UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica
Corso di Laurea Magistrale in Informatica

TESI DI LAUREA MAGISTRALE IN SOFTWARE ENGINEERING FOR ARTIFICAL INTELLIGENCE

Uno Studio Empirico sull'Impatto dei Data Smell sull'Accuracy e le Performance dei Modelli di Machine Learning

RELATORE
Prof. Fabio Palomba
Università degli Studi di Salerno

CANDIDATO
Davide La Gamba
Matricola: 0522501464

Anno Accademico 2023-2024

Questa tesi è stata realizzata nel

Alla mia famiglia

Abstract

L'Intelligenza Artificiale (AI) è sempre più impattante sulle nostre vite, essendo ormai presente in molti aspetti della società. I modelli di IA si basano principalmente sui dati che sono utilizzati per il loro addestramento, per cui risulta fondamentale analizzare l'importanza e l'impatto della loro qualità sulle performance ottenute. Il lavoro proposto punta ad analizzare l'impatto della presenza di alcuni data smell, definiti da Foidl et al. come degli indicatori latenti di problemi di qualità indipendenti dal contesto, all'interno dei dataset utilizzati per addestrare alcuni modelli di classificazione. L'analisi include la generazione di dati sintetici, nei quali verranno iniettati i data smell in analisi all'interno di feature in cui questi sono applicabili, con lo scopo di arricchire i dataset su cui sono addestrati i modelli di classificazione. I risultati mostrano un impatto significativo della presenza di alcuni dei data smell analizzati su parte delle metriche valutate quando analizzate in diversi scenari, relative all'accuracy e alle performance ottenute dai modelli di classificazione in alcune situazioni, denotando la necessità di introdurre tecniche di gestione della qualità nei sistemi di machine learning.

Indice

Elenco delle figure

Elenco delle tabelle

CAPITOLO 1

Introduzione

1.1 Motivazioni e Obiettivi

Negli ultimi anni l'**Intelligenza Artificiale (AI)** ha preso sempre più piede nelle nostre vite, divenendo presente anche in molti settori della nostra società [1]. I modelli di IA si basano principalmente sui dati utilizzati per il loro addestramento [2], e per questo risulta fondamentale porre attenzione alla loro qualità e a come questa possa influire sui risultati e sulle decisioni intraprese da essi. Una delle aree di ricerca in crescita in questo ambito è quella dei ***data smell***, che possono impattare la qualità dei sistemi AI causando *technical debt* [3], che sul lungo periodo può risultare pericoloso se non saldato [4]. Grazie al lavoro di Recupito et al. [5], è possibile analizzare quali siano i data smell più presenti in un insieme di dataset utilizzati per addestrare modelli di machine learning, permettendo di concentrare quindi le attenzioni su un sottoinsieme più ristretto di essi. Da questo punto di partenza, nasce quindi la necessità di capire il modo in cui i data smell che più frequentemente si annidano nei dataset impattano nell'accuracy e nelle performance dei modelli di classificazione. Per questo motivo, l'obiettivo di questo lavoro è di valutare in modo empirico l'impatto dell'iniezione di questi specifici tipi di data smell nei *training set* di molteplici modelli di classificazione (utilizzando delle tecniche di data augmentation per la generazione

di nuovi dati sintetici) per cercare di individuare delle relazioni tra la loro presenza e delle variazioni nelle metriche di valutazione rispetto allo scenario originale, facenti utilizzo dei medesimi dataset senza la presenza di tali data smell.

1.2 Risultati

In seguito alle fasi di design è stato possibile denotare come, nello scenario proposto e tra i data smell analizzati, le medie della metrica di Balanced Accuracy dei modelli di classificazione sia negativamente impattata dalla presenza di **Casing** smell, limitatamente allo scenario dei dataset con feature categoriche di dimensioni large, mentre le performance degli stessi siano peggiorate dalla presenza di **Suspect Sign** smell, considerando i dataset di dimensioni medium aventi feature numeriche. Inoltre, è emerso come gli specifici modelli di classificazione non siano impattati allo stesso modo e alla stessa misura dalla presenza di data smell, difatti si è denotato come i modelli **SVC** e **KNeighborsClassifier** siano i più impattati in termini di metriche di accuracy da **Casing** smell (nello scenario dei dataset con feature categoriche), mentre **Label Propagation**, **SVC** e **Label-Spreading** siano quelli più interessati a peggioramenti nelle performance risultanti dall'iniezione di *Suspect Sign* e *Casing* smell, negli scenari relativi.

1.3 Struttura della tesi

Il Capitolo 2 presenta una panoramica sugli argomenti che fanno da Background a quanto trattato in questo lavoro, ovvero il Technical Debt (Sezione 2.1), i Code Smell (Sezione 2.2) e i Data Smell (Sezione 2.3), utili per comprendere il contesto in cui si cala questa sperimentazione.
Il Capitolo 3 presenta un'analisi dello stato dell'arte sulle tecniche di identificazione dei Data Smell (Sezione 3.1), il loro impatto sulla qualità dei dati (Sezione 3.2), e le tecniche di Data Augmentation (Sezione 3.3), analizzando anche il possibile utilizzo dei Large Language Model per questo obiettivo (Sezione 3.4).
Nel Capitolo 4 viene presentato il design della sperimentazione effettuata, descrivendo l'Obiettivo (Sezione 4.1), i Dataset impiegati (Sezione 4.2), gli Strumenti utilizzati

(Sezione 4.3) e la Soluzione proposta (Sezione 4.4). I risultati delle sperimentazioni sono invece riportati nel Capitolo 5, dove viene descritto l'Approccio adottato alla validazione dei risultati 5.1, sono riportati gli effettivi Risultati ottenuti dai diversi confronti 5.2, vengono fornite delle risposte alle Research Question poste nella Sezione 4.1 (Sezione 5.3), e sono presentate le Minacce alla validità dei risultati individuate (Sezione 5.4).
Infine, nel Capitolo 6, sono riportate le Osservazioni che è possibile effettuare dai risultati ottenuti (Sezione 6.1) e i Possibili sviluppi futuri (Sezione 6.2). Il materiale sperimentale, inclusi il codice di analisi e i riferimenti ai dati utilizzati, è disponibile pubblicamente alla seguente repository Github:
`https://github.com/davide-lagamba/DataSmellsTesting`.

CAPITOLO 2

Background

2.1 Technical Debt

Per questa, la Sezione 2.3 e il Capitolo 3 sono state utilizzate diverse fonti raccolte da Recupito et al. [5] nel loro lavoro come punto di partenza per lo studio di questi temi, portando quindi ad osservazioni simili su questi argomenti.
Il "***Technical Debt***" è un concetto diffuso nell'ambito dell'ingegneria del software, e rappresenta un vero e proprio debito che si viene a formare durante lo sviluppo per permettere inizialmente di velocizzare i processi, ma che a lungo andare risulta pericoloso se non saldato [4]. Come riportato da Recupito et al. [5], Li et al. [6] hanno presentato una classificazione dei technical debt definiti in letteratura, che li divide in: debiti dei requisiti, architetturali, del codice, del testing, di design, di documentazione, di build, di infrastruttura, di versioning e dei difetti. I technical debt più analizzati nel tempo sono stati quelli relativi al codice e all'architettura, ma secondo Lenarduzzi et al. [7] mancano ancora strumenti specifici per la prioritizzazione dei debiti da considerare durante lo sviluppo, seppur in letteratura ne sono stati presentati diversi per l'individuazione di diversi tipi di technical debt [8, 9, 10], come evidenziato dalla fonte [5]. Dalla *Systematic Literature Review* di Alfayez et al. [11] emerge come ci sia ancora poca attenzione dal mondo dell'industria sulle tematiche relative ai technical

debt, individuando 24 approcci di prioritizzazione, di cui pochi che si concentrano sui vincoli di risorse e costi. La fonte [5] riporta anche come Sculley et al. [12] estendono il tema dei technical debt ai sistemi ML-based, che presentano anche dei particolari debiti specifici a livello di sistema, i quali minano la loro manutenibilità. Alcuni esempi di queste problematiche sono l'*Entanglement* di diversi *ML-based system*, ovvero una forte relazione tra diversi sistemi che può causare cambiamenti nel comportamento di un modello a fronte di piccole modifiche a poche feature, seguendo il principio ***CACE*** (*Changhing Anything Changes Everything*) [12] e i *Feedback Loops*, che consistono nell'avere un sistema che è influenzato dalle proprio azioni se aggiornato nel tempo, e può essere di tipo "diretto" o "nascosto" [12]. Per Sculley et al. [12] anche le dipendenze dai dati hanno una capacità di portare a del debito, essendo anche potenzialmente più difficili da individuare. Difatti queste dipendenze possono essere instabili, portando a cambiamenti nel comportamento di un modello anche a fronte di una modifica negli input nel corso del tempo, oppure possono essere poco utilizzate, portando a situazioni in cui sono presenti ad esempio $\epsilon - feature$. Inoltre, i tool per l'analisi statica di questo tipo di dipendenze non sono ancora molto diffusi [12]. In seguito, Bogner et al. [13] hanno definito il concetto di ***data debt*** nei sistemi *AI-based*, relativo a problematiche che coinvolgono la "*collezione, gestione e utilizzo dei dati, sia per il training che in produzione*" [13], che possono essere anche "latenti" nel sistema [13]. Questo tipo di debito è anche uno dei più frequenti nei sistemi *AI-based*, insieme ai *model antipattern*, denotando 22 antipattern unici per questa categoria dalla ricerca condotta da Bogner et al. [13]. Per Bosu e MacDonell [14] però, manca ancora una considerevole attenzione nell'ambito dell' "Empirical Software Engineering" verso i problemi di data quality e le fasi di data collection, al contrario invece di quanto dedicato al data pre-processing [14], come evidenziato anche da Recupito et al. [5].

2.2 Code Smell

Come raccolto da Lacerda et al. [15], uno dei fattori che può incrementare il technical debt è sicuramente la presenza di ***code smell***, ovvero delle "*violazioni di linee guida del design del codice*" [15], inserite spesso nei progetti nel corso delle fasi di

evoluzione del software, e che possono essere rimossi tramite tecniche di refactoring [15]. L'introduzione di questo tipo di anomalia può avere impatti negativi sulla leggibilità, sulla manutenibilità del codice e su altri aspetti, e quindi in generale sulla sua qualità [15]. La definizione di code smell è legata a quella del concetto di "***smell***" [15], la cui definizione riportata da Lacerda et al. [15] li cataloga come "problemi interni al software a livello di codice" ("*code smell*") o "di progettazione" ("*design smell*") [15]. La presenza di smell non è sempre causa diretta di errori nel software, ma può "impattare sulla manutenibilità" dello stesso [15]. In letteratura sono stati proposti molti code smell, e con essi diversi tipi di categorizzazioni. Una di queste è quella realizzata da Mäntylä e Lassenius [16], che raggruppa i code smells in 6 categorie, di seguito elencate:

- "Bloater": rappresentano elementi del codice che "hanno raggiunto dimensioni tali da condizionarne la gestione" [16]. Esempi di Bloater smell sono: *"Large Class"*, *"Long Method"* e *"Long Parameter List"* [16].

- "Object-Orientation Abusers": sono utilizzi impropri del paradigma dell'Object Orientation [16]. Esempi sono: "*Temporary Field*, *Switch Statements*" e "*Alternative Classes with Different Interfaces*" [16].

- "Change Preventers": sono code smell che "prevengono la modificabilità del software" a causa delle violazioni di alcuni principi di progettazione relativi alle relazioni tra le classi [16]. Questi smell sono: "*Divergent Change*, *Shotgun Surgery*" e "*Parallel Inheritance Hierarchies*" [16].

- "Dispensables": rappresentano "elementi non necessari nel codice" [16]. Si dividono in due sotto-categorie, ovvero "*dispensable classes*" ("*Lazy class*" e "*Data class*") e "*dispensable code*" (*"Duplicate code"*, *"Speculative generality"* e "*Dead code*") [16].

