UNIVERSITÀ POLITECNICA DELLE MARCHE
FACOLTÀ DI ECONOMIA "GIORGIO FUÀ"

Corso di laurea Corso di Laurea Magistrale o Specialistica in Data Science

Per l'Economia e le Imprese

ANALISI DEI COMPORTAMENTI DEGLI STUDENTI PER PREDIRNE LE PERFORMANCE: Un approccio basato sul Process Mining

ANALYSIS OF STUDENT BEHAVIOR TO PREDICT THEIR PERFORMANCE: A Process Mining-based approach

TIPO TESI: (di ricerca)

Relatore:
PROF. POTENA DOMENICO

Studente:
GALEAZZI LORENZO

Anno Accademico 2023 - 2024

Sommario

CAPITOLO 1 INTRODUZIONE 11
1.1 INTRODUZIONE ALL'EDUCATIONAL PROCESS MINING 11
1.2 IL CASO STUDIO 12
1.3 PANORAMICA DEL DATASET 14
1.4 STRUTTURA DELLA TESI 15

CAPITOLO 2 : CENNI TEORICI 16
2.1 PROCESS MINING 16
2.2 RETE DI PETRI 19
2.2.1 Componenti 20
2.2.2 Funzionamento 21
2.2.3 Applicazioni 21
2.2.4 Proprietà 22
2.3 MODELLI DI PROCESSO 23
2.3.1 Inductive Miner 23
2.3.2 Heuristic Miner 25
2.4 CONFORMANCE CHECKING 26
2.4.1 Replayed fitness. Approccio token based 28
2.4.2 Aligned fitness. Approccio alignment-based 31
2.5 GRAFI DIRETTI 34
2.6 ALGORITMI PER LA CLASSIFICAZIONE 36
2.6.1 Support Vector Machine 36
2.6.2 Decision Tree 39
2.6.3 Random Forest 42
2.6.4 XGBoost 44

CAPITOLO 3 DATASET 47
3.1 DATASET 1 – Esami Superati 47
3.2 DATASET 2 – Esami con Stati 48
3.3 DATASET 3 – Dati anagrafici degli studenti 49

CAPITOLO 4 : METODOLOGIA 50
4.1 PRE-PROCESSING 50
4.2 CREAZIONE DELLE VARIABILI PER LA PREDIZIONE 62
4.3 MODELLI DI PROCESSO 67

4.4 CONFORMANCE CHECHING 70
4.5 CREAZIONE GRAFI PER LA RAPPRESENTAZIONE DEL COMPORTAMENTO DEGLI STUDENTI 72
4.5.1 Instance Graphs 72
4.5.2 Sottografi 76
4.6 CREAZIONE DEI CLASSIFICATORI E VISUALIZZAZIONE DEI RISULTATI 78
4.6.1 Pre-processing 78
4.6.2 Hyperarameter tuning 80
4.6.3 Misure per valutare l'importanza delle feature 84

CAPITOLO 5 MODELLI DI PROCESSO 86
5.1 PETRI NET OTTENUTE TRAMITE L'INDUCTIVE MINER 86
5.1.1 Studenti early 86
5.1.2 Studenti One year late 89
5.1.3 Studenti Late 91
5.2 HEURISTIC NET OTTENUTE TRAMITE L'HEURISTIC MINER 93
5.2.1 Studenti Early 93
5.2.2 Studenti One year late 95
5.2.3 Studenti Late 96

CAPITOLO 6 CONFORMANCE CHECKING 99
6.1 RETE DI PETRI 99
6.2 CALCOLO DI TOKEN-BASED E ALIGNMENTS-BASED FITNESS 100

CAPITOLO 7 CREAZIONE GRAFI 104
7.1 DALLE TRACCE AGLI INSTANCE GRAPHS 106
7.2 DAGLI INSTANCE GRAPHS AI SOTTOGRAFI 108

CAPITOLO 8 PREDIZIONI DELLE PERFORMANCE DEGLI STUDENTI 113
8.1 DISTRIBUZIONI DELLE VARIABILI NUMERICHE 114
8.2 MIGLIORI PARAMETRI, RISULTATI E MATRICI DI CONFUSIONE PER I MODELLI ADDESTRATI SENZA SUB 116
8.3 MIGLIORI PARAMETRI, RISULTATI E MATRICI DI CONFUSIONE PER I MODELLI ADDESTRATI UTILIZZANDO LE SUB 118

CAPITOLO 9 VALUTAZIONE DELL'UTILITA' DEI SOTTOGRAFI 121
9.1 COMPARAZIONE DEI RISULTATI DELLE PREDIZIONI 121
9.2 IMPORTANZA DELLE FEATURES 123

CAPITOLO 10 CONCLUSIONI E LAVORI FUTURI 130

CAPITOLO 11 BIBLIOGRAFIA 132

Elenco delle tabelle

Tabella 1 Esempio di Alignments n°1 32

Tabella 2 Esempio di Alignments n°2 32

Tabella 3 Esempio di Alignments n°3 32

Tabella 4 Esempio di Alignments n°3 33

Tabella 5 Parametro “k” e metriche relative ai modelli di processo degli studenti Early (Inductive miner)..... 87

Tabella 6 Parametro “k” e metriche relative ai modelli di processo degli studenti One year late (Inductive miner)..... 89

Tabella 7 Parametro “k” e metriche relative ai modelli di processo degli studenti Late (Inductive miner)..... 91

Tabella 8 Rete di Petri Inductive miner studenti Late. k = 0.7 92

Tabella 9 Parametro “k” e metriche relative ai modelli di processo degli studenti Early (Heuristic miner) 93

Tabella 10 Parametro “k” e metriche relative ai modelli di processo degli studenti One year late (Heuristic miner) 95

Tabella 11 Parametro “k” e metriche relative ai modelli di processo degli studenti Late (Heuristic miner) 96

Tabella 12 Numero di varianti per numero di studenti 104

Tabella 13 Classifica SUBs per numero di occorrenze tra gli studenti..... 110

Tabella 14 Parametri scelti per i modelli senza SUB..... 116

Tabella 15 Accuracy dei modellli senza SUB..... 118

Tabella 16 Parametri scelti per i modelli utilizzando le SUB 118

Tabella 17 Accuracy dei modellli utilizzando le SUB..... 119

Elenco delle figure

Figura 2.1 Mappa concettuale sulla disciplina del Process Mining 18

Figura 2.2 Esempio di Rete di Petri con le sue componenti 20

Figura 2.3 Esempio di Process Tree contenente l'operatore sequenza e parallelismo 24

Figura 2.4 Traduzione del Process Tree in Petri Net 24

Figura 2.5 Diagnostiche nel Conformance Checking 26

Figura 2.6 Modello di Processo d' Esempio n°1 28

Figura 2.7 Modello di Processo d' Esempio n°2 28

Figura 2.8 Modello di Processo d' Esempio n°3 29

Figura 2.9 Modello di Processo d' Esempio n°4 29

Figura 2.10 Modello di Processo d' Esempio n°5 30

Figura 2.11 Modello di Processo d' Esempio n°6 30

Figura 2.12 Modello di processo d'esempio per l'aligned fitness 31

Figura 2.13 Esempio di grafo diretto 35

Figura 2.14 Esempio di iperpiano e margine separatore 37

Figura 2.15 Struttura dell'albero decisionale 40

Figura 2.16 Esempio del funzionamento del Random Forest 43

Figura 2.17 Esempio del funzionamento dell'XGBoost 45

Figura 4.1 Esempio del processo di creazione degli Instance Graphs 74

Figura 5.1 Rete di Petri Inductive miner studenti Early. k = 0.8 88

Figura 5.2 Rete di Petri Inductive miner studenti One year late. k = 0.7 90

Figura 5.3 Heuristic Net studenti Early. k = 0.92 94

Figura 5.4 Heuristic Net studenti One year late. k = 0.96 95

Figura 5.5 Heuristic Net studenti Late. k = 0.92/0.94 97
Figura 6.1 Rete di Petri standard per il Conformance Checking 100
Figura 6.2 Valore minimo e massimo per la replayed fitness 101
Figura 6.3 Valore minimo e massimo per l'aligned fitness 101
Figura 6.4 Distribuzioni tramite BoxPlot delle replayed fitness per le tre categorie di studenti 102
Figura 6.5 Distribuzioni tramite BoxPlot delle aligned fitness per le tre categorie di studenti 102
Figura 7.1 Top 9 Varianti per numero di studenti 105
Figura 7.2 Esempio di Instance Graph per uno specifico studente 107
Figura 7.3 Esempio di sottografi in formato testuale 109
Figura 7.4 SUB 2 111
Figura 7.5 SUB 3 111
Figura 7.6 SUB 22 111
Figura 7.7 SUB 84 112
Figura 8.1 Distribuzione di frequenza delle variabili rispetto alle classi Early e Late (gruppo 1) 114
Figura 8.2 Distribuzione di frequenza delle variabili rispetto alle classi Early e Late (gruppo 2) 115
Figura 8.3 Confusion Matrix SVM senza SUB 117
Figura 8.4 Confusion Matrix DT senza SUB 117
Figura 8.5 Confusion Matrix RF senza SUB 117
Figura 8.6 Confusion Matrix XGB senza SUB 117
Figura 8.7 Confusion Matrix SVM utilizzando le SUB 119
Figura 8.8 Confusion Matrix DT utilizzando le SUB 119
Figura 8.9 Confusion Matrix RF utilizzando le SUB 119

Figura 8.10 Confusion Matrix XGB utilizzando le SUB 119
Figura 9.1 Confronto dei livelli di accuracy delle predizioni dei modelli addestrati con e senza l'utilizzo delle SUB 121
Figura 9.2 Importanza relativa delle feature rispetto all'accuracy totale ottenuta con l'algoritmo XGBoost utilizzando l'intero set di variabili 124
Figura 9.3 Importanza cumulativa delle feature rispetto all'accuracy totale ottenuta con l'algoritmo XGBoost utilizzando l'intero set di variabili 124
Figura 9.4 SUB 3 125
Figura 9.5 SUB 39 125
Figura 9.6 SUB 16 125
Figura 9.7 SUB 7 126
Figura 9.8 SUB 13 126
Figura 9.9 Importanza delle feature basate su valori SHAP nel predire le performance degli studenti 127
Figura 9.10 SUB 70 128
Figura 9.11 SUB 98 128

Capitolo 1 INTRODUZIONE

1.1 INTRODUZIONE ALL'EDUCATIONAL PROCESS MINING

Nel contesto educazionale universitario, l'analisi dei percorsi degli studenti sta assumendo un'importanza sempre maggiore. Le università si stanno impegnando a migliorare costantemente i loro processi educativi per favorire il successo accademico e il benessere degli studenti. Un approccio promettente per raggiungere questi scopi è rappresentato dall'Educational Data Mining (EDM), una branca dell'analisi dati che si focalizza sull'estrazione di conoscenze dai dati educativi. L'EDM è utile per scoprire pattern, estrarre conoscenze utili e migliorare i processi educativi attraverso l'applicazione di tecniche di data mining. All'interno di questa disciplina l'Educational Process Mining (EPM) emerge con una metodologia potente per comprendere e ottimizzare i percorsi educativi. La forza dell'EPM risiede nella sua capacità di considerare dati sequenziali e procedurali, applicando tecniche di Process Mining per analizzare e migliorare i processi di apprendimento. Questo approccio permette di modellare, analizzare e migliorare i processi attraverso l'estrazione di informazioni dai dati raccolti sulle attività degli studenti, come registri di accesso ai sistemi di gestione dell'apprendimento, registri degli esami sostenuti dagli studenti e interazioni con piattaforme messe a disposizione dagli specifici istituti.

I motivi dell'utilizzo dell'EPM sono molteplici: ottimizzazione dei processi di apprendimento, personalizzazione dell'istruzione e miglioramento delle qualità

dell'insegnamento. Con questo approccio è possibile individuare i punti critici nei percorsi degli studenti e le deviazioni dai percorsi ottimali, in modo tale da permettere alle istituzioni educative di intervenire tempestivamente per supportare gli individui più in difficoltà. Analizzando il comportamento degli studenti è possibile personalizzare l'esperienza educativa, identificando le esigenze di coloro che si trovano in situazioni più problematiche. Intervenire per ridurre i tempi di laurea e diminuire il tasso di abbandono può avere benefici significativi sui singoli studenti, che vedranno aumentare le opportunità di carriera e diminuire i costi associati all'istruzione superiore, ma anche sull'istituto accademico migliorando la reputazione dell'università e favorendo maggiori finanziamenti e supporto. Infine, anche sulla società nel suo complesso attraverso l'aumento del capitale umano e diminuzione della disoccupazione.

1.2 IL CASO STUDIO

Alla luce dei potenziali benefici apportati dalla disciplina dell'Educational Process Mining, introduciamo il nostro caso di studio. Come menzionato nell'introduzione, l'EPM ha l'obiettivo di analizzare i percorsi degli studenti e identificare coloro che incontrano maggiori difficoltà nel conseguimento del titolo, con l'intento di proporre soluzioni personalizzate ed efficaci. Tali soluzioni mirano a ridurre i tempi di laurea e a diminuire i tassi di abbandono. Nel contesto di questo caso studio, ci proponiamo di analizzare, attraverso l'applicazione di tecniche di EPM e Machine Learning, i percorsi degli studenti nel loro primo anno accademico e verificare come i loro processi comportamentali influenzino le performance accademiche complessive.

Il presente studio si basa su un insieme di ex studenti laureati in Ingegneria Informatica e dell'Automazione, corso erogato dalla Facoltà di Ingegneria dell'Università Politecnica delle Marche. L'obiettivo della tesi è utilizzare tecniche atte a modellare i processi comportamentali degli studenti, al fine di identificare pattern e ricorrenze significative, valutando il loro impatto sul successo accademico. Gli studenti analizzati hanno frequentato gli anni accademici dal 2010/2011 al 2023/2024. I dati impiegati sono stati estratti dal sistema Esse3, ampiamente utilizzato dalle università italiane per la gestione della didattica. L'analisi si concentrerà sui comportamenti di tre categorie di studenti, classificati in base ai tempi di conseguimento del titolo: "Early", "One year late" e "Late". Nello specifico:

- Early: studenti che hanno completato il percorso di studi in meno di 3 anni e 6 mesi;
- One year late: studenti che hanno impiegato più di 3 anni e 6 mesi, ma meno di 4 anni e 6 mesi;
- Late: studenti che hanno completato il percorso in un tempo superiore ai 4 anni e 6 mesi;

Inizialmente, verranno analizzate le similarità e le dissimilarità tra queste categorie utilizzando tecniche di Process Mining. Successivamente, si procederà alla creazione di due tipi di variabili: le prime sono variabili continue che rappresentano sinteticamente le caratteristiche del percorso dello studente, le seconde sono utilizzare per rappresentare e modellare il comportamento dello studente nel sostenere gli esami, utilizzando solo le informazioni del primo anno accademico. Dopo una serie di analisi volte ad esplorare il legame trai comportamenti degli individui nel primo anno accademico e le loro performance complessive, verranno

addestrati diversi modelli di machine learning per predire le performance degli studenti, utilizzando le variabili appena create, con l'obiettivo di valutarne l'utilità.

1.3 PANORAMICA DEL DATASET

A partire da tre file csv abbiamo ottenuto i tre dataset cardine, sui quali abbiamo basato la nostra analisi. I file di partenza contenevano le seguenti informazioni:

- File 1: dati relativi agli esami superati. Ogni riga del file csv corrisponde ad un esame superato.
- File 2: a differenza del precedente ogni riga corrisponde ad un esame con il rispettivo stato. Non si tratta solo degli esami superati, bensì di tutte le attività riguardanti un certo esame per ogni studente.
- File 3: ogni riga corrisponde ad uno studente. Per ogni studente sono indicate delle informazioni anagrafiche quali l'identificativo univoco opportunamente anonimizzato, il sesso e il voto di diploma insieme ad altre informazioni.

A partire da questi tre file, abbiamo creato i tre dataset utilizzati per le analisi, che fanno riferimento a un totale di 707 studenti. Il primo dataset, chiamato "df_esami", include le informazioni relative agli esami superati, così come riportate nel primo file. Questo dataset fornisce una visione meno dettagliata del percorso degli studenti, poiché si limita agli esami che sono stati completati con successo.

Il secondo dataset, denominato "df_merged", è stato ottenuto tramite una join tra i tre file di partenza. L'implementazione dettagliata di questa operazione verrà descritta nel paragrafo 3.2, dedicato al pre-processing dei dati.

Il terzo, chiamato “df_anagrafiche”, contiene le informazioni del terzo file, ma con l’aggiunta di altre variabili costruite nel corso delle analisi. È il dataset utilizzato per l’induzione dei modelli di machine learning.

I tre dataset sono stati utilizzati congiuntamente nelle diverse analisi condotte nel corso di questa tesi.

1.4 STRUTTURA DELLA TESI

La tesi sarà organizzata nel seguente modo. A partire dal capitolo 2, verrà spiegato il funzionamento delle tecniche utilizzate in questa ricerca, fornendo una panoramica teorica sui metodi impiegati. Successivamente, nel capitolo 3, si procederà con la descrizione della metodologia adottata per ottenere i risultati, illustrando i passaggi chiave del processo di analisi. Il capitolo 4 sarà dedicato all’analisi dei dataset utilizzati, approfondendo la loro struttura e le caratteristiche principali. Nel capitolo 5 verranno esaminati i modelli di processo analizzati, spiegando come sono stati costruiti e interpretati. L'analisi dei risultati ottenuti attraverso le tecniche di Conformance Checking sarà presentata nel capitolo 6. Il cuore della ricerca sarà affrontato nei capitoli 7 e 8, dove verranno mostrate rispettivamente le tecniche impiegate per modellare il comportamento degli studenti e i risultati relativi alla predizione, ottenuti sia con che senza l’utilizzo dei dei comportamenti più frequenti tra gli studenti. L'utilità di queste variabili comportamentali sarà discussa nel capitolo 8, mentre il capitolo 9 concluderà il lavoro, presentando le conclusioni tratte riguardo l'intero progetto.

Capitolo 2: CENNI TEORICI

In questo capitolo sono stati illustrati i concetti teorici fondamentali alla base delle analisi svolte. Dopo una breve introduzione generale alla disciplina de Process Mining, si è approfondito l'uso della Petri net, uno degli strumenti più diffusi per la rappresentazione di modelli di processo. Successivamente, sono stati esaminati i principali algoritmi utilizzati per le analisi e la creazione dei modelli di processo, in particolare l'algoritmo Inductive ed Heuristic Miner. È stato inoltre introdotto il concetto di Conformance Checking, che consente di verificare la corrispondenza tra il modello teorico e il comportamento reale osservato nei dati. Infine, sono stati discussi i grafi diretti e le loro caratteristiche, con una panoramica sul loro utilizzo nell'analisi dei processi. A conclusione del capitolo, è stato fornito un approfondimento sul funzionamento degli algoritmi di Machine Learning impiegati nel corso delle analisi.

2.1 PROCESS MINING

Il Process Mining è una disciplina che si occupa dell'analisi dei processi. I processi sono delle successioni di eventi, esattamente come sono registrati nei vari sistemi informativi. Una delle principali sfide per le organizzazioni moderne è quella di estrarre informazioni rilevanti dai dati archiviati nei loro sistemi informativi. Il Process Mining si concentra sulle tracce lasciate dall'esecuzione di questi processi in

tali sistemi, aggiungendo una prospettiva di processo al Machine Learning e al Data Mining. Questa disciplina integra la gestione dei processi con metodi tradizionali e con la Data Science: nel primo caso, utilizzando metodi per scoprire modelli di processo direttamente dai dati, e nel secondo, fornendo una dimensione procedurale e sequenziale all'analisi dei dati tradizionali, che solitamente è statica. Inoltre, funge da collegamento tra queste aree, permettendo analisi complete e dettagliate dei processi. Attraverso tecniche come il Process Discovery, il Conformance Checking e l'Enhancement, il Process Mining consente di ottenere una visione chiara e approfondita delle dinamiche di processo. L'obiettivo di questa disciplina è estrarre valore dall'evoluzione degli eventi rappresentati dai dati nei sistemi informativi, comprendendo la dinamica dei processi coinvolti. Questo permette di identificare colli di bottiglia, fasi che richiedono più tempo e aree di miglioramento all'interno dei processi. Il Process Mining può ottimizzare processi come la gestione degli ordini all'interno di un'azienda, processi nel settore sanitario, come il trattamento dei pazienti e i loro percorsi di cura, oppure, come nel nostro caso, processi relativi al comportamento umano. Le principali tecniche analitiche del Process Mining includono il Process Discovery, il Conformance Checking e l'Enhancement.

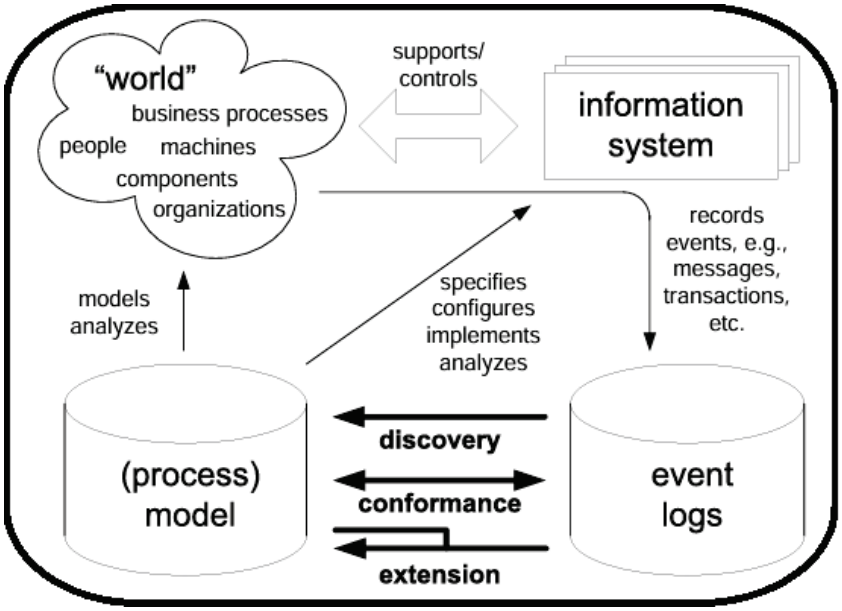


Figura 2.1 Mappa concettuale sulla disciplina del Process Mining

- Process Discovery: mira ad estrarre un modello di processo dai dati senza conoscenze preliminari. Attraverso questa tecnica è possibile prendere in input un event log e ottenere un modello che spiega i comportamenti catturati utilizzando diversi tipi di visualizzazioni e linguaggi di modellazione matematici.

- Conformace Checking: confronta l'esecuzione reale di un processo con un modello ideale di processo. Il modello ideale rappresenta come il processo dovrebbe essere eseguito in linea teorica.

- Enhancement: migliora un modello di processo esistente utilizzando le informazioni registrate nel log. Due sono i principali tipi di Enhancement, il "Repair" che modifica il modello per riflettere meglio la realtà e l'"Extension" che aggiunge nuove parti al modello.

Ogni analisi eseguita nel contesto del Process Mining ha come punto di partenza un event log, noto anche come registro degli eventi. Questo può essere rappresentato come un semplice dataframe, ad esempio un file csv, o in formati più specifici, come un file xes. Ogni riga in un event log rappresenta un evento, e ogni evento è caratterizzato da tre elementi fondamentali: la data dell'evento, il nome dell'attività svolta e un identificatore di caso, ovvero un codice o nome che identifica l'insieme degli eventi relativi allo stesso caso, oltre a un identificatore di evento che individua la singola riga. Ad esempio, se consideriamo l'event log 'df_esami' che vogliamo analizzare, l'identificatore di caso è il codice identificativo dello studente, l'attività è l'esame superato, e la data è quella del superamento dell'esame. Per quanto riguarda il dataset 'df_merged', l'attività è l'unione dell'esame e dello stato dell'esame, mentre l'informazione temporale corrisponde alla data dell'appello a cui si riferisce l'attività; l'identificatore di caso rimane lo stesso. La sequenza di eventi appartenenti allo stesso caso rappresenta una Traccia.

2.2 RETE DI PETRI

Le reti di Petri rappresentano un linguaggio di modellazione matematico usato per la descrizione e l'analisi di sistemi che sono caratterizzati da processi concorrenti, asincroni, distribuiti, paralleli non deterministici e stocastici.[1] Sono stati sviluppati da Carl Adam Petri nel 1962 come strumento attraverso il quale rappresentare sistemi che comprendono al loro interno processi concorrenti, in quanto i modelli sequenziali erano inadeguati a catturare e rappresentare la complessità di attività parallele.

[1] Petri nets: Properties, analysis and applications. T. Murata

2.2.1 *Componenti*

Gli elementi costitutivi di una rete di Petri sono i seguenti:

- Places (P): rappresentano le condizioni o gli stati del sistema. Solitamente sono rappresentati da tondi.
- Transitions (T): rappresentano eventi o azioni che causano cambi di stato. Sono quasi sempre rappresentati da rettangoli.
- Tokens: rappresentano l'occorrenza di una condizione o evento. Solitamente sono punti all'interno dei Places.
- Arcs: mostrano il flusso di relazioni, collegando i Places alle transizioni o viceversa, attraverso una freccia direzionale.

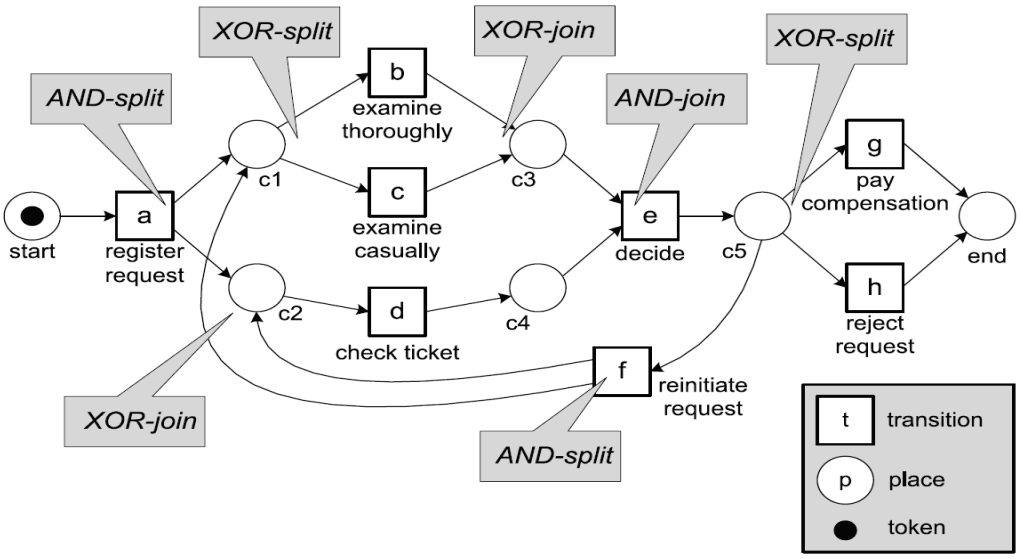


Figura 2.2 Esempio di Rete di Petri con le sue componenti

2.2.2 *Funzionamento*

- Stato iniziale: lo stato iniziale è definito dalla rete di Petri e dalla posizione dei token all'interno delle Places prima dell'esecuzione della prima transizione.
- Abilitazione della transizione: una transizione è abilitata se le Places in input a quella specifica transizione contengono il numero necessario di token.
- Attivazione della transizione: l'attivazione della transizione può avvenire solo se la transizione è abilitata. Quando la transizione è attivata i token nelle Places in input vengono consumati e vengono create token nelle Places in output. L'attivazione delle transizioni definisce un nuovo stato della Rete di Petri.

2.2.3 *Applicazioni*

Le Reti di Petri sono usate in diversi campi per modellare sistemi complessi di informazioni. Di seguito sono elencati alcuni dei molteplici ambiti di applicazione:

- System Modeling: Le Reti di Petri sono usate per modellare e analizzare il comportamento di sistemi concorrenti come protocolli di comunicazione e sistemi distribuiti.
- Process Analysis: Come nel nostro caso di studio, sono utilizzate nell'ambito del Process Mining per scoprire, monitorare e migliorare processi reali tramite l'estrazione di conoscenza dagli event log.

- Verificazione: le reti di Petri sono usate per verificare proprietà come la raggiungibilità (se un particolare stato delle cose può essere raggiunto), vitalità di un sistema (un certo sistema può continuare ad operare o raggiunge un limite?), limitazioni (assicura che un certo numero di token non superi un certo limite).
- Reti biologiche: le Reti di Petri possono modellare e analzzare comportamenti dinamici nei sistemi biologici come i percorsi metabolici.

