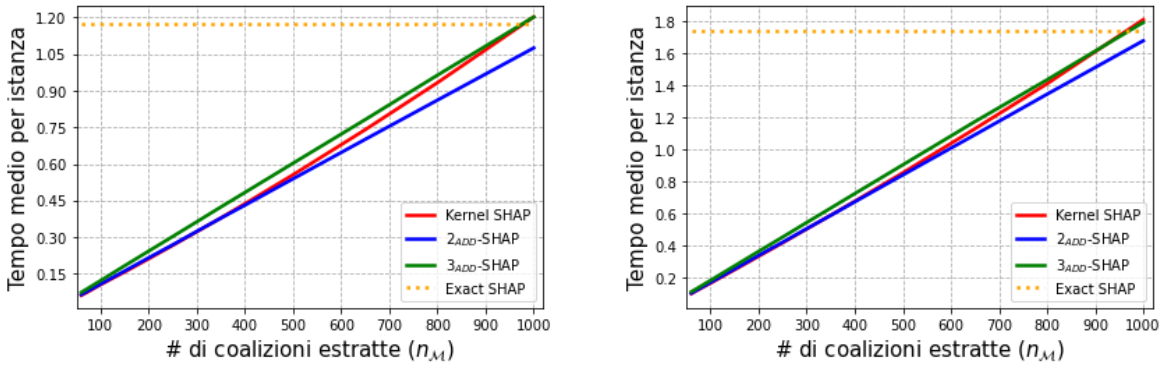

Figura 16: andamento del tempo medio (in secondi) necessario per l'utilizzo dei vari metodi di approssimazione dello SHAP *generalized value* su una singola istanza al variare del numero $n_{\mathcal{M}}$ di coalizioni estratte. Il grafico a sinistra si riferisce al NN, quello a destra all'XGB.

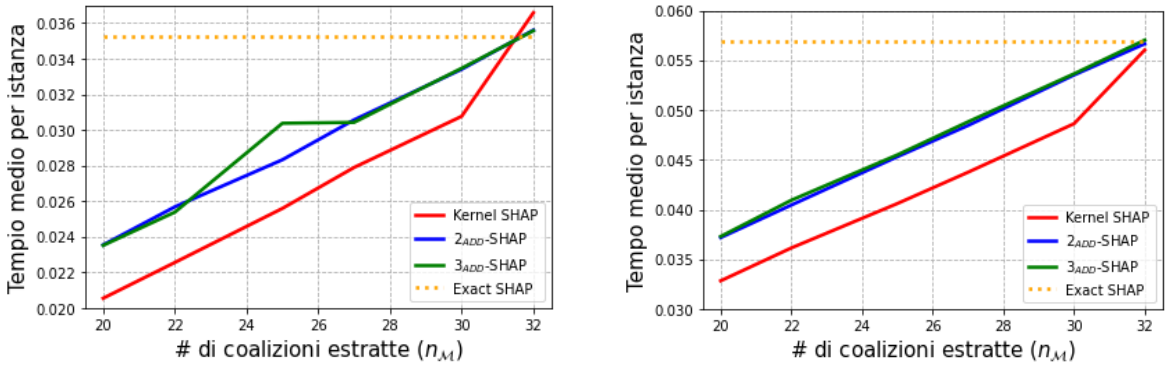

Figura 17: andamento del tempo medio (in secondi) necessario per l'utilizzo dei vari metodi di approssimazione del *Group* SHAP su una singola istanza al variare del numero $n_{\mathcal{M}}$ di coalizioni estratte. Il grafico a sinistra si riferisce al NN, quello a destra all'XGB.

10 Conclusioni

In questa tesi si è voluto presentare il funzionamento del metodo SHAP per l'interpretabilità locale delle previsioni prodotte dai modelli di *machine learning*. Tale metodo permette di assegnare alle *feature* utilizzate nell'addestramento del modello un valore reale, rappresentante il contributo (positivo o negativo) che queste hanno avuto nel determinare una certa previsione. Quest'informazione diventa di grande utilità nei contesti di *high stake decision making*, in quanto permette al *decision maker* di valutare l'affidabilità di tale previsione e, di conseguenza, di utilizzare il modello in un modo più consapevole, riducendo le possibili ripercussioni negative che possono gravare sui destinatari delle decisioni. L'applicazione nel contesto della *financial fraud detection* ha permesso di mostrare l'utilità di questo metodo (e delle sue estensioni per gruppi di variabili) in uno dei numerosi campi in cui l'utilizzo dei modelli di *machine learning* sta osservando una crescita importante. Malgrado le numerose proprietà desiderabili, l'applicazione dello SHAP nella pratica è fortemente limitata dalla sua complessità computazionale, che cresce esponenzialmente all'aumentare del numero di *feature*. Si è allora mostrato in che modo l'utilizzo di alcuni metodi di approssimazione possa ridurre drasticamente il numero di valutazioni necessarie per il calcolo dello SHAP, pur permettendone una stima molto accurata. Dai risultati ottenuti nell'analisi sperimentale effettuata nel Capitolo 9 sembra emergere che il 3_{ADD}-SHAP sia il migliore tra i metodi di approssimazione confrontati. Questo infatti sembra presentare il miglior livello di flessibilità: il *Kernel* SHAP, a causa della sua maggior parametrizzazione, richiede un maggior numero di osservazioni (in questo caso di coalizioni) per raggiungere un'approssimazione soddisfacente dello SHAP esatto; per contro, il 2_{ADD}-SHAP tende a risultare poco flessibile, comportando un'approssimazione meno accurata e, talvolta, il rischio di divergere rispetto al valore dello SHAP esatto. Richiedendo un minor numero di valutazioni, il 3_{ADD}-SHAP risulta essere anche il metodo più efficiente da un punto di vista computazionale, permettendo inoltre, rispetto al *Kernel* SHAP, il calcolo delle SHAP *interaction*, utili per fornire una spiegazione ancora più accurata della previsione prodotta dal modello. Va però ricordato che tutti i metodi presentati (sia lo SHAP esatto sia le sue approssimazioni) sfruttano l'assunzione di indipendenza tra le *feature* utilizzate per addestrare il modello, assunzione che risulta piuttosto irrealistica nei *dataset* reali. Un'eventuale violazione di questa assunzione porterebbe a spiegazioni completamente distorte e, di conseguenza, inutilizzabili. Lo sviluppo di soluzioni a questo tipo di criticità risulta quindi necessario affinché i metodi proposti possano trovare una diffusa applicazione nella pratica.

A Codici

In questa appendice vengono riportati i principali codici *Python* utilizzati per eseguire gli esperimenti descritti nel Capitolo 9. La versione utilizzata del *software* è la 3.8.8.