- "Couplers": sono gli smell legati a codice che viola i principi di accoppiamento nella progettazione [16], ad esempio *"Feature envy"*, *"Message chains"* e "*Middle man*" [16].

- "Other smells": sono quelli che "non rientrano nelle altre categorie" [16], come gli smell "*Comments*" e "*Incomplete Library Class*" [16].

Nonostante il refactoring sia il principale approccio per risolvere problematiche del codice di questo tipo e diminuire il technical debt, non sempre ciò accade, in quanto l'applicazione di refactoring superfluo potrebbe portare ad un calo della qualità [15]. Lacerda et al. [15] fanno emergere dalla loro *systematic review* come in letteratura manchi una standardizzazione nelle nomenclatura degli smell, che rendono più difficile una loro catalogazione, e una convergenza sui valori delle metriche del software da utilizzare come soglia di individuazione degli smell. Quest'ultimo aspetto è legato anche alla disparità nei risultati prodotti da diversi tool di detection degli smell [15], che spesso sono basati su diverse tecniche di individuazione, ad esempio essendo *metrics-baed* o *rules-based* [15]. Inoltre, sono ancora da approfondire le relazioni di co-occorrenza tra diversi smell [15].

2.3 Data Smell

L'intelligenza artificiale, sempre più presente nelle nostre vite, dipende fortemente dai dati sui quali viene addestrata [2]. Secondo Foidl et al. [3], una crescente area di ricerca in questo ambito è quella dei **data smell**, che hanno un impatto notevole sulla qualità dei sistemi AI, causando technical debt [3].

2.3.1 Definizione

Per Foidl et al. [3], i data smell sono degli "**indicatori latenti di problemi di qualità nei dati *value-based*, indipendenti dal contesto, causati da cattive pratiche e che possono portare a problemi futuri**". Questi possono essere messi in parallelo con i code smell [3], in quanto i data smell possono essere causati da errori nella gestione dei dati o violazione di buone pratiche e possono influire negativamente sulla manutenibilità dei sistemi basati su AI [3], ma a differenza dei code smell, questi smell relativi ai dati possono rappresentare di per sé degli errori [3].

Dalla fonte [3] emerge come le caratteristiche che denotano i data smell siano: "*grado moderato di sospetto*", "*indipendenza dal contesto*" e *"causati da cattive pratiche e che possano causare problemi in futuro"*, mentre per Shome et al. [17] questi siano degli "anti-pattern" sintomi di technical debt e altri problemi. Esistono numerose

diverse definizioni dei problemi legati ai dati, come *data issues*, *dirty data*, *data quality problem*, *data anomaly*, *data defect* e infine i *data lints*, che però sono specifici delle feature utilizzate per il machine learning, a differenza dei data smell [3].

2.3.2 Classificazione

Ge e Helfert [18] dividono i problemi di qualità nelle informazioni tra dipendenti e indipendenti dal contesto [18]. In questo scenario, Foidl et al. [3] si riferiscono alle problematiche dipendenti dal contesto come "***data errors***", i quali causano certamente problemi nel tempo, e dividono ulteriormente le problematiche indipendenti dal contesto tra quelle *obvious* e *latent* [3]. Gli *obvious data quality issues* sono facilmente identificabili da tecniche di statistica descrittiva e hanno valori o pattern con un elevato livello di sospetto, mentre quelli latenti coincidono con i **data smell** [3], ed entrambi possono poi divenire dei data errors quando calati in un contesto [3]. Per catalogare ulteriormente i data smell, Recupito et al. [5] propongono una classificazione estesa dei diversi tipi di data smell proposti in letteratura [5] :

- **Believability Smells**: problemi legati alla affidabilità dei dati [5]
- **Consistency Smells**: problemi legati alla consistenza con cui sono rappresentati i dati [5]
- **Understandability Smells**: dati con problemi di comprensibilità [5]
  - **Encoding Smells**: problemi legati alla codifica dei dati [5]
  - **Syntactic Smells**: problemi legati alla sintassi dei dati [5]
- **Redundant Value Smells**: feature con valori ridondanti, che aggiungono quindi poca informazione al modello di IA [5]
- **Distribution Smells**: problemi legati al range di valori assunti da una feature [5]
- **Miscellaneous Smells**: problemi dei dati che non rientrano nei precedenti tipi [5]

Foidl et al. [3] riportano anche degli esempi reali degli effetti causati dalla presenza di data smell nei dati su cui sono stati basati dei sistemi AI di ambito medico, relativi al campo oncologico e al COVID-19 [3].

CAPITOLO 3

Stato dell'arte

3.1 Identificazione dei Data Smell

In questa sezione vengono presentate alcune tecniche di identificazione di Data Smell basate su diverse metriche e criteri, partendo da un'analisi del concetto affine di *Data Validation* e descrivendone alcune delle metodologie presentate in letteratura.

3.1.1 Data Validation

Secondo Foidl et al. [3], il concetto di identificazione dei data smell si può intedere come legato a quello della data validation. Questa è una fase fondamentale per le pipeline di machine learning, in quanto i data error possono causare problemi nel modello risultante e propagarsi nella *pipeline* [19]. Per svolgere data validation, si possono impiegare dei *data schema*, ovvero dei modelli logici dei dati che ne rappresentano la semantica [20], e anche le proprietà attese dai dati, in modo da validarli e segnalare le anomalie [21], utilizzando ad esempio strumenti come Tensorflow Extended TFX, proposto da Baylor et al. [21]. Foidl et al. [3] tra le tecniche di data validation menzionano Deequ, TensorFlow Data Validation (TFDV), Data Sentinel,

MobyDQ, Data Quality Toolkit, Data Quality Advisor e Great Expectations, che saranno di seguito descritte.

Deequ è una API dichiarativa che permette di validare i dati utilizzando dei vincoli di qualità e con la possibilità di definirne di nuovi, permettendo quindi di sviluppare dei "*test di unità per i dati*" [22, 23].

TensorFlow Data Validation (TFDV) è parte integrante di TFX [24] e si compone di tre parti principali: un *Data Analyzer* che utilizza dei generatori di statistiche per estrarre informazioni su feature individuali in base al loro tipo e informazioni "*cross-feature*" [24]; un *Data Validator* che fa utilizzo di *schema* per valutare le caratteristiche dei dati, operando sia su un singolo batch di dati che su due, contrassegnando le deviazioni da quanto atteso come anomalie [24]; e un *Data Visualizer* che fornisce una visualizzazione tabellare di quanto individuato [24].

Data Sentinel è una piattaforma di data validation dichiarativa, impiegata in LinkedIn, che permette agli utenti di specificare quali controlli effettuare [25], potendo anche utilizzare *Data Sentinel Service (DSS)*, che fornisce diversi tipi di supporto, come nella definizione di controlli e nell'interpretazione dei risultati della validazione [25].

MobyDQ [1] è uno strumento per automatizzare controlli sulla qualità dei dati nella *data pipeline*, segnalando le anomalie di qualità, fornendo inoltre indicatori di data quality come "*Completezza*", "*Validità*", "*Freshness*" e "*Latency*" [26], che permettono di rispondere ad alcune domande su diversi aspetti della qualità dei dati su cui si va ad operare [26].

Data Quality Toolkit è una libreria basata sulle componenti di *Data Quality Measurement*, che valuta diversi aspetti di qualità dei dati, utilizzando più metriche e fornendo una descrizione testuale e consigli su come correggere gli errori [27]; *Data Remediation*, che consente di modificare quanto individuato dalla componente di analisi, permettendo di coinvolgere un operatore umano per fornire una review e

[1] `https://ubisoft.github.io/mobydq/`

un feedback prima di apportare le modifiche ai dati [27]; e la componente di *Data Readiness Report*, che crea un record contenente tutti i dettagli e gli aspetti di quanto individuato e modificato tramite il tool [27], come proposto da Afzal et al. nella definizione di questo tipo di report [28, 27].

Data Quality Advisor è un framework ideato per la scalabilità che analizza i dati per creare report comprensibili e ideando delle pipeline che si occupino della qualità dei dati [29]. I controlli avvengono nel *DQA Core*, composto dalle componenti *Data Quality DAG* e *Scalable Execution Engine* [29].

Great Expectations [2] permette di definire delle "*Expectations*" sui dati, ovvero delle asserzioni flessibili che possono essere integrate ai test sui dati [30].

Esistono anche tecniche basate su regole di data quality definite a priori per individuare data errors, nel contesto del *data cleaning* [31]. Tuttavia, per Foidl et al. [3] le tecniche di data validation andrebbero slegate da quelle per l'individuazione dei data smell, in quanto questi causerebbero troppe segnalazioni di anomalie, a causa della natura degli stessi, che richiede di analizzarne anche il contesto per poterli identificare come problematici [3]. Questa ingente quantità di anomalie segnalate potrebbe quindi andare contro gli obiettivi del *"Data-Centric AI Learn"* [3, 32]. Per questi motivi, secondo Foidl et al. [3] i data smell non sono catalogabili come data errors e non possono essere trattati con le medesime tecniche, necessitando quindi di approcci che siano in grado di individuarli.

3.1.2 Data Smells

Recupito et al. [5] hanno realizzato una panoramica su tre tecniche per l'individuazione di data smell tra quelle presenti in letteratura. Due di queste sono quelle proposte da Foidl et al. [3], che operano *offline* basandosi su due nuove metriche: *Data Smell Strength*, ovvero la probabilità che un valore o un pattern dei dati si configuri come data smell [3]; e la *Data Smell Density*, che indica la densità di data smell individuati in un attributo [3]. Le tecniche presentate da Foidl et al. [3] hanno rispettivamente un approccio *rule-based* e uno *ML-based*.

[2] `https://greatexpectations.io/`

Rule-based Detection [3] è basata sul tool *Great Expectations*[2], e si concentra sull'identificare data smell tramite l'utilizzo di regole, scegliendo anche il livello di sospetto per cui segnalare una anomalia [3]. Tra i data smell in grado di essere individuati con questo approccio ricadono ad esempio *Casing*, *Floating Point Number as String* e *Integer as Floating Point Number* [3].

Machine Learning-based Detection [4] è invece uno strumento che utilizza diversi algoritmi di machine learning (come Word2Vec), le librerie di Python Dedupe[5] e Keras[6] e reti neurali per individuare diversi tipi di data smell, come *inconsistency smells*, *Ambiguous Date/Time Format* e *Date/Time Format Inconsistency*, permettendo anche di ri-addestrare i modelli su dati specifici [3]. La terza tecnica riportata dalla fonte [5] è invece "Data Validator", la componente di TFX già trattata nella Sezione 3.1.1.
Dal lavoro di Recupito et al. [5] emerge come anche gli autori Shome et al. [17] abbiano realizzato un'analisi dei data smell presenti in diversi dataset pubblici. Da questa sono emerse quattro categorie di data smell, ovvero i "*Redundant value smells*", "*Categorical value smells*", "*Missing value smells*" e "*String value smells*" [5, 17], con in aggiunta tre data smell raggruppati nei "*Miscellaneous smells*" [17]. Queste categorie sono state incluse da Recupito et al. [5] nella loro analisi della classificazione dei data smell. I data smell individuati da Shome et al. [17], per categorie, sono:

- "Redundant Value Smells"
  - "Correlated features": la presenza di feature nel dataset che sono correlate tra loro [17].
  - "Unique identifiers": la presenza di feature che indicano degli identificatori, che può causare problemi potenziali [17].
  - "Duplicate examples": la presenza di più righe nel dataset che "indicano la medesima entità" [17], potendo quindi essere rimosse [17].