2.2.4 *Proprietà*

- Reachability: proprietà che definisce se un certo stato della rete di Petri può essere o meno raggiunto.
- Boundedness: assicura che il numero di token non superino una soglia limite prevenendo il cosiddetto overflow.
- Liveness: assicura che ogni transizione è in grado di attivarsi prima o poi, evitando una situazione chiamata deadlock.
- Safeness: è una specifica forma di boundedness dove il limite di token per ogni place è uguale a 1.
- Conservativeness: il numero di token rimangono invariati a prescindere dalle transizioni attivate.

2.3 MODELLI DI PROCESSO

I modelli di processo sono una rappresentazione formale che descrive visivamente le attività e interazioni presenti all'interno di un processo, utilizzando le informazioni presenti in un event log. È uno strumento fondamentale per documentare, analizzare e ottimizzare processi in quanto permette di gestire la complessità e di comunicare la realtà. Tuttavia, comprendere le caratteristiche dei processi individualmente all'interno di un registro eventi può essere troppo complicato. Per facilitare questo compito, esistono algoritmi di discovery come l'Inductive miner e l'Heuristic miner, che generano automaticamente modelli di processo a partire dai dati reali presenti nell'event log. Questi modelli non solo permettono di comprendere meglio la totalità dei processi dato un event log, ma offrono anche la flessibilità di essere tradotti in diverse notazioni, come le reti di Petri, BPMN, BPEL o i Diagrammi di Attività UML. Ogni algoritmo ha i suoi punti di forza e di debolezza, e la loro scelta dipende dall'obiettivo specifico dell'analisi. Alcuni algoritmi per la creazione di modelli di processo verranno discussi nei paragrafi seguenti.

2.3.1 *Inductive Miner*

L'Inductive Miner è un algoritmo di Process Discovery che mira a generare un modello in grado di rappresentare accuratamente i comportamenti osservati nel log degli eventi, garantendo un modello strutturato e comprensibile. L'algoritmo opera in modo ricorsivo, scomponendo il log degli eventi in parti più piccole e costruendo un modello di processo a partire da queste parti, utilizzando una serie di operatori come sequenza, parallelismo, scelta e loop. L'idea centrale è identificare pattern all'interno del log degli eventi che possano essere rappresentati da questi operatori. Le possibili visualizzazioni includono rappresentazioni ad albero e reti di Petri. L'algoritmo

procede sempre inizialmente con la rappresentazione ad albero, per poi derivarne una rete di Petri. Per creare il Process Tree il log degli eventi viene suddiviso in sottoinsiemi dei quali si vogliono identificare le relazioni reciproche che intercorrono. Una volta identificate tutte le relazioni tra questi, si continua ricorsivamente con le suddivisioni fino a che il sottoinsieme non rappresenta un caso base. Con l'insieme delle relazioni trovate tra le varie parti del log degli eventi, viene creato il Process Tree.

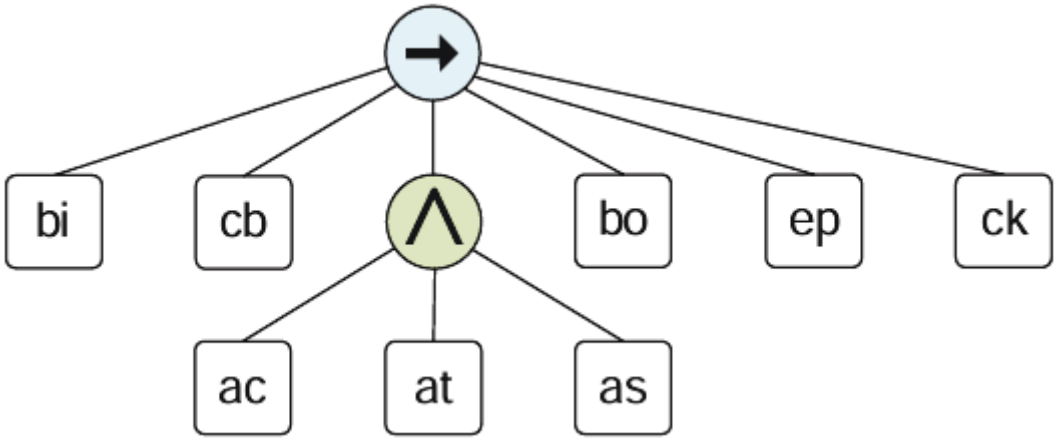


Figura 2.3 Esempio di Process Tree contenente l'operatore sequenza e parallelismo

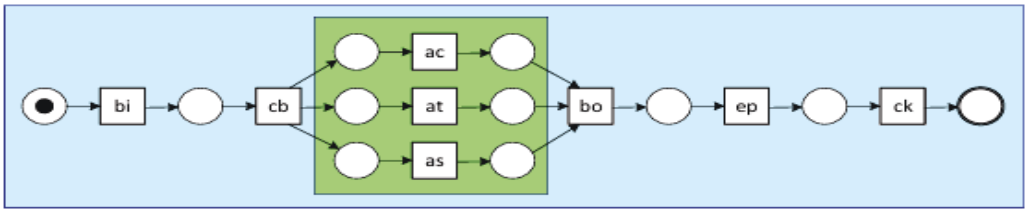


Figura 2.4 Traduzione del Process Tree in Petri Net

Nella Fig. 2.3 osserviamo che l'ordine gerarchico degli operatori è identificato dall'altezza in cui si trovano; quindi, consideriamo quello di sequenzialità per primo. La figura ci dice che l'evento "bi" viene sempre eseguito per primo e viene sempre seguito da "cb", per poi essere seguito da tre eventi paralleli tra loro. questa logica di funzionamento mappata dal Process Tree viene poi tradotta in Petri net come si vede

in Fig.2.4 L'Inductive miner inoltre è in grado di gestire il rumore, e pattern infrequenti assicurando al modello semplicità e comprensibilità.

2.3.2 *Heuristic Miner*

Il secondo algoritmo per la generazione di modelli di processo è l'Heuristic Miner. Questo algoritmo si focalizza sulla rappresentazione dei pattern più rilevanti, filtrando quelli meno frequenti. A differenza di altri modelli di processo, l'Heuristic Miner non si concentra esclusivamente sulle sequenze dirette tra eventi, ma utilizza un approccio euristico per gestire il rumore e scoprire modelli più precisi e accurati. Oltre all'utilizzo dell'event log, elementi fondamentali per questo algoritmo sono il concetto di Directly-Follows Graph (DFG), un grafo in cui i nodi rappresentano attività e gli archi diretti rappresentano relazioni sequenziali tra le attività, e i concetti di frequenza e dipendenza. Queste misure vengono utilizzate per determinare la forza delle relazioni tra le attività.

L'algoritmo inizia con l'estrazione delle relazioni dirette tra le attività dall'event log. Successivamente, viene calcolata la frequenza di ciascuna attività e delle relazioni estratte. Per ogni coppia di attività A e B, si calcola la misura di dipendenza (dependency measure) secondo la formula:

$$dep(A,B) = \frac{|A \to B| - |B \to A|}{|A \to B| + |B \to A| + 1}$$

Dopo aver calcolato queste misure, viene impostata una soglia (threshold) per filtrare le relazioni deboli o non statisticamente significative. Solo le relazioni che superano la soglia vengono incluse nel modello di processo finale. Questa tecnica di filtraggio

è utilizzata per ridurre la complessità del modello. Una volta mantenute le relazioni più rilevanti, si costruisce un modello di processo che prende la forma di un process tree, che può successivamente essere tradotto in una rete di Petri. Esistono diversi approcci, leggermente differenti tra loro, nell'utilizzo dell'Heuristic Miner; quello descritto è l'approccio standard.

2.4 CONFORMANCE CHECKING

Quando si parla di Conformance Checking nell'ambito del process mining, si fa riferimento al confronto tra un event log (ciò che è effettivamente avvenuto) e un modello di processo (ciò che dovrebbe succedere). Il log viene confrontato con il modello di processo, con l'obiettivo di determinare se i processi all'interno dell'event log rispettano le "regole" definite dal modello.

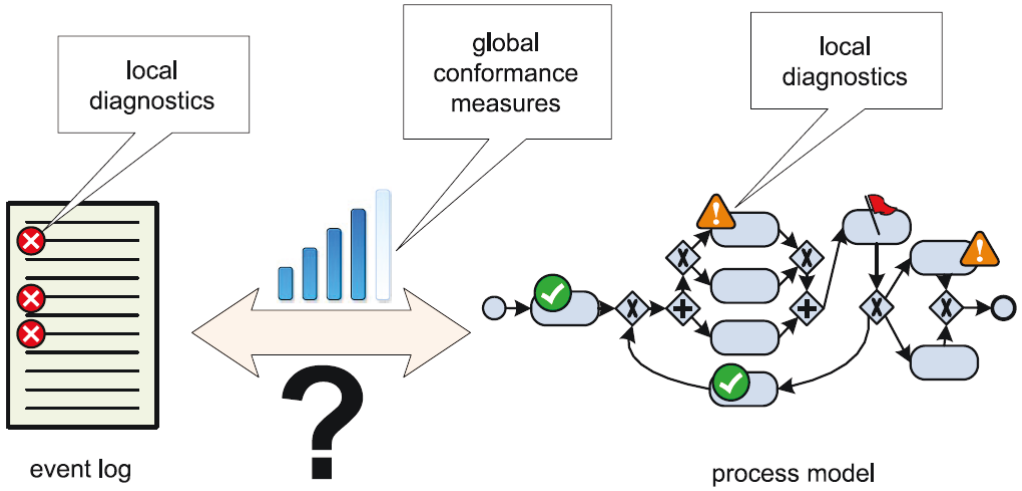


Figura 2.5 Diagnostiche nel Conformance Checking

Questo confronto permette di ottenere diagnostiche sia locali che globali. Le diagnostiche locali sull'event log mostrano le tracce che non aderiscono al modello di processo, mentre le diagnostiche locali sul modello evidenziano le parti del modello che sono presenti ma che non sono conformi a quello che succede nei processi

effettivi. Le diagnostiche globali, invece, forniscono una misura di conformità, o fitness, che, con un valore compreso tra 0 e 1, indica il livello di fitness generale dell'event log rispetto al modello di processo.

I motivi per effettuare il Conformance Checking possono essere molteplici. Uno, come già detto, è verificare che ciò che accade realmente sia conforme alle regole definite dal modello. Un altro motivo è capire quale sia il modello di processo migliore tra un certo numero di modelli disponibili. Nella nostra analisi verranno utilizzate due misure relative alla fitness dell'event log, quindi nel prossimo paragrafo introdurremo il concetto di token replay e allignment, che sono indispensabili per capire come sono state create. Prima di addentrarci nella spiegazione di questi concetti diamo una spiegazione generale del concetto di fitness nell'ambito del conformance checking.

Dato un event log L, una rete di petri N e un insieme di tracce $\sigma i = < \sigma 1, \sigma 2, \ldots, \sigma n >$ dove $\sigma = < a1, a2, \ldots, am >$ è possibile valutare il livello di fitness dell'event log rispetto al modello. L'idea è quella di vedere se le regole imposte dalla rete di Petri consentono l'esecuzione delle tracce presenti nell'event log. Un approccio "naive" sarebbe quello di contare il numero di tracce che possono essere riprodotte dal modello di processo. La fitness totale rappresenterebbe in questo caso la frazione di tracce che sono state riprodotte con successo sul totale. Questo approccio è però troppo drastico, prendiamo il caso di due event log, il primo contiene delle tracce per le quali un gran numero di attività non sono conformi alle regole definite dal modello, mentre nel secondo per ogni traccia abbiamo un solo evento che non è possibile riprodurre. In entrambi i casi otterremo una fitness pari a 0 nonostante il primo event log sia quello che differisce maggiormente dal modello. Per questo

motivo si usano delle definizioni di fitness definite a livello dell'evento invece che a livello dell'intera traccia. Esistono diverse nozioni di fitness in questo ambito, ma per i fini di questa tesi sarà necessario spiegarne solo due.

2.4.1 *Replayed fitness. Approccio token based*

Questo tipo di fitness è contruita a partire da quattro valori: token prodotti (p), token consumati (c), token mancanti (m), token rimanenti(r). Prendiamo la rete di Petri rappresentata in Fig. 2.6 e una traccia di cui vogliamo trovare la fitness rispetto alla rete di Petri: $\sigma = < a, d, c, e, h >$.

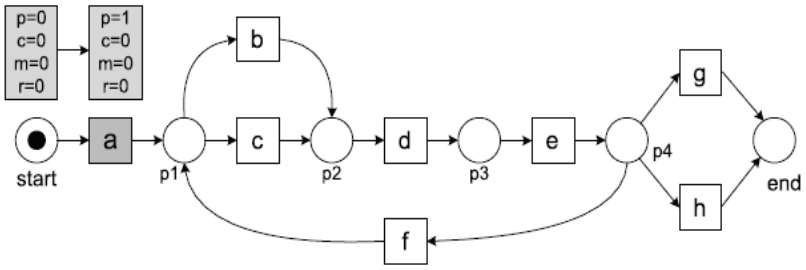


Figura 2.6 Modello di Processo d' Esempio n°1

Possiamo notare come nella place "start" sia stato inserito un token necessario per il replay che va ad incrementare il numero di token prodotti (p). In questo modo la transizione "a" è attivabile.

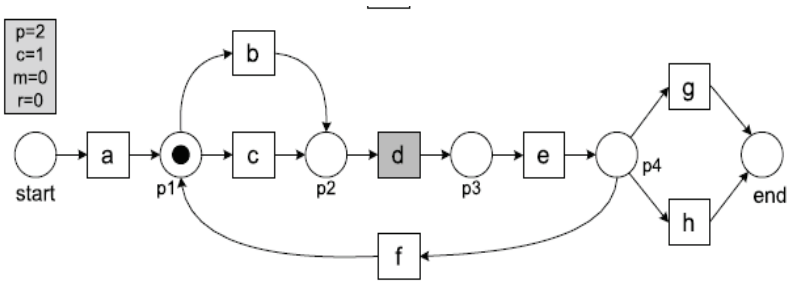


Figura 2.7 Modello di Processo d' Esempio n°2

Svolgendo l'attività "a" della traccia, viene attivata la transizione "a" della rete di Petri. Un token viene consumato, quindi ne viene prodotto un altro. Avendo attivato una transizione i token prodotti (p) diventano 2, mentre i token consumati (c) 1.

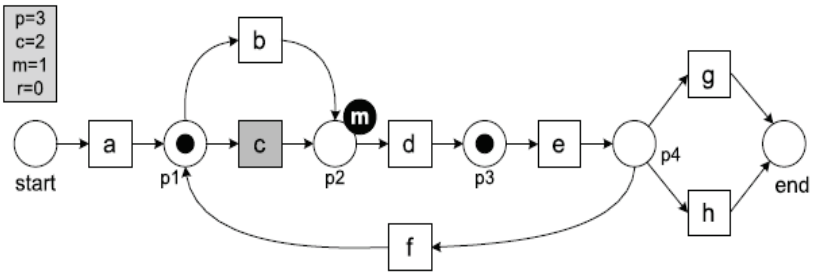


Figura 2.8 Modello di Processo d' Esempio n°3

La prima discrepanza tra la traccia e la rete di Petri si verifica in Fig 2.8 poiché la transizione "d" necessita di un token in "p2", che non è presente nella figura. Dobbiamo quindi produrre un token in "p2", consumarlo, e poi produrne un altro in "p3". Questa operazione aumenta di 2 il valore dei token prodotti (p), di 1 il valore dei token consumati, e incrementa di 1 il valore dei token mancanti (m).

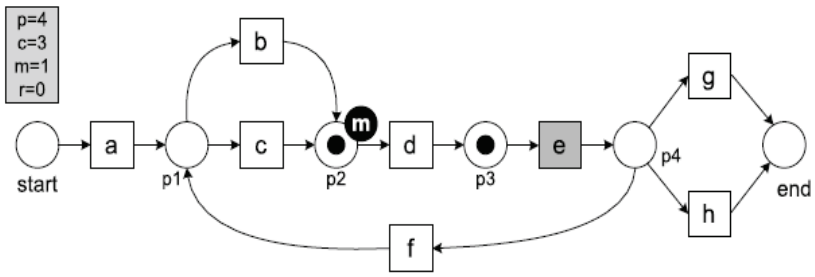


Figura 2.9 Modello di Processo d' Esempio n°4

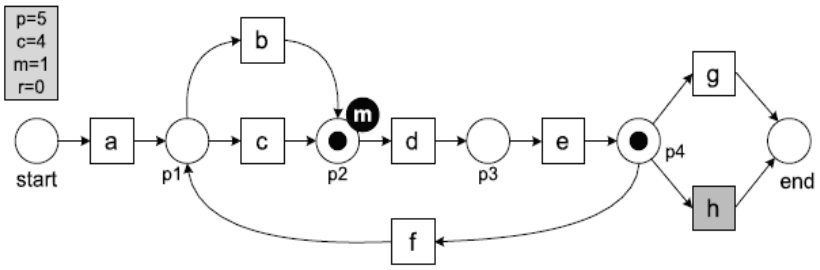


Figura 2.10 Modello di Processo d' Esempio n°5

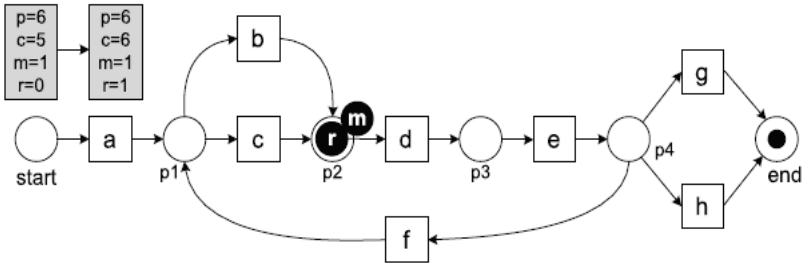


Figura 2.11 Modello di Processo d' Esempio n°6

Infine, riproducendo le ultime due attività vengono incrementati il numero di token prodotti e consumati come spiegato fino ad ora. L'ultima operazione è quella di consumo del token presente nel place "end" che aumenta di 1 i token consumati. Dato che in p2 è rimasto un token, il valore delle misure a seguito di queste operazioni di replay è: p = 6; c = 6; m = 1; r = 1.

A questo punto possiamo calcolare la fitness per questa osservazione:

$$fitness(\sigma_i, N) = \frac{1}{2}\left(1 - \frac{m}{c}\right) + \frac{1}{2}(1 - \frac{r}{p})$$

Nel nostro esempio la fitness è pari a:

$$\frac{1}{2}\left(1 - \frac{1}{6}\right) + \frac{1}{2}\left(1 - \frac{1}{6}\right) = 0.8\overline{3}$$

Effettuando il replay di un intero event log su un modello di processo è possibile ottenere la fitness totale dell'event log e non solo di una traccia. Per farlo si calcolano i valori di p, c, m, r per tutte le tracce, si sommano, e si calcola la fitness come indicato nella formula.

2.4.2 *Aligned fitness. Approccio alignment-based*

L'approccio precedentemente spiegato ha dei lati negativi. In caso di alte deviazioni delle tracce dal modello di processo la fitness rimane più alta del dovuto in quanto la rete di Petri si riempe di token che permettono qualsiasi tipo di comportamento. Inoltre, l'approccio token based può essere utilizzato esclusivamente su un modello di tipo Petri net.

Per calcolare le fitness attraverso questo approccio non si ricorre più a misure legate ai token, bensì ci si concentra sui concetti di alignments e costi.

Per spiegare il concetto di alignments consideriamo una traccia $\sigma = < a, b, d, f >$ e una rete di Petri N come quella mostrata in Fig 2.12.

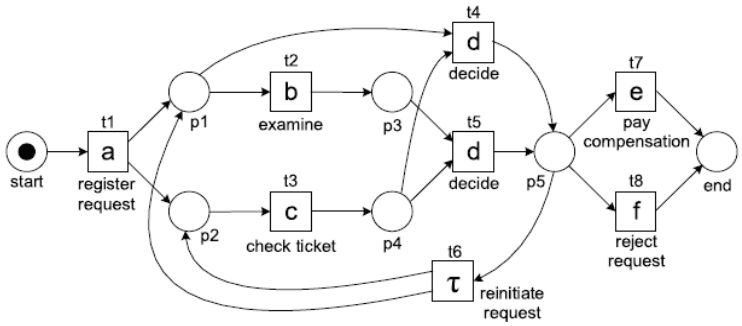


Figura 2.12 Modello di processo d'esempio per l'aligned fitness

La Fig. 2.12 mostra una rete di Petri N sulla quale possono essere eseguite molteplici tracce. Prendendo σ e N gli alignments ottimali sono quelli rappresentati nella tabelle 1 e 2.

La riga superiore rappresenta le mosse effettuate dalla traccia, mentre la riga inferiore un percorso possibile nella rete di Petri N, rappresentato dall'etichetta e dal id della transizione. il simbolo ">>" rappresenta una mossa non possibile nella traccia o nell'event log e prende il nome di disallineamento.

a	**b**	**>>**	**d**	**f**
a t1	b t2	c t3	d t5	f t8

Tabella 1 Esempio di Alignments n°1

a	**>>**	**b**	**d**	**f**
a t1	c t3	b t2	d t5	f t8

Tabella 2 Esempio di Alignments n°2

Ripercorriamo passo per passo gli step rappresentati dalla tabella 1: nella rete di Petri viene eseguita la transizione “a”, a cui corrisponde l’attività “a” della traccia, stessa cosa per “b”. In seguito, nella rete di Petri viene attivata la transizione “c”, che però non trova corrispondenze nella traccia. Questo disallineamento viene rappresentato con il simbolo “>>” nella riga superiore. Infine, senza ulteriori disallineamenti vengono eseguiti “d” e “f”. La tabella 2 mostra un alignment simile ma che prende come percorso nella rete di Petri (riga di sotto) una sequenza che inverte “c” con “b”. Le tabelle 1 e 2 sono considerate ottimali in quanto sono gli allineamenti che presentano il numero di disallineamenti minore data quella traccia e quella rete di Petri.

a	**>>**	**b**	**d**	**f**
a t1	c t3	>>	d t4	f t8

Tabella 3 Esempio di Alignments n°3

Nella tabella 3 troviamo un altro possibile allignment che però mostra al suo interno due disallineamenti. Una volta eseguita la transizione “a” che trova corrispondenza

nella traccia, viene eseguita “c” che non trova corrispondenza. Essendo “b” il prossimo evento nella traccia non si trova una corrispondenza nella rete di Petri, per questo motivo viene inserito il simbolo “>>” nella riga inferiore che rappresenta un’assenza di mossa nel modello. La riga superiore rappresenta le mosse dell’event log mentre quella inferiore consiste in una serie di mosse del modello (move on model). In generale una mossa è una coppia di mosse del log e mosse del modello del tipo $(x(y,t))$, dove x è una mossa del log, y una mossa del modello e t è la transizione a cui si riferisce y [2]. Una move è “legale” se:

- $x = y$ e y è un’etichetta visibile della transizione t (mossa sincrona);
- $x = \gg$ e y è un’etichetta visibile della transizione t (mossa nel modello);
- $x = \gg, y = \tau$ dove τ rappresenta una transizione silenziosa (mossa invisibile nel modello)
- $x \neq \gg, (y,t) = \gg$ (mossa nel log)

a	**b**	**d**	**f**	>>	>>	>>	>>
>>	>>	>>	>>	a t1	c t3	d t4	f t8

Tabella 4 Esempio di Alignments n°3

Un allignment è una serie di mosse tali per cui se togliamo tutti i simboli “>>” rimangono la traccia nella riga di sopra e un percorso di un token che nella rete di Petri parte dall’initial marking e finisce nel final marking.

[2] Come vediamo in Figura è possibile avere più transazioni con la stessa etichetta.

Data questa definizione di allignment è possibile crearne uno come quello mostrato nella tabella 4 che mostra il maggior numero possibile di asincronicità, con l'unica limitazione che nella riga inferiore può essere solo aggiunto il percorso più beve che conduce dallo stato iniziale a quello finale della rete di Petri. Prima abbiamo detto che gli allignment ottimali sono quelli che presentano il minor numero possibile di ">>" (assenza di mosse). In realtà ad ogni assenza di mosse viene associato un peso in base alla funzione di costo δ, che può variare in base all'importanza dell'attività che si prende in considerazione (un mancato pagamento può essere più grave di una mancata registrazione della richiesta). Per semplicità consideriamo che ogni assenza di mosse abbia lo stesso peso $\delta(\lambda) = 1$ dove λ rappresenta un allignment. Una volta calcolato il costo dell'allignment ottimale (che ha costo più basso tra tutti gli allignment possibili i.e tabella 1 o 2) e il costo dell'allignment peggiore (che ha costo più alto tra tutti gli allignment possibili i.e. tabella 4), ci basta dividere il primo per il secondo per ottenere l'alligned fitness:

$$alligned\ fitness(\sigma, N) = 1 - \frac{\delta(\lambda_{ottimale}(\sigma))}{\delta(\lambda_{peggiore}(\sigma))} = 1 - \frac{1}{8} = 0.875$$

2.5 GRAFI DIRETTI

Un grafo è un insieme di nodi e archi che collegano i nodi tra loro. In un grafo ci interessa solo se i nodi sono collegati o meno da un arco $e = \{u, v\}$ dove "u" e "v" sono nodi adiacenti in quanto connessi da e. In un grafo diretto invece i nodi sono collegati seguendo un ordinamento: $e = (u, v)$ non indica solo che "u" e "v" sono adiacenti, bensì che da "u"parte un arco che si connette a "v". Matematicamente un grafo diretto è quindi un insieme di nodi (V) e archi (E) tale per cui il set di nodi non

è vuoto e gli archi appartengono all'insieme di tutte le possibili combinazioni tra due nodi (V x V).

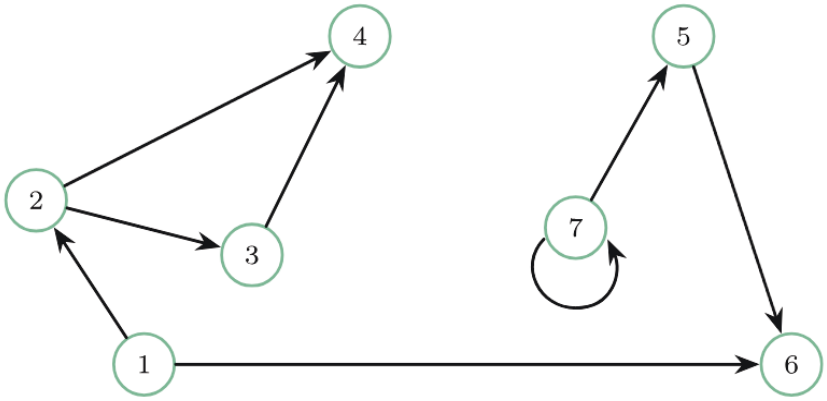


Figura 2.13 Esempio di grafo diretto

Il grafo diretto mostrato in Fig 2.13 può essere rappresentato testualmente. Molti algoritmi, che si occupano della manipolazione di questo tipo di oggetto, non agiscono su formati specifici come quello del DiGraph in Python, bensì utilizzano una formattazione testuale del tipo:

{v 1, v 2, v 3, v 4, v 5, v 6, v 7, e 1 2, e 1 6, e 2 3, e 2 4, e 3 4, e 5 6, e 7 7}

Dentro le parentesi graffe sono presenti il set di nodi (v) e archi (e) che compongono il grafico mostrato in figura. Il numero che contraddistingue nodi e archi nell'esempio testuale indica il loro ID, mentre in figura viene mostrata l'etichetta. In questo caso le etichette dei nodi mostrati corrispondono agli ID utilizzati nell'esempio di testo.

2.6 ALGORITMI PER LA CLASSIFICAZIONE

2.6.1 *Support Vector Machine*

L'SVM è un modello di classificazione in grado di creare sia margini di separazione lineari che non lineari. Dopo un'introduzione generale che spiega l'idea alla base di questo modello, si spiegherà la differenza tra l'SVM lineare e non lineare.

Un aspetto unico dell'SVM sta nel fatto che i margini decisionali utilizzano solo poche osservazioni del dataset per essere creati. Queste poche osservazioni sono chiamate vettori di supporto. L'idea è quella di trovare un iperpiano separatore delle osservazioni del training, che matematicamente può essere rappresentato da questa equazione:

$$w^T x + b = 0$$

Dove x rappresenta le features mentre (w,b) rappresentano i parametri che definiscono l'iperpiano. Dato che k è il numero di feature, l'iperpiano divide in due lo spazio k-dimensionale, e questo fa sì che un'osservazione x_i appartenga a una delle due parti dello spazio k-dimensionale. La sua posizione nello specifico dipende dal segno del valore di $w^T x + b$.