A.1 Modelli di *machine learning*

Per costruire i modelli di *machine learning* si è fatto uso del pacchetto *"sklearn"*. I codici in questa appendice sono in parte ripresi dal libro "Machine Learning for Financial Risk Management with Python" di Abdullah Karasan (2021) [28].

```
#Si importa il dataset e lo si divide in training e test set
data = pd.read_csv(...\data_SHAPexperiment_IV0.05)
data = data.drop(['Finanomalies97','Finanomalies112'], axis = 1)
X, y = data.iloc[:, 5:], data.iloc[:, 4]
X_train, X_test, y_train, y_test = train_test_split(X, y,random_state = 0,
    train_size = 0.8)

#Si addestra un modello logistico e se ne valuta la performance
model_log = sm.Logit(y_train, X_train)
model_log = model_log.fit()
pred_log = model_log.predict(X_test)
for i in range(58):
    if pred_log.iloc[i] > 0.5:
        pred_log.iloc[i] = 1
    else:
        pred_log.iloc[i] = 0
confusion_matrix(y_test, pred_log)
accuracy_score(y_test, pred_log)
recall_score(y_test, pred_log)

#Si addestra un extreme gradient boosting e se ne valuta la performance
param_XGB = {'learning_rate': [0.01, 0.05, 0.1], 'max_depth': [3, 5, 7, 10,
     20, 50, 100], 'subsample': [0.3, 0.5, 0.7], 'colsample_bytree': [0.3,
    0.5, 0.7], 'n_estimators': [10, 20, 30, 50, 100]}
random_state = np.random.RandomState(1)
model_XGB = RandomizedSearchCV(XGBClassifier(random_state = random_state),
    param_XGB, n_jobs = -1, random_state = random_state, scoring = 'accuracy
    ', cv = 10)
model_XGB.fit(X_train, y_train)
pred_XGB = model_XGB.predict(X_test)
confusion_matrix(y_test, pred_XGB)
accuracy_score(y_test, pred_XGB)
recall_score(y_test, pred_XGB)
```

```
#Si addestra un neural network e se ne valuta la performance
param_NN = {"hidden_layer_sizes": [(50,100),(200,100),(30,10),(100,50)], "
    solver": ["lbfgs"], "learning_rate_init": [0.001, 0.01, 0.05]}
random_state = np.random.RandomState(10)
model_NN = GridSearchCV(MLPClassifier(max_iter=10**6, random_state=
    random_state), param_NN, n_jobs = -1, cv = 10, scoring = "accuracy")
model_NN.fit(X_train, y_train)
pred_NN = model_NN.predict(X_test)
confusion_matrix(y_test,pred_NN)
accuracy_score(y_test, pred_NN)
recall_score(y_test, pred_NN)
```

A.2 Metodi di interpretabilità

In questa appendice vengono riportate le funzioni utilizzate per implementare i vari metodi di interpretabilità descritti. Parte di questi codici è stata ripresa dal lavoro di Guilherme Pelegrina[18] e da quello di Kang Lin e Yuzhuo Gao[19].

Exact SHAP

```
def exact_shap(X, X_train, model):
    """
    Calcola l'exact SHAP per un vettore di istanze. Misura anche il tempo
    medio impiegato per valutare ogni istanza e il tempo impiegato
    per valutare l'intero vettore di istanze.
    ----------
    PARAMETERS:
    X = design matrix delle istanze oggetto di interpretazione.
    X_train  = design matrix delle istanze usate per addestrare
    il modello.
    model = modello da interpretare.
    ------
    RETURN:
    shap_matrix = matrice con gli exact SHAP per ogni istanza e
    per ogni feature.
    time_tot = tempo impiegato per il calcolo dell'intera matrice.
    time_ind = tempo medio impiegato per valutare la singola istanza.
    """
    instances = range(len(X))
    features = X.columns
    X = np.array(X)
    shap = np.ones([len(X), len(features)])

```

[18] https://github.com/GuilhermePelegrina/k_addSHAP.
[19] https://codeocean.com/capsule/5373825/tree/v1.

```
    start = time.time()

    #Si costruiscono le 2^m coalzioni di feature
    all_subsets = []
    for k in range(len(features), -1, -1):
        for element in itertools.combinations(range(len(features)), k):
            all_subsets.append(list(element))

    #Si assegnano i pesi alle varie coalizioni
    omega = []
    for l in all_subsets[1:]:
        omegal = (math.factorial(len(l)) * math.factorial(len(features) -
len(l) - 1)) /math.factorial(len(features))
        omega.append(omegal)

    for i in range(len(instances)):
        print('instance', i)
        temp = np.tile(X[i], (len(X_train), 1))

        #Si calcolano i valori attesi condizionati per ogni coalizione
        #e relativamente l'istanza i-esima
        all_val = []
        for l in all_subsets:
            mat = np.array(X_train)
            mat[:, l] = temp[:, l]
            val = list(model.predict_proba(mat)[:,1])
            valS = np.array(val)
            valS = valS.mean()
            all_val.append(valS)

        for j in range(len(features)):

            #Si calcola lo SHAP per la feature j-esima e per l'istanza
            #i-esima
            Phi = 0
            for l in range(len(all_subsets)):
                if j not in all_subsets[l]:
                    valore = all_val[l]
                    peso = omega[l - 1]
                    lj = all_subsets[l] + [j]; lj.sort()
                    indice = all_subsets.index(lj)
                    valorej = all_val[indice]
                    phi = peso * (valorej - valore)
                    Phi += phi

            shap[i,j] = Phi
```

```
    end = time.time()
    time_tot = round(end - start, 8)
    time_ind = round(time_tot/len(X), 8)

    results = {'shap_matrix': shap, 'time_tot': time_tot, 'time_ind':
    time_ind}
    return results
```

***Exact Group* SHAP**

```
def exact_group_shap(X, X_train, model, groups):
    """
    Calcola l'exact Group SHAP per un vettore di istanze. Misura
    anche il tempo medio impiegato per valutare ogni istanza.
    ----------
    PARAMETERS:
    X = design matrix delle istanze oggetto di interpretazione.
    X_train  = design matrix delle istanze usate per addestrare il
    modello.
    model = modello da interpretare.
    groups = i gruppi di feature di cui di vuole valutare il contributo
    nelle previsioni.
    ------
    RETURN:
    shap_matrix = matrice con gli exact Group SHAP per ogni istanza
    e  per ogni features.
    time_tot = tempo impiegato per il calcolo dell'intera matrice.
    time_ind = tempo medio impiegato per valutare la singola istanza.
    """
    instances = range(len(X))
    features = groups
    X = np.array(X)
    shap = np.ones([len(X), len(features)])

    start = time.time()

    #Si costruiscono le 2^g coalizioni di gruppi
    all_subsets = []
    for k in range(len(features), -1, -1):
        for element in itertools.combinations(range(len(features)), k):
            all_subsets.append(list(element))

    #Si trasformano le coalizioni di gruppi in coalizioni di
    #singole feature
    combo = []
    for c in all_subsets:
        lis = []
```

```
        for element in c:
            for fea in features[element]:
                lis.append(fea)
        combo.append(lis)

    #Si assegnano i pesi alle varie coalizioni di gruppi
    omega = []
    for l in all_subsets[1:]:
        omegal = (math.factorial(len(l)) * math.factorial(len(features) -
    len(l) - 1)) / math.factorial(len(features))
        omega.append(omegal)

    for i in range(len(instances)):
        print('instance', i)
        temp = np.tile(X[i], (len(X_train), 1))

        #Si calcolano i valori attesi condizionati per ogni coalizione e
        #relativamente all'istanza i-esima
        all_val = []
        for l in combo:
            mat = np.array(X_train)
            mat[:, l] = temp[:, l]
            val = list(model.predict_proba(mat)[:,1])
            valS = np.array(val)
            valS = valS.mean()
            all_val.append(valS)

        for j in range(len(features)):

            #Si calcola l'exact Group SHAP per il j-esimo gruppo e per
            #l'istanza i-esima
            Phi = 0
            for l in range(len(all_subsets)):
                if j not in all_subsets[l]:
                    valore = all_val[l]
                    peso = omega[l - 1]
                    lj = all_subsets[l] + [j]; lj.sort()
                    indice = all_subsets.index(lj)
                    valorej = all_val[indice]
                    phi = peso * (valorej - valore)
                    Phi += phi

            shap[i,j] = Phi

    end = time.time()
    time_tot = round(end - start, 8)
    time_ind = round(time_tot/len(X), 8)
```

```
84
85      results = {'shap_matrix': shap, 'time_tot': time_tot, 'time_ind':
        time_ind}
86      return results
```