[3] https://github.com/mkerschbaumer/rb-data-smell-detection
[4] https://github.com/georg-wenzel/ml-data-smell-detection
[5] https://docs.dedupe.io/en/latest/
[6] https://keras.io/

- "Categorical Value Smells"
  - "Hierarchy from label encoding": la codifica in label di valori categorici "*sensitive*" [17] può portare alla realizzazione di una gerarchia dei valori, potenzialmente "introducendo bias nelle prestazioni del modello" [17].
  - "Binning categorical features": la presenza di molti valori unici in una feature categorica. Può essere risolta accomunando categorie in "*bin*" [17].
- "Miscellaneous Smells"
  - "Presence of sensitive features": la presenza di informazioni sensibili impattanti che possono portare ad un modello che prende decisioni "*biased* e *unfair*" [17].
  - "Imbalanced examples": la presenza di categorie fortemente sbilanciate nella loro rappresentazione nel dataset [17].
  - "Unknown unit of measure": la mancanza di informazioni sull'unità di misura che descrive dei valori numerici in una feature del dataset può portare a problemi potenziali [17].
- "Missing Value Smells"
  - "Nulltype missing values": la presenza di molti "valori mancanti" nel dataset [17].
  - "Special missing values": i valori mancanti sono rappresentati con dei "valori speciali" [17], necessitando di una accurata documentazione [17].
  - "Binary missing values": i valori mancanti potrebbero rappresentare il valore opposto a quello delle istanze che contengono invece dati non-mancanti [17], portando a casi in cui i valori potrebbero quindi avere dei "significati impliciti" [17].
- "String Value Smells"
  - "Strings with special characters": la presenza di "spazi e caratteri speciali" in dati strutturati può portare a problemi nell'analisi del dataset [17].

  - "Numerical features as string": la presenza di feature in formato stringa che esprimono informazioni numeriche può rappresentare un problema nell'analisi effettuata dai tool [17].
  - "Strings in human-friendly formats": alcuni dati numerici potrebbero essere espressi in un formato di stringa per facilitarne la lettura, ma portando a possibili problemi nell'analisi [17].

Dall'analisi condotta da Shome et al. [17] emerge come i data smell di tipo Redundant e Categorical siano i più comuni, mentre quelli relativi alle stringhe e ai valori mancanti siano i meno presenti [17].

Cao et al. [33] presentano invece una tecnica per individuare *data-defect*, con cui gli autori indicano degli outlier. Questa metodologia fa utilizzo del "*Kernel-Neighbor-Density-Change Outlier Factor (KNDCOF)*", e quindi della "*Kernel Neighbor Density*", che rappresenta la densità degli oggetti presenti nei dati da analizzare [33]. Questo valore è usato come base di una serie di operazioni che portano al calcolo del KNDCOF, che viene in definitiva utilizzato dagli autori per valutare il grado con cui un oggetto può essere considerato un outlier rispetto agli altri dati presenti [33].

3.2 Impatto dei Data Smell

Dopo aver analizzato come può avvenire l'identificazione dei data smell, risulta importante capire quale sia effettivamente l'impatto che questi apportano alle metriche di valutazione dei dati stessi. Recupito et al. [5] a tal proposito hanno effettuato un'analisi, basata su quella condotta da Le Quy et al. [34], su 19 dataset di diversi domini [5]. Le metriche scelte dagli autori [5] per l'analisi sono alcune di quelle proposte da Elouataoui et al. [35] che mirano a valutare la qualità dei dati [5]. Le metriche scelte da Recupito et al. [5] sono le seguenti:

- "**Completezza**": la percentuale di valori non vuoti sul totale [5].
- "**Unicità**": la percentuale di righe nel dataset uniche sul totale [5].
- "**Consistenza**": la percentuale di valori con una "struttura consistente con lo schema dei dati" sul totale [5].

- "**Comprensibilità**": la percentuale di valori senza "errori di spelling o altri errori semantici" [5].

Dopo la scelta delle metriche, gli autori Recupito et al. [5] hanno verificato l'esistenza di una correlazione statisticamente significativa tra la presenza di alcuni data smell e l'andamento delle metriche selezionate, denotando delle correlazioni significative con alcune delle metriche di data quality, seppur con diversi livelli di significatività statistica [5], di seguito riportate:

- "**Extreme Value Smell**": correlazione positiva con Unicità, Consistenza e Comprensibilità; correlazione negativa con Completezza [5].
- "**Missing Value Smell**": correlazione positiva con Unicità; correlazione negativa con Consistenza e Completezza [5].
- "**Casing Smell**": correlazione positiva con Unicità; correlazione negativa con Comprensibilità [5].
- "**Suspect Sign Smell**": correlazione positiva con Unicità e Comprensibilità [5].
- "**Floating Point Number As String Smell**": correlazione positiva con Consistenza e Comprensibilità [5].

In seguito a questa analisi, Recupito et al. [5] hanno anche creato dei modelli GLM (*Generalized Linear Model*) per analizzare nel dettaglio le relazioni tra data smell e metriche emerse nel precedente studio [5]. Questo approfondimento ha rivelato come, anche se l'impatto di una singola occorrenza di un data smell su una particolare metrica di data quality sembri molto limitato, il livello di significatività emerso dall'analisi delle correlazioni ne rende pericolosa la presenza quando in grandi quantità [5]. In particolare, le correlazioni maggiormente significative per le diverse metriche di data quality sono le seguenti [5]:

- "**Completezza**": "Missing Value Smell" (corr. negativa) [5].
- "**Unicità**": "Casing Smell" (corr. positiva) [5].
- "**Consistenza**": "Missing Value Smell" (corr. negativa) [5].

- "**Comprensibilità**": "Floating Point Number As String Smell" (corr. negativa), "Casing Smell" (corr. negativa) e "Extreme Value Smell" (corr. positiva) [5].

Queste analisi condotte da Recupito et al. [5] sono quindi focalizzate sul valutare l'impatto dei data smell su metriche relative alla qualità dei dati, in particolare concentrandosi su delle metriche indipendenti dal contesto. Tuttavia, i data smell individuati potrebbero influire anche su metriche di qualità context-dependent, ovvero legate al loro contesto di utilizzo. Per questo motivo, l'attenzione di questo lavoro si concentra su questo aspetto, investigando in che modo la presenza di questi data smell possa impattare nel contesto del training di modelli di machine learning. Anche Foidl et al. [3] hanno previsto di valutare gli effetti della presenza di *data issue* sui modelli di machine learning, tuttavia gli aspetti in cui questo lavoro si differenza dalla proposta di Foidl et al. [3] sono riportati nella Sezione 4.

Per valutare possibili modi in cui arricchire i dataset su cui andranno iniettati i data smell, è ora necessario effettuare un'analisi sul concetto di *data augmentation*.

3.3 Tecniche di Data Augmentation

Di seguito sono descritte alcune delle tecniche di *Data Augmentation* presenti in letteratura, in particolare concentrandosi su quelle che operano su dati tabulari, partendo con una panoramica su come viene definita questa pratica.

3.3.1 Definizioni

La data augmentation è una pratica utilizzata per aumentare i campioni da utilizzare per l'addestramento di modelli di machine learning, in scenari in cui tipicamente i dati non sono molto disponibili, andando a ottenere dataset con maggiori volumi, qualità e differenze nei campioni che lo compongono [36]. A. Mumuni e F. Mumuni [36] descrivono formalmente il task della data augmentation come la "trasformazione di campioni di training S di un dataset D, aventi delle label associate L, per la creazione di nuovi campioni S', mantenendo le label L" [36]. Questa è una pratica molto importante nell'ambito della computer vision, che può essere condotta anche tramite l'utilizzo di reti neurali, GAN e meta-learning [36]. A. Mumuni e F. Mumuni

[36] suddividono questi approcci, specifici per lo scenario d'utilizzo nella computer vision, in tecniche di *"Data transformation"*, di *"Data synthesis"* e di *"Meta-learning"*, con quest'ultima che può essere anche applicata alle prime due [36]. Le tecniche di Data transformation si possono basare sul modificare in modo diretto l'immagine di input, seguendo logiche geometriche o fotometriche (approcci *"traditional"*) oppure a livello di campione, di regione o di pixel (approcci *"advanced"*), utilizzando anche strumenti di deep learning come layer CNN e GAN [36]. Anche i metodi di "*Feature space*" fanno parte degli approcci di Data transformation, e si concentrano sul modificare le "*feature map*" delle deep CNN per andare ad ottenere in questo modo nuovi campioni, utilizzando le tecniche di "*Feature addition*", "*Feature dropping*", "*Feature mixing*" e "*Feature interpolation*" [36]. Gli approcci di "Data synthesis" mirano a generare nuove immagini da zero, utilizzando diverse tecniche, come ad esempio il *neural rendering* e il *Generative modeling*, che fa utilizzo di GAN e di CGAN [36]. Infine, gli approcci di "Meta-learning" si concentrano nell'operare sui meta-dati in modo da poter catturare informazioni da usare su diversi task, oppure mirano a identificare le migliori operazioni di data augmentation in base alla specifica casistica, andando ad automatizzare il processo [36].

3.3.2 Data augmentation per dati tabulari

Il lavoro di Cui et al. [37] si concentra invece sull'analizzare le tecniche di data augmentation in uno scenario diverso da quello trattato fin'ora, ovvero quello dei dati tabulari, che fa parte di un mercato in crescita e di grande impatto economico [37]. In questo caso, la ***Tabular Data Augmentation*** è definita come "un processo per arricchire i dataset tabellari originali, utilizzando dati da fonti esterne oppure generati sinteticamente, per migliorare le performance dei modelli di machine learning che ne fanno utilizzo" [37]. Questo task risulta essere particolarmente complesso a causa della struttura dei dati tabulari, che possono presentare organizzazioni gerarchiche, avendo milioni di righe e dati eterogenei tra loro [37]. Cui et al. [37] racchiudono le principali tecniche di TDA in due categorie, ovvero "***Retrieval-based TDA***" e "***Generation-based TDA***". La Retrieval-based TDA genera nuovi dati attingendo da fonti di dati esterni realistici e *table pools*, mentre la Generation-based TDA non ne

fa utilizzo e mira a generare dati sintetici utilizzando spesso modelli generativi [37]. Entrambe queste categorie si suddividono in sotto-task per la data augmentation che operano a diversi livelli di dettaglio, ovvero a *row-level*, andando a generare nuove righe, a *column-level*, generando nuove colonne, a *cell-level*, generando specifiche celle per determinate righe che potrebbero presentare dati mancanti, e a *table-level*, generando sia nuove righe che nuove colonne [37]. Dati gli scopi di questo lavoro, questa analisi si concentrerà solo sui sotto-task di *row-level*, che sono ***Entity Augmentation*** e ***Record generation*** [37]. L'Entity Augmentation è il sotto-task Retrieval-based che arricchisce un dataset di partenza utilizzando righe, o parte di esse, prese dai table-pool, cercando le tabelle adeguate per l'unione tramite tecniche di rappresentazione che consistono nel convertire i record in *latent-space vectors*, per poi classificare i livelli di legame della tabella originale con quelle target [37]. Questo sotto-task si suddivide ulteriormente in 4 categorie, ovvero ***Statistical***, ***KB-based***, ***Graph-based*** e ***PLM-based***, di seguito descritti [37]. Gli approcci di tipo statistico fanno utilizzo di modelli statistici per valutare la possibilità di unione tra tabelle, tramite tecniche come la *cosine similarity* sui vettori TF-IDF oppure tramite altre misure statistiche, ma in generale non scalano bene per dataset di grandi dimensioni [37]; gli approcci KB-based fanno utilizzo anche di knowledge-graph per valutare la possibilità di unione tra dataset tramite diversi metodi, ma possono portare ad una bassa *recall* [37]; gli approcci graph-based mirano a rappresentare le tabelle sotto forma di grafi, per poi calcolare le possibilità di unione tra esse tramite le relazioni che sussistono tra le loro rappresentazioni, in base a diversi criteri e tecniche di rappresentazione diverse, che possono rendere questo approccio più costoso in termini di risorse [37]; e gli approcci basati su pre-trained language model (PLM), che spesso valutano le relazioni tra le tabelle grazie ad informazioni estratte dai PLM tramite la codifica delle stesse, ma sono limitati da alcune caratteristiche dei dati tabellari, come la loro strutturazione [37]. Il sotto-task di Record generation invece si occupa di generare nuove righe sintetiche partendo dai dataset originali oppure da delle *sub-tables*, suddividendo il task in due categorie, in base all'approccio che si vuole ottenere sulla distribuzione dei dati generati rispetto a quella originale, ovvero la *distribution-preserving record generation* e la *class-imbalance-aware record generation* [37]. La *distribution-preserving record generation* calcola la distribuzione statistica dei dati originali e mira a generare

record che la mantengano, utilizzando approcci statistici, come ad esempio le reti Bayesiane, e strumenti come le reti neurali [37]. In particolare è diffuso l'utilizzo di: generative-adversarial-network (GAN), che possono migliorare l'aderenza dei dati alla semantica originale; *diffusion models*, utilizzando diversi approcci come ad esempio il processo di *reverse diffusion*, oppure utilizzando delle condizioni per permettere ai dati generati di rispettare determinati criteri; e LLM, che possono apprendere diverse caratteristiche dei dati tramite il fine-tuning [37]. La *class-imbalance-aware record generation* mira invece a generare dei dati che siano in grado di bilanciare il dataset originale utilizzando delle varianti delle GAN, come ad esempio le CTGAN (Xu et al. [38]), in grado di imporre condizioni su un valore discreto, delle Wasserstein GAN oppure delle Sequence-to-Sequence Recurrent Neural Network (RNN) per generare dati sintetici in grado di compensare gli sbilanciamenti originali [37]. Cui et al. [37] propongono anche una comparison tra i diversi metodi di generazione ai medesimi livelli di dettaglio, dalla quale emerge che i metodi Retrieval-based di generazione a row-level si concentrino più sull'analisi delle relazioni tra i dataset in input e i table pool [37], mentre quelli Generation-based si focalizzano sul modo in cui i nuovi dati influiscono le distribuzioni di partenza, portando a dati più coerenti a quelli di partenza e quindi senza poter introdurre dati completamente nuovi [37].