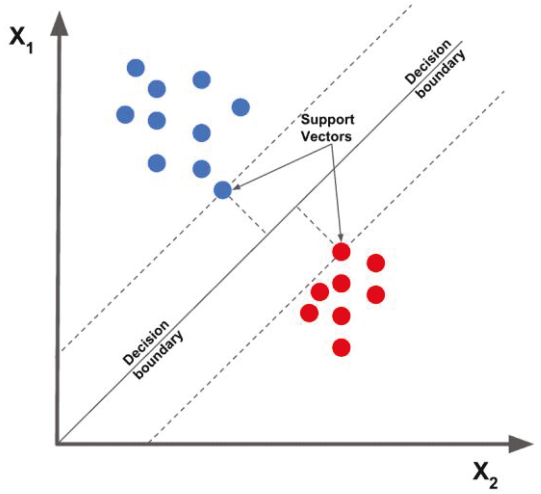


Figura 2.14 Esempio di iperpiano e margine separatore

Quando esiste un iperpiano in grado di separare perfettamente tutte le osservazioni del dataset diciamo che il dataset è linearmente separabile. La Fig. 2.14 mostra un esempio di dati linearmente separabili in una situazione in cui abbiamo due classi. Dato che per separare le due classi possiamo utilizzare un'infinità di piani separatori, con posizioni e inclinazioni diverse, quale scegliamo? L'idea è quella di scegliere l'iperpiano in modo tale da poter essere solido rispetto a piccole perturbazioni delle osservazioni. La solidità rispetto a perturbazioni significa che anche in presenza di piccoli cambiamenti nel valore delle osservazioni, l'iperpiano continua ad essere in grado di dividere le due classi. Per trovare un iperpiano solido dobbiamo introdurre il concetto di margine. Creiamo per ogni possibile iperpiano B_i altri due iperpiani (b_{1i}, b_{2i}) paralleli, detti iperpiani margine, che giacciono sui due punti del dataset più vicini all'iperpiano separatore, anche detti vettori di supporto. Il miglior iperpiano separatore è quello che massimizza la distanza trai due iperpiani margine anche detta margine. Un margine ampio genera una maggior robustezza rispetto a perturbazioni.

Il problema di trovare il miglior iperpiano è un problema di minimizzazione matematica:

$$\min(w, b) \frac{||w||^2}{2}$$

Soggetta al seguente vincolo: $y_i(w^T x_i + b) \geq 1$

Per come è stato definito adesso l'SVM trova l'iperpiano separatore in modo tale da non misclassificare nemmeno un'osservazione del training set, facendo sì che nessun punto cada all'interno dello spazio definito dai due iperpiani margine. Una soluzione meno drastica è quella definita dal soft margin SVM che consente a qualche osservazione di cadere all'interno dello spazio definito dai due iperpiani in modo tale da avere un margine più grande e che quindi permetta al modello di generalizzare maggiormente. Questo può essere fatto introducendo una nuova variabile che prende il nome di slack variable $\varepsilon \geq 0$ per ogni variabile del training set.

$$\min(w, b, \varepsilon_i) \frac{||w||^2}{2} + C \sum_{i=1}^{n} \varepsilon_i$$

Soggetta ai seguenti vincoli: $y_i(w^T x_i + b) \geq 1 - \varepsilon_i$ e $\varepsilon_i \geq 0$

C è un iperparametro che definisce il trade off tra la massimizzazione del margine e la minimizzazione dell'errore di training.

Come abbiamo visto, il compito principale dell'SVM lineare è trovare l'iperpiano che massimizza il margine, ossia la distanza tra i vettori di supporto delle due classi. Tuttavia, non tutti i dataset sono linearmente separabili; in altre parole, non sempre esiste un iperpiano che separa perfettamente le classi nel loro spazio originale.

Qui entra in gioco l'SVM non lineare. Quando i dati non possono essere separati linearmente, possiamo utilizzare una tecnica chiamata kernel trick. Il kernel trick

consiste nel mappare i dati originali in uno spazio di dimensione superiore, dove è più probabile che possano essere separati linearmente. Questa mappatura avviene attraverso una funzione kernel che calcola il prodotto scalare tra due vettori nel nuovo spazio senza dover effettivamente calcolare le coordinate nel nuovo spazio, risparmiando così risorse computazionali. Esistono varie funzioni kernel, ognina delle quali consente di separare dati non lineari. In questo lavoro le funzioni kernel utilizzate sono state Polinomiale e Radial Basis Function(RBF). l'SVM lineare è una scelta semplice ed efficace quando i dati sono sufficientemente separabili in uno spazio lineare, mentre l'SVM non lineare, grazie all'uso dei kernel, permette di gestire con successo situazioni più complesse in cui la separazione lineare non è possibile.

2.6.2 *Decision Tree*

Il Decision Tree è un modello di machine learning utilizzato sia per la classificazione che per la regressione. La struttura ad albero è simile a un digramma di flusso e viene creata in fase di addestramento. Una struttura ad albero ha le seguenti componenti:

a) Nodo radice: rappresenta l'intero set dei dati che abbiamo a disposizione per l'analisi e quindi qua deve essere presa la decisione iniziale. Questo nodo ha solamente rami uscenti;
b) Nodi interni: rappresentano delle condizioni sugli attributi del dataset. Ogni nodo interno ha un ramo in entrata e due o più rami in uscita;
c) Nodi foglia: sono associati ai valori delle label della variabile dipendente. Questi nodi hanno un ramo in entrata e nessun ramo in uscita;

d) Rami: rappresentano il percorso derivante da una decisione o di un test, che collega un nodo superiore ad uno inferiore;

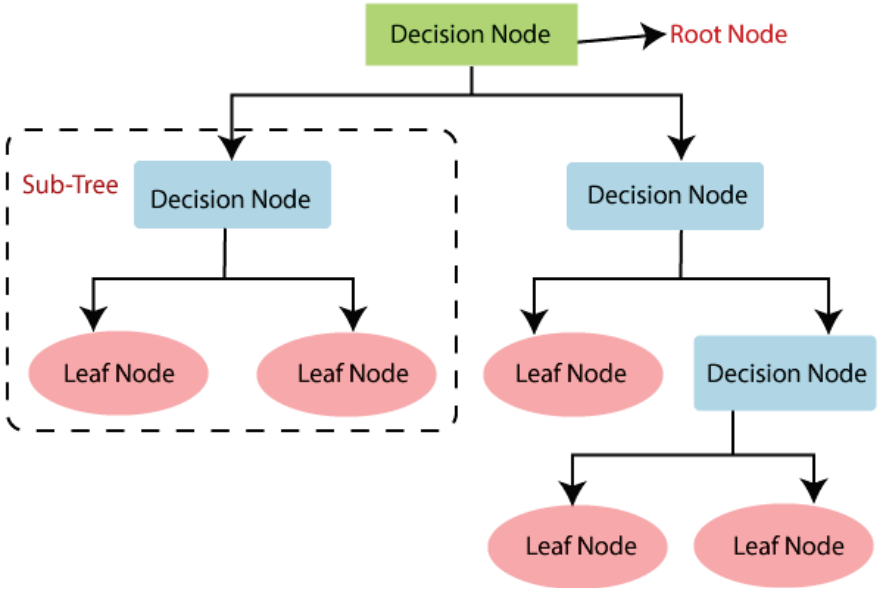


Figura 2.15 Struttura dell'albero decisionale

L'algoritmo alla base del Decision Tree è l'algoritmo di Hunt il quale stabilisce che, dato un training set D e un insieme di classi $\{y_1, y_2, \ldots, y_c\}$ e considerando $D_t \subseteq D$ l'insieme delle osservazioni di training associate ad uno specifico nodo t, se tutti i dati in D_t sono appartenenti alla stessa classe y_t, allora t è un nodo foglia e viene assegnato alla classe y_t. Se invece, le osservazioni in D_t appartengono a classi diverse, verrà scelto un nuovo criterio per dividere i dati. In tal caso applichiamo una condizione sull'attributo preso in considerazione per effettuare uno split su D_t; per ogni sottoinsieme verrà creato un nodo figlio e la procedura verrà ripetuta per ognuni di questi nodi figlii.

Lo *splitting* tramite cui viene suddiviso il sottoinsieme di dati D_t può essere effettuato in diversi modi a seconda delle caratteristiche delle osservazioni. Lo *split* binario suddivide il nodo padre in soli due nodi figli mentre lo *split* cosiddetto *multiway*,

partiziona il nodo padre in più di due nodi figli. La tipologia di split viene scelta in base alla natura degli attributi considerati dall'algoritmo in quel momento. In particolare, per gli attributi binari, l'unica metodologia di divisione possibile è lo split binario, mentre per gli attributi nominali, ordinali e continui è possibile operare una scelta. Nel caso degli attributi continui, la partizione può essere sia binaria, scegliendo un valore per la specifica caratteristica e definendo due classi (una con valori inferiori e l'altra con valori superiori al valore scelto A), sia non binaria, definendo classi di valori nel seguente modo: $v_i \leq A \leq v_{i+1}$, $per\ i = 1,2, \dots, k$. L'obiettivo dell'algoritmo è identificare la migliore condizione di split a ogni iterazione, in modo tale da massimizzare l'omogeneità (o minimizzare l'eterogeneità) delle classi nei nodi figli rispetto a quella del nodo padre. L'omogeneità o eterogeneità all'interno di ciascun nodo figlio può essere misurata utilizzando tre differenti metriche:

a) Entropia: Entropia(t) = $-\ \sum_{I=0}^{C-1} P_I\ (t) log_2 p_i(t)$, dove pi è la probabilità di classificare un'istanza per una specifica classe. Questa misura indica perciò l'impurità all'interno del nodo
b) Indice di Gini: Gini(t) = $1 -\ \sum_{I=0}^{C-1} P_I\ (t)^2$. Misura la probabilità di una classificazione sbagliata di una nuova istanza se questa è classificata in modo casuale secondo la distribuzione delle classi nel dataset;
c) Errore di classificazione: $Error(t)\ =\ 1 -\ \max\ (p_i)$, indica la frequenza con cui un dato nel nodo viene classificato erroneamente.

Una volta deciso quale metrica utilizzare per il calcolo dell'eterogeneità/omogeneità si individua il miglior split in base al gain ratio relativo a una particolare condizione di split. Il gain ratio consiste in una differenza tra l'eterogeneità/omogeneità del nodo

padre prima di applicare lo split e la media dell'eterogeneità/omogeneità dei nodi figli a seguito dello split, ponderata per il numero di osservazione dei nodi figli.
Una caratteristica di particolare rilevanza del Decision Tree risiede nella sua capacità di mitigare il problema dell'overfitting, un fenomeno che può insorgere durante i processi di classificazione. Questo obiettivo viene raggiunto attraverso due strategie principali: da un lato, limitando gli split ai soli nodi binari (approccio utilizzato nell'algoritmo CART); dall'altro, tenendo in considerazione il numero di nodi figli generati durante la suddivisione mediante l'uso del gain ratio, privilegiando un gain ratio elevato (poiché all'aumentare del numero di nodi figli, il gain ratio tende a diminuire).
Infine, l'algoritmo dell'albero decisionale viene interrotto quando:

- Tutti i nodi appartengono alla stessa classe;
- Il numero di record all'interno del nodo è sotto un valore soglia definito, ossia ulteriori suddivisioni sarebbero poco sensate;
- Tutti i record hanno valori simili degli attributi;
- Lo split non migliora la misura di omogeneità.

2.6.3 *Random Forest*

L'algoritmo del Random Forest rappresenta il primo dei due metodi di "ensemble" che verranno impiegati nella nostra analisi. Come già accennato in precedenza, questa tipologia di algoritmi combina i risultati di diversi modelli per ottenere una previsione complessiva più accurata. Nello specifico, i modelli utilizzati nel Random Forest sono alberi decisionali (da cui il termine "Forest"). Per mitigare il rischio di

overfitting, viene adottato il metodo del bagging, secondo il quale ciascun albero decisionale viene addestrato su un sottoinsieme casuale dei dati (da cui il termine "Random"), generato tramite campionamento con ripetizione (campionamento bootstrap). Inoltre, durante l'addestramento di ogni Decision Tree, ad ogni iterazione, solo un sottoinsieme delle feature disponibili viene considerato per selezionare il miglior criterio di splitting. Successivamente, la predizione del Random Forest si basa sulla combinazione dei risultati provenienti da ciascun albero decisionale, tenendo conto della natura del task per cui l'algoritmo è stato applicato.

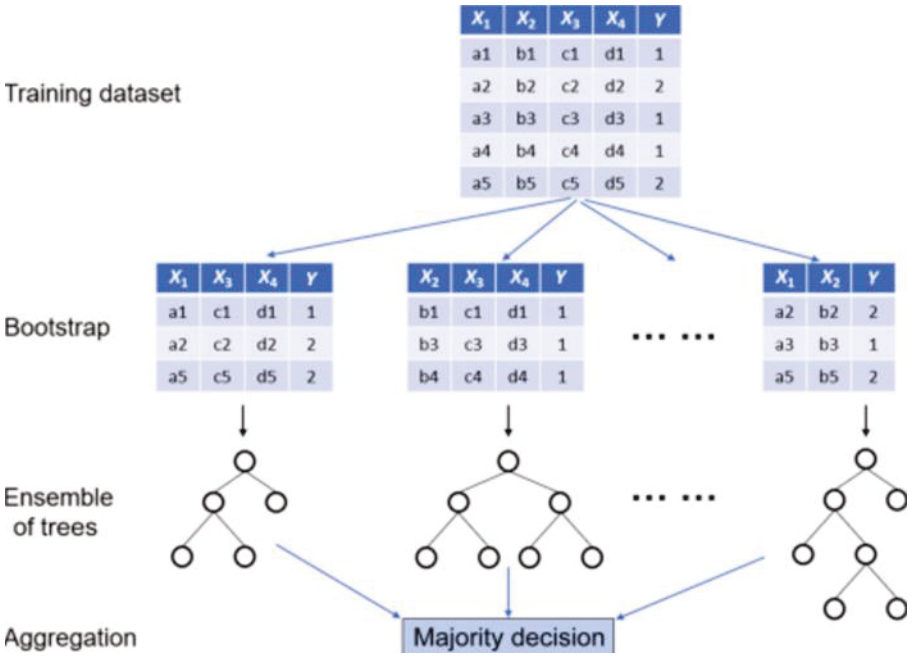


Figura 2.16 Esempio del funzionamento del Random Forest

Considerando B il numero degli alberi decisionali utilizzati, per b = 1 a B, l'algoritmo del Random Forest si articola nel seguente modo:

a. Viene estratto un campione bootstrap di dimensione N dai dati di training disponibili;

b. Viene costruito un albero decisionale denominato T_b a partire dal campione ottenuto. Per ciascun nodo terminale dell'albero vengono eseguiti i seguenti passaggi fino a quando il numero di osservazioni nel nodo non raggiunge la soglia minima prestabilita: 1) Viene selezionato un sottoinsieme casuale di variabili m dall'insieme delle variabili; 2) Viene scelta la migliore variabile tra le m per effettuare lo split; 3) il nodo viene splittato come per un normale albero decisionale in base alla condizione che massimizza il gain.

Il risultato è un insieme degli output degli alberi $\{T_b\}\ per\ b = 1\ a\ B$

Per effettuare la predizione di un nuovo punto, come accennato precedentemente, occorre considerare l'obiettivo dell'analisi: nel caso di un problema di classificazione, verrà scelta la classe maggioritaria tra tutte quelle predette da ciascun albero decisionale; nel caso di un problema di regressione, verrà calcolata la media dei valori previsti dagli alberi.

2.6.4 *XGBoost*

L'XGBoost è il secondo algoritmo utilizzato che rientra nella categoria dei metodi ensemble ed è noto per la sua potenza ed efficienza. Si basa sulla tecnica del boosting, una tecnica che combina più modelli deboli, generalmente alberi decisionali, per creare modelli complessivi robusti e accurati. L'XGBoost si distingue per la capacità di incorporare tecniche di regolarizzazione che aiutano a prevenire il sovra-adattamento dai dati di training per non creare problemi di generalizzazione. La regolarizzazione è effettuata tramite penalizzazioni L1 e L2 note rispettivamente come Lasso e Ridge. Queste regolano la complessità del

modello permettendogli di generalizzare meglio, Un'altra caratteristica importante dell'XGBoost molto utile in applicazioni reali è la capacità di gestire dati sparsi e valori mancanti.

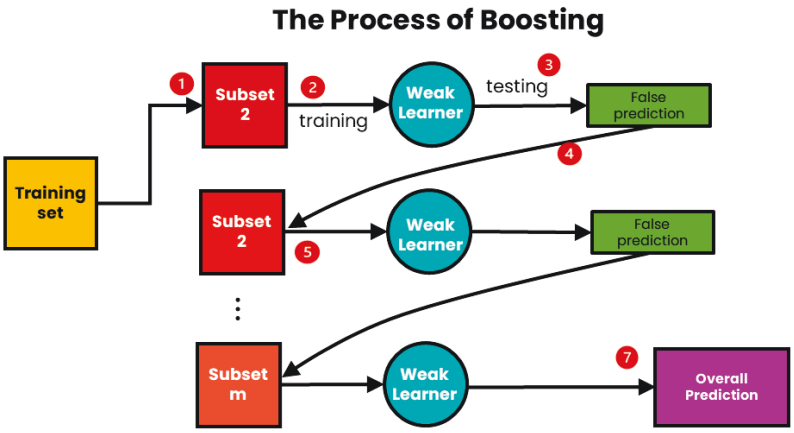


Figura 2.17 Esempio del funzionamento dell'XGBoost

L'XGBoost funziona aggiungendo sequenzialmente alberi a un insieme, dove ogni nuovo albero corregge gli errori commessi dai precedenti. Come mostrato dalla Fig. 2.17 il modello inizia con una previsione semplice, solitamente la media dei valori target, e affina iterativamente questa previsione aggiungendo alberi che si concentrano sui residui (la differenza tra i valori effettivi e quelli previsti). L'obiettivo è minimizzare una funzione di perdita, tipicamente l'errore quadratico medio per la regressione o la log loss per la classificazione.

Meccanismo di Gradient Boosting:

- Previsione Iniziale: Il processo inizia con un modello base, spesso un singolo albero decisionale che fornisce una previsione iniziale per tutti i campioni.
- Calcolo dei Residui: Dopo la previsione iniziale, vengono calcolati i residui, che rappresentano l'errore tra i valori previsti e quelli effettivi.

- Adattamento del Nuovo Albero ai Residui: Un nuovo albero decisionale viene quindi addestrato su questi residui. Lo scopo di questo albero è prevedere i residui dal passaggio precedente.
- Aggiornamento della Previsione: Le previsioni del nuovo albero vengono aggiunte alle previsioni precedenti, migliorando gradualmente l'accuratezza.
- Ripetizione: Questo processo viene ripetuto per un numero predefinito di iterazioni o fino a quando il modello smette di migliorare. Ogni nuovo albero viene aggiunto all'insieme, e le previsioni del modello diventano progressivamente più accurate.

Quindi l'XGBoost è un algoritmo flessibile e potente che combina i principi del gradient boosting con tecniche di ottimizzazione avanzate.

Capitolo 3 DATASET

Come anticipato nell'introduzione i dataset sono stati estratti dalla piattaforma Esse3 ed estratti tramite query SQL. Il risultato di questa estrazione sono stati tre dataset, ognuno contenente informazioni diverse sugli stessi studenti. Come anticipato nell'introduzione, nella parte relativa al dataset, questi ultimi sono stati pre-processati in modo tale che fossero adatti alle analisi successive. Prima di cominciare con il pre-processing effettuiamo una breve spiegazione delle variabili presenti nei dataset di partenza. Per questioni di semplicità descriviamo solo le variabili che sono state utilizzate nell'analisi.

3.1 DATASET 1 – Esami Superati

Questo dataset come anticipato nell'introduzione contiene una riga per ogni esame superato dallo studente. Le variabili sono le seguenti:

- STU_ID: Codice alfa-numerico opportunamente anonimizzato che identifica univocamente lo studente;
- DES: Nome dell'esame superato dallo studente;
- MOD_VAL_COD: Codice che identifica la modalità di valutazione. Prova Finale; OFA e altri esami presentano una modalità di valutazione diversa dai normali esami;
- PESO_AD: Peso (Crediti formativi) relativo all'esame;

- TIPO_INS_DESC: descrizione sul tipo di insegnamento (Obbligatorio/Facoltativo);
- ANNO_CORSO: Anno in cui dovrebbe teoricamente essere sostenuto l'esame;
- DATA_SUP (aa-mm-gg): Data di superamento dell'esame;
- VOTO: Valutazione dell'esame superato (Min 18, Max 30);
- DATA_ISCR: Data di iscrizione al corso universitario;
- Tempo_laurea: Giorni impiegati per l'ottenimento del titolo di laurea dal giorno in cui ci si è iscritti;
- Tempo_laurea_AA: Giorni impiegati per l'ottenimento del titolo di laurea contando dal giorno d'inizio dell'anno accademico in cui ci si è iscritti;

3.2 DATASET 2 – Esami con Stati

In questo dataset a differenza del primo ogni riga rappresenta un'attività legata ad un esame. Fornisce informazioni più dettagliate sul percorso dello studente in quanto contiene al suo interno tutte le attività di uno studente in relazione all'esame, come l'iscrizione, l'avvenuta verbalizzazione, la bocciatura, ecc. le variabili sono le seguenti:

- STU_ID: Codice alfa-numerico opportunamente anonimizzato che identifica univocamente lo studente;
- STA_REG_DES: Descrizione dello stato dell'esame (Prenotato, Chiuso, Caricato, Verbalizzato);
- DATA_APP: Data in cui è avvenuta un'attività relativa ad un esame;

- AD_DES: Nome della materia d'esame per la quale è stata effettuata l'attività;
- RITIRATO_FLG: Variabile binaria che indica con il valore 1 l'azione del ritiro dall'esame;
- ASSENTE_FLG: Variabile binaria che indica con il valore 1 l'assenza dello studente all'esame;
- PROVA_SUP_FLG: Variabile binaria che indica con il valore 1 il superamento dell'esame;
- Insuf_flg: Variabile binaria che indica con il valore 1 la bocciatura a causa di insufficienza;

3.3 DATASET 3 – Dati anagrafici degli studenti

In questo dataset ogni riga rappresenta uno studente.

- STU_ID: Codice alfa-numerico opportunamente anonimizzato che identifica univocamente lo studente;
- AA_IMM: Anno immatricolazione dello studente;
- SESSO: Maschio/Femmina;
- Voto: voto del diploma;

Capitolo 4: METODOLOGIA

In questo capitolo descriveremo le analisi che abbiamo condotto per ottenere i risultati presentati nei capitoli successivi. L'obiettivo dell'analisi è triplice:

1. Estrarre i comportamenti più frequenti tra gli studenti e formattarli in modo adeguato agli scopi successivi dell'analisi.
2. Indurre modelli predittivi in grado di prevedere le performance degli studenti, misurate in termini di tempo necessario per il conseguimento del titolo.
3. Valutare l'utilità di queste variabili comportamenti utilizzate nei modelli predittivi.

Il punto di partenza della ricerca sono stati i tre event log descritti nel precedente capitolo. Per raggiungere gli obiettivi che si prefigge questa tesi abbiamo eseguito una serie di analisi tramite codice Python che verranno descritte dettagliatamente in questo capitolo.

4.1 PRE-PROCESSING

Dopo aver importato i dataset descritti nel paragrafo precedente abbiamo effettuato un'operazione di join trai tre dataset di partenza. Il risultato di quest'operazione è un dataset chiamato "df_merged", utilizzato per la maggior parte delle analisi successive. Lo scopo di questo join è quello di trasporre informazioni presenti nel primo e nel terzo dataset sul secondo. Il codice utilizzato è il seguente:

“df_merged = pd.merge(df_esami, df_exam_states, left_on=['STU_ID', 'DES'], right_on=['STU_ID', 'AD_DES'], how='right')

df_merged = pd.merge(df_merged, df_anagrafiche, on='STU_ID', how='left')”

Come vediamo dal codice il join col DATASET 1 (“df_esami”) viene fatto utilizzando le corrispondenze tra le colonne “STU_ID” e “DES”. “how='right” indica che tutte le righe del DATASET 2 (“df_exam_states”) saranno mantenute, e verranno unite con le righe corrispondenti del DATASET 1 basate sui valori delle colonne specificate. Viene fatta la stessa cosa, utilizzando solo “STU_ID”, per la join con DATASET 3 (“df_anagrafica”).

Con la seguente funzione abbiamo eliminato i duplicati: “df_merged = df_merged.drop_duplicates()”.

Siccome erano presenti degli individui che avevano dei tempi di laurea molto bassi (726 e 772 giorni) li abbiamo considerati outlier e quindi eliminati dal dataset con il seguente codice:

“valori_da_elimin = [726, 772]

df_merged = df_merged[~df_merged['tempo_laurea_AA'].isin(valori_da_elimin)]”.

Lo step successivo è stato quello di andare a creare due colonne chiamate “SEMESTRE_ATTESO” e “ANNO_ATTESO” nel seguente modo:

“df_merged['SEMESTRE_ATTESO'] = df_merged['AD_DES'].map(sem_att_esami_primo)

df_merged['ANNO_ATTESO'] = df_merged['AD_DES'].map(anni_att_esami_completo)”.

Le variabili “sem_att_esami_primo” e “anni_att_esami_completo” sono dei dizionari contenenti come chiave il nome dell’esame e come valore l’anno in cui dovrebbe essere dato l’esame.

Dopo aver creato queste due colonne, che rappresentano l’anno e semestre in cui gli esami dovrebbero essere dati, andamo a creare anche le colonne che indicano l’anno e il semestre in cui l’esame è stato effettivamente superato:

“def calcola_semestre_superato(data):

```
    mese = data.month
    giorno = data.day
    if (mese > 10 or (mese == 10 and giorno >= 2)):
        return 1
    elif (mese < 3 or (mese == 3 and giorno <= 2)):
        return 1
    else:
        return 2
```

df_merged['SEMESTRE_SUPERATO'] =

df_merged['DATA_SUP'].apply(calcola_semestre_superato)”.

La colonna “SEMESTRE_SUPERATO” è stata creata in base a “DATA_SUP”. se quest’ultima è compresa tra il 2 ottobre e il 2 marzo dell’anno successivo allora significa che l’esame è stato superato nel primo semestre, in ogni altra data, alla colonna viene associato valore 2.

Lo step successivo riguarda il tempo di laurea, per ogni studente infatti è stata creata una variabile categorica ordinale “STATO_LAUREA” che può assumere valore “Early”, “One year late” e “Late” in base alla colonna “tempo_laurea_AA”. A tempi

minori di 1305 corrispondeva l'etichetta "Early", trai 1305 e 1670 l'etichetta "One year late", tutti gli altri "Late". L'ultima riga del sottostante frammento di codice invece rimuove tutti gli studenti che hanno impiegato più di 2400 giorni ovvero 6 anni e mezzo per laurearsi, in quanto ritenuti outlier.

```
"def determina_stato_laurea(tempo):
    if tempo < 1305:
        return 'Early'
    elif tempo >= 1305 and tempo < 1670:
        return 'One_year_late'
    else:
        return 'Late'
df_merged['STATO_LAUREA'] =
df_merged['tempo_laurea_AA'].apply(determina_stato_laurea)
df_merged = df_merged[df_merged['tempo_laurea_AA'] <= 2400]"
```

Questa variabile seppur modificata verrà usata in seguito come variabile target nella predizione.

Oltre alle righe presenti nei dataframe di partenza sono state inserite righe artificiali con la funzione "append_end_semester" mostrata di seguito. Le righe in questione rappresentano la fine del primo semestre nei primi due anni e fine dell'anno per i primi sei anni. In questo modo è stato possibile avere una visione più dettagliata sul momento in cui uno specifico studente ha effettuato un esame o una certa attività legata all'esame:

```
"def append_end_semesters(df):
    new_rows = []
```

```
student_ids = df["STU_ID"].unique()


for stu_id in student_ids:
    record = df[df["STU_ID"] == stu_id].iloc[0]
    anno_inizio = record["AA_ISCR_ID"]
    data_iscr = record["DATA_ISCR"]
    stato_carr = record["STATO_LAUREA"]
    dates = {
        "Fine primo semestre": fine_primo_semestre(anno_inizio + 1),
        "Fine primo anno": fine_primo_anno(anno_inizio + 1),
        "Fine terzo semestre": fine_terzo_semestre(anno_inizio + 1),
        "Fine secondo anno": fine_primo_anno(anno_inizio + 2),
        "Fine terzo anno": fine_primo_anno(anno_inizio + 3),
        "Fine quarto anno": fine_primo_anno(anno_inizio + 4),
        "Fine quinto anno": fine_primo_anno(anno_inizio + 5),
        "Fine sesto anno": fine_primo_anno(anno_inizio + 6)  }
    for description, date in dates.items():
        new_rows.append({
            'STU_ID': stu_id,
            'DES': description,
            'DATA_SUP': date,
            'DATA_ISCR': data_iscr,
            'STATO_LAUREA': stato_carr,
            'DATA_APP': date})
```

return pd.DataFrame(new_rows)".