Exact* SHAP *generalized value

```
1  def exact_generalized_shap(X, X_train, model, groups):
2      """
3      Calcola l'exact generalized SHAP per un vettore di istanze. Misura
       anche il
4      tempo medio impiegato per valutare ogni istanza.
5      ----------
6      PARAMETERS:
7      X = design matrix delle istanze oggetto di interpretazione.
8      X_train  = design matrix delle istanze usate per addestrare
9      il modello.
10     model = modello da interpretare.
11     groups = i gruppi di feature di cui di vuole valutare il contributo
12     nelle previsioni.
13     ------
14     RETURN:
15     shap_matrix = matrice con gli exact generalized shap per ogni istanza
16     e per ogni features.
17     time_tot = tempo impiegato per il calcolo dell'intera matrice.
18     time_ind = tempo medio impiegato per valutare la singola istanza.
19     """
20     instances = range(len(X))
21     features = groups
22     features0 = [m for n in features for m in n]
23     X = np.array(X)
24     shap = np.ones([len(X), len(features)])
25
26     start = time.time()
27
28     #Si costruiscono le coalizioni ottenute fissando un gruppo alla
29     #volta e trattando le altre variabili individualmente
30     all_subsets = []
31     for j in range(len(features)):
32         exclusive = features.copy()
33         del(exclusive[j])
34         exclusive = [m for n in exclusive for m in n]
35         for k in range(len(exclusive), -1, -1):
36             for element in itertools.combinations(exclusive, k):
37                 all_subsets.append(list(element))
38                 all_subsets.append(list(element) + features[j])
39
```

```
    for el in all_subsets:
        el = el.sort()

    #Si eliminano le coalizioni doppione
    sub = []
    for el in all_subsets:
        if sub.count(el) < 1:
            sub.append(el)

    for i in range(len(instances)):
        print('instance:', i)
        temp = np.tile(X[i], (len(X_train), 1))

        #Si calcolano i valori attesi condizionati per ogni coalizione e
        #relativamente all'istanza i-esima
        all_val = []
        for l in sub:
            mat = np.array(X_train)
            mat[:, l] = temp[:, l]
            val = list(model.predict_proba(mat)[:,1])
            valS = np.array(val)
            valS = valS.mean()
            all_val.append(valS)

        for j in range(len(features)):

            #Si calcola l'exact generalized SHAP per il j-esimo gruppo
            #e per l'istanza i-esima
            Phi = 0
            for l in range(len(sub)):
                num = 0
                #Si individuano le coalizioni che non contengono
                #il j-esimo gruppo
                for element in features[j]:
                    if element not in sub[l]:
                        num += 1
                if num == len(features[j]):
                    valore = all_val[l]
                    peso = (math.factorial(len(sub[l])) * math.factorial(
len(features0) - len(sub[l]) - len(features[j]))) / math.factorial(len(
features0)-len(features[j])+1)
                    lj = sub[l] + features[j]; lj.sort()
                    indice = sub.index(lj)
                    valorej = all_val[indice]
                    phi = peso * (valorej - valore)
                    Phi += phi

```

```
            shap[i, j] = Phi

    end = time.time()
    time_tot = round(end - start, 8)
    time_ind = round(time_tot/len(X), 8)

    results = {'shap_matrix': shap, 'time_tot': time_tot, 'time_ind':
    time_ind}
    return results
```

Kernel SHAP per singole *feature*

```
def kernel_shap(X, X_train, model, nm, R):
    """
    Calcola i kernel SHAP per un vettore di istanze utilizzando 'nm'
    coalizioni estratte dal power set di M. Ripete la procedura per
    R differenti estrazioni di uguale dimensione 'nm'. Misura anche
    il tempo medio necessario per fare questi calcoli.
    ----------
    PARAMETERS:
    X = design matrix delle istanze oggetto di interpretazione.
    X_train  = design matrix delle istanze usate per addestrare
    il modello.
    model = modello da interpretare.
    nm = numero di coalizioni utilizzate nel calcolo.
    R = numero di ripetizioni dell'estrazione.
    ----------
    RETURN:
    shap_array = array contenente una matrice di kernel SHAP per ogni
    estrazione r-esima. Una singola matrice contiene i kernel SHAP
    per ogni feature e per ogni istanza.
    time_ind = tempo mediamente necessario per calcolare i kernel SHAP
    di una singola istanza.
    time_set = tempo mediamente necessario per calcolare una matrice.
    """
    features = X.columns
    instances = range(len(X))
    X = np.array(X)

    #Si costruiscono le 2^m coalizioni di feature
    PL = []
    for k in range(len(features), -1, -1):
        for element in itertools.combinations(range(len(features)), k):
            PL.append(list(element))

    #Si trasformano le coalizioni in vettori binari
    PM = np.zeros([2**len(features), len(features)])
```

```
    for l in range(2**len(features)):
        PM[l, PL[l]] = 1

    #Si assegna un kernel weight a ogni coalizione
    weights_PM = []
    for m in range(2**len(features)):
        w = (len(features) - 1)/(comb(len(features), PM[m].sum())*PM[m].sum
()*(len(features) - PM[m].sum()))
        weights_PM.append(w)
    weights_PM[0] = 10**6
    weights_PM[-1] = 10**6
    #Si costruiscono i pesi normalizzati da usare nelle estrazioni
    weights_PM_norm = weights_PM/np.array(weights_PM).sum()

    nm_max = len(PL)

    if nm <= nm_max:

        phi_array = np.ones([len(X), len(features) + 1, R])

        start = time.time()
        for r in range(R):
            print('simulation:', r)

            #Si estraggono 'nm' coalizioni
            SM = np.ones([nm, len(features)])
            SL = []
            weights = np.ones([nm])
            I = range(2**len(features))
            np.random.seed(r)
            estrazioni = np.random.choice(I, size = nm, replace = False, p
= weights_PM_norm)

            for s in range(nm):
                SM[s] = PM[estrazioni[s]]
                weights[s] = weights_PM[estrazioni[s]]
                sl = PL[estrazioni[s]]
                SL.append(sl)

            #Date le coalizioni estratte si costruisce la matrice S
            #che non dipende dalla specifica istanza
            Z = np.concatenate((np.ones([nm,1]), SM), axis = 1)
            W = np.diag(weights)
            S = np.linalg.inv(Z.T @ W @ Z) @ Z.T @ W

            phi_matrix = np.ones([len(X), len(features) + 1])