Synthetic Data Vault (SDV)

Patki et al. [39] hanno proposto un sistema chiamato **"Synthetic Data Vault" (SDV)**, che ha come obiettivo quello di generare dati sintetici, partendo da interi database o da singole tabelle, creando dei modelli generativi che riescano ad emulare i dati di partenza sia da un punto di vista statistico che strutturale [39]. La sequenza di operazioni che permette a tale tool di generare i nuovi dati richiesti dagli utenti inizia con il fornire in input il database comprensivo dei metadati che descrivono le tabelle, in modo da catturare anche le relazioni che sono presenti tra di esse [39]. In seguito, è possibile apprendere i modelli generativi per ogni singola tabella tramite SDV, permettendo poi di generare la quantità richiesta di nuovi dati [39]. Il cuore del processo risiede ovviamente nella fase di modellazione, che avviene utilizzando anche un metodo denominato "Conditional Parameter Aggregation" (CPA), che per-

mette di catturare nel processo le relazioni esistenti tra le diverse tabelle del database da apprendere [39]. Il modello generativo per una singola tabella, che comprende tutte le colonne contenenti solo dati numerici, è composto dalle **distribuzioni** dei dati e dalle **covarianze** tra le colonne [39]. Per carpire le distribuzioni dei dati di ogni colonna, queste vengono confrontate con alcune distribuzioni comuni, come la Gaussiana, la Gaussiana troncata, la distribuzione uniforme, ed altre possibili distribuzioni implementabili, tramite il test di Kolmogorov-Smirnov, scegliendo la distribuzione per la quale viene fornito il *p-value* maggiore [39]. Per carpire invece le covarianze, si utilizza la copula Gaussiana per convertire tutte le colonne ad una distribuzione normale standard [39]. Nel caso in cui la tabella in analisi non sia una tabella "foglia", ma quindi che questa sia referenziata da altre tabelle "figlie", allora tramite il metodo CPA vengono calcolate ed aggiunte nuove colonne alla tabella in analisi che rappresentano le distribuzioni e le covarianze dei dati a essa legati, in aggiunta ad una informazione sul numero di figli di ogni elemento genitore [39]. In questo modo è possibile carpire anche le dipendenze tra i "*parametri condizionali*" (ovvero le matrici di distribuzioni e di covarianze) e quelli della tabella originale. [39]. Per poter eseguire questi passi è però necessario che le colonne non abbiano dati mancanti e che siano tutte numeriche; nel caso in cui il database fornito a SDV non rispetti tali condizioni, questo deve essere modificato attraverso una fase di pre-processing [39], che si può comporre anche di più round nel caso in cui siano necessarie più trasformazioni [39]. Nei casi in cui i dati mancanti abbiano un significato all'interno del database, ad esempio se un dato è stato omesso volontariamente o se questa scelta è legata a specifici valori di altre colonne, i valori *null* devono essere modellati da SDV e inglobati nelle possibili casistiche in cui i dati generati potranno ricadere [39]. Questa scelta viene mantenuta anche nel caso in cui la presenza di dati mancanti non dia informazioni aggiuntive, poiché non è possibile per il tool discernere questi diversi scenari [39]. Per questi motivi, quando SDV incontra una colonna con almeno un valore mancante, separa la colonna in due, generando una nuova colonna con dati casuali presi dalla colonna originale per rimpiazzare i valori mancanti [39], ed una colonna aggiuntiva che segnala la presenza o meno di un dato mancante nell'informazione originale [39]. Anche i dati categorici devono essere modificati in quanto non possono essere trattati dal metodo CPA [39], per cui questi sono rimpiazzati da

valori nel range "[0, 1]" utilizzando la distribuzione di probabilità cumulativa delle categorie e per ognuna di esse campionando i valori da una distribuzione Gaussiana troncata costruita in base al suo intervallo. [39]. Anche le colonne rappresentanti dei *timestamp* vengono convertite andando a rimpiazzarle con il numero di secondi trascorsi dall'inizio del 1970 fino alla data da rappresentare [39]. Passando invece alla fase di generazione di nuovi dati sintetici, questa può avvenire in modalità Model-Based, se si vuole usufruire interamente del modello generativo così creato, o in modalità Knowledge-Based, se invece si vogliono integrare dei dati già presenti con informazioni da generare [39]. Con l'approccio Model-Based, utilizzando le distribuzioni e le covarianze apprese nello step precedente, è possibile andare a campionare i valori numerici per ogni colonna, riconvertendo anche i dati nei loro tipi originali effettuando i passi inversi [39], e quindi inserendo anche dei valori nulli nei casi in cui la colonna ausiliaria li indichi come presenti [39]. Per sintetizzare un'intera riga viene quindi impiegato questo processo, distinguendo i casi in cui la tabella sia o meno figlia di un'altra [39]. Per lo scenario Knowledge-Based è invece necessario cambiare questo approccio, dovendo aggiungere due metodi chiamati *Sampling Updates*, che va a modificare la struttura dei dati in base agli input forniti, modificando anche le distribuzioni e le covarianze, e quindi impattando sul processo di campionamento dei dati da generare [39], e *Parent Inference*, che invece mira ad inferire i valori assunti dalle righe di una tabella basandosi sui valori forniti nelle righe che le sono figlie [39]. Da un esperimento condotto da Patki et al. [39] emerge come non sussista una differenza significativa nell'accuracy ottenuta da modelli predittivi costruiti con dati sintetici rispetto a quella ottenuta da modelli che usano solo i dati reali originali [39], portando quindi a valutare l'utilizzo di questo tool per le sperimentazioni di questo lavoro.

Per la gestione di database contenenti più tabelle, viene applicato il metodo di CPA ricorsivo (RCPA), che consiste nell'applicare CPA ricorsivamente a tutte le tabelle figlie di una data tabella, e calcolando quindi le distribuzioni e le covarianze del database [39].

GAN e CTGAN

Il lavoro di Xu et al. [38], citato anche da Cui et al. [37], dettaglia invece sull'utilizzo delle GAN nella generazione di dati tabulari e propone il metodo delle **Conditional Tabular GAN (CTGAN)**, che mira a ridurre i limiti alla generazione di dati tabulari tramite GAN, relativi alle metriche di "*likelihood fitness*" e "*machine learning efficacy*" [38]. La generazione di dati sintetici tramite GAN richiede l'addestramento di un sintetizzatore su una tabella che contiene feature numeriche sia discrete che continue, trattate come variabili casuali con distribuzione congiunta sconosciuta [38], cercando di generare dei dati sintetici che seguano le medesime distribuzioni congiunte dei dati originali (valutando la metrica *"likelihood fitness"*) e di ottenere prestazioni simili nel creare un classificatore o un regressore sui dati sintetici rispetto a quelle ottenute usando i dati originali (valutando la metrica *"machine learning efficacy"*) [38]. I limiti riportati da [38] su questa tecnica sono legati alle caratteristiche che possono avere i dati tabulari e che non sono ben catturate dalle GAN, come la presenza di "dati con tipi misti", "dati con distribuzioni non-Gaussiane", la presenza di "distribuzioni multi-modali non rappresentabili in dataset bidimensionali", la presenza di "feature categoriche molto sbilanciate" e "l'apprendimento di dati rappresentati con dei vettori *one-hot encoded*, e quindi sparsi" [38]. Xu et al. [38] propongono quindi le CTGAN per superare alcuni di questi problemi tramite delle tecniche chiamate "***mode-specific normalization***", "***conditional generator***" e "***training-by-sampling***" [38]. La normalizzazione *mode specific* modella le distribuzioni delle colonne con valori continui tramite un "*variational Gaussian mixture model*" (VGM) [38], e per ogni valore assunto nelle colonne computa la probabilità che questo provenga da ogni moda, per poi campionare la moda da inserire in un vettore *one-hot encoded* insieme ad uno scalare che indichi il valore, in modo quindi da rappresentare anche distribuzioni più complesse [38]. L'utilizzo di un *conditional generator* mira invece a superare i limiti delle GAN con i dati sbilanciati, che derivano dal loro utilizzo di distribuzioni standard multivariate (MVN) per campionare i vettori da dare in input al generatore [38], poiché nell'ottenere delle trasformazioni deterministiche che mappino le MVN nella distribuzione dei dati non vengono apprese correttamente le categorie poco rappresentate [38]. L'obiettivo di questa tecnica è quindi quello di campionare i dati

in modo equo nel training, seppur non sempre in modo uniforme, per poi passare alla vera distribuzione dei dati nella fase di test [38]. Per raggiungere questo obiettivo sono utilizzati tre elementi chiave: il *conditional vector*, la *generator loss* e il metodo *training-by-sample*, che permette di stimare meglio la distanza tra la distribuzione condizionale appresa e quella reale [38]. Per la *network structure* sono poi utilizzate delle reti *generator* e *critic*, realizzate tramite due *fully-connected hidden layers* [38]. Dagli esperimenti condotti da Xu et al. [38] emerge come questo tipo di modello presentato possa apprendere una distribuzione migliore rispetto a quanto fatto dalle reti Bayesiane, fornendo quindi una valida tecnica per la data augmentation di dati tabulari [38].

Basandosi su questo lavoro appena descritto ([38]), è stato anche implementato un sintetizzatore di tipo CTGAN nel tool SDV[7] ([39]).

3.4 Utilizzo dei Large Language Model

Come riportato da Cui et al. [37], l'utilizzo della IA generativa per la data augmentation in scenari con dati tabulari sta ancora muovendo i suoi primi passi [37]. Li et al. [40] riportano una collezione di lavori in letteratura che trattano dell'utilizzo dei LLM per l'augmentation di dataset e per la generazione totale di essi, affermando come le tecniche di data augmentation facenti utilizzo di LLM abbiano avuto dei risultati positivi se comparate ad altri approcci [40], mentre la valutazione sulla generazione totale di dataset sia ancora incerta [40].

3.4.1 Data augmentation tramite LLM

Di seguito è riportata una descrizione di alcune fonti citate da Li et al. [40] che si occupano di realizzare *data augmentation* tramite LLM.