Nel dataset contenente solo gli esami superati ("df_esami") abbiamo utilizzato le stesse funzioni che abbiamo descritto fino ad ora per "df_merged". Di seguito sono riportate solo le applicazioni delle funzioni e non la loro definizione per evitare ripetizioni.

```
"df_esami = df_esami.drop_duplicates()
df_esami = df_esami.dropna(subset =["DATA_SUP"])
df_esami['SEMESTRE_ATTESO'] = df_esami['DES'].map(sem_att_esami_primo)
df_esami['ANNO_ATTESO'] = df_esami['DES'].map(anni_att_esami_completo)
df_esami['SEMESTRE_SUPERATO'] =
df_esami['DATA_SUP'].apply(calcola_semestre_superato)
df_esami['STATO_LAUREA'] =
df_esami['tempo_laurea_AA'].apply(determina_stato_laurea)
df_esami = df_esami[df_esami['tempo_laurea_AA'] <= 2400]
df_esami = pd.concat([df_esami, append_end_semesters(df_esami)],
ignore_index=True)
df_esami = assign_AA_year(df_esami)".
```

Come possiamo vedere dal codice, sono state eliminate le righe contenenti valori "NA" nella colonna "DATA_SUP", e sono stati creati gli anni e i semestri in cui gli esami dovevano essere sostenuti, oltre a quelli in cui gli esami sono stati effettivamente superati. È stata inoltre creata la variabile categoriale "STATO_LAUREA" nello stesso modo utilizzato per il dataset precedente. Gli studenti che hanno impiegato più di 2400 giorni per conseguire il titolo sono stati

rimossi dal dataset, e sono state aggiunte righe artificiali che rappresentano la fine dei primi due semestri e dei primi sei anni.

Le operazioni successive, applicate su entrambi i dataset "df_esami" e "df_merged", hanno come obiettivo la creazione di nuovi dataset contenenti informazioni dettagliate sulla carriera accademica degli studenti nel corso del primo anno.

Cominceremo quindi con l'analisi del dataset "df_merged".

```
“def filtra_primo_anno(df):
    date_fine_primo_anno = df[df['DES'] == "Fine primo anno”]
.set_index('STU_ID')['DATA_SUP']
    df['DATA_FINE_PRIMO_ANNO'] = df['STU_ID'].map(date_fine_primo_anno)
    df['DIFF_ISCR_SUP'] = (df['DATA_SUP'] - df['DATA_ISCR']).dt.days
    df['DIFF_ISCR_FINE'] = (df['DATA_FINE_PRIMO_ANNO'] -
df['DATA_ISCR']).dt.days
    df_filtrato = df[(df['DIFF_ISCR_SUP'] <= df['DIFF_ISCR_FINE'])]
    return df_filtrato
df_merged_filtrato = filtra_primo_anno(df_merged_1_anno)”.
```

La funzione “filtra_primo_anno” ha come obiettivo filtrare un dataframe, mantenendo solo le righe relative agli esami sostenuti dagli studenti durante il loro primo anno accademico. Il processo inizia con l'estrazione della data di fine del primo anno accademico per ciascuno studente. Successivamente, viene creata una nuova colonna che assegna a ogni studente la data estratta precedentemente.

Per ogni esame sostenuto, viene quindi calcolata la differenza in giorni tra la data di iscrizione all’università e la data dell’esame stesso. Una volta ottenute queste differenze, si procede a calcolare la differenza in giorni tra la data di iscrizione e la

data di fine del primo anno accademico. Infine, vengono confrontate queste differenze e vengono mantenuti solo gli esami in cui la differenza tra la data d'iscrizione e la data dell'esame è inferiore alla differenza tra la data d'iscrizione e la data di fine del primo anno. Poiché in alcuni casi un esame potrebbe essere sostenuto nel primo anno anche se non fa parte del programma del primo anno, utilizziamo una funzione che filtra ulteriormente il dataset, mantenendo solo gli esami del primo anno presenti nella lista "lista_esami_primo", applicata nel seguente modo: "df_merged_filtrato= df_merged_filtrato [df_merged_filtrato ['DES'].isin(lista_esami_primo)]".

Una volta creato "df_merged_filtrato" applichiamo le funzioni qui di seguito che hanno l'obiettivo di arricchire il dataframe con nuove righe "artificiali" che rappresentano le attività di "Start" ed "End" per ogni studente, calcolare lo stato dell'esame per ogni riga, e infine creare una descrizione finale basata sullo stato dell'esame e sulla descrizione dell'attività.

"def subtract_one_month_one_day(date):
    return date - relativedelta(months=1, days=1)
def add_one_month_one_day(date):
    return date + relativedelta(months=1, days=1)

def determine_status(row):
    if row['DES'] in ['Fine primo semestre', 'Fine primo anno', 'Start', 'End']:
        return row['DES']
    if row['STA_REG_DES'] == 'Caricato':
        return 'Passato'

```
    elif row['STA_REG_DES'] == 'Chiuso' and row['TIPO_NOVERB_DES'] in
['Assente', 'Respinto/Ritirato']:
        return 'Non Passato'
    elif row['STA_REG_DES'] == 'Verbalizzato' and row['GIUD_ESA_DES'] ==
'Insufficiente':
        return 'Non Passato'
    else:
        return None


def create_passato_nonpassato_merged_start_end(df):
    df['DATA_APP'] = pd.to_datetime(df['DATA_APP'])
    new_rows = []
    for stu_id, group in df.groupby('STU_ID'):
        earliest_date = group['DATA_APP'].min()
        new_date = subtract_one_month_one_day(earliest_date)
        new_rows.append({'STU_ID': stu_id, 'DES': 'Start', 'DATA_APP': new_date})
    for stu_id, group in df.groupby('STU_ID'):
        latest_date = group['DATA_APP'].max()
        new_date = add_one_month_one_day(latest_date)
        new_rows.append({'STU_ID': stu_id, 'DES': 'End', 'DATA_APP': new_date})
    new_rows_df = pd.DataFrame(new_rows)
    df = pd.concat([df, new_rows_df], ignore_index=True)
    df['STATO_ESAME'] = df.apply(determine_status, axis=1)
    df = df[df['STATO_ESAME'].notna()]
```

```
    df['DESC_FINALE'] = df['DES'] + "-" + df['STATO_ESAME']
    df = df.drop_duplicates()
    return df

df_merged_caricato =
create_passato_nonpassato_merged_start_end(df_merged_filtrato)".
```

La funzione “create_passato_nonpassato_merged_start_end”, applicata a “df_merged_filtrato” per creare “df_merged_caricato”, inizia convertendo la colonna “DATA_APP” in formato datetime per consentire operazioni di manipolazione delle date. Successivamente, si crea una lista “new_rows” per raccogliere le nuove righe. La funzione itera poi sul DataFrame raggruppato per “STU_ID”, aggiungendo per ogni gruppo una riga "Start", basata sulla data più precoce trovata nel gruppo, e una riga "End", basata sulla data più recente. Per calcolare le date "Start" ed "End", si utilizzano le funzioni “subtract_one_month_one_day” e “add_one_month_one_day”, che rispettivamente sottraggono o aggiungono un mese e un giorno alla data fornita. Queste nuove righe vengono convertite in un DataFrame separato e concatenate al DataFrame originale.

Successivamente, la funzione “determine_status” viene applicata a ciascuna riga del DataFrame per determinare lo stato dell'esame. Questa funzione esamina diverse condizioni: se la descrizione dell'attività (“DES”) è "Fine primo semestre", "Fine primo anno", "Start" o "End", lo stato dell'esame corrisponderà alla descrizione stessa. Se “STA_REG_DES” è "Caricato", lo stato sarà "Passato". Se lo stato è "Chiuso" e “TIPO_NOVERB_DES” è "Assente" o "Respinto/Ritirato", o se lo stato

è "Verbalizzato" e “GIUD_ESA_DES” è "Insufficiente", lo stato sarà "Non Passato". Per tutti gli altri casi, la funzione restituisce None, che indica che la riga non deve essere considerata.

Dopo aver determinato lo stato dell'esame, il DataFrame viene filtrato per mantenere solo le righe in cui lo stato dell'esame non è nullo. Successivamente, viene creata una descrizione finale combinando la descrizione dell'attività (DES) e lo stato dell'esame (STATO_ESAME) in una nuova colonna “DESC_FINALE”. Infine, vengono rimosse eventuali righe duplicate e il DataFrame risultante, che include tutte queste nuove informazioni, viene restituito.

Le righe “Start” ed “end” sono state aggiunte in modo tale da avere un punto di inizio e di fine comune trai percorsi di tutti gli studenti. In questo modo gli Instance Graph, e i modelli di processo sono più facilmente analizzabili. “STATO_ESAME” e “DESC_FINALE” invece, sono state due nuove variabili create in modo tale da avere un dataset che ci indichi quali sono stati i tentativi malriusciti degli studenti. Infatti “STATO_ESAME” è una colonna che indica se l’esame che quella riga rappresenta è stato passato o meno, mentre “DESC_FINALE” è una colonna che ha come valore una stringa che è data dall’unione tra il nome dell’esame e lo stato dell’esame[3] .Con “df_merged_caricato” abbiamo un dataset che ci permette di avere una visione dettagliata del percorso degli studenti nel primo anno (a differenza di “df_esami” che ha informazioni solo sugli esami superati).

[3] Esempio: “ANALISI 1 – Passato” o “FISICA 2 – Non Passato”

Lo step successivo è quello di cominciare ad analizzare i comportamenti degli studenti attraverso tecniche di Process Mining. Dato che lo scopo di questo lavoro è quello di analizzare se c'è un legame trai comportamenti degli studenti nel primo anno e le loro performance in termini di tempo impiegato per il conseguimento del titolo, proseguiamo con la divisione del dataset "df_merged_caricato" e "df_esami_filtrato[4]" in altri tre dataset in base alle performance:

"df_merged_caricato_early =
df_merged_caricato[df_merged_caricato["STATO_LAUREA"] =="Early"]
df_merged_caricato_1_year_late =
df_merged_caricato[df_merged_caricato["STATO_LAUREA"] =="One_year_late"]
df_merged_caricato_late =
df_merged_caricato[df_merged_caricato["STATO_LAUREA"] =="Late"]
df_esami_early = df_esami_filtrato[df_esami_filtrato["STATO_LAUREA"]
=="Early"]
df_esami_1_year_late = df_esami_filtrato[df_esami_filtrato["STATO_LAUREA"]
=="One_year_late"]
df_esami_late = df_esami_filtrato[df_esami_filtrato["STATO_LAUREA"]
=="Late"]".

[4] "df_esami_filtrato" è "df_esami" al quale sono state applicate le funzioni "filtra_esami_primo_anno" e "filtra_primo_anno" come spiegato per "df_merged"

4.2 CREAZIONE DELLE VARIABILI PER LA PREDIZIONE

Partendo dagli event log iniziali sono state create delle variabili quantitative che rappresentano in maniera generica[5] il comportamento e le performance degli studenti nel loro primo anno accademico. I risultati sono stati inseriti in dizionari contenenti come chiavi le matricole degli studenti e come valori le variabili calcolate, per poi essere trasformati in dataset ed essere inserite alla fine all'interno del dataset "df_anagrafiche" che è stato utilizzato come dataset per l'induzione dei modelli predittivi. Di seguito un elenco puntato che per ogni variabile descrive cosa rappresenta e come è stata ottenuta.

- Media voti: Questa metrica è una semplice media ponderata per i crediti ("PESO_AD") di ogni esame superato dagli studenti del primo anno.

```
"def calcola_voto_medio(df):
    df = df.dropna(subset=['VOTO', 'PESO_AD'])
        group_by_stu_id = df.groupby("STU_ID")
    dizio = {}
    for stu_id, group in group_by_stu_id:
        somma_voti_pesati =
group['VOTO'].multiply(group['PESO_AD']).sum()
        somma_pesi = group['PESO_AD'].sum()
            if somma_pesi != 0:
            media_pesata = (somma_voti_pesati / somma_pesi)
            dizio[stu_id] = media_pesata
```

[5] Per generica intendiamo che non danno nessuna informazione sullo specifico percorso seguito nel sostenere gli esami.

```
    return dizio
```

voti_medi_diz = calcola_voto_medio(df_esami_filtrato)".

- Basi di laurea: Proiezione del voto di partenza di laurea calcolata a partire dalla media dei voti. $B = \frac{m*11}{3}$ dove "m" è la media dei voti nel primo anno.

```
"def calcola_base_di_laurea(df):
        df = df.dropna(subset=['VOTO', 'PESO_AD'])
        group_by_stu_id = df.groupby("STU_ID")
    dizio = {}
    for stu_id, group in group_by_stu_id:
        somma_voti_pesati =
group['VOTO'].multiply(group['PESO_AD']).sum()
        somma_pesi = group['PESO_AD'].sum()
        if somma_pesi != 0:
            media_pesata = (somma_voti_pesati / somma_pesi) * (11 / 3)
            dizio[stu_id] = media_pesata
    return dizio
```

basi_di_laurea_diz = calcola_base_di_laurea(df_esami_filtrato)".

- Esami dati nel primo e nel secondo semestre: conteggio degli esami dati nei due diversi semestri.

```
"esami_dati_s1 =
df_esami_filtrato[df_esami_filtrato['SEMESTRE_SUPERATO'] == 1
].groupby('STU_ID').size().reset_index(name='ES_DATI_s1')
```

esami_dati_s2 =

df_esami_filtrato[df_esami_filtrato['SEMESTRE_SUPERATO'] == 2

].groupby('STU_ID').size().reset_index(name='ES_DATI_s2')".

- Crediti dati nel primo e nel secondo semestre: somma dei crediti corrispondenti agli esami superati.

  "creds_s1 = df_esami_filtrato[df_esami['SEMESTRE_SUPERATO'] ==

  1].groupby('STU_ID')['PESO_AD'].sum()

  creds_s1 = creds_s1.reset_index(name='CREDS_s1')

  creds_s2 = df_esami_filtrato[df_esami['SEMESTRE_SUPERATO'] ==

  2].groupby('STU_ID')['PESO_AD'].sum()

  creds_s2 = creds_s2.reset_index(name='CREDS_s2')".

- Crediti dati nei primi tre anni: calcolati come i crediti dei vari semestri ma dove si prendono come riferimento gli anni in cui gli esami sono stati presi in considerazione:

  "creds_y1 = df_esami[df_esami['AA_DATO'] ==

  1].groupby('STU_ID')['PESO_AD'].sum()

  creds_y1 = creds_y1.reset_index(name='CREDS_Y1')

  creds_y2 = df_esami[df_esami['AA_DATO'] ==

  2].groupby('STU_ID')['PESO_AD'].sum()

  creds_y2 = creds_y2.reset_index(name='CREDS_Y2')

  creds_y3 = df_esami[df_esami['AA_DATO'] ==

  3].groupby('STU_ID')['PESO_AD'].sum()

  creds_y3 = creds_y3.reset_index(name='CREDS_Y3')".

- Crediti attesi per I primi tre anni: Numero di crediti che teoricamente dovrebbero raggiungere nei vari anni per ogni studente. Se messi a rapporto come denominatore con i crediti effettivi si ottiene una misura che rappresenta il livello di adempimento degli studenti rispetto agli esami che avrebbero dovuto dare.

  “expected_creds_y1 = df_esami[df_esami['ANNO_ATTESO'] == 1].groupby('STU_ID')['PESO_AD'].sum()

  expected_creds_y1 = expected_creds_y1.reset_index(name='EXP_CREDS_Y1')

  expected_creds_y2 = df_esami[df_esami['ANNO_ATTESO'] == 2].groupby('STU_ID')['PESO_AD'].sum()

  expected_creds_y2 = expected_creds_y2.reset_index(name='EXP_CREDS_Y2')

  expected_creds_y3 = df_esami[df_esami['ANNO_ATTESO'] == 3].groupby('STU_ID')['PESO_AD'].sum()

  expected_creds_y3 = expected_creds_y3.reset_index(name='EXP_CREDS_Y3')”.
- Numero bocciature: numero di volte in cui un individuo non ha superato un esame nel primo anno.

  “n_bocciato = df_merged_caricato[df_merged_caricato['STATO_ESAME'] == "Non Passato"].groupby('STU_ID').size().reset_index(name='NUM_BOC')”.
- Costanza nel dare gli esami: questa misura consiste nella deviazione standard dell’intervallo medio tra due esami consecutivi. Se uno studente da esami

sempre distanziati tra loro dallo stesso intervallo temporale la misura sarà 0, altrimenti un valore alto.

```
“df_esami_sorted =
df_esami_filtrato[~df_esami_filtrato['DES'].isin(righe_artificiali)].sort_values(by=['STU_ID', 'DATA_SUP'])
df_esami_sorted['TIME_INTERVAL'] =
df_esami_sorted.groupby('STU_ID')['DATA_SUP'].diff().dt.days
mean_intervals =
df_esami_sorted.groupby('STU_ID')['TIME_INTERVAL'].mean().reset_index()
mean_intervals.columns = ['STU_ID', 'MEAN_INTERVAL']
consistency_measure =
df_esami_sorted.groupby('STU_ID')['TIME_INTERVAL'].std().reset_index()
”.
```

- Costanza nei voti: questa misura funziona come la precedente dove al posto dell’intervallo medio viene usato il voto medio. Una volta trovato il voto medio si calcola la deviazione standard nel seguente modo.

```
“consistency_measure_voti =
df_esami_sorted.groupby('STU_ID')['VOTO'].std().reset_index()
consistency_measure_voti.columns = ['STU_ID',
'CONSISTENCY_STD_VOTO']”.
```

- Replayed Fitness/ Aligned Fitness: queste due variabili rappresentano il grado di aderenza del percorso seguito dagli studenti rispetto al percorso ideale del

primo anno. Il codice utilizzato per la creazione di queste due variabili viene spiegato nel paragrafo 4.4 dedicato al Conformance Cheching

4.3 MODELLI DI PROCESSO

Come spiegato nei cenni teorici i modelli di processo sono degli strumenti che ci permettono di comprendere e sintetizzare e rappresentare la struttura dei processi presenti all'interno di un'event log, attraverso reti di Petri e altre notazioni. Nel nostro caso siamo interessati ad osservare i modelli di processo creati sui percorsi degli studenti per due motivi:

- Il primo motivo è di natura più generale ed esplorativa. Siccome siamo interessati ad analizzare i comportamenti degli studenti nel primo anno, avere un modello di processo ci permette di acquisire informazioni su tendenze, varianti di processo frequenti e comportamenti ricorrenti.
- Il secondo motivo è propedeutico alla generazione degli Instance Graph (grafi diretti) la cui creazione verrà spiegata successivamente. L'idea è quella di capire dai modelli di processo se ci sono evidenze relative alla concorrenza tra due o più esami in modo tale da poter utilizzare questa regola di concorrenza nella creazione degli Instance Graphs.

Di seguito verrà spiegata la metodologia seguita per la creazione dei modelli di processo. Con le operazioni spiegate fino a questo punto, abbiamo creato due dataset adatti ad essere utilizzati per la creazione dei modelli di processo. Il primo dataset mostra per ogni studente la sequenza degli esami superati, mentre il secondo mostra la sequenza degli esami tentati, di cui l'esito può essere sia positivo che negativo. Ognuno di questi dataset è composto da studenti che si sono laureati in tempi ottimali

(“STATO_LAUREA” = “Early”), massimo un anno in più rispetto agli ottimali (“STATO_LAUREA” = “One_year_late”) e più di un anno in più rispetto agli ottimali (“STATO_LAUREA” = “Late”). Per ogni categoria di studente e per i dataset completi, abbiamo applicato la medesima metodologia utilizzando due diversi algoritmi (Inductive ed Heuristic miner). Il codice implementato in Python è il seguente[6]:

“result0 = pd.DataFrame(columns=['Threshold','Fitness','Precision', 'Generalization', 'Semplicity'])

```
for k in np.arange(0,1.1,0.1):
    mod = pm4py.discovery.discover_process_tree_inductive(df_merged_caricato, k)
    net, initial_marking, final_marking = pm4py.convert_to_petri_net(mod)
    fitness = pm4py.fitness_token_based_replay(df_merged_caricato, net, initial_marking, final_marking)
    if __name__ == "__main__":
        prec = pm4py.precision_token_based_replay(df_merged_caricato, net, initial_marking, final_marking)
        gen = generalization_evaluator.apply(df_merged_caricato, net, initial_marking, final_marking)
        simp = simplicity_evaluator.apply(net)
    row = pd.DataFrame({'Threshold':k,'Fitness':fitness['average_trace_fitness'],
                        'Precision':prec, 'Generalization':gen, 'Semplicity':simp},index=[0])
```

[6] Nell’esempio utilizziamo “df_merged_caricato”, ovvero il dataframe contenente gli esami superati e non superati.

```
    result0 = pd.concat([result0, row])


tree = pm4py.discovery.discover_process_tree_inductive(df_merged_caricato,
best_k_in_result0)
net, initial_marking, final_marking = pm4py.convert_to_petri_net(tree)
pm4py.view_petri_net(net, initial_marking, final_marking)".
```

Il codice Python fornito si compone di due parti: la prima parte serve per capire quale parametro k scegliere per creare un modello di processo che non semplifichi troppo i dati ma che allo stesso tempo rappresenti la realtà senza troppe distorsioni; la seconda parte serve a creare la rete di Petri per i miglior parametro k trovato, e a visualizzarla. Per fare ciò abbiamo creato un ciclo for che crea 10 modelli di processo attraverso l'Inductive Mincr iterando il parametro k in un range da 0.1 a 1. Per ognuna di queste iterazioni abbiamo creato un Process Tree che è stato convertito in rete di Petri. Su ogni rete di Petri sono state calcolate quattro misure: Precision, Fitness, Semplicity e Generalization, che sono state inserite all'interno di un dataframe. Alla fine di ogni iterazione queste misure sono state inserite all'interno del dataframe "result0" per essere ispezionate. Il k scelto è quello che corrisponde alla migliore combinazione di questi quattro valori. Non c'è una regola precisa che ci dice quale sia il miglior k; quindi, ci siamo affidati alla nostra opinione, cercando di trovare una soluzione che rappresenti il dataset ma che sia comunque comprensibile. Con le ultime tre righe di codice viene visualizzata l'immagine della rete di Petri estratta con l'Inductive Miner.

Se al posto di “discover_process_tree_inductive” utiliziamo “discover_heuristics_net” utiliziamo l’algoritmo Heuristic Miner, che come spiegato nei cenni teorici, utilizza altre regole per creare modelli di processo. Con questo algoritmo di discovery abbiamo anche la possibilità di visualizzare l’Heuristic net con il seguente codice:

“tree = pm4py.discovery.discover_process_tree_inductive(df_merged_cluster0, 0.9)
pm4py.view_heuristics_net(tree)”.

In quest’analisi è stato utilizzato anche il Fuzzy miner come processo di discovery, ma che non verrà mostrato nella metodologia in quanto viene fornito direttamente dal software “Disco” una volta caricato il dataset/event log.

4.4 CONFORMANCE CHECHING

Il Conformance Checking è un'altra tecnica di Process Mining impiegata in questo lavoro e spiegata nei cenni teorici. Come anticipato questa tecnica è stata utilizzata per mostrare il grado con cui gli studenti hanno rispettato il percorso ideale delineato dall’università per gli studenti del primo anno. La fitness viene calcolata inizialmente per ogni singolo studente viene poi utilizzata per mostrare la distribuzione di questi valori relativamente alle categorie di studenti (“Early”, “One_year_late” e “Late”). A differenza dei modelli di processo, per il calcolo di questa variabile è stato utilizzato solo il dataset relativo agli esami superati nel primo anno, che prende il nome di “df_esami_filtrato”.

“filtrato_log = df_esami_filtrato[['STU_ID', 'DES', 'DATA_SUP']].copy()

filtrato_log.columns = ['case_id', 'activity', 'timestamp']

filtrato_log['timestamp'] = pd.to_datetime(filtrato_log['timestamp'])

filtrato_log = filtrato_log.sort_values(by=['case_id', 'timestamp'])

log = pm4py.format_dataframe(filtrato_log, case_id='case_id',

activity_key='activity', timestamp_key='timestamp').

net, initial_marking, final_marking = pnml_importer.apply(petri_net_cc_path) #

importo petrinet

pm4py.view_petri_net(net, initial_marking, final_marking)".

Il codice sopra citato trasforma "df_esami_filtrato" in event log contenente come case id la matricola degli studenti, come activity il nome dell'esame e come time-stamp la data di superamento dell'esame. I dati vengono poi ordinati per case id e time-stamp. Viene infine importata la rete di Petri[7].

"replayed_traces = pm4py.conformance_diagnostics_token_based_replay(log, net,

initial_marking, final_marking)

case_ids = filtrato_log['case_id'].unique()

fitness_replayed_values = []

for result in replayed_traces:

    fitness = result['trace_fitness']

    fitness_replayed_values.append(fitness)

dict_replayed_fitness = {key:value for key,value in

zip(case_ids,fitness_replayed_values)}".

[7] La rete di Petri in questione viene mostrata nel capitolo 6

Questo blocco di codice esegue un'analisi di conformità basata sulla riproduzione di tracce di un log rispetto a un modello di processo, come spiegato nei cenni teorici. Successivamente crea un dizionario che associa gli identificativi dei casi ("case_id") ai valori di fitness delle tracce riprodotte. Come nel caso delle variabili continue i valori vengono inseriti in un dizionario che verrà poi trasformato in un dataframe per effettuare il join su "df_anagrafiche".

"aligned_traces = pm4py.conformance_diagnostics_alignments(log, net, initial_marking, final_marking)

fitness_alligned_values = []

for result in aligned_traces:

    fitness = result['fitness']

    fitness_alligned_values.append(fitness)

dict_alligned_fitness1 = {key:value for key,value in zip(case_ids,fitness_alligned_values1)}".

Questo blocco invece esegue anch'esso un'analisi di conformità ma basata stavolta sugli allignments, spiegati nei cenni teorici. Come nel caso precedente anche qui i valori vengono inseriti all'interno di un dizionario per essere trasformati in nuove features in "df_anagrafiche".

4.5 CREAZIONE GRAFI PER LA RAPPRESENTAZIONE DEL COMPORTAMENTO DEGLI STUDENTI

4.5.1 *Instance Graphs*

Gli instance graphs sono dei grafici diretti che rappresentano un'istanza di comportamento individuata nel dataset. Ogni instance graph costruito rappresenta

quindi il comportamento di uno o più studenti attraverso una struttura a grafo contenente una serie di nodi e archi direzionali, che possono quindi catturare rapporti di sequenzialità e parallelismo tra esami sostenuti[8] dagli studenti nel primo anno. L'algoritmo creato per la creazione degli instance graphs è il seguente e verrà spiegato dettagliatamente in seguito:

```
"def create_exam_pairs_merged(df_merged_caricato):
    student_data = {}
    for stu_id, group in df_merged_caricato.groupby('STU_ID'):
        group = group.sort_values('DATA_APP')
        exams = group['DESC_FINALE'].tolist()
        data_sup = group['DATA_APP'].apply(pd.to_datetime).tolist()
        study_starts = group['DATA_START'].apply(pd.to_datetime).tolist()
        graph = nx.DiGraph()
        for idx, exam in enumerate(exams):
            graph.add_node(idx, label=exam)
        non_restrictive_activities = ["Fine primo anno-Fine primo anno", "Fine primo semestre-Fine primo semestre", "Start-Start", "End-End"]
                for i in range(len(exams)):
            for j in range(i + 1, len(exams)):
                if exams[i] in non_restrictive_activities or exams[j] in non_restrictive_activities:
                    graph.add_edge(i, j)
```

[8] E non superati, in quanto viene usato "df_merged_caricato" e non "df_esami_filtrato".

```
            elif not check_concurrency_merged(study_starts[i], data_sup[i],
study_starts[j], data_sup[j], exams[i], exams[j]):
                graph.add_edge(i, j)
        reduced_graph = transitive_reduction(graph)
        label_dict = {node: graph.nodes[node]['label'] for node in graph.nodes()}
        nx.set_node_attributes(reduced_graph, label_dict, 'label')
        student_data[stu_id] = reduced_graph
    return student_data".
```

Le funzioni "create_exam_pairs_merged" e "check_concurrency_merged" lavorano insieme per analizzare le attività di esame degli studenti e rappresentarle come grafi diretti. La concorrenza è stata definita facendo un'assunzione, ovvero che un esame necessita circa di un mese per essere preparato. Ciò significa che, se due esami vengono dati a distanza di meno di un mese l'uno dall'altro possono essere considerati concorrenti in quanto per un periodo lo studio dei due si è sovrapposto.