```

```
            for i in range(len(instances)):

                #Si costruisce il vettore f di valori attesi per
                #l'i-esima istanza
                f = []
                temp = np.tile(X[i], (len(X_train), 1))

                for j in SL:
                    mat = np.array(X_train)
                    mat[:, j] = temp[:, j]
                    val = list(model.predict_proba(mat)[:,1])
                    valS = np.array(val)
                    valS = valS.mean()
                    f.append(valS)

                #Si calcolano i kernel SHAP relativi a ogni feature
                #per l'i-esima istanza
                phi = S @ f
                phi_matrix[i] = phi

            phi_array[:,:,r] = phi_matrix

        end = time.time()
        time_cons = end - start
        tempo_medio_tot = round(time_cons/R,8)
        tempo_medio_ind = round(tempo_medio_tot/len(X),8)

        results = {'shap_array': phi_array, 'time_ind': tempo_medio_ind, '
    time_set': tempo_medio_tot}
        return results

    else:
        print('"nm" supera il valore massimo. Valore massimo di "nm":',
    nm_max)
```

***Kernel Group* SHAP**

```
def kernel_group_shap(X, X_train, model, groups, nm, R):
    """
    Calcola i kernel Group SHAP per un vettore di istanze
    utilizzando 'nm' coalizioni estratte dal power set di M. Ripete
    la procedura per R differenti estrazioni di uguale dimensione
    'nm'.Misura anche il tempo medio necessario per fare questi calcoli.
    ----------
    PARAMETERS:
    X = design matrix delle istanze oggetto di interpretazione.
    X_train  = design matrix delle istanze usate per addestrare
```

```
    il modello.
    model = modello da interpretare.
    groups = lista dei gruppi considerati.
    nm = numero di coalizioni utilizzate nel calcolo.
    R = numero di ripetizioni dell'estrazione.
    ----------
    RETURN:
    shap_array = array contenente una matrice di kernel Group SHAP
    per ogni estrazione r-esima. Una singola matrice contiene i kernel
    Group SHAP per ogni feature e per ogni istanza.
    time_ind = tempo mediamente necessario per calcolare i kernel
    Group SHAP per una singola istanza.
    time_set = tempo mediamente necessario per calcolare una matrice.
    """
    features = groups
    instances = range(len(X))
    X = np.array(X)

    #Si costruiscono le 2^g coalizioni di gruppi
    PL = []
    for k in range(len(features), -1, -1):
        for element in itertools.combinations(range(len(features)), k):
            PL.append(list(element))

    #Si trasformano le coalizioni in vettori binari
    PM = np.zeros([2**len(features), len(features)])
    for l in range(2**len(features)):
        PM[l, PL[l]] = 1

    #Si assegnano i kernel weight alle varie coalizioni
    weights_PM = []
    for m in range(2**len(features)):
        w = (len(features) - 1)/(comb(len(features), PM[m].sum())*PM[m].sum
    ()*(len(features) - PM[m].sum()))
        weights_PM.append(w)
    weights_PM[0] = 10**6
    weights_PM[-1] = 10**6
    #Si costruiscono i pesi normalizzati da usare nelle estrazioni
    weights_PM_norm = weights_PM/np.array(weights_PM).sum()

    nm_max = len(PL)

    if nm <= nm_max:

        phi_array = np.ones([len(X), len(features) + 1, R])

        start = time.time()
```

```
        for r in range(R):
            print('simulation:', r)

            #Si estraggono 'nm' coalizioni
            SM = np.ones([nm, len(features)])
            SL = []
            weights = np.ones([nm])
            I = range(2**len(features))
            np.random.seed(r)
            estrazioni = np.random.choice(I, size = nm, replace = False, p
    = weights_PM_norm)

            for s in range(nm):
                SM[s] = PM[estrazioni[s]]
                weights[s] = weights_PM[estrazioni[s]]
                sl = PL[estrazioni[s]]
                SL.append(sl)

            #Date le coalizioni estratte si calcola la matrice S che non
            #dipende dalla specifica istanza
            Z = np.concatenate((np.ones([nm,1]), SM), axis = 1)
            W = np.diag(weights)
            S = np.linalg.inv(Z.T @ W @ Z) @ Z.T @ W

            phi_matrix = np.ones([len(X), len(features) + 1])

            #Si trasformano le coalizioni di gruppi in coalizioni
            #di feature
            combo = []
            for c in SL:
                lis = []
                for element in c:
                    for fea in features[element]:
                        lis.append(fea)
                combo.append(lis)

            for i in range(len(instances)):
                #Si calcola il vettore f di valori attesi per
                #l'i-esima istanza
                f = []
                temp = np.tile(X[i], (len(X_train), 1))

                for j in combo:
                    mat = np.array(X_train)
                    mat[:, j] = temp[:, j]
                    val = list(model.predict_proba(mat)[:,1])
                    valS = np.array(val)
```

```
                    valS = valS.mean()
                    f.append(valS)

                #Si calcolano i kernel Group SHAP relativamente a ogni
                #gruppo di feature per l'i-esima istanza
                phi = S @ f
                phi_matrix[i] = phi

            phi_array[:,:,r] = phi_matrix

        end = time.time()
        time_cons = end - start
        tempo_medio_tot = round(time_cons/R,8)
        tempo_medio_ind = round(tempo_medio_tot/len(X),8)

        results = {'shap_array': phi_array, 'time_ind': tempo_medio_ind, '
    time_set': tempo_medio_tot}
        return results

    else:
        print('"nm" supera il valore massimo. Valore massimo di "nm":',
    nm_max)
```