Sahu et al. [41] propongono una tecnica basata sui prompt per la generazione di dati di training già etichettati per poter poi effettuare intent classification, utilizzando

[7] `https://docs.sdv.dev/sdv/single-table-data/modeling/synthesizers/ctgansynthesizer`

diverse versioni di GPT-3 come modelli per la *data augmentation* (Ada, Davinci, Babbage e Curie) e senza necessitare fine-tuning task-specific [41]. L'approccio di prompting è del tipo ***few-shots***, fornendo quindi pochi esempi al pre-trained LLM, come l'esempio riportato da Sahu et al. nella Figura 1 del loro lavoro [41]. Il template per questi prompt fornisce quindi una categoria ed alcuni esempi, lasciando l'ultimo esempio vuoto per l'auto-completamento di GPT-3, che però in alcuni casi non risulta adeguato [41]. Il task affrontato da Sahu et al. [41] è quindi quello di classificazione di intent in un ambiente di agenti conversazionali, utilizzando BERT-Large *fine-tuned* e validato con diversi approcci sui dataset scelti [41]. Dai risultati ottenuti dagli autori emerge come una maggiore fedeltà ai dati forniti nella generazione non implica sempre una maggiore qualità degli stessi, in quanto una alta fedeltà spesso risulta in frasi troppo simili a quelle d'esempio [41]. I risultati ottenuti dagli autori indicano come la *data augmentation* ottenuta con gli approcci *full few-shot* e *partial few-shot* porti generalmente a miglioramenti nell'accuracy delle classificazioni effettuate, seppur con qualche difficoltà in alcune casistiche [41]. Sahu et al. [41] fanno notare come l'utilizzo di questi tool potrebbe potenzialmente portare a problemi di *bias* e di tossicità nei contenuti generati, che richiederebbero quindi una supervisione umana [41].

Il lavoro di Hartvigsen et al . [42] è molto legato a questo tema, in quanto tratta della realizzazione di un dataset sintetico contenente frasi sia con *hate speech* che senza, chiamato ToxiGen, utilizzando GPT-3 [42]. L'obiettivo di questo dataset è quello di effettuare il fine-tuning di un classificatore per individuare la tossicità in modo accurato [42]. Nel processo di generazione, gli autori hanno utilizzato il "*demonstration-based prompting*" per la creazione sia di frasi benigne sia di frasi tossiche, con impliciti riferimenti a minoranze , fornendone quindi delle dimostrazioni al LLM [42]. Per ottenere numerosi esempi, gli autori hanno espanso un insieme di frasi collezionate da varie fonti, tra cui social network, con delle altre generate dal LLM e filtrate manualmente [42]. In seguito a questa fase, in una delle versioni di ToxiGen, Hartvigsen et al. [42] hanno introdotto una fase di *adversarial decoding* per generare delle frasi più difficilmente categorizzabili per un classificatore di tossicità pre-addestrato (HateBERT e OffensEval), in modo da ottenere frasi più

utili al miglioramento del modello, mirando a ottenere falsi positivi e falsi negativi [42]. Durante la valutazione dei risultati, Hartvigsen et al. [42] hanno denotato come effettuare il fine-tuning di modelli pre-addestrati su ToxiGen abbia migliorato le performance di classificazione su tre dataset [42].

Yoo et al. [43] propongono un diverso metodo di *data augmentation* per task di classificazione, chiamato GPT3Mix, che genera dati sintetici realistici da esempi reali [43]. Questa tecnica, ispirata da Mixup (Zhang et al. [44]), parte da un dataset di riferimento per costruire dei prompt da esempi e "*meta-information*" riferite al dataset per quindi generare nuovi campioni utilizzando versioni di GPT-3 [43]. La scelta e l'ordine degli esempi del dataset selezionati sembra impattare notevolmente nella generazione dei dati [43] , e in questo caso viene utilizzata una distribuzione uniforme per la scelta dei *k* esempi, che sono poi inseriti in un prompt strutturato per indicare al LLM di generare i dati nel formato richiesto, descrivendo in modo chiaro al LLM le parti che lo compongono e fornendo gli esempi individuati [43]. Grazie al prompt, il LLM genera quindi nuovi esempi in un formato strutturato, che ne facilita l'estrazione [43]. Questa tecnica di generazione combina la *text-perturbation*, lo *pseudo-labeling* (ovvero utilizzare le predizioni di un modello come etichette per dati che ne sono sprovvisti) e la *knowledge distillation* (ovvero una tecnica che utilizza gli output di un classificatore per effettuare l'addestramento di un altro classificatore, ma di dimensioni ridotte) [43]. I risultati riportati dagli autori su sette dataset sembrano essere molto migliori di alcuni approcci per la *data augmentation* [43], ed in generale presentano un comportamento molto consistente, divenendo quindi una alternativa valida al fine-tuning [43].

Kumar et al. [45] hanno affrontato il task della "*conditional data augmentation*" studiando le performance di GPT-2, BERT e BART [45]. Nei metodi proposti da Kumar et al. [45] viene generato un campione sintetico da ogni esempio di training, al quale è associata la label [45]. In particolare, per i pre-trained language model, gli autori hanno analizzato due metodi per aggiungere la label alle sequenze di dati di training, e quindi per realizzare la ***conditional*** *data augmentation* [45], che consistono nell'aggiungere (metodo *expand*) o meno (metodo *prepend*) la label al vocabolario del

modello, dopo averla preposta ad ogni sequenza di training [45]. Anche per BART gli autori hanno preposto le label a tutti gli esempi di una determinata classe, addestrando poi il modello con la tecnica di *masking* sia a livello di *word* che di *sub-word* [45]. I risultati della classificazione realizzata dai modelli addestrati, ottenuti su tre dataset distinti, sono stati poi comparati dagli autori con tre approcci baseline, realizzando una valutazione estrinseca e una intrinseca [45]. Tra i risultati ottenuti, è emerso come gli autori [45] consiglino la tecnica *prepend* per effettuare data augmentation tramite dei pre-trained model. [45]. Kumar et al. [45] hanno denotato come questi modelli analizzati possano essere un'alternativa per il task di *conditional data augmentation* effettuandone il fine-tuning come descritto, ovvero preponendo la class label ai dati d'esempio [45].

3.4.2 Generazione completa di dataset usando LLM

Ye et al. [46] propongono il metodo di zero-shot learning denominato "*ZeroGen*", che addestra un "*tiny task model* (TAM)", come ad esempio un LSTM, utilizzando un dataset generato tramite dei language models pre-addestrati per uno specifico task in logica non supervisionata [46], per avere quindi un modello di dimensioni molto inferiori rispetto a quelle di un language model pre-addestrato, e non necessitando quindi di annotazioni manuali dei dati [46]. La generazione dei dati tramite language model avviene tramite dei prompt progettati dagli autori per essere task-specific, come quelli riportati dagli autori Ye et al. nella Tabella 11 del loro lavoro [46].
I dati così generati sono quindi forniti ad un TAM per effettuarne l'addestramento supervisionato, in quanto il dataset sintetico contiene anche le label associate ai campioni, per poi concludere con la fase di evalutation [46]. I risultati ottenuti da ZeroGen, valutati usando tre diversi language model e due diversi TAM, sembrano essere migliori, sulla maggior parte dei dataset testati, dei corrispettivi ottenuti dai pre-trained language model in logica zero-shot, utilizzando anche molti parametri in meno [46]. ZeroGen raggiunge risultati anche migliori di quelli ottenuti da modelli addestrati in modo supervisionati con dati etichettati manualmente, in scenari *low-resourced* [46].
Per gli autori, le performance dei task sono anche da trovarsi nella qualità dei dataset

sintetici generati, ad esempio nel compromesso che viene effettuato tra *Diversity* e *Correctness* [46].

Gao et al. [47] hanno proposto un framework, chiamato SunGen, per creare dati per task di classificazione *zero-shot*, cercando di risolvere i problemi di qualità delle generazioni di esempi causati dal "*data-generation based zero-shot learning*" di Ye et al. ([46]), causati dall'overfitting dei modelli ai dati generati, che porta ad un declino dell'accuracy nella fase di test [47]. La tecnica proposta in questo caso consiste in un framework che effettua un "*reweighting*" dei pesi tramite una ottimizzazione su due livelli, partendo da un validation set sintetico che presenta dati "*noisy*" [47]. Il framework apprende quindi i pesi dei campioni andando a valutare la qualità dei dati, in modo da poter ottenere un dataset "pulito" senza la necessità di interventi manuali, aiutando a migliorare l'accuracy in alcuni task di classificazione [47].

Tang et al. [48] hanno proposto un framework per la generazione di dati sintetici etichettati nel contesto dell'*healthcare*, per evitate le problematiche legate alla privacy e alle risorse richieste per la collezione di dati strutturati da dare in input ai LLM per effettuarne il training [48]. Per fare ciò, viene proposto un nuovo approccio al training, che utilizza i LLM per la generazione di dati sintetici tramite degli specifici prompt, che sono poi filtrati in base alle loro caratteristiche, per poi effettuare il fine-tuning di un PLM locale [48]. L'obiettivo di Tang et al. [48] è poi quello di realizzare i task di Named Entity Recognition e Relation Extraction, poiché ChatGPT non produce risultati soddisfacenti quando impiegato per questi task in contesti medici, forse a causa dei pochi dati di questo genere presenti nel training del modello, secondo la fonte [48]. I cinque prompt considerati dagli autori per la generazione dei dati, consigliati proprio da ChatGPT, sono stati impiegati per generare nuovi esempi, da poi utilizzare per il fine-tuning di BERT, RoBERTa e BioBERT [48]; il miglior tipo di prompt per la Named Entity Recognition consiste in una richiesta a ChatGPT di generazione di nuovi esempi che contengano determinate parole [48], aggiungendo dettagli su come la qualità dei dati debba essere simile a quella degli articoli di stampo medico e su come non debbano contenere altri dettagli [48], mentre il prompt per la Relation Extraction contiene più dettagli su come formattare

gli esempi, includendo anche in questo caso la richiesta di imitare i dati presenti sulla banca dati indicata [48]. I risultati ottenuti dagli autori denotano come i tre modelli fine-tuned sui dati sintetici abbiano migliori performance nei task di Named Entity Recognition Extraction rispetto a ChatGPT (con paradigma zero-shot) [48], e comparabili in alcuni casi anche a quelle ottenute quando fine-tuned sui dati originali, con il task di Recognition Extraction che mostra le minori differenze in quest'ultimo confronto [48].

Da un'analisi degli autori è emerso come i dati generati non siano eccessivamente simili rispetto a quelli originali, spiegando così il peggioramento delle performance dei modelli fine-tuned sui dati sintetici rispetto a quelli originali [48], che però apporta comunque un notevole miglioramento rispetto all'impiego di zero-shot ChatGPT [48].

Wang et al. [49] propongono una procedura per la generazione di dati sintetici chiamata "*Unsupervised Data Generation*", utilizzando un approccio di prompting di tipo "*few-shot*" [49]. Gli autori utilizzano Large Language Model come "*few-shot generators*" per ottenere dati sintetici etichettati da utilizzare per effettuare il fine-tuning di altri modelli, utilizzando la tecnica del *top-k sampling* [49].

I risultati ottenuti da questa tecnica nel task di unsupervised text classification, con una fase di filtro dei dati tramite *Noisy Label Annealing*, vanno generalmente a superare quelli derivanti da tecniche few-shot, in particolare UDA [49]; per il task di unsupervised language undestanding, il modello proposto dagli autori ha risultati migliori delle altre tecniche analizzate di tipo *few-shot* nella maggior parte dei casi [49]; nello scenario in cui viene utilizzata la tecnica di Unsupervised Data Generation per effettuare *data augmentation*, andando poi a svolgere il fine-tuning di T5-XXL, i risultati migliorano ulteriormente, superando in media gli altri approcci [49]. I template per i prompt utilizzati da Wang et al. [49] per effettuare text classification sui diversi dataset sono riportati nell'Appendice C del loro lavoro [49].