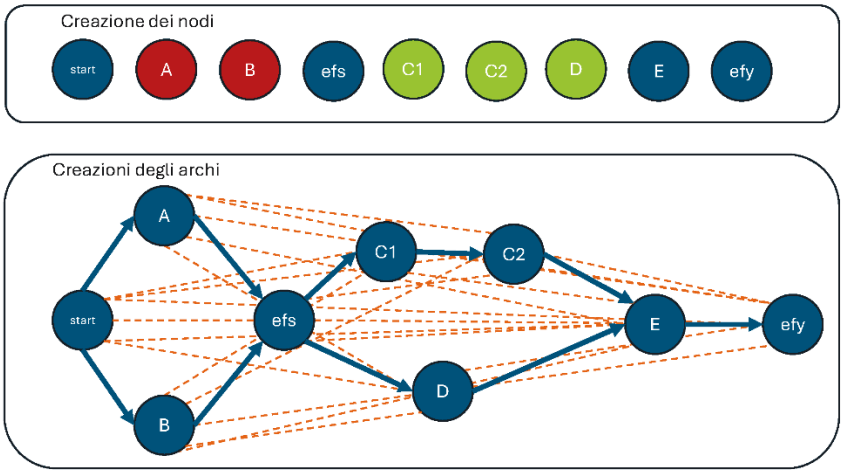


Figura 4.1 Esempio del processo di creazione degli Instance Graphs

Il processo esemplificato in Fig. 4.1 funziona nel seguente modo:

- Preparazione dei dati: La funzione create_exam_pairs_merged inizia prendendo i dati relativi agli esami degli studenti, contenuti in un DataFrame, e li raggruppa per ogni studente (STU_ID). All'interno di ogni gruppo, le attività vengono ordinate cronologicamente sulla base della data di esecuzione (DATA_APP). Questo garantisce che tutte le analisi successive rispettino l'ordine temporale in cui gli esami sono stati effettivamente sostenuti.
- Costruzione del Grafo Diretto: Dopo aver ordinato le attività, la funzione costruisce un grafo diretto (DiGraph) per ciascun studente. In questo grafo, ogni attività di esame è rappresentata come un nodo, con un'etichetta che descrive l'attività (DESC_FINALE). I nodi vengono identificati con indici che seguono l'ordine temporale delle attività. Nella parte superiore della figura i nodi sono ordinati cronologicamente.
- Aggiunta degli Archi e Verifica della Concorrenza: Gli archi, che rappresentano le relazioni tra le attività, vengono aggiunti al grafo sulla base di regole di concorrenza. Alcune attività, come "Fine primo anno", "Fine primo semestre", "Start" e "End", sono considerate non restrittive e quindi sempre collegate a tutte attività successive, non solo a quelle immediatamente successive.

  Per tutte le altre attività, la funzione “check_concurrency_merged” viene chiamata per determinare se due attività possono considerarsi concorrenti, ovvero se avvengono nello stesso intervallo di tempo. Per ogni coppia di attività questa funzione esamina sia le descrizioni delle attività (Esame) che i loro intervalli di tempo: a) Se le descrizioni indicano attività diverse e sono a

distanza di meno di un mese l'una dall'altra, sono considerate concorrenti. In tal caso, non viene creato alcun arco diretto tra di esse nel grafo, indicando che queste attività non sono sequenziali ma avvengono in parallelo. b) Se le descrizioni indicano attività diverse ma sono distanti più di un mese l'una dall'altra, non sono concorrenti e viene aggiunto un arco, che rappresenta una relazione temporale diretta tra le due attività. c) Se le descrizioni indicano attività uguali a prescindere dall'intervallo temporale che le separano verranno considerate non concorrenti e verrà aggiunto un arco che le collega.

- Riduzione Transitiva: Una volta che tutti gli archi sono stati aggiunti, viene applicata una riduzione transitiva al grafo. Questo processo semplifica il grafo rimuovendo gli archi ridondanti che possono essere impliciti in altri percorsi all'interno del grafo, mantenendo solo le relazioni essenziali tra le attività.
- Memorizzazione dei Grafi: Infine, il grafo ridotto per ciascuno studente viene memorizzato in un dizionario, con l'identificativo dello studente (STU_ID) come chiave.

4.5.2 *Sottografi*

Una volta costruiti gli Instance Graphs si sfrutta l'algoritmo SUBDUE per l'estrazione dei comportamenti più rilevanti che prenderanno il nome di sottografi (SUBs). L'algoritmo SUBDUE è una tecnica di clustering gerarchico che individua parti di grafo frequenti tra gli Instance Graphs di partenza e li ordina in una struttura in base alla loro rilevanza e al loro livello di astrazione. In questa struttura i pattern

più comuni e generali si trovano al livello superiore, mentre quelli più specifici e meno frequenti sono a livelli inferiori. Questa tecnica di clustering esplora iterativamente gli Instance Graphs. Ogni iterazione del processo identifica una SUB rilevante che viene compressa in un singolo nodo, il processo continua finché non sono possibili altre compressioni. Questi pattern vengono utilizzati nel nostro caso per analizzare i comportamenti degli studenti relativamente agli esami sostenuti. Una volta trovati le SUB è possibile valutarli utilizzando quattro misure principali:

- Diversity: misura l'eterogeneità trai cluster delle SUB. È calcolata sul numero di modifiche necessarie per trasformare un cluster in un altro. Un'alta diversità indica che le SUB rappresentano comportamenti distinti e quindi a rigor di logica sono più utili in quanto meno sovrapposti.
- Coverage: indica quanto il dataset iniziale, e quindi il set di Instance Graphs è rappresentato dalle SUB scoperte. Una copertura elevata indica che i pattern scoperti rappresentano una parte significativa dei processi osservati.
- Rapresentativeness: definita su ogni SUB, misura la percentuale di Instance Graphs che contengono almeno una volta quel comportamento.
- Frequency. Misura il numero di occorrenze di un SUB all'interno del set di Instance Graphs. È diverso dalla rappresentatività in quanto se un comportamento è contenuto più volte all'interno di un Instance Graphs viene contato tutte le volte in cui è presente e non solo una volta.

Queste misure aiutano a garantire che i pattern scoperti siano sia rilevanti che rappresentativi dei comportamenti osservati.

4.6 CREAZIONE DEI CLASSIFICATORI E VISUALIZZAZIONE DEI RISULTATI

In questo capitolo procederemo alla spiegazione della metodologia implementata per la creazione dei modelli predittivi. Come anticipato gli algoritmi utilizzati sono stati: SVM, Decision Tree, Random Forest e XGBoost. Uno degli scopi di questo lavoro è quello di confermare l'utilità delle variabili comportamentali nella predizione delle performance degli studenti; per farlo abbiamo seguito la seguente procedura: a) abbiamo utilizzato solo le variabili continue per la predizione e osservato i risultati per tutti e quattro i modelli. b) abbiamo aggiunto le variabili comportamentali sotto forma di variabili binarie e osservato se le prestazioni dei modelli miglioravano. c) abbiamo utilizzato i risultati dei modelli predittivi e un'explainable AI per confermare l'utilità/inutilità di queste SUB.

4.6.1 *Pre-processing*

Prima di procedere con la creazione dei modelli predittivi, sono state eseguite una serie di operazioni sul dataset iniziale "df_anagrafiche", al quale sono state aggiunte tutte le variabili continue create e le SUB. Il seguente codice mostra il processo di pulizia del dataset, partendo dall'eliminazione di osservazioni con valori nulli nella variabile dipendente ("STATO_LAUREA"). Successivamente, sono state eliminate colonne non utili, come identificativi e codici, oltre a variabili altamente correlate con quelle mantenute. Inoltre, sono state rimosse alcune variabili non rilevanti per singoli esperimenti, ma che potrebbero essere utilizzate in fasi successive. I valori mancanti in sei variabili sono stati sostituiti con 0:

“data = df_anagrafiche.dropna(subset=['STATO_LAUREA'])

data_updated = data.drop(columns=columns_to_remove)

data_updated["CREDS_Y1"].fillna(0, inplace=True)

data_updated["CREDS_s1"].fillna(0, inplace=True)

data_updated["CREDS_s2"].fillna(0, inplace=True)

data_updated["NUM_NOT_PASSED"].fillna(0, inplace=True)

data_updated["EX_PASSED_S1"].fillna(0, inplace=True)

data_updated["EX_PASSED_S2"].fillna(0, inplace=True)”.

Successivamente, le etichette relative ai tempi di laurea sono state modificate. Tutti gli studenti considerati in ritardo di un anno (One_year_late) sono stati inseriti nella categoria dei ritardatari (Late), in modo da bilanciare meglio le classi e semplificare il compito di predizione. Infine, sono stati creati i dataset “X” e “y”, dove X rappresenta il dataset contenente le variabili indipendenti utilizzate per la predizione, ossia le variabili comportamentali e altre caratteristiche degli studenti, mentre y contiene la variabile dipendente (STATO_LAUREA).

“data_updated['STATO_LAUREA'] =

data_updated['STATO_LAUREA'].replace('One_year_late', 'Late')

X = data_updated.drop("STATO_LAUREA", axis=1).copy()

feature_names = X.columns.tolist()

y = data_updated["STATO_LAUREA"].copy()

dopo la creazione di questi due dataset, si procede con l’imputazione dei valori nulli superstiti. Per queste altre variabili abbiamo considerato più opportuno utilizzare valori medi invece che 0 per sostituire quelli mancanti.

“Float_col_missing_val =

["MEAN_GRADES","CONSISTENCY_STD","MEAN_INTERVAL","CONSISTE

NCY_STD_GRADES","HIGH_SCHOOL_GRADE"]

imputer = SimpleImputer(strategy='mean')

X[Float_col_missing_val] = imputer.fit_transform(X[Float_col_missing_val])”.

A questo punto non ci manca che associare 0 agli studenti “Early” e 1 ai “Late”, creare test e training set rispettando la proporzione di 0.3/0.7 e standardizzare le prime 12 colonne che sono quelle continue[9].

“y[y == "Early"] = 0

y[y =="Late"] = 1

y=y.astype(int)

X_train, X_test, y_train, y_test = train_test_split(X, y,

test_size=0.3,random_state=109)

scaler = StandardScaler()

X_train.iloc[:, :12] = scaler.fit_transform(X_train.iloc[:, :12])

X_test.iloc[:, :12] = scaler.fit_transform(X_test.iloc[:, :12])”.

4.6.2 *Hyperarameter tuning*

Come abbiamo anticipato prima, “colums_to_remove” è una lista di variabili che togliamo dal dataset prima di finire il pre-processing e di cominciare la predizione vera e propria. Dato che in “df_anagrafiche” sono presenti le SUB, e dato che

[9] Tutte le altre variabili dopo la dodicesima sono variabili binarie che rappresentano i comportamenti degli studenti, quindi non necessitano di essere standardizzate

vogliamo cominciare la predizione senza utilizzare le SUB, in un primo momento il nome di queste sarà contenuto nella lista “colums_to_remove”. Una volta creati i modelli e ottenuti i risultati senza SUB, le togliamo dalla lista in modo che siano presenti nel dataset “X” e ripetiamo la procedura. Di seguito sarà presentato il codice utilizzato per entrambi i set di variabili su ognuno dei quattro modelli. La procedura è stata la stessa per ognuno dei modelli:

1. Creazione di matrici contenenti una serie di valori per ogni hyperparametro specifico del modello.
2. Utilizzo della Grid search per creare una serie di modelli addestrati sui dati di train, ognuno con una combinazione diversa degli hyperparametri presenti nella matrice. La grid search crea una serie di modelli dei quali controlla le performance tramite 5 fold cross-validation. In base ai risultati ottenuti, vengono individuati e salvati i parametri del miglior modello.
3. Addestramento dei modelli su tutti i dati di train utilizzando i migliori hyperparametri trovati tramite grid search.
4. Calcolo delle capacità predittive del modello sulle osservazioni del test set.

1.

```
param_grid_SVM = {
    'C': [0.1, 1, 10, 100],
    'kernel': ['linear', 'rbf', 'poly'],
    'gamma': ['scale', 'auto']
}
```

2.

```
grid_search_SVM = GridSearchCV(SVC(random_state=110), param_grid_SVM,
cv=5, n_jobs=1, scoring='accuracy')
grid_search_SVM.fit(X_train, y_train)
```

3.

```
best_model_SVM = SVC(
    C=best_params_SVM['C'],
    gamma =best_params_SVM['gamma'],
    kernel=best_params_SVM['kernel'],
    random_state=110)
best_model_SVM.fit(X_train, y_train)
```

4.

```
y_pred_SVM_ALL = best_model_SVM.predict(X_test)
accu_SVM_ALL = accuracy_score(y_test, y_pred_SVM_ALL)
```

Per semplicità mostreremo solo la parte di codice relativa all' SVM in quanto per gli altri classificatori è stata implementata la stessa metodologia. L'unica differenza sta nelle matrici dei parametri utilizzate. Di seguito le matrici dell' Decision Tree, Random Forest e XGBoost:

```
param_grid_DT = {
    'criterion': ['gini', 'entropy'],
```

```
    'splitter': ['best', 'random'],
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 5, 10]}

param_grid_RF = {
    'n_estimators': [50, 100, 200],
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 5, 10],
    'bootstrap': [True, False]
param_grid_XGB = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 6, 10],
    'learning_rate': [0.01, 0.1, 0.2],
    'subsample': [0.8],
    'colsample_bytree': [0.6, 0.8, 1.0]}
```

4.6.3 *Misure per valutare l'importanza delle feature*

Per tutti e quattro i modelli e per i due set di variabili sono state visualizzate le performance in termini di accuracy[10] e sono state confrontate in un istogramma. Per il modello migliore sono state poi individuate l'importanza delle feature in due modi diversi. Il primo metodo utilizzato è quello fornito dalla funzione "xgb.feature_importances". Questa funzione restituisce un valore relativo a ogni variabile, il "gain", che riflette il miglioramento apportato da una feature quando è usata per separare i dati dei nodi del decision tree. Il miglioramento apportato può essere visto come aumento dell'accuracy o diminuzione della log loss. Il secondo metodo consiste nell'utilizzo di valori SHAP. Questi valori offrono un livello di interpretazione più dettagliato. Non solo forniscono informazioni globali, ma spiegano anche come ogni variabile ha contribuito sulle singole predizioni. Dato che il miglior modello alla fine degli esperimenti è stato XGBoost, mostriamo il codice relativo a quel modello per la creazione dell'importanza delle feature secondo le due metodologie spiegate.

```
"best_model_XGB = xgb.XGBClassifier(
    n_estimators=200,
    max_depth=3,
    learning_rate=0.01,
    subsample=0.8,
    colsample_bytree=0.8,
    use_label_encoder=False,
```

[10] Ricordiamo che per valutare le performance dei modelli è stata utilizzata l'accuracy poiché le classi "Early" e "Late" sono bilanciate.

```
    eval_metric='logloss',
    random_state=110)


XGB_importance
best_model_XGB.fit(X_train, y_train)
importances = best_model_XGB.feature_importances_
best_model_XGB_importances = pd.Series(importances, index=feature_names)
best_model_XGB_importances =
best_model_XGB_importances.sort_values(ascending=False)
best_model_XGB_importances /= best_model_XGB_importances.sum()
cumulative_importances = best_model_XGB_importances.cumsum()


SHAP
best_model_XGB.fit(X_train, y_train)
explainer = shap.Explainer(best_model_XGB)
shap_values = explainer.shap_values(X_train)”
```

Capitolo 5 MODELLI DI PROCESSO

Questo capitolo si concentra sull'analisi dei modelli di processo estratti tramite Inductive ed Heuristic miner. Questi modelli verranno creati per ognuna delle tre categorie di studenti che abbiamo, in base ai tempi di laurea, ovvero "Early","One year late" e "Late". Lo scopo di questo capitolo è indagare sulle differenze e similarità comportamentali tra queste categorie di studenti. Per creare i modelli di processo abbiamo usato solo "df_merged_caricato", che rappresenta il dataset con gli stati relativi agli esami. La metodologia è la seguente, dopo la valutazione del miglior parametro K, che regola il filtraggio dei processi rappresentati nel modello in base alla frequenza, abbiamo costruito il modello di processo. I primi risultati che vengono mostrati sono i modelli di processo e i parametri relativi all'Inductive miner, i secondi sono relativi all'Heuristic miner. Il dataset di partenza "df_merged_caricato" è stato suddiviso in base alla categoria di studente in "df_merged_caricato_early","df_merged_caricato_one_year_late" e "df_merged_caricato_late". Per ogni tipo di modello di processo sono stati utilizzati tutti e tre i dataset.

5.1 PETRI NET OTTENUTE TRAMITE L'INDUCTIVE MINER

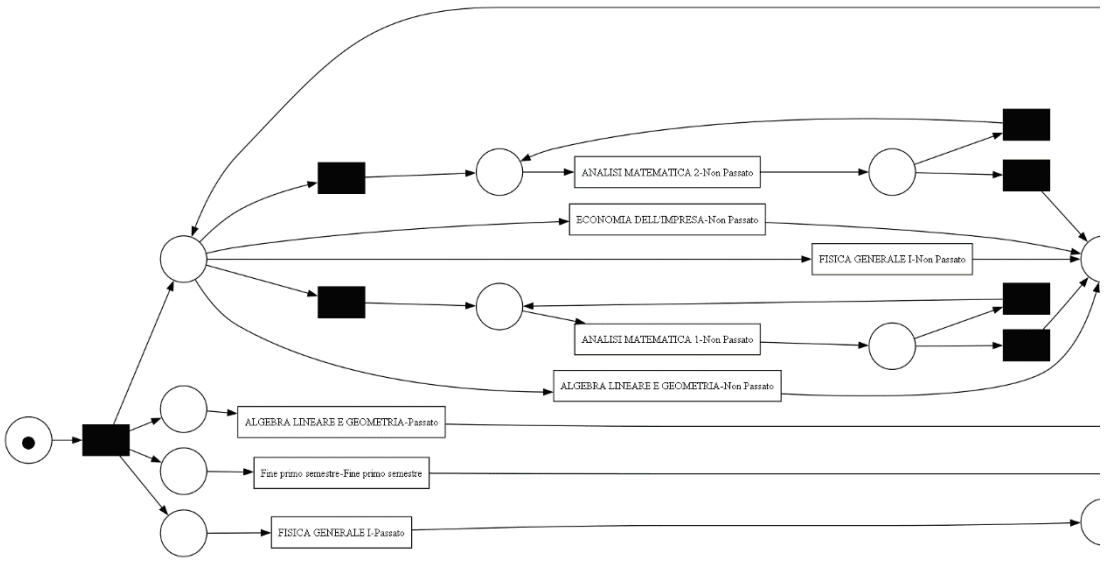
5.1.1 *Studenti early*

Nella tabella 5 sono riportati il parametro K e quattro metriche relative ai 10 modelli di processo creati per gli studenti Early. Ricordiamo che il miglior modello è quello caratterizzato dal miglior trade-off tra le quattro metriche; pertanto, è raro se non impossibile trovare un modello oggettivamente migliore degli altri.

Threshold	Fitness	Precision	Generalization	Semplicity
0	1	0.344348	0.864076	0.610738
0.1	1	0.397652	0.86295	0.605442
0.2	0.992433	0.480466	0.880777	0.62963
0.3	0.967122	0.461998	0.766045	0.621622
0.4	0.751758	0.47976	0.791356	0.6
0.5	0.713149	0.511254	0.781062	0.642857
0.6	0.5944	0.43036	0.750683	0.632653
0.7	0.644313	0.557911	0.771107	0.670886
0.8	0.735025	0.678607	0.784054	0.634409
0.9	0.636634	0.506331	0.707531	0.642105
1	0.891628	0.615261	0.577099	0.652174

Tabella 5 Parametro "k" e metriche relative ai modelli di processo degli studenti Early (Inductive miner)

Dalla tabella 5 i1 modello scelto per essere visualizzato è quello con k = 0.8. Questo modello presenta un valore discreto di fitness rispetto ai dati, il più alto livello di precision tra tutti i modelli considerati, e una generalizzazione di livello medio rispetto agli altri. Il livello di semplicità, invece, risulta basso per tutti i modelli analizzati; pertanto, non è significativo considerare questo aspetto nella valutazione complessiva.



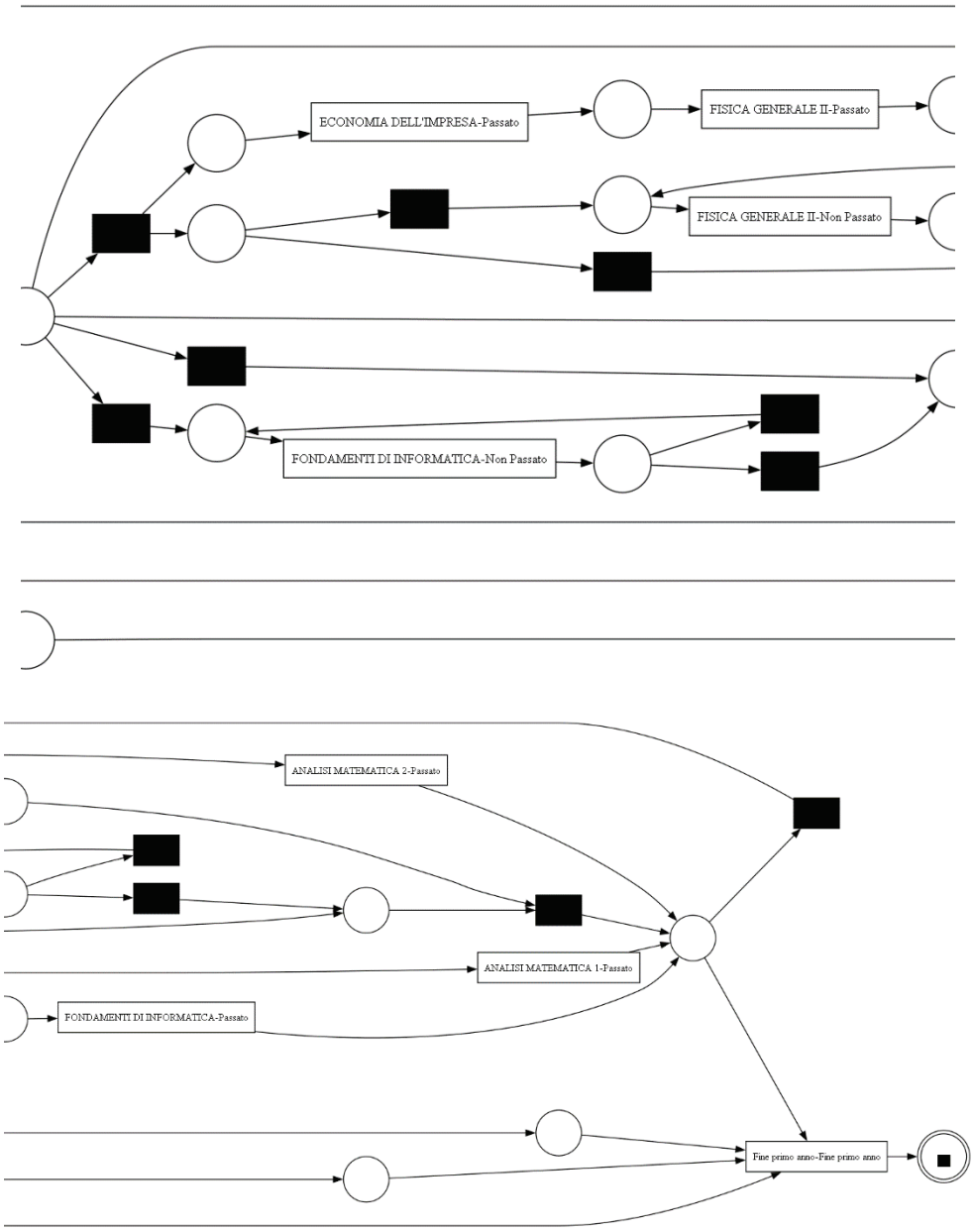


Figura 5.1 Rete di Petri Inductive miner studenti Early. k = 0.8

Nonostante si stia parlando dei migliori studenti in termini di performance, notiamo che anche qui sono presenti loop su “Analisi Matematica 1”, “Analisi Matematica 2”, “fondamenti di Informatica” e “Fisica Generale 2”. Questi cicli indicano che gli esami citati non sono stati passati più di una volta da alcuni di questi studenti.

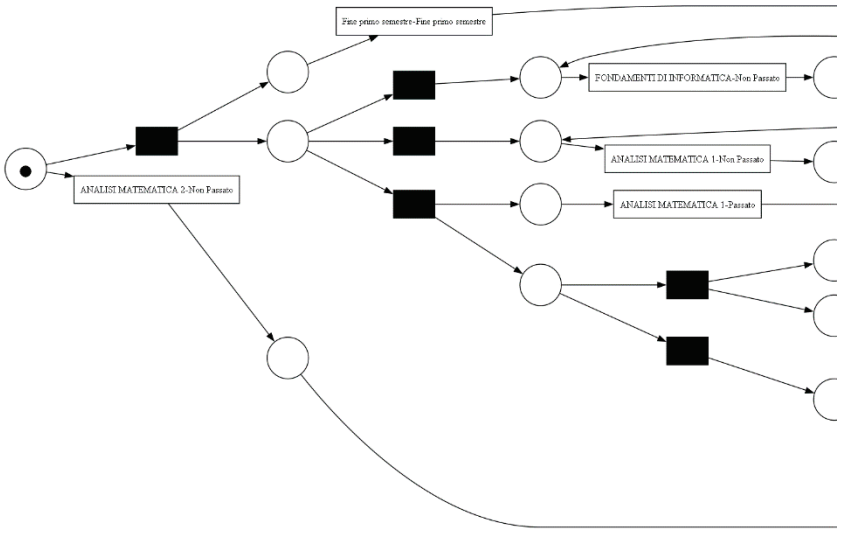
5.1.2 *Studenti One year late*

La seguente è la stessa tabella di prima ma realizzata per gli studenti One year late.

Threshold	Fitness	Precision	Generalization	Semplicity
0	1	0.380277	0.782739	0.622378
0.1	0.996415	0.493215	0.833996	0.623188
0.2	0.889084	0.810245	0.808098	0.625
0.3	0.954302	0.536138	0.795512	0.635036
0.4	0.951295	0.56729	0.78145	0.621622
0.5	0.7461	0.629281	0.786629	0.656566
0.6	0.697415	0.630555	0.778606	0.677419
0.7	0.783782	0.739444	0.771369	0.708333
0.8	0.685324	0.657535	0.679454	0.684211
0.9	0.700647	0.628668	0.66484	0.684211
1	0.606609	0.633242	0.64333	0.555556

Tabella 6 Parametro "k" e metriche relative ai modelli di processo degli studenti One year late (Inductive miner)

In questo caso la semplicità dei dieci modelli ha un range più ampio rispetto a prima. Per questo motivo è stato scelto il modello ottenuto con K = 0.7. Il modello infatti è quello che rappresenta in maniera più semplice il comportamento di questi individui, ma allo stesso tempo mostra buoni livelli di fitness, precision e generalization, ognuna attorno a 0.75.