Kernel* SHAP *generalized value

```
def kernel_generalized_shap(X, X_train, model, groups, nm, R):
    """
    Calcola i kernel generalized SHAP per un vettore di istanze utilizzando
    'nm' coalizioni estratte dal power set di M. Ripete la procedura
    per R differenti estrazioni di uguale dimensione 'nm'. Misura
    anche il tempo medio necessario per fare questi calcoli.
    ----------
    PARAMETERS:
    X = design matrix delle istanze oggetto di interpretazione.
    X_train  = design matrix delle istanze usate per addestrare
    il modello.
    model = modello da interpretare.
    groups = lista dei gruppi.
    nm = numero di coalizioni utilizzate nel calcolo.
    R = numero di ripetizione dell'estrazione.
    ----------
    RETURN:
    shap_array = array contenente una matrice di kernel generalized
    SHAP per ogni estrazione r-esima. Una singola matrice contiene i
    kernel generalized SHAP per ogni feature e per ogni istanza.
    time_ind = tempo mediamente necessario per calcolare i kernel
    generalized SHAP per una singola istanza.
```

```
    time_set = tempo mediamente necessario per calcolare una matrice.
    """
    features = groups
    features0 = [m for n in features for m in n]
    instances = range(len(X))
    X = np.array(X)

    #Si costruiscono le coalizioni fissando un gruppo alla volta e trattando
    le altre variabili individualmente
    all_subsets = []
    for j in range(len(features)):
        exclusive = features.copy()
        del(exclusive[j])
        exclusive = [m for n in exclusive for m in n]
        for k in range(len(exclusive), -1, -1):
            for element in itertools.combinations(exclusive, k):
                all_subsets.append(list(element))
                all_subsets.append(list(element) + features[j])

    for el in all_subsets:
        el = el.sort()

    #Si eliminano le coalizioni doppione
    PL = []
    for el in all_subsets:
        if PL.count(el) < 1:
            PL.append(el)

    #Si trasformano le coalizioni in vettori binari
    PM = np.zeros([len(PL), len(features0)])
    for l in range(len(PL)):
        PM[l, PL[l]] = 1

    #Si assegna un kernel weight a ogni coalizione
    weights_PM = []
    for m in range(len(PL)):
        w = (len(features0) - 1)/(comb(len(features0), PM[m].sum())*PM[m].
    sum()*(len(features0) - PM[m].sum()))
        weights_PM.append(w)
    for m in range(len(weights_PM)):
        if weights_PM[m] == float('inf'):
            weights_PM[m] = 10**6
    #Si calcolano i pesi normalizzati da utilizzare nelle estrazioni
    weights_PM_norm = weights_PM/np.array(weights_PM).sum()

    nm_max = len(PL)
```

```
68
69      if nm <= nm_max:
70
71          phi_array = np.ones([len(X), len(features), R])
72
73          start = time.time()
74
75          for r in range(R):
76              print('simulation:', r)
77
78              #Si estraggono 'nm' coalizioni
79              SM = np.ones([nm, len(features0)])
80              SL = []
81              weights = np.ones([nm])
82              I = range(len(PL))
83              np.random.seed(r)
84              estrazioni = np.random.choice(I, size = nm, replace = False, p
    = weights_PM_norm)
85
86              for s in range(nm):
87                  SM[s] = PM[estrazioni[s]]
88                  weights[s] = weights_PM[estrazioni[s]]
89                  sl = PL[estrazioni[s]]
90                  SL.append(sl)
91
92              f = np.ones([len(instances), nm])
93
94              #Si calcola il vettore di valori attesi per ogni istanza
95              #di interesse
96              for i in range(len(instances)):
97                  ff = []
98                  temp = np.tile(X[i], (len(X_train), 1))
99
100                 for l in SL:
101                     mat = np.array(X_train)
102                     mat[:, l] = temp[:, l]
103                     val = list(model.predict_proba(mat)[:,1])
104                     valS = np.array(val)
105                     valS = valS.mean()
106                     ff.append(valS)
107
108                 f[i,:] = ff
109
110             phi_matrix = np.ones([len(X), len(features)])
111
112             for j in range(len(features)):
113
```

```
                Z = np.ones([nm, 1])
                wj = np.ones([nm, 1])
                K = len(features0) - len(features[j]) + 1

                for n in range(nm):

                    #Si assegna un kernel weight a ogni coalizione
                    #che presenta tutte o nessuna  delle feature
                    #del gruppo j-esimo
                    if SM[n, features[j]].sum() == len(features[j]):
                        Z[n] = 1
                        omega = SM[n].sum() - len(features[j]) + 1
                        wj[n] = (K - 1)/(comb(K, omega) * omega * (K -
omega))

                    elif SM[n, features[j]].sum() == 0:
                        Z[n] = 0
                        wj[n] = (K - 1)/(comb(K, SM[n].sum()) * (SM[n].sum
()) * (K - SM[n].sum()))

                    #Si assegna peso 0 alle coalizioni che presentano
                    #solo alcune delle feature del j-esimo gruppo
                    else:
                        Z[n] = 0
                        wj[n] = 0

                    for m in range(len(wj)):
                        if wj[m] == float('inf'):
                            wj[m] = 10**6

                #Si calcola la matrice S che non dipende dalla specifica
                #istanza ma dipende dal gruppo j-esimo
                ZJ = np.concatenate((np.ones([nm, 1]), SM[:, 0:features[j
][0]], Z, SM[:, features[j][-1] + 1:len(features0)]), axis=1)
                W = wj * np.identity(nm)
                S = np.linalg.inv(ZJ.T @ W @ ZJ) @ ZJ.T @ W

                #Si calcola il kernel generalized SHAP per il solo gruppo
                #j-esimo ma relativamente ad ogni istanza
                phi = S @ f.T
                pos = features[j][0] + 1
                phi_matrix[:, j] = phi[pos, :]

            phi_array[:,:,r] = phi_matrix

        end = time.time()
        time_cons = end - start
```

```
        tempo_medio_tot = round(time_cons/R,8)
        tempo_medio_ind = round(tempo_medio_tot/len(X),8)

        results = {'shap_array': phi_array, 'time_ind': tempo_medio_ind, '
    time_set': tempo_medio_tot}
        return results

    else:
        print('"nm" supera il valore massimo. Valore massimo di "nm":',
    nm_max)
```

k-additive SHAP per singole *feature*

```
def kadd_shap(X, X_train, model, k, nm, R):
    """
    Calcola i kadd SHAP per un vettore di istanze utilizzando 'nm'
    coalizioni estratte dal power set di M. Ripete la procedura per R
    differenti estrazioni di uguale dimensione 'nm'. Misura anche il
    tempo medio necessario per fare questi calcoli.
    ----------
    PARAMETERS:
    X = design matrix delle istanze oggetto di interpretazione.
    X_train  = design matrix delle istanze usate per addestrare
    il modello.
    model = modello da interpretare.
    k = grado di additivita'.
    nm = numero di coalizioni utilizzate nel calcolo.
    R = numero di ripetizioni dell'estrazione.
    ----------
    RETURN:
    shap_array = array contenente una matrice di kadd SHAP per ogni
    estrazione r-esima. Una singola matrice contiene i kadd SHAP per ogni
    feature, per ogni interazione di grado minore o uguale a k e per ogni
    istanza.
    time_ind = tempo mediamente necessario per calcolare i kadd SHAP per
    una singola istanza.
    time_set = tempo mediamente necessario per calcolare una matrice.
    """
    features = X.columns
    instances = range(len(X))
    X = np.array(X)

    #Si costruiscono le 2^m possibili coalizioni di feature
    PL = []
    for j in range(len(features) + 1):
        for element in itertools.combinations(range(len(features)), j):
            PL.append(list(element))
```

```
35
36      #Si individuano le coalizioni con cardinalita' inferiore a k
37      DL = []
38      for el in PL:
39          if len(el) <= k:
40              DL.append(el)
41
42      n_par = len(DL)
43      bern = bernoulli(k)
44
45      #Si trasformano le coalizioni in vettori binari
46      PM = np.zeros([2**len(features), len(features)])
47      for l in range(2**len(features)):
48          PM[l, PL[l]] = 1
49
50      DM = np.zeros([n_par, len(features)])
51      for ll in range(n_par):
52          DM[ll, DL[ll]] = 1
53
54      #Si calcolano i kernel weight e li si normalizzano in modo da usarli
55      #per effettuare le estrazioni
56      weights_PM = []
57      for m in range(2**len(features)):
58          w = (len(features) - 1)/(comb(len(features), PM[m].sum())*PM[m].sum
        ()*(len(features) - PM[m].sum()))
59          weights_PM.append(w)
60      weights_PM[0] = 10**6
61      weights_PM[-1] = 10**6
62      weights_PM_norm = weights_PM/np.array(weights_PM).sum()
63
64      #Si assegnano i pesi alle varie coalizioni
65      weights_k = np.ones(2**len(features))
66      weights_k[0] = 10**6
67      weights_k[-1] = 10**6
68
69      nm_max = len(PL)
70
71      if nm <= nm_max:
72          phi_array = np.ones([len(X), n_par, R])
73
74          start = time.time()
75          for r in range(R):
76              print('simulation:', r)
77
78              #Si estraggono 'nm' coalizioni
79              SM = np.ones([nm, len(features)])
80              SL = []
```

```
            weights = np.ones([nm])
            I = range(2**len(features))
            np.random.seed(r)
            estrazioni = np.random.choice(I, size = nm, replace = False, p
    = weights_PM_norm)

            for s in range(nm):
                SM[s] = PM[estrazioni[s]]
                weights[s] = weights_k[estrazioni[s]]
                sl = PL[estrazioni[s]]
                SL.append(sl)

            #Si costruisce la transformation matrix
            T = np.ones([nm, n_par])
            for m in range(nm):
                for n in range(n_par):
                    Dcard = int(sum(DM[n,:]))
                    ADcard = int(sum(DM[n,:]*SM[m,:]))
                    gamma = 0
                    for lll in range(ADcard + 1):
                        gamma += comb(ADcard, lll) * bern[Dcard - lll]
                    T[m,n] = gamma

            W = np.diag(weights)
            #Si calcola la matrice S che non dipende
            #dalla specifica istanza
            S = np.linalg.inv(T.T @ W @ T) @ T.T @ W

            phi_matrix = np.ones([len(X), n_par])

            for i in range(len(instances)):

                #Si costruisce il vettore f di valori attesi per
                #l'i-esima istanza
                f = []
                temp = np.tile(X[i], (len(X_train), 1))
                phi0 = model.predict_proba(X_train)[:,1].mean()

                for j in SL:
                    mat = np.array(X_train)
                    mat[:, j] = temp[:, j]
                    val = list(model.predict_proba(mat)[:,1])
                    valS = np.array(val)
                    valS = valS.mean()
                    f.append(valS)

                f_bar = f - phi0
```

```
                #Si calcola il kadd SHAP per ogni feature e relativamente
                #all'i-esima istanza
                phi = S @ f_bar
                phi_matrix[i] = phi

            phi_array[:,:,r] = phi_matrix

        end = time.time()
        time_cons = end - start
        tempo_medio_tot = round(time_cons/R,8)
        tempo_medio_ind = round(tempo_medio_tot/len(X),8)

        results = {'shap_array': phi_array, 'time_ind': tempo_medio_ind, '
    time_set': tempo_medio_tot}
        return results

    else:
        print('"nm" supera il valore massimo. Valore massimo di "nm":',
    nm_max)
```