Dopo aver analizzato questi lavori, sarebbe possibile quindi concludere che l'utilizzo dei LLM porti grandi vantaggi ai processi di *data augmentation* e che possano essere impiegati dei framework che permettono anche di generare interi dataset sintetici per poter eseguire determinati task. Tuttavia, Li et al. [40], avendo analizzato

le fonti precedentemente descritte in questa Sezione, pongono l'attenzione su quale sia il discriminante che permette di utilizzare in maniera efficiente i LLM per la *data augmentation* e la generazione di dati [40]. In particolare, gli autori Li et al. [40] ipotizzano che questo fattore sia il livello di **soggettività** del task di classificazione in esame, analizzando come questo vada ad impattare sulle performance [40]. Dopo aver quindi analizzato diversi task di classificazione, sia con approcci di generazione di tipo *zero-shot* che di tipo *few-shot*, Li et al. [40] hanno esaminato la relazione che intercorre tra i risultati e la soggettività *task-level*, relativa al tipo di task di classificazione [40], e quella che si denota invece con la soggettività *instance-level*, legata invece alla soggettività emersa dalla classificazione manuale [40]. Per lo scenario *zero-shot*, gli autori Li et al. [40] hanno fornito a GPT-3.5-Turbo prima un prompt per impostare il contesto relativo ai dati da generare [40], poi con un secondo tipo di prompt hanno realizzato la generazione dei dati, indicando anche le caratteristiche che questi devono avere [40], e infine con un terzo prompt viene chiesto di diversificare i prossimi dati generati rispetto alle ultime generazioni realizzate [40]. Per lo scenario *few-shot* invece, dopo il prompt mirato ad impostare il contesto, sono aggiunti degli esempi prima di richiedere la generazione di nuovi dati [40], specificando anche di non limitarsi a modificare i dati forniti o ripeterli per la creazione dei nuovi [40]. I risultati ottenuti da Li et al. [40] mostrano come, per entrambi gli scenari descritti, generalmente all'aumento dei livelli di soggettività, sia a task-level che ad instance-level, corrisponda un peggioramento nelle metriche di valutazione ottenute dai modelli, quando questi sono addestrati su questi dati sintetici [40].

A seguito di queste analisi, risulta quindi interessante impiegare i LLM per questo lavoro, valutandone le loro capacità nel generare dati affetti da data smell.

CAPITOLO 4

Design

4.1 Obiettivo

L'obiettivo di questo lavoro è di realizzare uno studio empirico sull'impatto dei data smell sulle metriche di accuracy e di performance di alcuni modelli di classificazione, utilizzando anche tecniche di generazione di dati sintetici. Anche Foidl et al. [3] nel loro lavoro propongono di valutare l'effetto della presenza di *data issue* sui modelli di machine learning, tuttavia in questo caso l'analisi sarà concentrata solo su un sottoinsieme dei data smell, e inoltre saranno utilizzate tecniche di iniezione dei data smell che non fanno utilizzo del medesimo framework che Foidl et al. [3] intendono applicare, e aggiungendo inoltre la generazione di dati sintetici alla sperimentazione e alla valutazione dei risultati, differenziando quindi dalla proposta di Foidl et al. [3].

Per definire formalmente l'obiettivo di questo studio è stato utilizzato il template *Goal Question Metric* proposto da Caldiera et al. [50].

◎ Our Goal.
Purpose: Valutare
Issue: l'impatto dell'iniezione di alcuni tipi di data smell nei training set
Object: sull'accuracy e sulle performance dei modelli di classificazione,
Viewpoint: dal punto di vista dei data scientist.

Dall'obiettivo definito vengono poi individuate due Research Question (RQ), di seguito proposte:

RQ$_1$. *Come impatta l'iniezione di data smell in termini di metriche di accuracy dei modelli?*

RQ$_2$. *Come impatta l'iniezione di data smell in termini di metriche delle performance dei modelli?*

Da queste Research Question è possibile formulare delle ipotesi nulle. Per ogni data smell in analisi d in D, e per ogni metrica di valutazione dell'accuratezza e delle performance per i modelli di classificazione m in M, vengono valutate le ipotesi nulle strutturate come segue:

> "Non esiste una differenza statisticamente significativa nei valori della metrica $m \in M$ tra quelli ottenuti dai modelli di classificazione quando addestrati su dataset arricchiti con dati contenenti il data smell $d \in D$ e quelli ottenuti quando addestrati su dataset privi di dati che presentano tale data smell d"

Nella sperimentazione saranno analizzate anche le capacità dei LLM, in particolare di ChatGPT[1], nell'iniettare data smell tramite few-shot learning e prompt accurati.

Per condurre queste analisi è stato preso come riferimento principale il lavoro di Recupito et al. [5], andando ad utilizzare le informazioni sui dataset ottenute, la catalogazione dei data smell, le informazioni sulla loro presenza e il tool da impiegare per l'individuazione di essi nei dataset.

I dataset scelti sono prima pre-processati per poter applicarvi una moltitudine di modelli di classificazione, in modo da poter avere una baseline di metriche di valutazione per ogni modello e di tempo di addestramento necessario.

Per l'iniezione dei data smell, questi sono stati scelti considerando i 5 data smell ritenuti più frequenti da Recupito et al. [5] dalle analisi svolte sui dataset di riferimento, ovvero "***Extreme Value Smell***", "***Missing Value Smell***", "***Casing Smell***", "***Suspect Sign Smell***" e "***Floating Point Number As String Smell***" [5]. Le loro definizioni sono presenti nella Sezione 4.4.2, mentre le tecniche con cui si realizza l'iniezione di tali

[1] `https://chatgpt.com/`

data smell sono descritte nella Sezione 4.4. Durante lo sviluppo della soluzione proposta, è stato deciso tuttavia di non procedere con le sperimentazioni riguardanti *Extreme Value Smell*, a causa dei problemi nell'iniettare tale data smell.

A seguito della verifica sulla presenza dei data smell nei nuovi dataset tramite il tool Data Smell Detection[2] (DSD), i medesimi modelli di classificazione verranno riaddestrati sui nuovi dataset e le metriche di valutazione dell'accuracy e delle performance ne saranno comparate con quelle baseline. Queste ultime fasi della sperimentazione verranno realizzate indipendentemente per ogni data smell testato.

4.2 Dataset impiegati

Per poter avere dei confronti più rigorosi, è necessario realizzare delle versioni dei dataset che hanno come unica differenza la presenza, o meno, di un particolare tipo di data smell su una specifica feature, scelta secondo diversi criteri. Per questo motivo, i dataset utilizzati per questa analisi sono i cinque indicati da Recupito et al. [5] come privi di data smell, attraverso l'utilizzo del tool DSD. Questa scelta permette di confrontare dei dataset che presentano caratteristiche simili rispetto alla presenza di data smell, avendo quindi modo di analizzare l'impatto della loro iniezione con meno variabili da considerare. L'individuazione dei dataset è avvenuta confrontando gli ID dei dataset mancanti nella Figura 2 del lavoro di Recupito et al. [5], che riporta i data smell individuati per dataset, con l'ordinamento dei dataset presenti nel file "input/paths.csv" del replication package fornito da Recupito et al. [5]. I cinque dataset presentano caratteristiche molto diverse tra loro, di cui la principale è il numero di record che li compongono. Per questo motivo, nelle successive analisi questi sono stati divisi in due gruppi, che saranno chiamati "**Large Dataset**" e "**Medium Dataset**" in base a se i record che li compongono sono maggiori o minori di 1000 e maggiori o minori di 100. Questa suddivisione deriva da quanto raccolto in letteratura da Bouget et al. [51], che separano dataset di immagini in Small (meno di 100 record), Medium (tra 100 e 1000) e Large (più di 1000 record) [51]. Questa scelta permette di fare confronti più accurati e di trarre delle conclusioni restringendo il campo d'osservazione. I gruppi individuati sono composti dai seguenti dataset:

[2] `https://github.com/DinoDx/rb-data-smell-detection`

- Large Dataset
  - kdd-census (69850 record)
  - diabetic_data (101766 record)
  - nursery (12960 record)
- Medium Dataset
  - Firefighter_Promotion_Exam_Scores (118 record)
  - tae (151 record)

Come riportato anche da Recupito et al. [5], questi dataset provengono da due collezioni realizzate da Hirzel e Feffer [52] e da Le Quy et al. [53], e sono tutti mirati a task di classificazione. Le collezioni [52] [53] mirano a raggruppare degli insiemi di dataset legati a tematiche di fairness connesse al machine learning, e quindi contengono attributi sensibili [5]. In particolare, i dataset "tae" e "nursery" sono menzionati dal lavoro di Hirzel e Feffer [52], che indica come questi siano provenienti da OpenML[3], mentre i restanti sono analizzati da Le Quy et al. [34]. Tutti questi dataset presentano una struttura che li rende idonei ad essere analizzabili dai tool di detection, ovvero una forma tabellare. I tipi delle feature che compongono i dataset non sono tuttavia omogenei, in quanto sono così composti:

- Large Dataset
  - kdd-census - variabili categoriche e numeriche intere - classificazione binaria
  - diabetic_data - variabili categoriche e numeriche intere - classificazione multi-class
  - nursery - solo variabili categoriche - classificazione binaria
- Medium Dataset
  - Firefighter_Promotion_Exam_Scores - variabili categoriche e numeriche decimali, con e senza virgola - classificazione binaria

[3] https://www.openml.org/

  - tae - solo variabili numeriche intere - classificazione multi-class

Inoltre, alcune le variabili categoriche sono di tipo nominali mentre altre di tipo ordinali.

4.2.1 Pre-processing sui dati

Prima di iniziare le sperimentazioni, sono stati valutati nuovamente i data smell in analisi sui dataset selezionati. Da queste analisi è emerso come, in realtà, due dei dataset selezionati presentino dei data smell, non rilevati da DSD quando questi vengono caricati per intero, probabilmente a causa della loro dimensione. Andando a dividere i dataset in sotto-dataset, questi vengono tuttavia rilevati, con le configurazioni di default del tool. Di seguito sono elencati i data smell individuati in questo modo, assicurandosi che il numero di *faulty element* indicato fosse abbastanza alto, sommando le occorrenze dei sotto-dataset, da sollevare il data smell anche nel dataset integrale dato il tipo di data smell, ovvero se le occorrenze superassero il 10% del dataset integrale (per Missing Value e Casing Smells):

- kdd-census
  - Missing Value
    - "migration_msa", "migration_reg", "migration_sunbelt", "migration_within_reg": 23227 occorrenze, $\approx$ 33,25% del dataset integrale
- diabetic_data
  - Missing Value
    - "A1Cresult": 84748 occorrenze, $\approx$ 83,28% del dataset integrale
    - "max_glu_serum": 96420 occorrenze, $\approx$ 94,75% del dataset integrale
  - Casing
    - "medical_specialty": 13704 occorrenze, $\approx$ 13,47% del dataset integrale (utilizzando la configurazione "Medium" su uno dei sotto-dataset, le occorrenze sono invece 15798, $\approx$ 15,52% del dataset integrale)

Queste feature così individuate sono quindi rimosse dai dataset prima di procedere con i successivi passi di pre-processing, in modo da avere tutti dataset senza data smell in esame rilevati con la configurazione "Tolerant". Altri data smell rilevati nei dataset sono i "Duplicated Value", che DSD[2] presenta quando ci sono più istanze con lo stesso valore in una feature, che però non sono oggetto delle analisi, e quindi non saranno trattati ulteriormente. Applicando invece i criteri di individuazione "Strict", sono individuati anche degli smell di tipo "Extreme Value" e "Integer As String Smell" (quest'ultimo però non rientra tra quelli oggetto di analisi). Tuttavia, essendo poche occorrenze anche rispetto ai sotto-dataset, non verrebbero individuati nei dataset integrali, e per questo motivo non sono considerati nelle fasi di pre-processing. Per poter valutare diversi modelli di classificazione sui dati, questi necessitano di essere in una forma standardizzata. Per questo motivo, è stato necessario realizzare una fase di pre-processing sui dataset, volti a prepararli al tool **LazyPredict**[4], descritto nella Sezione 4.3. In particolare, sono state rimosse le feature indicanti degli id univoci per i record dei dataset, in modo da non influenzare eccessivamente i classificatori in esame. Inoltre, tutte le variabili di tipo categorico vengono trasformate prima di essere elaborate, andando a fornire una rappresentazione numerica di queste tramite la funzione `pandas.factorize()`. In questo modo però, vengono perse eventuali informazioni sulle relazioni d'ordine tra i diversi valori delle feature che vengono codificate, imponendo quindi un nuovo ordine anche alle feature che erano di tipo nominale. Ciò viene quindi assunta come possibile limitazione alle interpretazioni dei risultati e quindi come minaccia alla validità dello studio, ma permette di valutare un numero maggiore di classificatori. Inoltre, occorre notare che è LazyPredict ad occuparsi di selezionare le feature su cui poi vengono effettivamente addestrati i modelli di classificazione. Il dataset "tae" ha invece richiesto una modifica dei nomi delle feature che lo compongono, andando a sostituire gli spazi presenti nelle parole che li compongono con degli "_", per permettere a DSD di valutare queste feature.