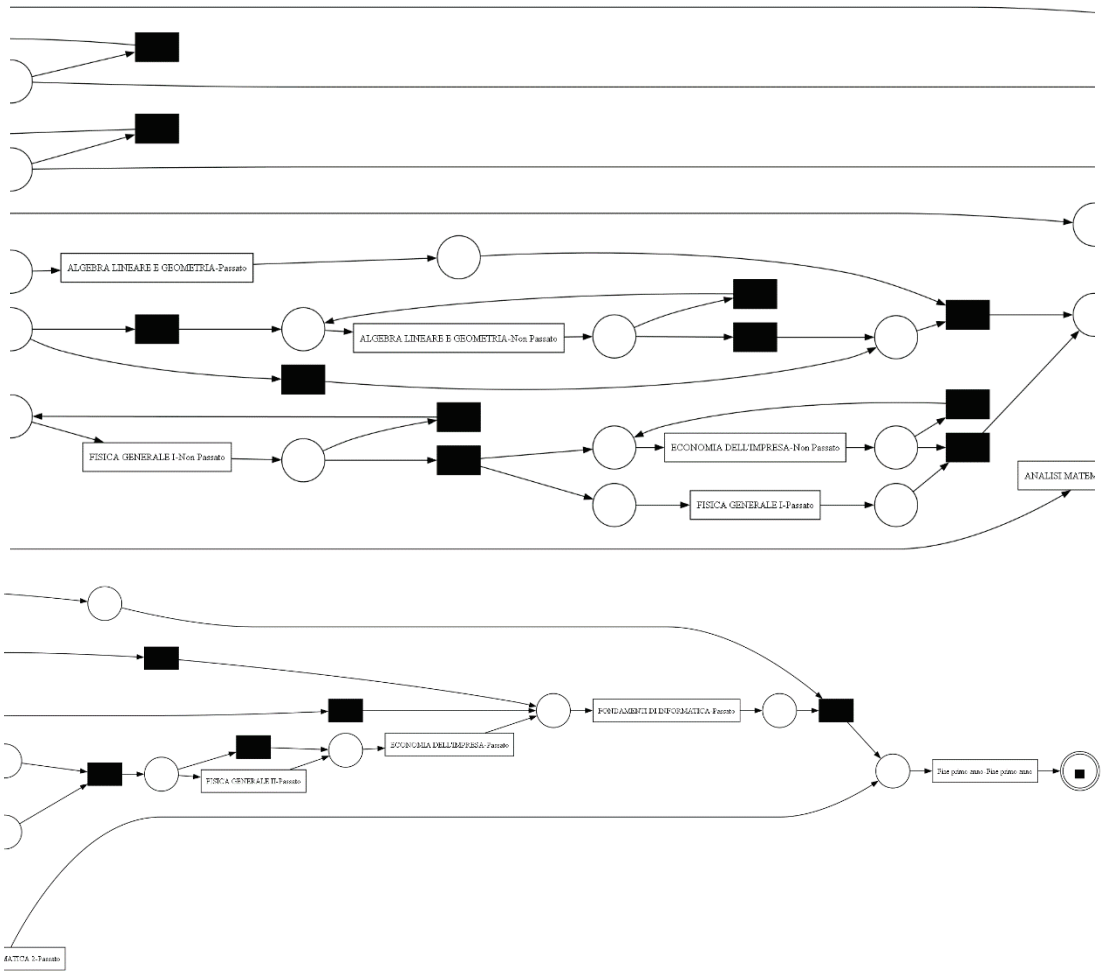


Figura 5.2 Rete di Petri Inductive miner studenti One year late. k = 0.7

Anche in questo caso, emerge una serie di esami ripetuti più volte dagli studenti, tra cui "Fondamenti di Informatica", "Analisi Matematica 1", "Algebra Lineare e Geometria", "Fisica Generale 1" ed "Economia dell'Impresa". Tuttavia, a differenza del modello di processo relativo agli studenti Early, in questo caso il modello consente di concludere il primo anno accademico senza aver sostenuto alcun esame, o avendone sostenuti pochi, in alcuni casi senza nemmeno averli superati.

Sebbene la differenza non sia particolarmente evidente, i comportamenti degli studenti della categoria One year late mostrati dalla rete di Petri appaiono leggermente peggiori rispetto a quelli della categoria Early. Questo suggerisce che il

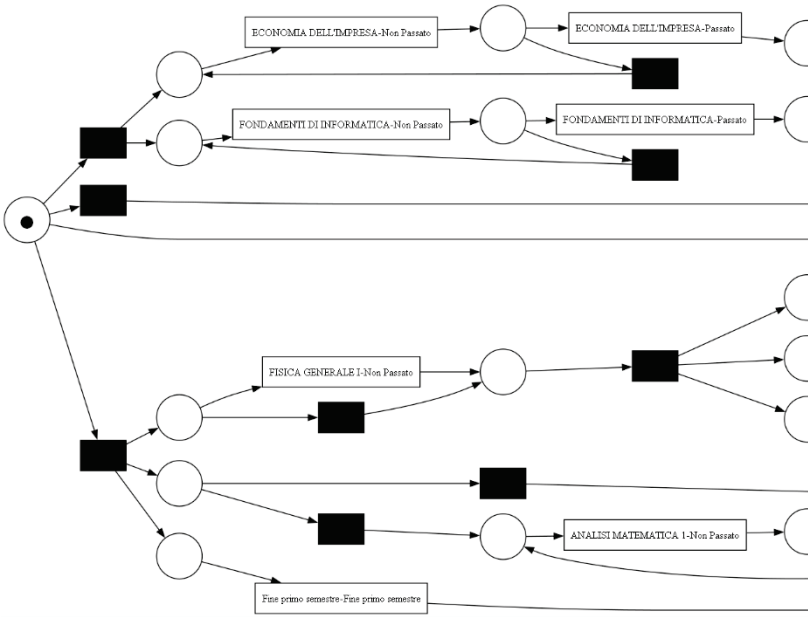
comportamento degli studenti durante il primo anno di studi possa avere un impatto significativo sui tempi di conseguimento del titolo.

5.1.3 *Studenti Late*

Threshold	Fitness	Precision	Generalization	Semplicity
0	1	0.49607	0.799774	0.627586
0.1	0.999779	0.476484	0.803171	0.62963
0.2	0.974962	0.544266	0.805319	0.642857
0.3	0.916473	0.433536	0.663922	0.619048
0.4	0.791847	0.625689	0.777918	0.662338
0.5	0.781478	0.588044	0.744638	0.647059
0.6	0.740228	0.587872	0.729121	0.641026
0.7	0.740228	0.587872	0.729121	0.641026
0.8	0.715621	0.588677	0.701786	0.633803
0.9	0.729562	0.569256	0.679545	0.625
1	0.947564	0.717581	0.587378	0.692308

Tabella 7 Parametro "k" e metriche relative ai modelli di processo degli studenti Late (Inductive miner)

Il modello di processo selezionato per questa categoria di studenti è quello con K = 0.7.



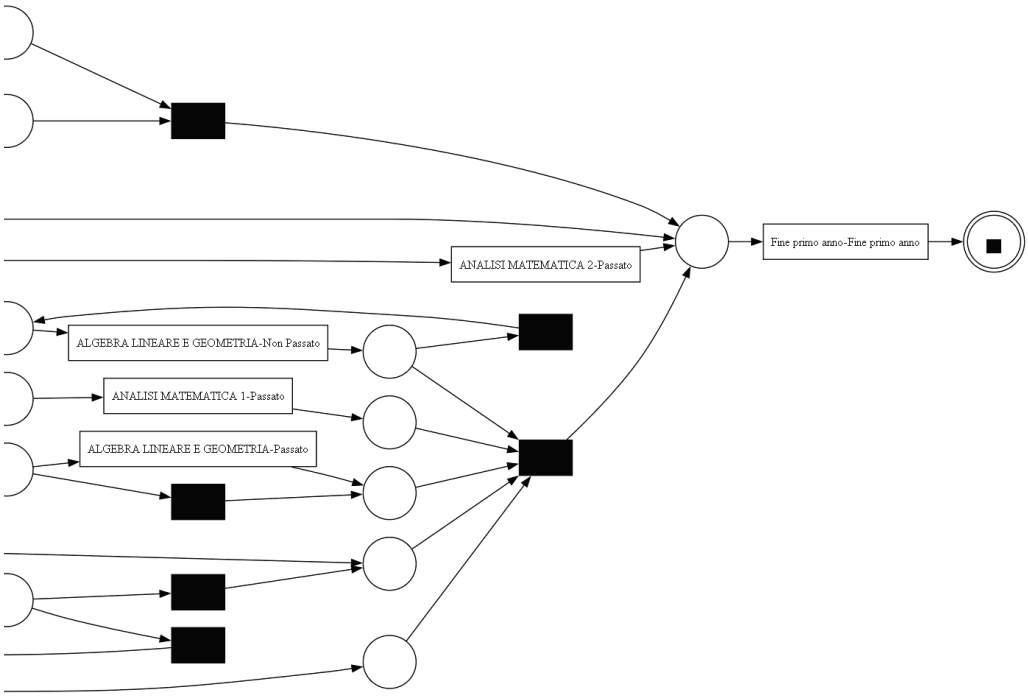


Tabella 8 Rete di Petri Inductive miner studenti Late. k = 0.7

Il modello di processo, come nei casi precedenti, contiene alcuni cicli relativi a esami non superati, tra cui “Economia delle imprese”, “Fondamenti di Informatica”, “Analisi Matematica 1” e “Algebra Lineare e Geometria”. Analogamente agli studenti della categoria One year late, anche in questo caso alcuni studenti hanno concluso il primo anno senza aver superato alcun esame o, in alcuni casi, senza averne sostenuti. Il fatto che questa categoria di studenti abbia sostenuto un numero ridotto di esami è evidente dal numero inferiore di attività presenti nel modello di processo, che risulta decisamente più scarno rispetto alle altre due categorie. Questo riflette una partecipazione limitata agli esami da parte di questi studenti durante il loro percorso accademico.

5.2 HEURISTIC NET OTTENUTE TRAMITE L'HEURISTIC MINER

I modelli di processo ottenuti tramite l'Inductive Miner forniscono alcune informazioni sul comportamento delle diverse categorie di studenti, ma senza un livello di dettaglio sufficiente. Per ottenere un'analisi più approfondita, si procede con la costruzione di un altro tipo di modello di processo, chiamato Heuristic Net, utilizzando l'Heuristic Miner. La metodologia applicata è la stessa impiegata per l'Inductive Miner, con l'unica differenza nel range di K. Mentre in precedenza il modello veniva creato con valori di K compresi tra 0 e 1, con incrementi di 0.1, in questo caso il modello è stato generato con K che va da 0.8 a 1, con incrementi di 0.02. Il valore iniziale di K è più alto poiché, per questo tipo di algoritmo, un filtraggio maggiore è fondamentale per evitare la generazione di modelli eccessivamente complessi, detti "spaghetti-like", i quali contengono troppo rumore e risultano di difficile interpretazione.

5.2.1 *Studenti Early*

Threshold	Fitness	Precision	Generalization	Semplicity
0.80	0.708312	0.950575	0.734217	0.5
0.82	0.753161	0.956536	0.746536	0.514019
0.84	0.757266	0.960875	0.743272	0.511962
0.86	0.760567	0.960694	0.747643	0.523316
0.88	0.752776	0.83396	0.75553	0.530055
0.90	0.764228	0.814938	0.766808	0.536424
0.92	0.768082	0.817934	0.777648	0.544828
0.94	0.685857	0.724436	0.825666	0.643836
0.96	0.722825	0.591917	0.791655	0.698113
0.98	0.878192	0.456838	0.943393	1
1	0.746985	0.415232	0.896772	1

Tabella 9 Parametro "k" e metriche relative ai modelli di processo degli studenti Early (Heuristic miner)

Il modello scelto tra questi è stato quello con K = 0.92. Questo modello presenta buoni valori per le prime tre metriche ma un livello di semplicità basso, ciononostante è stato ritenuto il migliore in quanto i modelli successivi avrebbero filtrato troppi comportamenti.

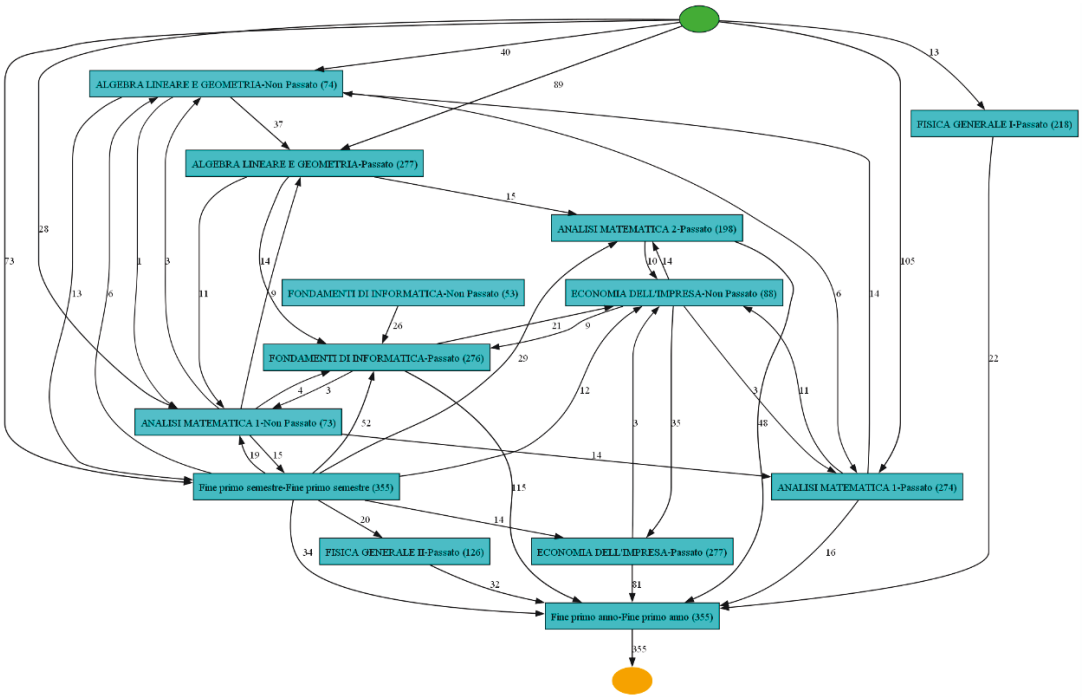


Figura 5.3 Heuristic Net studenti Early. k = 0.92

Come si evince dall'Heuristic Net, gli esami maggiormente sostenuti da questa categoria di studenti sono “Algebra Lineare e Geometria” , “Economia dell’Impresa” e “Fondamenti d’Informatica”, i primi due superati da 277 studenti su 355 totali, mentre l’ultimo da 276. In generale, osserviamo un buon numero di esami sia sostenuti che superati, il che dimostra che questa categoria di studenti si distingue per un maggiore impegno durante il primo anno accademico. Il numero di bocciature risulta relativamente basso rispetto al numero di esami superati, ulteriore conferma del fatto che questi studenti sono più efficaci nel sostenere e completare gli esami.

Questi dati suggeriscono che il loro approccio agli studi nel primo anno è più strutturato e produttivo.

5.2.2 *Studenti One year late*

Threshold	Fitness	Precision	Generalization	Semplicity
0.80	0.776691	0.93893	0.690955	0.534091
0.82	0.769016	0.948906	0.751695	0.558824
0.84	0.768198	0.0.943898	0.738297	0.578125
0.86	0.76623	0.79944	0.716188	0.567568
0.88	0.762481	0.809856	0.744023	0.588235
0.90	0.763415	0.809856	0.744568	0.6
0.92	0.779565	0.809856	0.744776	0.608247
0.94	0.775231	0.823088	0.748038	0.634146
0.96	0.777525	0.974774	0.745423	0.652174
0.98	0.690713	0.174428	0.918107	1
1	0.808229	0.454978	0.837476	1

Tabella 10 Parametro "k" e metriche relative ai modelli di processo degli studenti One year late (Heuristic miner)

Per quanto riguarda i modelli degli One year Late il migliore è quello con K = 0.96

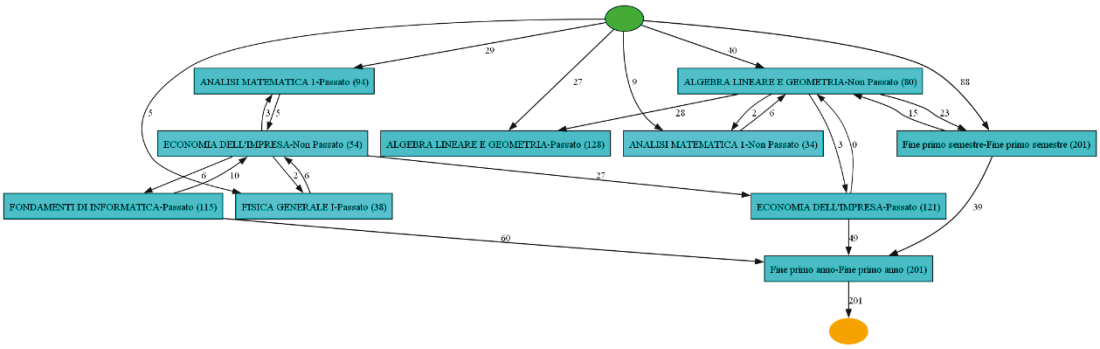


Figura 5.4 Heuristic Net studenti One year late. k = 0.96

In questo modello, analogamente al caso degli studenti Early, notiamo che i tre esami più superati sono "Algebra Lineare e Geometria", "Economia dell'Impresa" e "Fondamenti d'Informatica". Tuttavia, una differenza rispetto al modello precedente è il fatto che il rapporto tra il numero di bocciature e quello dei superamenti risulta

più elevato. Un altro elemento significativo è l'assenza dell'attività "Fisica 2" nel modello, a indicare che pochi studenti hanno sostenuto questo esame nel primo anno accademico, motivo per cui l'attività è stata filtrata dal modello. Per quanto riguarda gli studenti della categoria One year late, si osserva che circa uno studente su due ha concluso il primo semestre senza sostenere nemmeno un esame, contro uno su cinque per la categoria Early. Nel secondo semestre, il rapporto di studenti che non hanno sostenuto esami è uno su cinque per gli One year late e uno su undici per gli Early. Questo conferma ulteriormente che gli studenti Early mostrano un impegno significativamente maggiore durante il primo anno accademico.

5.2.3 *Studenti Late*

Threshold	Fitness	Precision	Generalization	Semplicity
0.80	0.868481	0.936765	0.631254	0.555556
0.82	0.883874	0.921867	0.62995	0.557522
0.84	0.876378	0.935756	0.618876	0.572816
0.86	0.867144	0.820136	0.729641	0.578947
0.88	0.861652	0.794366	0.72419	0.594203
0.90	0.852584	0.960543	0.791792	0.652174
0.92	0.88789	0.970918	0.78932	0.707317
0.94	0.88789	0.970918	0.78932	0.707317
0.96	0.914735	0.377286	0.894488	1
0.98	1	0.824734	0.918621	1
1	0.864217	0.546016	0.807004	1

Tabella 11 Parametro "k" e metriche relative ai modelli di processo degli studenti Late (Heuristic miner)

I miglior modello è quello costruito utilizzando K = 0.92/0.94.

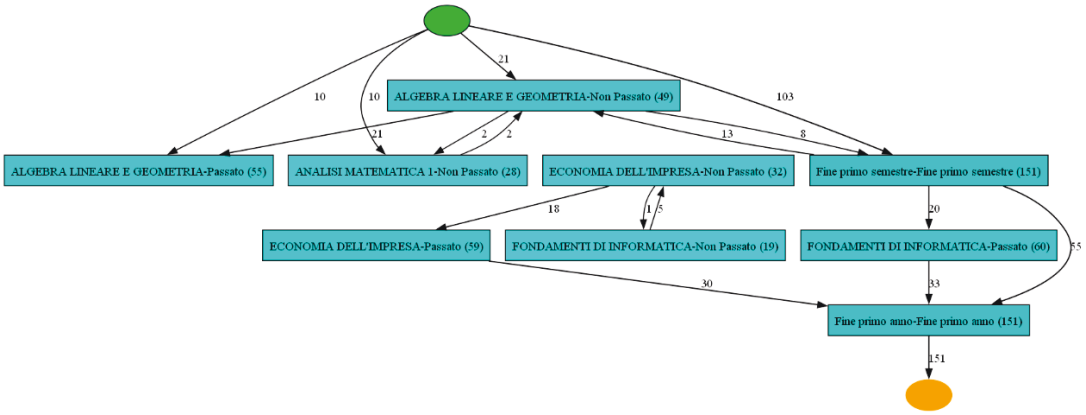


Figura 5.5 Heuristic Net studenti Late. k = 0.92/0.94

Per gli studenti della categoria Late, l'Heuristic Net, come la Petri Net, mostra un livello di semplicità maggiore rispetto a quello delle altre due categorie di studenti. Le uniche materie superate visibili nel modello sono “Algebra Lineare e Geometria”, “Economia dell’Impresa” e “Fondamenti d’Informatica”. Tutte le altre materie sono state superate da un numero così esiguo di studenti che non sono state nemmeno incluse nel modello. Oltre al basso numero di esami superati, si osserva anche un ridotto numero di tentativi complessivi. Questo è evidente dal fatto che esami come “Analisi Matematica 2 - NonPassato” e “Fisica Generale 2 - NonPassato” non compaiono nel modello, a riprova del limitato numero di tentativi per queste materie. Il risultato più sorprendente riguarda il numero di studenti che non hanno sostenuto esami nel primo semestre: infatti, 2 studenti su 3 non hanno sostenuto alcun esame nel primo semestre, mentre circa 1 su 3 non ne ha sostenuti nel secondo semestre. Questi dati confermano che un maggiore tempo per laurearsi corrisponde a un minore impegno da parte degli studenti, evidenziando una correlazione tra il comportamento accademico nei primi anni e i tempi di conseguimento del titolo.

In conclusione, entrambi i tipi di modello di processo sono stati utili ad analizzare come cambia il comportamento degli studenti in relazione al tempo che hanno impiegato per laurearsi.

Capitolo 6 CONFORMANCE CHECKING

Nel capitolo precedente abbiamo messo in luce, attraverso modelli di processo, come un maggiore impegno durante il primo anno accademico sia associato a una migliore performance complessiva. L'obiettivo di questo capitolo è confermare ulteriormente questa correlazione utilizzando una tecnica diversa di Process Mining, il Conformance Checking.

L'analisi ha un duplice scopo: da un lato, intendiamo verificare le distribuzioni della fitness dei percorsi accademici delle tre categorie di studenti rispetto al percorso ideale stabilito dall'università; dall'altro, miriamo a calcolare il livello di fitness per ciascun singolo studente, così da poter utilizzare queste misure come variabili per l'induzione di modelli di Machine Learning volti alla classificazione, che saranno oggetto di analisi nei capitoli successivi. A differenza dei modelli di processo questa volta utilizziamo "df_esami", ovvero il dataset che contiene la sequenza degli esami superati da ogni studente.

6.1 RETE DI PETRI

Come spiegato nel Capitolo 1 sui Cenni Teorici, il Conformance Checking è una tecnica di Process Mining che consente di misurare il livello di fitness tra un event log e un modello di processo. Il modello può essere estratto direttamente da un event log o definito artificialmente. In questo caso, il modello utilizzato è una rete di Petri

che rappresenta il percorso ideale da seguire durante il primo anno di corso, come illustrato in Figura 7.1:

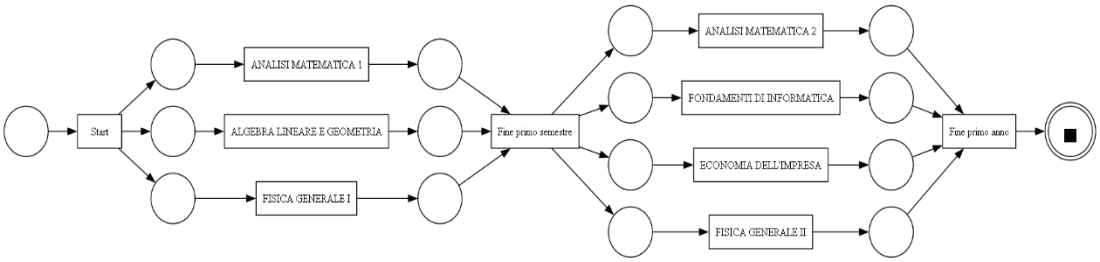


Figura 6.1 Rete di Petri standard per il Conformance Checking

Come possiamo osservare, la rete di Petri si articola in due parti, corrispondenti al primo e al secondo semestre. Il percorso ideale delineato nella rete prevede che gli studenti completino con successo gli esami di "Analisi Matematica 1", "Algebra Lineare e Geometria" e "Fisica Generale 1" entro la fine del primo semestre. Successivamente, durante il secondo semestre, il percorso ottimale richiede il superamento di "Analisi Matematica 2", "Fondamenti d'Informatica", "Economia dell'Impresa" e "Fisica Generale 2" entro la fine del primo anno accademico. Questo modello ideale riflette il flusso degli esami e delle attività che gli studenti dovrebbero seguire secondo le linee guida accademiche, e il Conformance Checking ci permette di confrontare i percorsi effettivi degli studenti con questo tracciato teorico, misurando la loro aderenza o deviazione.

6.2 CALCOLO DI TOKEN-BASED E ALIGNMENTS-BASED FITNESS

Entrambe le misure di fitness verranno calcolate utilizzando lo stesso event log e la medesima rete di Petri, e terranno conto del livello di fitness delle diverse tracce rispetto al modello. Tuttavia, le due misure, pur basandosi sugli stessi dati, presentano valori differenti poiché, come spiegato nei cenni teorici, vengono

calcolate con metodi diversi. Sia la replayed fitness che la aligned fitness considerano l'ordine in cui gli esami vengono superati. Facciamo un esempio pratico: supponiamo che uno studente superi tutti gli esami del primo anno entro il suo primo anno accademico. Se lo studente, però, avesse superato "Analisi Matematica 1" nel secondo semestre, entrambe le misure di fitness, pur essendo alte, non risulterebbero pari a 1. Questo accade perché la rete di Petri non prevede questo tipo di comportamento, ovvero il superamento di questo specifico esame dopo il primo semestre. Le misure teoriche di fitness dovrebbero variare tra 0 e 1, ma poiché ogni traccia dell'event log creato con "df_esami" contiene le tre attività artificiali "Start", "Fine primo semestre" e "Fine primo anno", tutte le tracce presenteranno comunque un livello minimo di fitness diverso da 0.

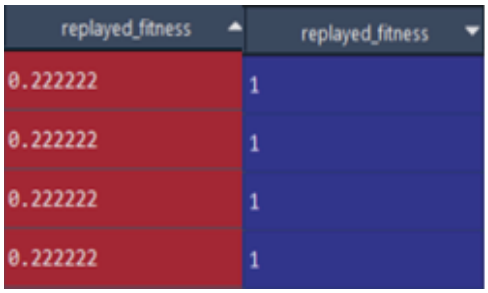


Figura 6.2 Valore minimo e massimo per la replayed fitness

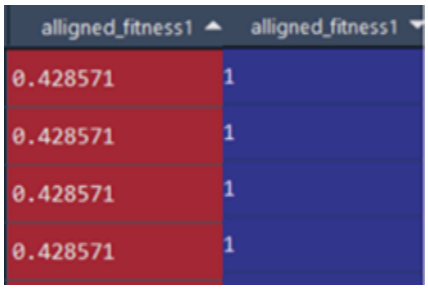


Figura 6.3 Valore minimo e massimo per l'aligned fitness

Come vediamo nella Fig. 7.2 e 7.3 il minimo è di 0.2 circa per la replayed fitness e di 0.43 circa per l'alligned fitness, mentre il massimo in entrambi i casi è pari a 1.

Nelle Fig 7.4 e 7.5 sono stati riportati dei boxplot che rappresentano rispettivamente replayed ed aligned fitness per le tre categorie di studenti.

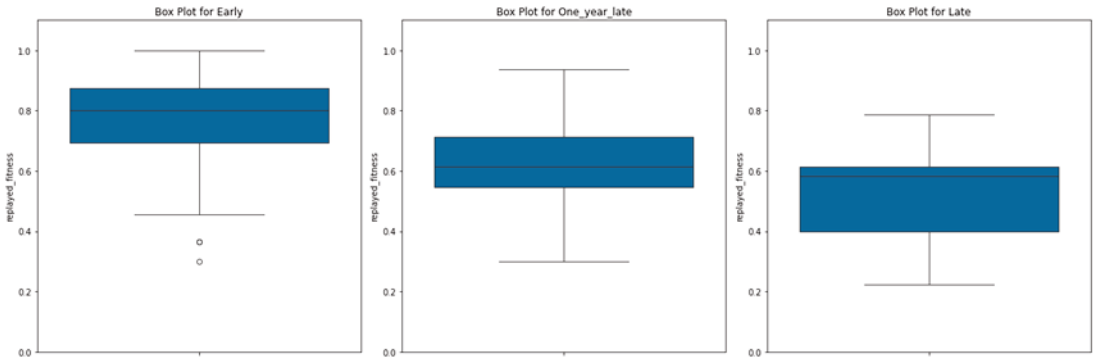


Figura 6.4 Distribuzioni tramite BoxPlot delle replayed fitness per le tre categorie di studenti

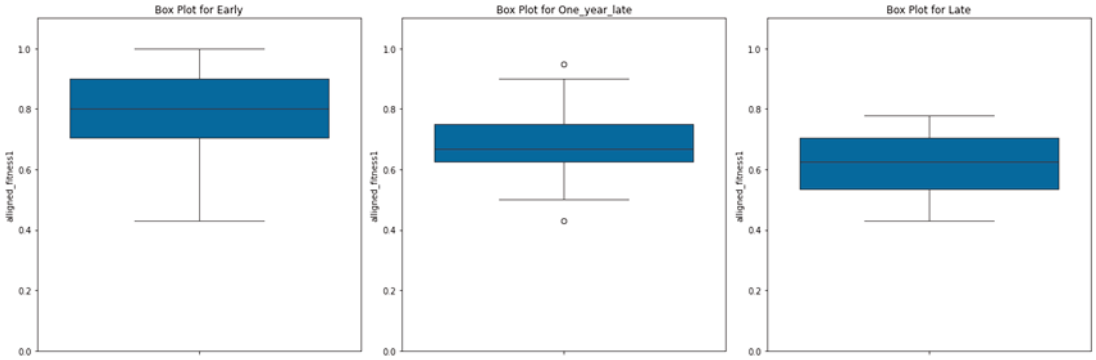


Figura 6.5 Distribuzioni tramite BoxPlot delle aligned fitness per le tre categorie di studenti

Come possiamo osservare dai boxplot in entrambe le figure, la classe Early presenta i livelli di fitness più alti, che diminuiscono progressivamente per gli One year late e ulteriormente per i Late. Inoltre, solo la categoria Early mostra studenti con un livello di fitness massimo. Sebbene le distribuzioni evidenzino un calo costante dei livelli di fitness dalla migliore alla peggiore categoria, notiamo che le distribuzioni degli studenti One year late e Late sono più simili tra loro rispetto a quelle degli Early, soprattutto in riferimento all'aligned fitness. Un aspetto degno di nota riguarda il limite inferiore della distribuzione degli studenti Early. Per la replayed fitness, il limite inferiore è più alto rispetto alle altre due categorie, fatta eccezione per un individuo con livelli di fitness molto bassi. Tuttavia, nel caso dell'aligned fitness, il margine inferiore è simile a quello delle altre categorie. Questo suggerisce che la variabilità del fitness all'interno della classe Early è leggermente maggiore rispetto alle altre due categorie.