k-additive Group SHAP

```
def kadd_group_shap(X, X_train, model, k, groups, nm, R):
    """
    Calcola i kadd Group SHAP per un vettore di istanze utilizzando
    'nm' coalizioni estratte dal power set di M. Ripete la procedura
    per R differenti estrazioni di uguale dimensione 'nm'. Misura
    anche il tempo medio necessario per fare questi calcoli.
    ----------
    PARAMETERS:
    X = design matrix delle istanze oggetto di interpretazione.
    X_train  = design matrix delle istanze usate per addestrare
    il modello.
    model = modello da interpretare.
    groups = lista dei gruppi da analizzare.
    k = grado di additivita'.
    nm = numero di coalizioni utilizzate nel calcolo.
    R = numero di ripetizioni dell'estrazione.
    ----------
    RETURN:
    shap_array = array contenente una matrice di kadd Group SHAP
    per ogni estrazione r-esima. Una singola matrice contiene i kadd Group
    SHAP per ogni gruppo di feature, per ogni interazione fino al grado
    k-esimo e per ogni istanza.
    time_ind = tempo mediamente necessario per calcolare i kadd
    Group SHAP per una singola istanza.
    time_set = tempo mediamente necessario per calcolare una matrice.
```

```
    """
    features = groups
    instances = range(len(X))
    X = np.array(X)

    #Si costruiscono le 2^g coalizioni di gruppi
    PL = []
    for j in range(len(features) + 1):
        for element in itertools.combinations(range(len(features)), j):
            PL.append(list(element))

    #Si individuano le coalizioni con cardinalita' inferiore a k
    DL = []
    for el in PL:
        if len(el) <= k:
            DL.append(el)

    n_par = len(DL)
    bern = bernoulli(k)

    #Si trasformano le coalizioni in vettori binari
    PM = np.zeros([2**len(features), len(features)])
    for l in range(2**len(features)):
        PM[l, PL[l]] = 1

    DM = np.zeros([n_par, len(features)])
    for ll in range(n_par):
        DM[ll, DL[ll]] = 1

    #Si costruiscono i kernel weight e li si normalizzano in modo da
    #utilizzarli per le estrazioni
    weights_PM = []
    for m in range(2**len(features)):
        w = (len(features) - 1)/(comb(len(features), PM[m].sum())*PM[m].sum
    ()*(len(features) - PM[m].sum()))
        weights_PM.append(w)
    weights_PM[0] = 10**6
    weights_PM[-1] = 10**6
    weights_PM_norm = weights_PM/np.array(weights_PM).sum()

    #Si assegnano i pesi alle varie coalizioni
    weights_k = np.ones(2**len(features))
    weights_k[0] = 10**6
    weights_k[-1] = 10**6

    nm_max = len(PL)

```

```
    if nm <= nm_max:

        phi_array = np.ones([len(X), n_par, R])

        start = time.time()
        for r in range(R):
            print('simulation:', r)

            #Si estraggono 'nm' coalizioni
            SM = np.ones([nm, len(features)])
            SL = []
            weights = np.ones([nm])
            I = range(2**len(features))
            np.random.seed(r)
            estrazioni = np.random.choice(I, size = nm, replace = False, p
= weights_PM_norm)

            for s in range(nm):
                SM[s] = PM[estrazioni[s]]
                weights[s] = weights_k[estrazioni[s]]
                sl = PL[estrazioni[s]]
                SL.append(sl)

            #Si calcola la transformation matrix
            T = np.ones([nm, n_par])
            for m in range(nm):
                for n in range(n_par):
                    Dcard = int(sum(DM[n,:]))
                    ADcard = int(sum(DM[n,:]*SM[m,:]))
                    gamma = 0
                    for lll in range(ADcard + 1):
                        gamma += comb(ADcard, lll) * bern[Dcard - lll]
                    T[m,n] = gamma

            W = np.diag(weights)
            #Si costruisce la matrice S che non dipende dalla
            #specifica istanza
            S = np.linalg.inv(T.T @ W @ T) @ T.T @ W

            #Si trasformano le coalizioni di gruppi in coalizioni
            #di singole feature
            combo = []
            for c in SL:
                lis = []
                for element in c:
                    for fea in features[element]:
                        lis.append(fea)
```

```
                combo.append(lis)

            phi_matrix = np.ones([len(X), n_par])

            for i in range(len(instances)):

                #Si calcola il vettore f di valori attesi per
                #l'istanza i-esima
                f = []
                temp = np.tile(X[i], (len(X_train), 1))
                phi0 = model.predict_proba(X_train)[:,1].mean()

                for j in combo:
                    mat = np.array(X_train)
                    mat[:, j] = temp[:, j]
                    val = list(model.predict_proba(mat)[:,1])
                    valS = np.array(val)
                    valS = valS.mean()
                    f.append(valS)

                f_bar = f - phi0
                #Si calcola il kadd SHAP per ogni gruppo di feature
                #relativamente all'istanza i-esima
                phi = S @ f_bar
                phi_matrix[i] = phi

            phi_array[:,:,r] = phi_matrix

        end = time.time()
        time_cons = end - start
        tempo_medio_tot = round(time_cons/R,8)
        tempo_medio_ind = round(tempo_medio_tot/len(X),8)

        results = {'shap_array': phi_array, 'time_ind': tempo_medio_ind, '
    time_set': tempo_medio_tot}
        return results

    else:
        print('"nm" supera il valore massimo. Valore massimo di "nm":',
    nm_max)
```