Per la preparazione dei dataset all'addestramento dei diversi modelli di classificazione di LazyPredict, questi sono stati suddivisi in train e test set, che rispettivamente rappresentano l'80% e il 20% del dataset originale (percentuali che Gholamy et al. [54] consigliano). La fase di iniezione dei data smell viene invece realizzata sui dataset

[4] `https://lazypredict.readthedocs.io/en/latest/`

originali, prima che questi siano modificati per poter essere elaborati da LazyPredict, fatta eccezione per la modifica ai nomi delle feature del dataset "tae" e per la rimozione delle feature con data smell e rappresentanti id univoci.

4.3 Strumenti utilizzati

I principali strumenti utilizzati per la soluzione proposta in definitiva da questo lavoro sono stati: il tool Data Smell Detection (DSD)[2], il tool Weka[5], SDV[6] e LazyPredict[4] rispettivamente per l'individuazione dei data smell, per la scelta delle feature in cui iniettare i data smell, per la generazione di dati sintetici e per l'addestramento dei modelli di classificazione. DSD è stato utilizzato mantenendo le configurazioni di default impostate dal tool. Della libreria LazyPredict è stata utilizzata in particolare la funzione LazyClassifier, che permette di valutare 26 diversi classificatori, in modo da confrontarne i valori ottenuti di Accuracy, Balanced Accuracy, ROC AUC, F1 Score e tempo impiegato. I modelli confrontati sono: *XGBClassifier, LGBMClassifier, NearestCentroid, RandomForestClassifier, ExtraTreesClassifier, AdaBoostClassifier, BaggingClassifier, SVC, DecisionTreeClassifier, BernoulliNB, LinearDiscriminantAnalysis, LogisticRegression, CalibratedClassifierCV, LinearSVC, RidgeClassifier, RidgeClassifierCV, LabelSpreading, LabelPropagation, KNeighborsClassifier, ExtraTreeClassifier, PassiveAggressiveClassifier, SGDClassifier, Perceptron, QuadraticDiscriminantAnalysis, GaussianNB* e *DummyClassifier*. In aggiunta a questi, viene valutato il modello *NuSVC* ma solo per alcuni dataset che rispettano determinate caratteristiche, che in questo caso sono i dataset "nursery" e "Firefighter_Promotion_Exam_Scores".

[5] `https://ml.cms.waikato.ac.nz/weka`

[6] `https://sdv.dev/`

4.4 Soluzione proposta

Di seguito sono descritti i processi per la realizzazione delle diverse fasi che hanno composto la sperimentazione.

4.4.1 Elaborazione risultati baseline

I dataset originali, pre-processati con le fasi descritte nella Sezione 4.2.1, sono forniti alla funzione LazyClassifier di LazyPredict per valutare le metriche ottenute dai modelli di classificazione, in modo da poter ottenere i risultati baseline da confrontare con quelli che saranno ottenuti sui dataset arricchiti con record che presentano i data smell.

4.4.2 Scelta delle feature su cui iniettare i data smell

Non tutti i data smell considerati, ed elencati nella Sezione 4.1, possono essere iniettati su ogni feature, in quanto alcuni di essi sono relativi a specifici tipi di feature, come quelle testuali. Per questo motivo, è risultato fondamentale individuare la feature su cui iniettare ogni data smell. Per fare ciò, le feature di ogni dataset pre-processato, quindi le versioni fornite ai modelli di classificazione utilizzati per il confronto dei risultati, sono state valutate tramite il tool Weka attraverso la funzione di *feature selection*. In particolare è stata scelta una tecnica che permettesse di classificare le feature e che fosse applicabile a tutti i dataset in esame. L'unica tecnica individuata che rispetta queste caratteristiche è ***ReliefFAttributeEval***, che valuta quanto una feature sia utile a distinguere un record del dataset dagli altri record simili[7] [55]. I risultati di questa analisi, per ogni dataset, sono riportati in "DatasetAnalysisAndPreprocessing.ipynb", all'interno della repository Github indicata nella Sezione 1.3 . Per ogni data smell da iniettare, è stata quindi selezionata la feature valida che si è classificata prima dai risultati di questa analisi. Non tutti i data smell considerati sono applicabili ad ogni dataset preso in considerazione, a causa dei

[7] `https://weka.sourceforge.io/doc.dev/weka/attributeSelection/ReliefFAttributeEval.html`

tipi di feature che li compongono. Di seguito sono riportati i dataset in cui sono applicabili i data smell considerati, con le loro definizioni:

- **Missing Value**: è presente quando ci sono dati mancanti in una colonna [56].
  - kdd-census
  - diabetic_data
  - Firefighter_Promotion_Exam_Scores
  - tae
- **Casing**: è presente quando non sono usati in modo consistente il maiuscolo ed il minuscolo per i dati testuali (definizione riportata da un catalogo online fornito nella Sezione 4.2 del lavoro di Foidl et al. [3]).
  - kdd-census
  - diabetic_data
  - Firefighter_Promotion_Exam_Scores
  - nursery
- **Suspect Sign**: viene individuato dal tool DSD^2 quando dei dati in una feature hanno un segno diverso dalla maggioranza.
  - kdd-census
  - diabetic_data
  - Firefighter_Promotion_Exam_Scores
  - tae
- **Extreme Value**: è presente quando dei dati numerici differiscono notevolmente dalla distribuzione di quella feature [5].
  - kdd-census
  - diabetic_data
  - Firefighter_Promotion_Exam_Scores
  - tae

- **Floating Point Number As String**: è presente quando un numero *floating-point* è rappresentato sotto forma di stringa (definizione riportata da un catalogo online fornito nella Sezione 4.2 del lavoro di Foidl et al. [3]).
  - nursery

Il data smell Missing Value è rilevabile dal tool DSD solo quando inserito in feature numeriche, per cui non è iniettabile in dataset con sole feature categoriche. Viene segnalato dal tool DSD quando sono individuati dei valori del tipo "NaN".

4.4.3 Iniezione dei data smell

Sperimentazione esplorata

La prima sperimentazione esplorata per raggiungere l'obiettivo indicato nella Sezione 4.1 riguarda l'utilizzo di LLM per la generazione di dati sintetici.

Creazione del prompt L'iniezione dei data smell per questa sperimentazione esplorata è stata realizzata utilizzando ChatGPT[8] e nello specifico il modello **gpt-4o**, con una tecnica di prompting basata sul *few-shot learning*, similmente a come fatto da diversi lavori analizzati nella Sezione 3.4.1. Per ogni fase di iniezione di un nuovo data smell in un dataset, sono stati forniti 25 campioni del dataset originale, assieme allo schema del dataset, tramite il seguente prompt:

> Given the following features and examples of records in a dataset, write 10 rows with different records that are realistic to the examples given, containing a data smell of type '*data_smell_type*' only in the feature '*feature_for_data_smell*'. The values of feature '*target_feature*' must contain exactly only the values in the examples.
>
> The data smell 'Casing' must be inserted in every new record using values with an unusual use of upper and lower case (mixed case, upper only, lower only).The records must be produced in csv format, as supplied. Be sure that the csv has the first row containing the features, and after that 10 rows containing the new records. Don't add anything else to the answer.

[8] `https://chatgpt.com/`

> Features and records examples:
>
> ...
>
> ...
>
> ...

In aggiunta a questo prompt, per i *large dataset* sono stati richiesti altri 10 campioni tramite il seguente prompt:

> Generate exactly other 10 records with the same instructions. Start from where you stopped, without inserting the features row

Per la creazione del prompt, è stato anche tentato un approccio ispirato dal ***meta-prompting*** [57] , per tentare sia di migliorare i risultati ottenuti con questo prompt, sia di generare da zero un prompt adeguato, ma non ottenendo miglioramenti. I prompt utilizzati per questi esperimenti sono quelli riportati nelle Figure 4.1 e 4.2. I campioni scelti sono stati selezionati tramite la funzione "*train_test_split*"[9] di scikit-learn, impostando come dimensione 25 campioni ed il parametro "*stratified*" a *true*, in modo da avere dei campioni che rappresentano la proporzione delle classi della variabile target nel dataset integrale, all'interno del file "DatasetAnalysisAndPreprocessing.ipynb" presente all'interno della repository del progetto indicata nella Sezione 1.3 .

Limiti dell'approccio con gpt-4o Il numero di campioni da generare è stato scelto a seguito di un'analisi con il tool DSD, andando a identificare la quantità di data smell necessaria a far individuare il problema dal tool. Per quattro dei cinque data smell analizzati, ovvero Extreme Smell, Casing Smell, Suspect Sign Smell e Missing Value Smell, la configurazione di default di DSD (ovvero quella "Tolerant"), segnala un data smell solo se questo affligge almeno il 10% del dataset fornito in input. Per il Floating Point Number As String Smell, DSD sembra rilevare lo smell se questo è presente nel 10% o meno dei dati, segnalando poi come "Faulty element" i restanti dati non espressi sotto forma di stringa. Volendo quindi utilizzare i dataset per intero, è stato necessario calcolare il numero di dati da iniettare per far si che questi siano

[9] `https://scikit-learn.org/1.5/modules/generated/sklearn.model_selection.train_test_split.html`

il 10% del totale. Ciò è stato realizzato tramite la seguente proporzione, di cui se ne descrivono gli elementi:

$$\begin{gathered} ceiling = funzione\ per\ l'arrotondamento\ di\ un\ valore\ al\ minimo\ numero\ intero \\ maggiore\ o\ uguale\ rispetto\ ad\ esso \\ \#record\_df = il\ numero\ di\ record\ presenti\ nel\ dataset\ in\ esame \\ numero\ dati\ da\ generare = ceiling((\#record\_df * 100)/90) - \#record\_df \end{gathered} \tag{4.4.1}$$

Per il dataset "nursery", è stato necessario aggiungere un'altra riga a quelle calcolate da generare tramite questa formula, in modo da raggiungere la percentuale di dati iniettati richiesta, a seguito dell'unione con il dataset originale, per rilevare i data smell. Utilizzando questa formula per il dataset più grande, ovvero il diabetic_data, composto da 101766 record, si evince debbano essere generati **11308** nuovi record. Questa necessità si è scontrata con i tempi di generazione delle risposte di gpt-4o, che richiede diversi secondi per generare 10 campioni, rallentando molto il processo. Inoltre, per alcune combinazioni di data smell da iniettare e dataset (ad esempio Casing Smell nel dataset diabetic_data), i risultati generati sono stati molto imprecisi, generando dei dati che non rispecchiavano l'header del dataset, oppure iniettando il data smell non in tutti i record, oppure ancora in delle feature errate. Richiedendo invece di generare in un solo prompt, tutti i record richiesti, il modello gpt-4o produce uno script python per generare casualmente dei nuovi campioni utilizzando i valori forniti nel prompt, ed iniettando in questa fase il data smell, utilizzando dei valori "smelly" rispetto ai dati forniti. Un esempio di tale generazione è presente nella Figura 4.3. Seppur i dati così ottenuti siano validi, non rispecchia l'obiettivo della sperimentazione ed il tipo di data-augmentation che si vuole utilizzare, introducendo inoltre altri tipi di minacce alla validità dei risultati, per cui tali script generati da ChatGPT non sono stati utilizzati. Le medesime sperimentazioni sono state tentate anche con il modello gpt-4o mini, ottenendo generalmente risultati peggiori. Per questi motivi, è stato scelto di scartare la sperimentazione con GPT e concentrare le ricerche sulla soluzione facente utilizzo di SDV.