Questi risultati confermano che il comportamento degli studenti che hanno conseguito la laurea in tempi più brevi è generalmente più conforme al percorso ideale. Questo ha validato l'opportunità di includere i livelli di fitness calcolati per ogni singolo studente all'interno del dataset destinato all'analisi predittiva.

Capitolo 7 CREAZIONE GRAFI

Dopo aver analizzato le differenze comportamentali tra le tre categorie di studenti utilizzando tecniche di Process Mining il passo successivo consiste nella creazione degli Instance Graphs e dei sottografi. L'obiettivo di questa fase è estrarre i comportamenti individuali degli studenti in un formato che possa essere utilizzato come variabili per la costruzione di modelli predittivi in grado di classificare correttamente "STATO_LAUREA". Una soluzione immediata sarebbe stata quella di sfruttare le varianti presenti all'interno dell'event log, creando per ciascuna variante una variabile binaria che assume valore 1 se lo studente ha seguito il percorso rappresentato da quella variante, e 0 in caso contrario. Tuttavia, questo approccio presenta dei limiti significativi in un contesto come questo, dove gli studenti sono liberi di sostenere gli esami quando e come preferiscono, il che porta a un numero elevato di varianti rispetto al numero di tracce. In particolare, l'event log creato utilizzando il dataset "df_merged_caricato" contiene 526 varianti distribuite su 707 tracce.

N° varianti diverse	N° studenti caratterizzati dalle varianti
481	1
34	2
5	3
3	11
1	8
1	12
1	126

Tabella 12 Numero di varianti per numero di studenti

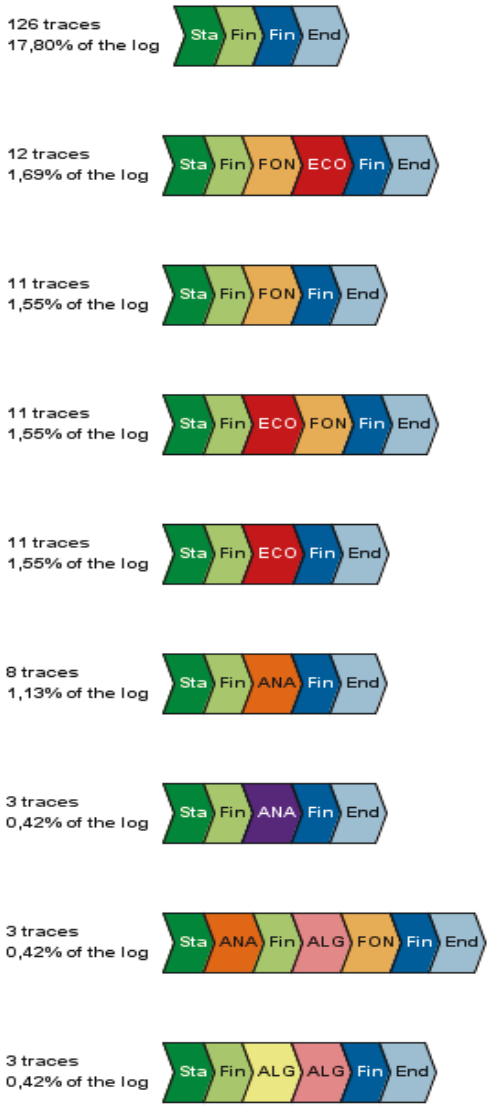


Figura 7.1 Top 9 Varianti per numero di studenti

Come mostrato nella tabella 12, 481 varianti di processo contengono una sola traccia, il che evidenzia l'estrema eterogeneità e mancanza di struttura nei comportamenti degli studenti. Questa variabilità comportamentale tra gli individui rende difficile l'uso diretto delle varianti per la predizione, poiché molte di esse sono uniche. La figura 8.1 illustra che l'unica variante dell'event log associata a un numero consistente di tracce è quella che inizia con "Start", seguita da "Fine primo semestre" e "Fine primo anno", per concludersi con "End". Questa variante caratterizza tutti

quegli studenti che, nel corso del loro primo anno accademico, non hanno sostenuto nemmeno un esame.

7.1 DALLE TRACCE AGLI INSTANCE GRAPHS

Data l'impossibilità di utilizzare le varianti per la creazione di variabili predittive, si procede con la costruzione degli Instance Graphs, ovvero grafi diretti. A differenza delle varianti presenti nell'event log, questi grafi sono in grado di rappresentare la concorrenza tra due o più attività, una volta esplicitato il concetto di concorrenza adottato. Come spiegato nel capitolo dedicato alla metodologia, consideriamo due attività concorrenti se sono state eseguite a meno di un mese di distanza l'una dall'altra. La Figura 8.2 illustra chiaramente cosa intendiamo per concorrenza: il nodo "Start" rappresenta l'inizio del primo anno accademico e si collega ad "Analisi Matematica 1 – Passato" e "Algebra Lineare e Geometria – Non Passato". Questo significa che lo studente ha sostenuto questi due esami a distanza di meno di un mese l'uno dall'altro, e quindi sono considerate attività concorrenti. Anche se "Algebra Lineare e Geometria – Non Passato" e "Algebra Lineare e Geometria – Passato" sono distanti meno di un mese l'una dall'altra, non vengono considerate tali secondo la regola stabilita, poiché si tratta di tentativi sulla stessa materia d'esame. Lo stesso principio viene applicato a qualsiasi coppia di attività che includa un'attività artificiale, come "Start", "Fine primo semestre" o "Fine primo anno", che non sono considerate concorrenti con altre attività. Anche "Analisi Matematica 1 – Passato", "Fisica Generale 1 – Passato" e "Algebra Lineare e Geometria – Passato" sono state rappresentate come parallele in quanto concorrenti.

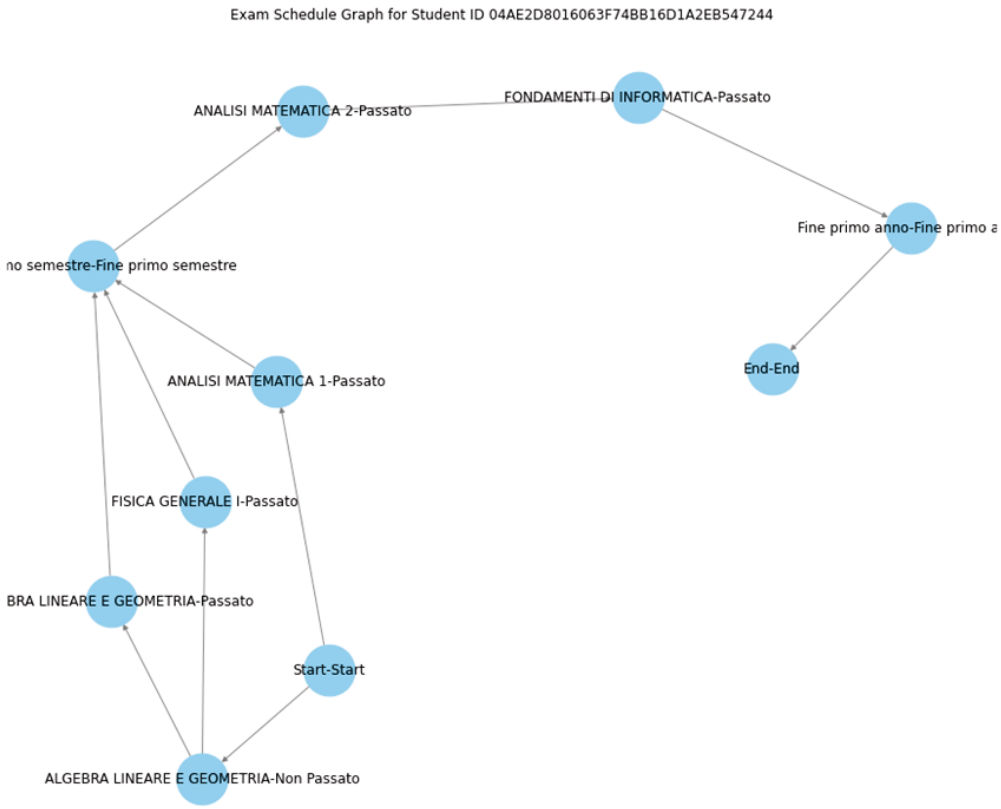


Figura 7.2 Esempio di Instance Graph per uno specifico studente

La rappresentazione della concorrenza consente di ottenere lo stesso Instance Graph partendo da varianti diverse. Ad esempio, supponiamo di avere due tracce, T1 e T2, definite rispettivamente come T1 = << a, b, c, d >> e T2 = << a, c, b, d >>, dove "b" e "c" sono state eseguite a meno di un mese di distanza l'una dall'altra. In questo caso, l'Instance Graph risultante sarà lo stesso per entrambe le tracce, poiché "b" e "c" verranno rappresentate come attività parallele all'interno dei grafi, indicando la loro concorrenza.

Allo stesso modo, è possibile ottenere Instance Graphs diversi partendo dalla stessa traccia. Supponiamo di avere T1 = T2 = << a, b, c, d >>. Se nella prima traccia "b" è concorrente con "c", mentre nella seconda non lo è, il primo grafo rappresenterà questo parallelismo, evidenziando la concorrenza tra le due attività, mentre il

secondo grafo non mostrerà tale parallelismo. Il numero di Instance Graphs diversi ottenuti dalla conversione delle tracce dell'event log è di 514 grafi, distribuiti su 707 tracce. Similmente a quanto osservato con le varianti, anche il numero di modelli comportamentali rappresentati dagli Instance Graphs è troppo elevato per poter essere utilizzato direttamente nei modelli predittivi. Questa sovrabbondanza di grafi, dovuta all'elevata variabilità dei comportamenti tra gli studenti, rende necessario un ulteriore processo di sintesi o riduzione delle informazioni, in modo da estrarre le caratteristiche salienti.

7.2 DAGLI INSTANCE GRAPHS AI SOTTOGRAFI

Il problema emerso alla fine del capitolo precedente trova una soluzione con la creazione dei sottografi utilizzando l'algoritmo SUBDUE. Questi sottografi rappresentano i comportamenti più frequenti rilevati negli Instance Graphs. Nella Fig. 8.3 sono riportati alcuni esempi di sottografi, che da ora abbrevieremo come "SUB". Come illustrato nei cenni teorici e nella metodologia, i sottografi sono grafi diretti estratti tramite tecniche di clustering gerarchico dagli Instance Graphs e sono organizzati in una struttura gerarchica in base alla loro rilevanza e al livello di astrazione. In questa struttura, i pattern più comuni e generali si trovano al livello superiore, mentre quelli più specifici e meno frequenti occupano i livelli inferiori. Le SUB a livello inferiore, oltre a essere caratterizzate da una frequenza minore, tendono a includere nodi costituiti da altre SUB, creando talvolta una struttura annidata.

```
S
v 1 Start-Start
v 2 ALGEBRALINEAREEGEOMETRIA-Passato
v 3 Fineprimosemestre-Fineprimosemestre
d 1 2 Start-Start_ALGEBRALINEAREEGEOMETRIA-Passato
d 2 3 ALGEBRALINEAREEGEOMETRIA-Passato_Fineprimosemestre-Fineprimosemestre

S
v 1 Start-Start
v 2 ALGEBRALINEAREEGEOMETRIA-NonPassato
v 3 ALGEBRALINEAREEGEOMETRIA-Passato
v 4 Fineprimosemestre-Fineprimosemestre
d 1 2 Start-Start_ALGEBRALINEAREEGEOMETRIA-NonPassato
d 2 3 ALGEBRALINEAREEGEOMETRIA-NonPassato_ALGEBRALINEAREEGEOMETRIA-Passato
d 3 4 ALGEBRALINEAREEGEOMETRIA-Passato_Fineprimosemestre-Fineprimosemestre

S
v 1 SUB_1
v 2 ECONOMIADELLIMPRESA-Passato
d 2 1 ECONOMIADELLIMPRESA-Passato_Fineprimoanno-Fineprimoanno

S
v 1 SUB_3
v 2 SUB_1
d 1 2 Fineprimosemestre-Fineprimosemestre_Fineprimoanno-Fineprimoanno

S
v 1 SUB_2
v 2 ECONOMIADELLIMPRESA-Passato
d 2 1 ECONOMIADELLIMPRESA-Passato_FONDAMENTIDIINFORMATICA-Passato
```

Figura 7.3 Esempio di sottografi in formato testuale

I grafi diretti in forma testuale mostrati in Fig. 8.3 devono essere interpretati nel seguente modo: la lettera "S" dopo uno spazio indica l'inizio di una SUB, mentre la lettera "v" rappresenta i nodi. Alla destra della "v", in ordine, sono riportati l'identificativo del nodo e la sua etichetta. La lettera "d" rappresenta un arco, e alla sua destra vengono indicati, in ordine, l'identificativo del nodo da cui parte l'arco, l'identificativo del nodo di arrivo, l'etichetta del nodo di partenza e l'etichetta del nodo di arrivo.

SUBs	Early	One year late	Late	Tot
SUB 1	355	201	151	707
SUB 2	163	75	43	281
SUB 3	73	88	103	264
SUB 7	150	73	39	262
SUB 4	111	35	4	150
SUB 5	94	29	12	135
SUB 22	88	27	12	127
SUB 8	33	39	54	126
SUB 20	55	45	21	121
SUB 9	54	25	13	92
…	…	…	…	…
SUB 81	1	0	0	1
SUB 97	0	1	0	1

Tabella 13 Classifica SUBs per numero di occorrenze tra gli studenti

Con l'utilizzo delle varianti e degli Instance Graphs, avevamo oltre 500 comportamenti distinti, che raramente coinvolgevano più di uno studente. Tuttavia, con l'introduzione delle SUB, il numero di comportamenti si è ridotto drasticamente a soli 100, e la maggior parte di questi caratterizza più di uno studente. In Tabella 13 sono riportate le prime dieci e le ultime due SUB per numero di occorrenze negli Instance Graphs degli studenti.

La SUB 1 è presente in ogni Instance Graph in quanto rappresenta un grafo che collega “Fine primo anno” a “End” e che quindi, per come sono state costruite le attività “artificiali”, è presente nell’Instance Graph di ogni studente. Tutte le altre

coinvolgono solo una parte degli studenti, di seguito sono riportati alcuni esempi di SUB, non in formato testuale.[11]

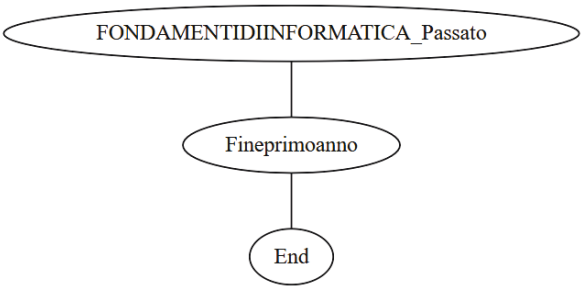


Figura 7.4 SUB 2

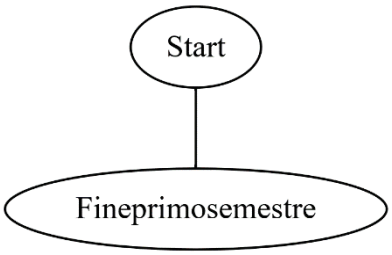


Figura 7.5 SUB 3

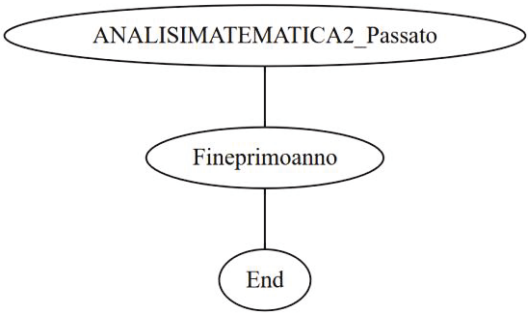


Figura 7.6 SUB 22

[11] Il verso degli archi non è specificato, perciò si consideri che gli archi partono sempre dal nodo più in alto per arrivare al nodo più in basso.

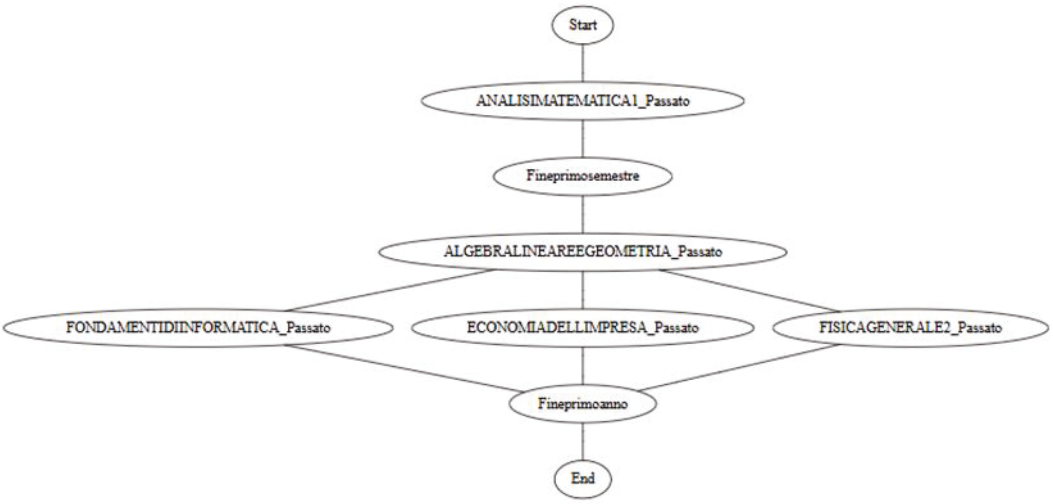


Figura 7.7 SUB 84

Le SUB illustrate nelle Figg. 8.4, 8.5, 8.6, e 8.7 mostrano come una SUB possa essere più o meno complessa. Comportamenti relativamente semplici, come quelli delle SUB 2, 3 e 22, hanno maggiori occorrenze nel dataset poiché sono meno specifici e più generali. Al contrario, comportamenti più complessi, come quello rappresentato dalla SUB 84, sono meno frequenti e corrispondono a un numero inferiore di studenti, nello specifico 8 individui. Nell'esempio illustrato in Fig. 8.7 si può osservare un caso di annidamento: questo grafo contiene la SUB 2, la quale, a sua volta, contiene la SUB 1. Dal punto di vista matematico, una SUB che contiene al suo interno un'altra SUB non può avere una frequenza maggiore rispetto a quella della seconda, poiché rappresenta un pattern più specifico e complesso che compare solo in un sottoinsieme degli stessi dati.

Capitolo 8 PREDIZIONI DELLE PERFORMANCE DEGLI STUDENTI

L'obiettivo di questa tesi è valutare l'impatto del processo seguito dagli studenti nel sostenere gli esami durante il loro primo anno accademico sulle loro performance di laurea, misurate in termini di tempo necessario per conseguire il titolo. Questo impatto può essere valutato impostando l'analisi come un problema di classificazione. L'obiettivo è utilizzare le SUB, insieme ad altre misure descritte nei capitoli precedenti, come caratteristiche degli studenti, mentre le classi di "STATO_LAUREA" fungeranno da variabile target. Le SUB rappresentano i comportamenti frequenti estratti dai processi degli studenti, definiti come sequenze di esami sostenuti durante il primo anno. Le altre misure, invece, descrivono in maniera più generica il primo anno accademico, senza tuttavia riflettere comportamenti specifici. In questo capitolo, procederemo all'induzione di una serie di modelli predittivi per identificare le relazioni tra le caratteristiche degli studenti e la variabile target.

I paragrafi seguenti saranno organizzati come segue: In primo luogo, verranno indotti modelli predittivi utilizzando esclusivamente il set di variabili generali (ad esempio, media dei voti, numero di bocciature), e le loro performance saranno calcolate. Successivamente, si procederà con l'induzione di nuovi modelli predittivi includendo anche le variabili che rappresentano i comportamenti (SUB). Infine, verrà valutata

l'utilità delle SUB confrontando i risultati ottenuti dai due set di modelli e utilizzando tecniche che esaminano l'importanza delle caratteristiche (features) impiegate nell'analisi. Visto che il dataset è sbilanciato e che i comportamenti degli One year late sono sotto certi punti di vista più simili a quelli degli studenti Late, abbiamo accorpato queste due classi denominandole come Late. A questo punto il dataset sui 707 studenti è diviso nel seguente modo: 355 studenti Early e 352 studenti Late.

8.1 DISTRIBUZIONI DELLE VARIABILI NUMERICHE

Come anticipato all'inizio di questo capitolo i primi modelli verranno addestrati senza l'utilizzo delle SUB. Nel paragrafo 2.3 vengono elencate tutte le variabili utilizzate e come sono state costruite. In questa sezione mostreremo come si distribuiscono i loro valori rispetto alle classi "Early" e "Late".

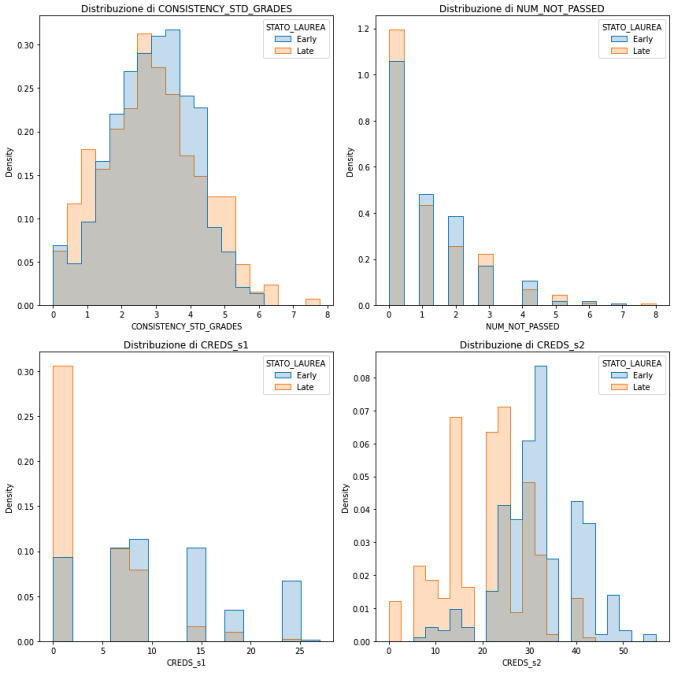


Figura 8.1 Distribuzione di frequenza delle variabili rispetto alle classi Early e Late (gruppo 1)

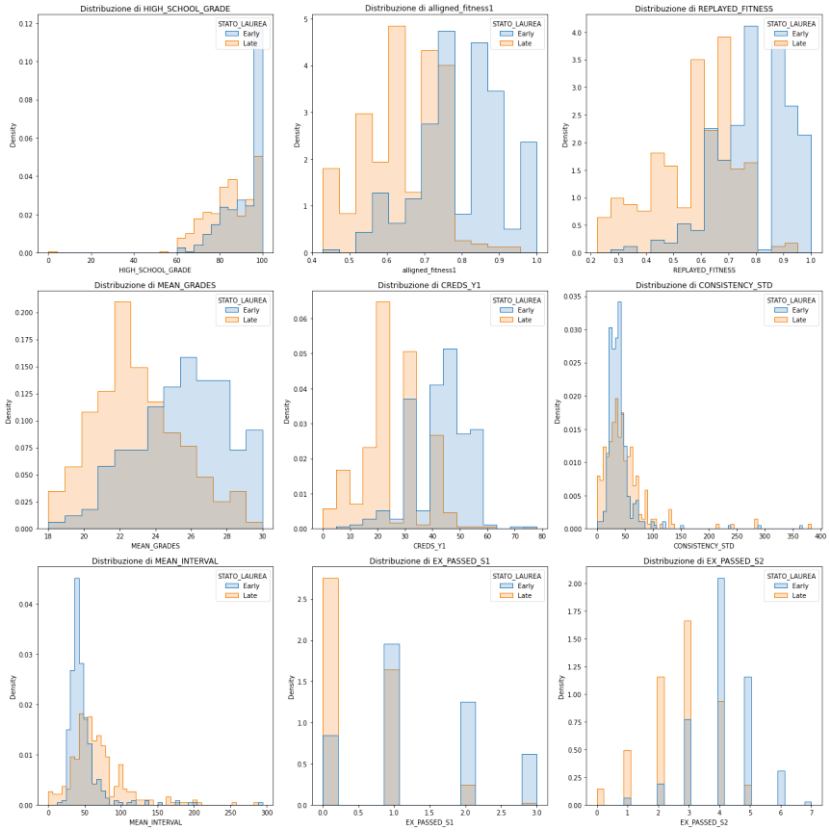


Figura 8.2 Distribuzione di frequenza delle variabili rispetto alle classi Early e Late (gruppo 2)

In cima ad ogni quadrante sono riportati i nomi delle variabili. In arancione sono rappresentate le distribuzioni degli studenti Late mentre in blu quelle degli studenti Early. Nonostante gli studenti One year late siano stati inclusi nella classe Late, si può comunque osservare che le distribuzioni di frequenza sono diverse tra le due classi. Le uniche eccezioni riguardano le variabili "CONSISTENCY_STD_GRADES" e "NUM_NOT_PASSED", che rispettivamente rappresentano la costanza dei voti ottenuti dagli studenti e il numero di volte in cui uno studente è stato bocciato durante il primo anno accademico. Queste due variabili mostrano una distribuzione simile tra le due classi, suggerendo che, per questi aspetti, i comportamenti accademici degli studenti One year late e Late risultano

comparabili. Ciò nonostante, queste due variabili sono comunque state inserite nel dataset utilizzato per addestrare i modelli predittivi in quanto potrebbero ancora fornire un po' d'informazione utile nel discriminare le due classi. L'aligned fitness invece è stata rimossa in quanto altamente correlata con la replayed fitness. Nei prossimi sotto-paragrafi verranno mostrate le matrici di confusione ottenute dalle predizioni effettuate dai quattro modelli usando questo set di variabili. Inoltre, verrà mostrato il set di parametri migliori trovati per i modelli utilizzando Grid Search con 5 fold-cross validation.

8.2 MIGLIORI PARAMETRI, RISULTATI E MATRICI DI CONFUSIONE PER I MODELLI ADDESTRATI SENZA SUB

Come anticipato nella metodologia, i parametri sono stati ottenuti tramite Grid Search effettuando un 5-Fold Cross Validation e scegliendo i parametri che minimizzano l'errore nel classificare le osservazioni del training. Di seguito nella tabella 14 sono riportati i parametri utilizzati nei quattro modelli.