k-additive* SHAP *generalized value

```
def kadd_generalized_shap(X, X_train, model, k, groups, nm, R):
    """
    Calcola i kadd generalized SHAP per un vettore di istanze utilizzando
    'nm' coalizioni estratte dal power set di M. Ripete la procedura
```

```
    per R differenti estrazioni di uguale dimensione 'nm'. Misura
    anche il tempo medio necessario per fare questi calcoli.
    ----------
    PARAMETERS:
    X = design matrix delle istanze oggetto di interpretazione.
    X_train  = design matrix delle istanze usate per addestrare
    il modello.
    model = modello da interpretare.
    k = grado di additivita'
    groups = lista dei gruppi.
    nm = numero di coalizioni utilizzate nel calcolo.
    R = numero di ripetizioni dell'estrazione.
    ----------
    RETURN:
    shap_array = array contenente una matrice di kadd generalized
    SHAP per ogni estrazione r-esima. Una singola matrice contiene i
    kadd generalized SHAP per ogni gruppo di feature (ma non per le
    interazioni le quali vengono calcolate ma non raccolte) e per ogni
    #istanza.
    time_ind = tempo mediamente necessario per calcolare i kadd
    generalized SHAP per una singola istanza.
    time_set = tempo mediamente necessario per calcolare una matrice.
    """
    instances = range(len(X))
    features = groups
    features0 = [m for n in features for m in n]
    X = np.array(X)
    bern = bernoulli(k)

    #Si costruiscono le coalizioni ottenute fissando un gruppo e trattando
    #le altre variabili individualmente
    all_subsets = []
    for j in range(len(features)):
        exclusive = features.copy()
        del(exclusive[j])
        exclusive = [m for n in exclusive for m in n]
        for p in range(len(exclusive) + 1):
            for element in itertools.combinations(exclusive, p):
                all_subsets.append(list(element) + features[j])
                all_subsets.append(list(element))

    for el in all_subsets:
        el = el.sort()

    #Si eliminano le coalizioni doppione
    sub = []
    for el in all_subsets:
```

```
        if sub.count(el) < 1:
            sub.append(el)

    #Si trasformano le coalizioni in vettori binari
    PM = np.zeros([len(sub), len(features0)])
    for l in range(len(sub)):
        PM[l, sub[l]] = 1

    #Si calcolano i kernel weight e li si normalizzano in modo da
    #utilizzarli per l'estrazione delle coalizioni
    weights_PM = []
    for m in range(len(sub)):
        w = (len(features0) - 1)/(comb(len(features0), PM[m].sum())*PM[m].
    sum()*(len(features0) - PM[m].sum()))
        weights_PM.append(w)
    for m in range(len(weights_PM)):
        if weights_PM[m] == float('inf'):
            weights_PM[m] = 10**6
    weights_PM_norm = weights_PM/np.array(weights_PM).sum()

    nm_max = len(sub)

    if nm <= nm_max:

        phi_array = np.ones([len(X), len(features), R])

        start = time.time()
        for r in range(R):
            print('simulation:', r)

            #Si estraggono 'nm' coalizioni
            SM = np.ones([nm, len(features0)])
            SL = []
            I = range(len(sub))
            np.random.seed(r)
            estrazioni = np.random.choice(I, size = nm, replace = False, p
    = weights_PM_norm)

            for s in range(nm):
                SM[s] = PM[estrazioni[s]]
                sl = sub[estrazioni[s]]
                SL.append(sl)

            f = np.ones([len(instances), nm])
            phi0 = model.predict_proba(X_train)[:,1].mean()

            #Si calcola il vettore di valori attesi per ogni istanza
```

```
            for i in range(len(instances)):
                ff = []
                temp = np.tile(X[i], (len(X_train), 1))

                for l in SL:
                    mat = np.array(X_train)
                    mat[:, l] = temp[:, l]
                    val = list(model.predict_proba(mat)[:,1])
                    valS = np.array(val)
                    valS = valS.mean()
                    ff.append(valS)

                f[i,:] = ff

            f_bar = f - phi0

            phi_matrix = np.ones([len(X), len(features)])

            for j in range(len(features)):

                #Si individuano le sole coalizioni che contengono
                #tutte o nessuna delle feature relative al gruppo j-esimo
                SLJ = []
                for m in range(nm):

                    if SM[m, features[j]].sum() == len(features[j]):
                        SLJ.append(SL[m])

                    elif SM[m, features[j]].sum() == 0:
                        SLJ.append(SL[m])

                SMJ = np.zeros([len(SLJ), len(features0)])
                f_barJ = np.ones([len(instances), len(SLJ)])
                Z = np.ones([len(SLJ),1])
                for l in range(len(SLJ)):
                    SMJ[l, SLJ[l]] = 1
                    f_barJ[:,l] = f_bar[:, SL.index(SLJ[l])]
                    if SMJ[l, features[j]].sum() == 0:
                        Z[l] = 0

                #Si assegnano i pesi alle coalizioni
                weights_k = np.ones(len(SLJ))
                weights_k[SLJ.index(features0)] = 10**6
                weights_k[SLJ.index([])] = 10**6


                ZJ = np.concatenate((SMJ[:, 0:features[j][0]], Z, SMJ[:,
```

```
    features[j][-1] + 1:len(features0)]), axis=1)
144                 W = np.diag(weights_k)
145
146                 #Si individuano le coalizioni con cardinalita'
147                 #inferiore a k
148                 DL = []
149                 for d in range(k + 1):
150                     for element in itertools.combinations(range(np.shape(ZJ
    )[1]), d):
151                         DL.append(list(element))
152
153                 n_par_j = len(DL)
154
155                 #Si trasformano tali coalizioni in vettori binari
156                 DM = np.zeros([n_par_j, np.shape(ZJ)[1]])
157                 for ll in range(n_par_j):
158                     DM[ll, DL[ll]] = 1
159
160                 #Si costruisce la transformation matrix
161                 T = np.ones([len(SLJ), n_par_j])
162                 for m in range(len(SLJ)):
163                     for n in range(n_par_j):
164                         Dcard = int(sum(DM[n,:]))
165                         ADcard = int(sum(DM[n,:]*ZJ[m,:]))
166                         gamma = 0
167                         for lll in range(ADcard + 1):
168                             gamma += comb(ADcard, lll) * bern[Dcard - lll]
169                         T[m,n] = gamma
170
171                 #Si costruisce la matrice S che non dipende dalla
172                 #specifica istanza ma che dipende dal gruppo j-esimo
173                 S = np.linalg.inv(T.T @ W @ T) @ T.T @ W
174                 #Si calcola il kadd generalized SHAP per il solo gruppo
175                 #j-esimo e relativamente a tutte le istanze
176                 phi = S @ f_barJ.T
177                 pos = features[j][0] + 1
178                 phi_matrix[:, j] = phi[pos, :]
179
180             phi_array[:,:,r] = phi_matrix
181
182         end = time.time()
183         time_cons = end - start
184         tempo_medio_tot = round(time_cons/R,8)
185         tempo_medio_ind = round(tempo_medio_tot/len(X),8)
186
187
188         results = {'shap_array': phi_array, 'time_ind': tempo_medio_ind, '
```

```
    time_set': tempo_medio_tot}
189         return results
190
191     else:
192         print('"nm" supera il valore massimo. Valore massimo di "nm":',
    nm_max)
```

Riferimenti bibliografici

[1] AAS, K., JULLUM, M., AND LØLAND, A. Explaining individual predictions when features are dependent: More accurate approximations to shapley values. *Artificial Intelligence 298* (2021), 103502.