Modello	Parametri
SVM	"C": **0.1**, "gamma": "**scale**", "kernel": "**linear**"
Decision Tree	"max_depth": **None**, "min_samples_leaf": **10**, "min_samples_split": **2**, "splitter": "**random**"
Random Forest	"bootstrap": "**True**", "criterion": "**entropy**", "max_depth": **10**, "min_sample_leaf": **2** "min_sample_split": **2**, "n_estimators": **100**
XGBoost	"colsample_bytree": **0.6**, "learning_rate": **0.1**, "max_depth": **3**, "n_estimators":**50**, "subsample": **0.8**

Tabella 14 Parametri scelti per i modelli senza SUB

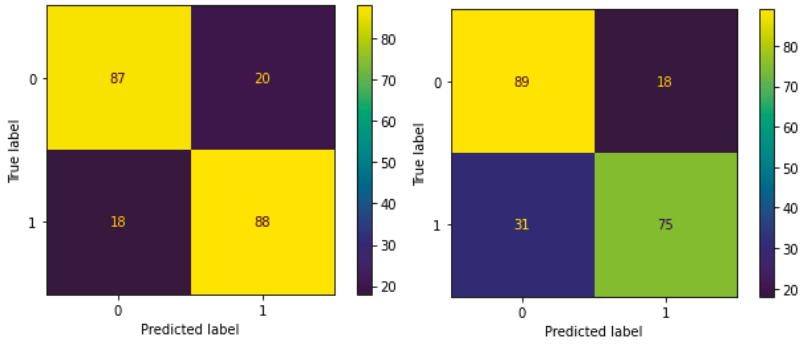


Figura 8.3 Confusion Matrix SVM senza SUB

Figura 8.4 Confusion Matrix DT senza SUB

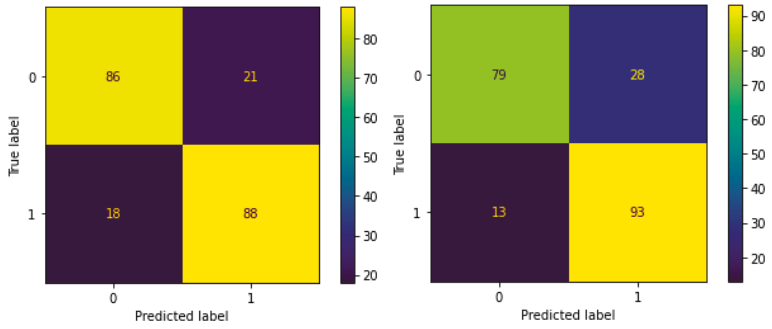


Figura 8.5 Confusion Matrix RF senza SUB

Figura 8.6 Confusion Matrix XGB senza SUB

Nelle Figg. 9.3,9.4,9.5 e 9.6 è possibile vedere le matrici di confusione di rispettivamente SVM, Decision Tree, Random Forest, XGBoost. La classe Early è stata codificata come 0 mentre quella Late come 1. Come spiegato nella metodologia il dataset originale che conta 707 studenti, è stato suddiviso in training e test mantenendo il 70% delle osservazioni nel primo e il 30% nel secondo.

Modello	**Accuracy**
SVM	82.16 %
DT	76.99 %

RF	81.69 %
XGB	80.75 %

Tabella 15 Accuracy dei modellli senza SUB

Come possiamo vedere dalla tabella 14 tutti i modelli tranne il Decision Tree hanno delle performance simili poco sopra l'80%. Il test-set contiene al suo interno 213 studenti, questo fa sì che un errore in più nella classificazione di uno studente comporta la diminuzione dell'accuracy di 0.47%. Per questo motivo sarebbe azzardato dire che l'SVM performa meglio del Random Forest, ciò nonostante, l'SVM è comunque il modello che al momento ha prodotto le migliori classificazioni della variabile "STATO_LAUREA".

8.3 MIGLIORI PARAMETRI, RISULTATI E MATRICI DI CONFUSIONE PER I MODELLI ADDESTRATI UTILIZZANDO LE SUB

Adesso mostriamo i risultati come fatto in precedenza, ma dei modelli addestrati utilizzando le SUB. La metodologia seguita è la stessa, verranno quindi mostrati i parametri che massimizzano le accuracy, le matrici di confusione e le accuracy dei modelli.

Modello	Parametri
SVM	"C": **0.1**, "gamma": "**scale**", "kernel": "**linear**"
Decision Tree	"max_depth": **None**, "min_samples_leaf": **10**, "min_samples_split": **2**, "splitter": "**random**"
Random Forest	"bootstrap": "**False**", "criterion": "**entropy**", "max_depth": **10**, "min_sample_leaf":**1** "min_sample_split": **10**, "n_estimators": **100**
XGBoost	"colsample_bytree": **1**, "learning_rate": **0.01**, "max_depth": **3**, "n_estimators": **200**, "subsample": **0.8**

Tabella 16 Parametri scelti per i modelli utilizzando le SUB

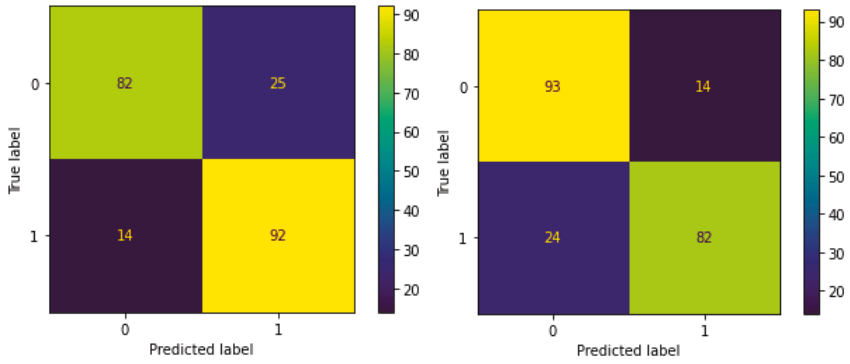


Figura 8.7 Confusion Matrix SVM utilizzando le SUB

Figura 8.8 Confusion Matrix DT utilizzando le SUB

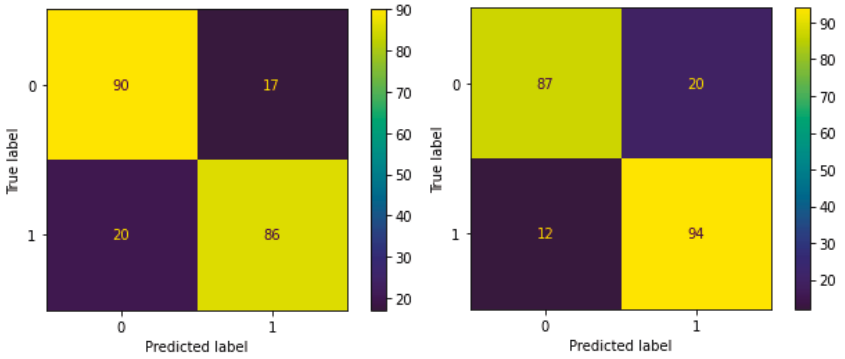


Figura 8.9 Confusion Matrix RF utilizzando le SUB

Figura 8.10 Confusion Matrix XGB utilizzando le SUB

Modello	Accuracy
SVM	81.69 %
DT	82.16 %
RF	82.62 %
XGB	84.98 %

Tabella 17 Accuracy dei modellli utilizzando le SUB

In questo caso il risultato migliore è quello ottenuto utilizzando l'XGBoost. Questo modello ha infatti predetto correttamente la classe di 181 studenti su 213, cinque osservazioni in più rispetto al secondo modello con le performance migliori.

Capitolo 9 VALUTAZIONE DELL'UTILITA' DEI SOTTOGRAFI

In questo capitolo andremo a verificare l'utilità dell'utilizzo delle variabili comportamentali rappresentate dalle SUB. Nel primo paragrafo confronteremo i risultati delle predizioni dei vari modelli addestrati con e senza le SUB, mentre nel secondo analizzeremo l'importanza delle feature utilizzate per osservare nel dettaglio quali di queste hanno avuto un'importanza maggiore nel discriminare le classi.

9.1 COMPARAZIONE DEI RISULTATI DELLE PREDIZIONI

Nella figura di seguito vengono rappresentate in un grafico a barre i livelli di accuracy dei modelli addestrati con e senza l'utilizzo delle variabili comportamentali.

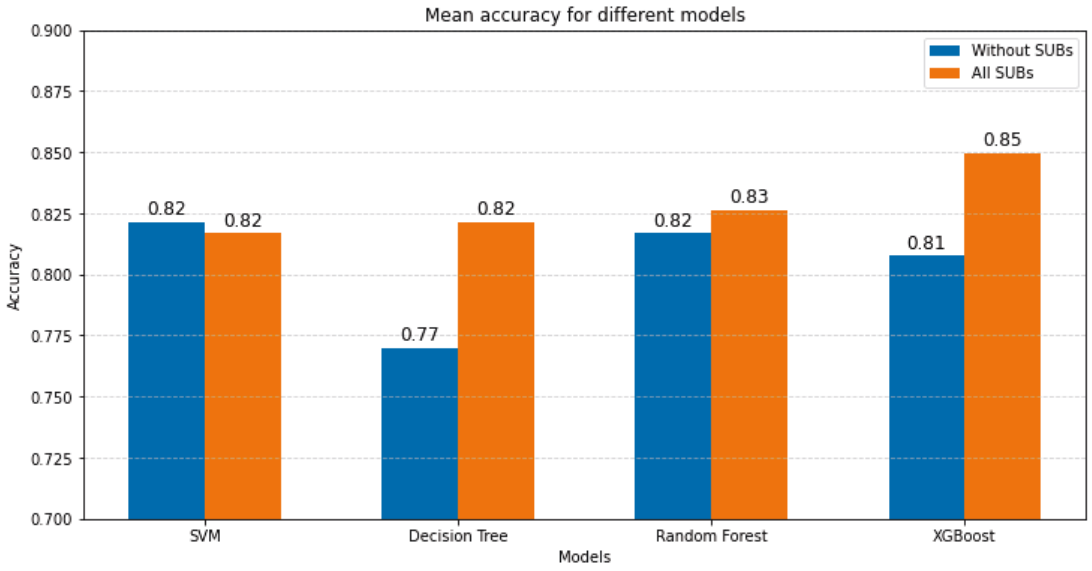


Figura 9.1 Confronto dei livelli di accuracy delle predizioni dei modelli addestrati con e senza l'utilizzo delle SUB

La Fig. 10.1 mostra i migliori risultati ottenuti dai quattro algoritmi su entrambi i dataset. I livelli di accuracy senza l'utilizzo delle variabili comportamentali (Tab. 15) sono rappresentati dalle colonne blu, mentre quelli ottenuti aggiungendo le 100 variabili binarie (Tab. 17) sono mostrati in arancione. Si può osservare come l'accuracy di SVM e Random Forest sia rimasta pressoché invariata. L'algoritmo SVM mostra una leggera diminuzione delle performance, con una sola osservazione in più classificata in modo errato. Un possibile motivo per cui l'SVM non ha migliorato le proprie performance potrebbe essere legato al problema della curse of dimensionality. Mentre gli altri algoritmi impiegati, come Random Forest, Decision Tree e XGBoost, dispongono di meccanismi intrinseci di feature selection che permettono di gestire efficacemente l'aumento del numero di variabili, l'SVM non possiede un sistema interno per la selezione delle caratteristiche più rilevanti. Di conseguenza, il passaggio da 12 a 112 variabili potrebbe aver introdotto troppa complessità, influenzando negativamente le performance dell'algoritmo. La Random Forest, invece, registra un lieve miglioramento, con due misclassificazioni in meno. Per questi due modelli, non si può dunque affermare che l'aggiunta delle SUB sia stata utile nel migliorare la capacità di discriminare le performance degli studenti. Al contrario, la situazione per il Decision Tree e l'XGBoost è significativamente diversa. Entrambi i modelli hanno tratto vantaggio dall'inclusione delle SUB durante la fase di addestramento, migliorando notevolmente le loro performance. Sebbene il Decision Tree resti comunque il modello con le performance peggiori, ha classificato correttamente undici osservazioni in più rispetto al primo esperimento. L'XGBoost, che già aveva raggiunto un buon livello di accuracy, ha classificato correttamente

dieci osservazioni in più, diventando così il modello migliore, con un distacco del 3% rispetto al secondo classificato. Questo dimostra che, mentre per alcuni modelli l'aggiunta delle variabili comportamentali non ha apportato un miglioramento significativo, per altri, come il Decision Tree e l'XGBoost, l'utilizzo delle SUB ha contribuito a un sostanziale incremento delle capacità predittive.

Alla luce di questi risultati, è importante considerare l'affidabilità delle performance osservate. Tranne che per l'SVM, gli altri tre algoritmi (Random Forest, Decision Tree e XGBoost) sono sensibili al random seed utilizzato durante la fase di addestramento. Ciò implica che, pur utilizzando gli stessi dati per addestrare i modelli, il cambiamento del random seed può portare a parametri interni diversi e quindi a modelli diversi e, di conseguenza, a predizioni differenti sulle osservazioni del test-set. I nostri esperimenti sono stati condotti con un random state fissato a 110 per tutti e quattro i modelli, e i risultati riportati si basano sulle prime esecuzioni ottenute. È quindi possibile che modificando il random seed i risultati possano variare leggermente. Questo va tenuto presente quando si valuta la stabilità e l'affidabilità dei risultati ottenuti.

9.2 IMPORTANZA DELLE FEATURES

Alla luce dei confronti effettuati tra le predizioni dei diversi algoritmi, è importante evidenziare quali feature abbiano avuto il maggior impatto nella discriminazione delle diverse classi per il miglior modello e verificare se tra queste figurano alcune delle SUB. A tal fine, è stato creato un grafico che illustra il contributo apportato dalle singole feature rispetto all'accuracy complessiva ottenuta dal XGBoost con il set completo di variabili.

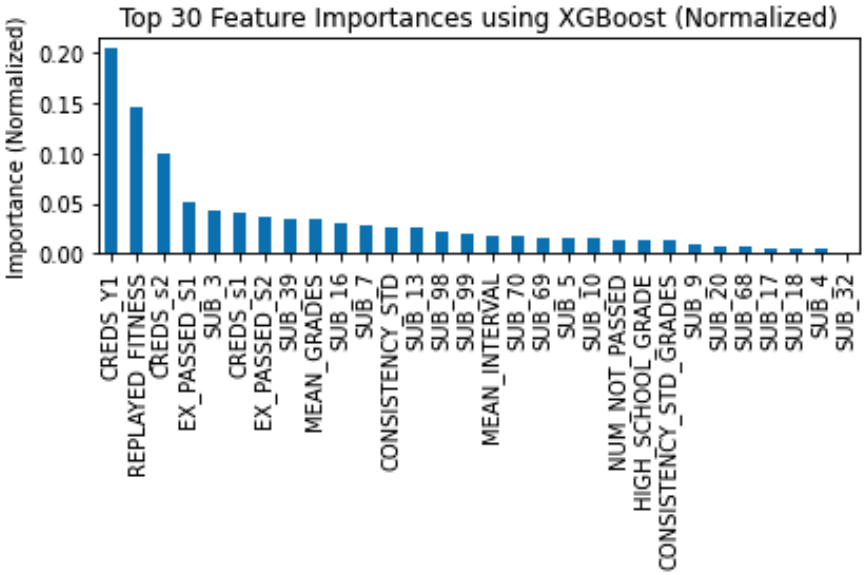


Figura 9.2 Importanza relativa delle feature rispetto all'accuracy totale ottenuta con l'algoritmo XGBoost utilizzando l'intero set di variabili

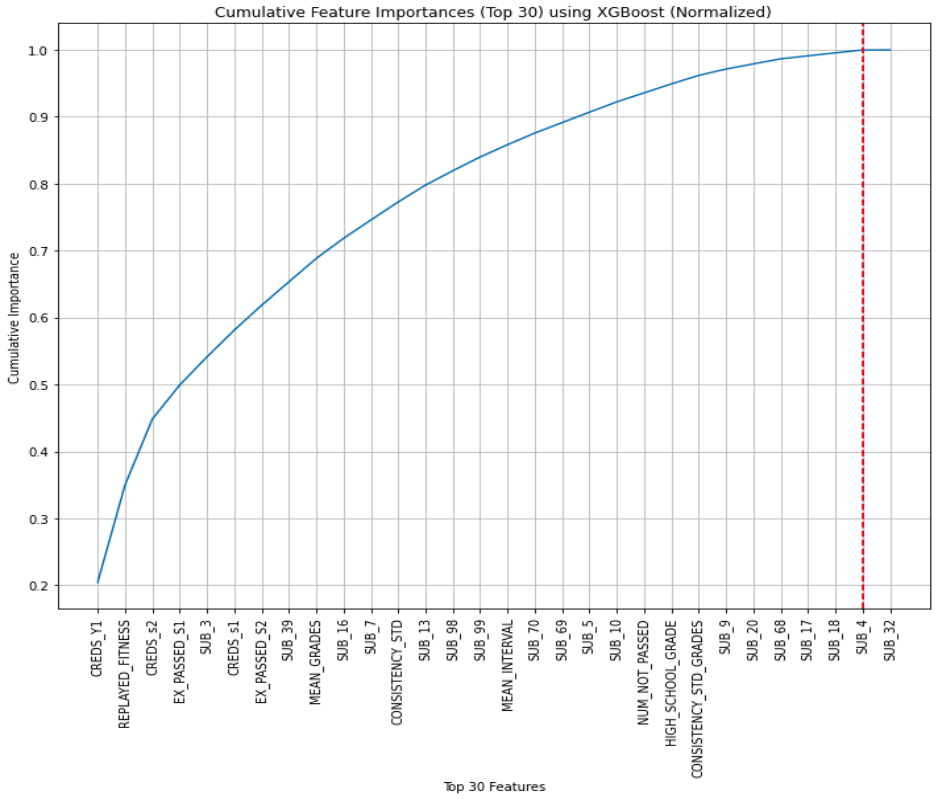


Figura 9.3 Importanza cumulativa delle feature rispetto all'accuracy totale ottenuta con l'algoritmo XGBoost utilizzando l'intero set di variabili

La curva normalizzata dei pesi in Fig 10.3 mostra che solo 29 delle 112 variabili utilizzate hanno contribuito al 100% dell'accuracy ottenuta dal modello e molte di queste variabili sono SUB. Nonostante il numero relativamente ingente di SUB notiamo dalla Fig. 10.2 che singolarmente le SUB hanno un peso abbastanza basso. In ordine le dieci feature che hanno avuto un peso maggiore sono: i crediti ottenuti nel primo anno, la replayed fitness, i crediti ottenuti nel secondo semestre, SUB 3, i crediti ottenuti nel primo semestre, gli esami passati nel secondo semestre, SUB 39, la media dei voti del primo anno, SUB 16, SUB 7. Il fatto che quattro tra le prime dieci feature siano SUB, conferma che siano importanti nel predire le performance degli studenti. Un'altra utilità sta nel fatto che permettono di capire quali sono i comportamenti che incidono positivamente o negativamente sulle performance.

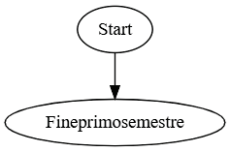


Figura 9.4 SUB 3

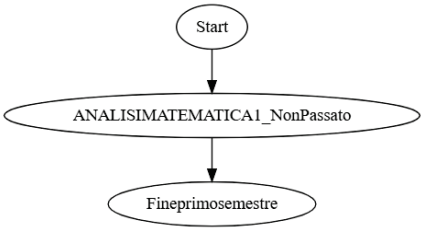


Figura 9.5 SUB 39

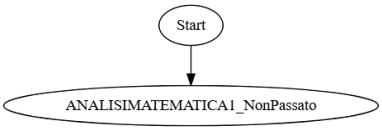


Figura 9.6 SUB 16

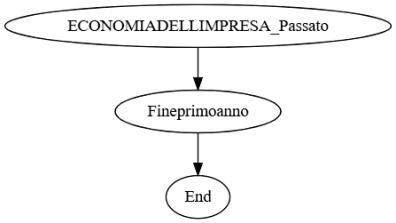


Figura 9.7 SUB 7

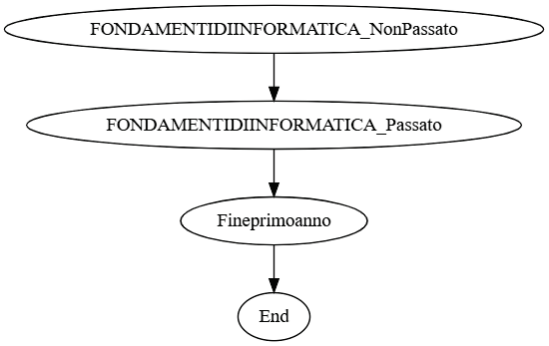


Figura 9.8 SUB 13

Nelle Figg. sopra sono riportate le prime cinque SUB in ordine di importanza. La SUB che ha assunto maggiore rilevanza è rappresentata in Fig. 10.4. La SUB 3 evidenzia un pessimo inizio del primo anno accademico, poiché rappresenta un comportamento in cui lo studente non ha sostenuto nemmeno un esame durante il primo semestre. Analogamente, anche la SUB 39 indica un comportamento negativo, in quanto mostra come unico tentativo nel primo semestre una bocciatura in Analisi Matematica 1. La SUB 16, pur essendo simile alla 39, fornisce un'informazione diversa, in quanto descrive un cattivo inizio, ma non l'assenza totale di esami superati nel primo semestre. Le SUB 7 e 13 sono più difficili da interpretare. La SUB 7 segnala che lo studente ha concluso il suo anno accademico superando un esame, mentre la SUB 13 mostra che, dopo una bocciatura in Fondamenti di Informatica, lo studente è riuscito a superare l'esame entro la fine del primo anno. Prese

singolarmente, queste ultime due SUB non indicano necessariamente comportamenti chiaramente positivi o negativi, ma potrebbero acquisire un significato diverso a seconda delle altre SUB che caratterizzano uno specifico studente. Potrebbero infatti rappresentare comportamenti positivi o negativi solo in relazione a ulteriori comportamenti riscontrati nel percorso dello studente.

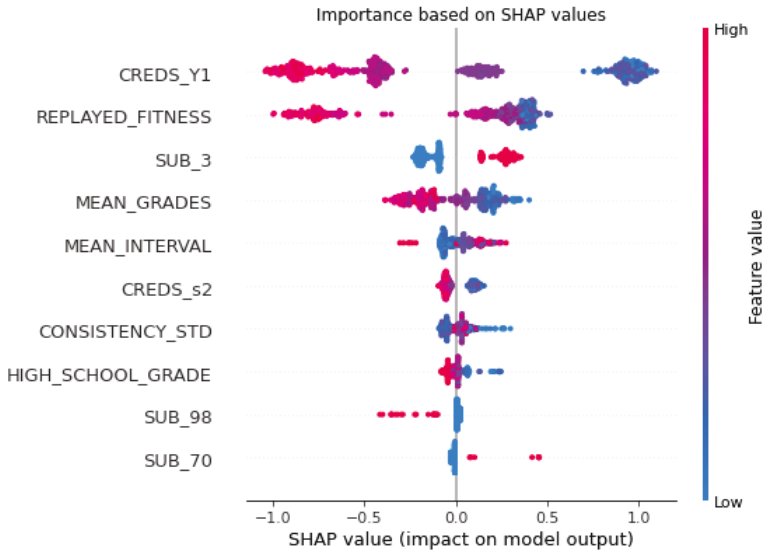


Figura 9.9 Importanza delle feature basate su valori SHAP nel predire le performance degli studenti

In Fig. 10.9 abbiamo utilizzato un'explainable AI per identificare il contributo delle feature nel predire le osservazioni. Per capirne il funzionamento prendiamo come esempio “”CREDS_Y1”, i punti sulla riga che corrispondono a questa variabile rappresentano le osservazioni, il loro colore rappresenta il valore di quella variabile per una certa osservazione, mentre la posizione sull'asse delle x rappresenta il valore SHAP della feature ed indica quanto quella specifica feature, in combinazione con

tutte le altre, sposta la predizione rispetto alla baseline rappresentata dalla linea verticale. Un'osservazione a destra del valore 0 indica che quella specifica feature contribuisce a classificare come 1 l'osservazione.

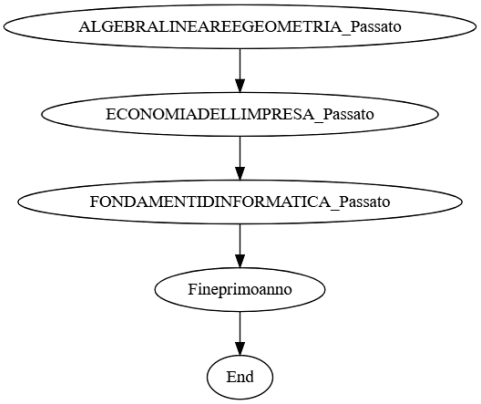


Figura 9.10 SUB 70

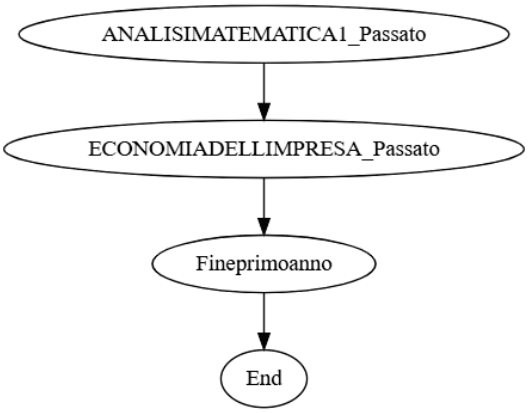


Figura 9.11 SUB 98

Come possiamo osservare dal grafico che riporta i valori SHAP, le prime 10 feature per importanza includono SUB diverse rispetto a quelle mostrate in Fig. 10.2. Questo è dovuto al fatto che, mentre il grafico SHAP considera l'interazione tra le feature, il grafico in Fig. 10.2 non lo fa, il che spiega la differenza nei risultati.

Nella Fig. 10.9 emergono risultati controversi per le SUB 70 e 98. Queste due SUB rappresentano il superamento di una serie di esami prima della fine del primo anno. Nonostante la SUB 70 includa un esame in più rispetto alla SUB 98, il comportamento rappresentato, pur intuitivamente positivo, sembra paradossalmente contribuire all'associazione di alcuni studenti con la classe "Late". Forse perché “Algebra Lineare e Geometria” essendo un esame del primo semestre, è importante che non venga dato tra gli ultimi esami del primo anno.

Capitolo 10 CONCLUSIONI E LAVORI FUTURI

Questo studio si è concentrato sull'analisi dei comportamenti degli studenti durante il loro primo anno. L'obiettivo era determinare se e in che modo questi comportamenti influenzassero i tempi di laurea. Per fare ciò dopo aver creato delle variabili che rappresentassero i comportamenti più frequenti degli studenti, sono state utilizzate queste variabili per l'induzione di modelli predittivi, affiancate ad un altro set di variabili che davano una rappresentazione più generica e meno dettagliata del comportamento degli studenti. Nonostante i risultati ottenuti mostrino che i comportamentali nel primo anno accademico degli studenti abbia un impatto sulle loro performance, ci sono degli aspetti dell'analisi che potrebbero essere migliorati. In primo luogo, le SUB per come sono estratte, presentano alti livelli di correlazione elevate tra loro, e risultano in qualche caso essere anche ripetitive. Pertanto, sarebbe auspicabile l'utilizzo di un altro algoritmo per l'estrazione dei comportamenti più frequenti. Un'alternativa a SUBDUE potrebbe essere quella di passare dagli Instance Graph alla creazione degli alberi di processo e quindi dei sub-tree, ovvero le componenti degli alberi. Un'altra considerazione da fare riguarda il concetto di concorrenza. Due esami sono stati considerati concorrenti se effettuati a distanza di meno di un mese l'uno dall'altro. Questo intervallo di tempo è stato definito arbitrariamente, sulla base dell'idea che un mese sia all'incirca il tempo necessario per preparare un esame. Ciò significa che per due esami dati a meno di un mese di distanza i periodi di studio delle due materie si sono sovrapposti. Questo concetto di

concorrenza può essere modificato in modo tale da ottenere Instance Graphs diversi e quindi sottoprocessi diversi. Infine, puntiamo a tradurre le intuizioni emerse da questa analisi in raccomandazioni pratiche. Queste indicazioni consentiranno ai coordinatori dei corsi di laurea di proporre percorsi ottimali agli studenti al momento dell'iscrizione o di suggerire le migliori scelte di carriera in diverse fasi del percorso, considerando i loro progressi attuali.

Capitolo 11 BIBLIOGRAFIA

1. VAN DER AALST, WILMP. (2016). *Process mining: Data science in action* (2nd ed.). Springer.
2. ALIZADEH, MAHDI, DE LEONI, MASSIMILIANO, & ZANNONE, NICOLA. (2014). *History-based construction of alignments for conformance checking: Formalization and implementation*. Eindhoven University of Technology.
3. Coursera. *Process mining: Data science in action*. From https://www.coursera.org/learn/process-mining/home
4. TAN, PANG-NING, & STEINBACH, MICHAEL. (2019). *Introduction to data mining* (Global ed.). Pearson.
5. MURATA, TADAO. (1989). *Petri nets: Properties, analysis and applications*. *Proceedings of the IEEE*.
6. DESEL, JÖRG, & REISIG, WOLFGANG. (2015). *The concepts of Petri nets*. *Software and Systems Modeling*.
7. Javatpoint. *Machine learning decision tree classification algorithm*. From https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm
8. Documentazione pm4py: https://processintelligence.solutions/pm4py
9. Documentazione scikit-learn: scikit-learn: machine learning in Python — scikit-learn 1.5.2 documentation
10. Documentazione SHAP: Welcome to the SHAP documentation — SHAP latest documentation