[2] AL-HASHEDI, K. G., AND MAGALINGAM, P. Financial fraud detection applying data mining techniques: A comprehensive review from 2009 to 2019. *Computer Science Review 40* (2021), 100402.

[3] ALGABA, E., FRAGNELLI, V., AND SÁNCHEZ-SORIANO, J. *Handbook of the Shapley value*. CRC Press, 2019.

[4] ALPAYDIN, E. *Introduction to machine learning*. MIT press, 2020.

[5] AUMANN, R. J. *Game Theory*. Palgrave Macmillan UK, London, 2016, pp. 1–40.

[6] BATISTA, G. E., PRATI, R. C., AND MONARD, M. C. A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD explorations newsletter 6*, 1 (2004), 20–29.

[7] BHATTACHARYYA, S., JHA, S., THARAKUNNEL, K., AND WESTLAND, J. C. Data mining for credit card fraud: A comparative study. *Decision support systems 50*, 3 (2011), 602–613.

[8] BILLINGSLEY, P. *Probability and measure*. John Wiley & Sons, 1995.

[9] BISHOP, C. *Pattern recognition and machine learning*. Springer, 2006.

[10] BOEHMKE, B., AND GREENWELL, B. M. *Hands-on machine learning with R*. CRC press, 2019.

[11] BRYANT, V. *Aspects of combinatorics: a wide-ranging introduction*. Cambridge university press, 1993.

[12] CARVALHO, D. V., PEREIRA, E. M., AND CARDOSO, J. S. Machine learning interpretability: A survey on methods and metrics. *Electronics 8*, 8 (2019), 832.

[13] CHAINALYSIS. The 2023 crypto crime report. Tech. rep., 2023.

[14] DAS, A., AND RAD, P. Opportunities and challenges in explainable artificial intelligence (xai): A survey. *arXiv preprint arXiv:2006.11371* (2020).

[15] DYCK, A., MORSE, A., AND ZINGALES, L. Who blows the whistle on corporate fraud? *The journal of finance 65*, 6 (2010), 2213–2253.

[16] EC. On artificial intelligence: A european approach to excellence and trust. *White Paper* (2020).

[17] ECB. Report on card fraud in 2020 and 2021. Tech. rep., 2023.

[18] FRYER, D., STRÜMKE, I., AND NGUYEN, H. Shapley values for feature selection: The good, the bad, and the axioms. *IEEE Access 9* (2021), 144352–144360.

[19] GIUDICI, P., AND RAFFINETTI, E. Shapley-lorenz explainable artificial intelligence. *Expert systems with applications 167* (2021), 114104.

[20] GOODFELLOW, I., BENGIO, Y., AND COURVILLE, A. *Deep learning*. MIT press, 2016.

[21] GRABISCH, M. *Set functions, games and capacities in decision making*, vol. 46. Springer, 2016.

[22] GRABISCH, M., AND ROUBENS, M. An axiomatic approach to the concept of interaction among players in cooperative games. *International Journal of game theory 28* (1999), 547–565.

[23] GUIDOTTI, R., MONREALE, A., RUGGIERI, S., TURINI, F., GIANNOTTI, F., AND PEDRESCHI, D. A survey of methods for explaining black box models. *ACM computing surveys (CSUR) 51*, 5 (2018), 1–42.

[24] GUNNING, D., STEFIK, M., CHOI, J., MILLER, T., STUMPF, S., AND YANG, G.-Z. Xai—explainable artificial intelligence. *Science robotics 4*, 37 (2019).

[25] HALMOS, P. R. *Naive set theory*. van Nostrand, 1960.

[26] JAMES, G., WITTEN, D., HASTIE, T., TIBSHIRANI, R., AND TAYLOR, J. *An introduction to statistical learning: With applications in python*. Springer Nature, 2023.

[27] JULLUM, M., REDELMEIER, A., AND AAS, K. groupshapley: Efficient prediction explanation with shapley values for feature groups. *arXiv preprint arXiv:2106.12228* (2021).

[28] KARASAN, A. *Machine Learning for Financial Risk Management with Python*. O'Reilly Media, Inc., 2021.

[29] KOHAVI, R. Glossary of terms. *Machine learning 30* (1998), 271–274.

[30] LIN, K., AND GAO, Y. Model interpretability of financial fraud detection by group shap. *Expert Systems with Applications 210* (2022), 118354.

[31] LUNDBERG, S. M., ERION, G. G., AND LEE, S.-I. Consistent individualized feature attribution for tree ensembles. *arXiv preprint arXiv:1802.03888* (2018).

[32] LUNDBERG, S. M., AND LEE, S.-I. A unified approach to interpreting model predictions. *Advances in neural information processing systems 30* (2017).

[33] MARICHAL, J.-L., KOJADINOVIC, I., AND FUJIMOTO, K. Axiomatic characterizations of generalized values. *Discrete Applied Mathematics 155*, 1 (2007), 26–43.

[34] MASCHLER, M., ZAMIR, S., AND SOLAN, E. *Game theory*. Cambridge University Press, 2020.

[35] MERRICK, L., AND TALY, A. The explanation game: Explaining machine learning models using shapley values. In *Machine Learning and Knowledge Extraction: 4th IFIP TC 5, TC 12, WG 8.4, WG 8.9, WG 12.9 International Cross-Domain Conference, CD-MAKE 2020, Dublin, Ireland, August 25–28, 2020, Proceedings 4* (2020), Springer, pp. 17–38.

[36] MURPHY, K. P. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[37] NASDAQ-VERAFIN. Global financial crime report. Tech. rep., 2024.

[38] PELEGRINA, G. D., DUARTE, L. T., AND GRABISCH, M. A k-additive choquet integral-based approach to approximate the shap values for local interpretability in machine learning. *Artificial Intelligence 325* (2023), 104014.

[39] RAVISANKAR, P., RAVI, V., RAO, G. R., AND BOSE, I. Detection of financial statement fraud and feature selection using data mining techniques. *Decision support systems 50*, 2 (2011), 491–500.

[40] REURINK, A. Financial fraud: A literature review. *Contemporary Topics in Finance: A Collection of Literature Surveys* (2019), 79–115.

[41] RIBEIRO, M. T., SINGH, S., AND GUESTRIN, C. "why should i trust you?" explaining the predictions of any classifier. *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining* (2016), 1135–1144.

[42] RICHHARIYA, P., AND SINGH, P. K. A survey on financial fraud detection methodologies. *International journal of computer applications 45*, 22 (2012), 15–22.

[43] ROTH, A. E. *The Shapley value: essays in honor of Lloyd S. Shapley*. Cambridge University Press, 1988.

[44] RUDIN, C. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature machine intelligence 1*, 5 (2019), 206–215.

[45] SHAPLEY, L. S. A value for n-person games. *Annals of Mathematics Studies 28* (1953), 307–317.

[46] SOKOL, K., AND FLACH, P. Interpretable representations in explainable ai: From theory to practice. *arXiv preprint arXiv:2008.07007* (2020).

[47] WEST, J., AND BHATTACHARYA, M. Intelligent financial fraud detection: a comprehensive review. *Computers & security 57* (2016), 47–66.

[48] YOUNG, H. P. Monotonic solutions of cooperative games. *International Journal of Game Theory 14*, 2 (1985), 65–72.