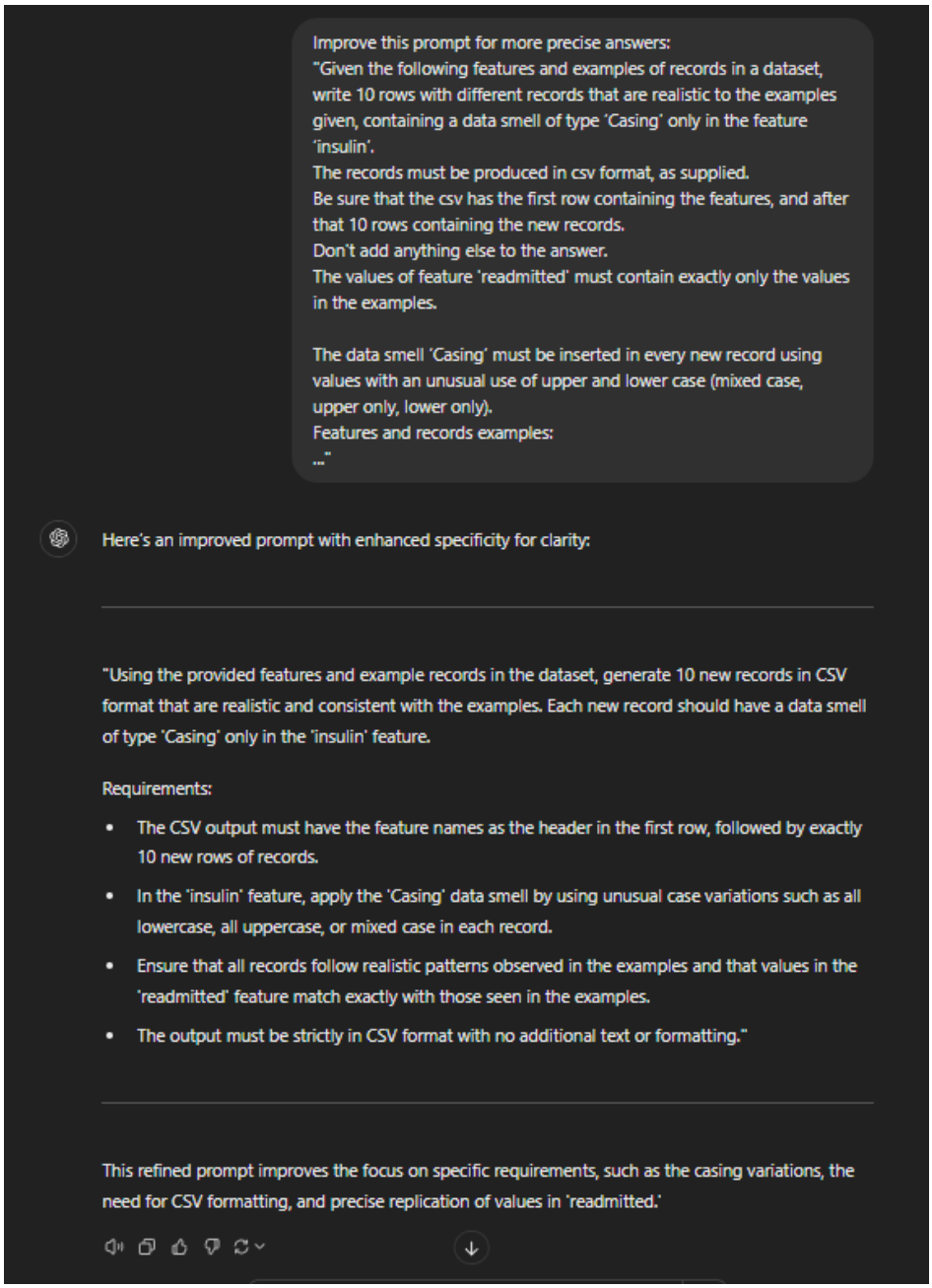


Figura 4.1: Esempio di interazione con gpt-4o per il miglioramento di un promtp per l'iniezione di data smell.

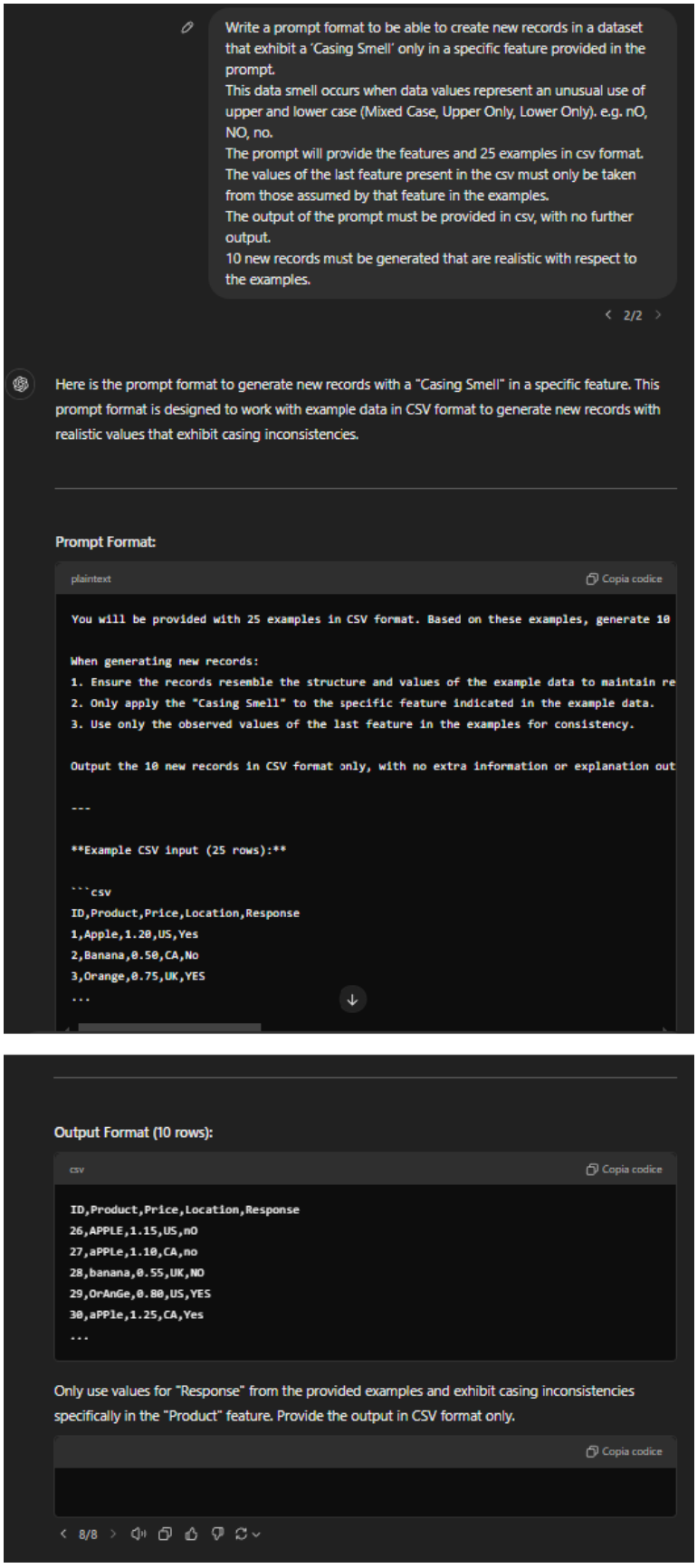
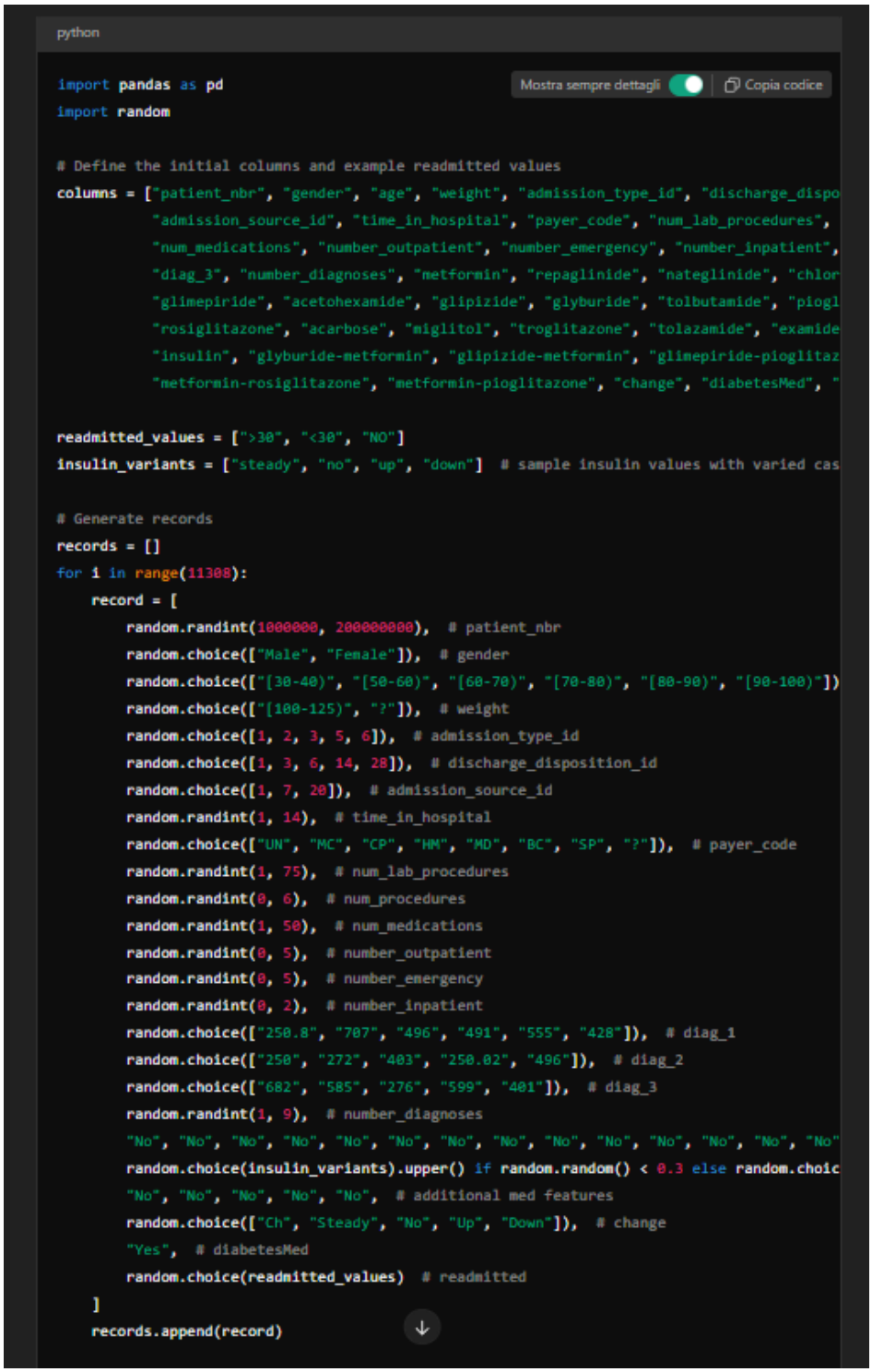


Figura 4.2: Esempio di interazione con gpt-4o per la generazione di un promtp per l'iniezione di data smell.



```python
import pandas as pd
import random

# Define the initial columns and example readmitted values
columns = ["patient_nbr", "gender", "age", "weight", "admission_type_id", "discharge_dispo
           "admission_source_id", "time_in_hospital", "payer_code", "num_lab_procedures",
           "num_medications", "number_outpatient", "number_emergency", "number_inpatient",
           "diag_3", "number_diagnoses", "metformin", "repaglinide", "nateglinide", "chlor
           "glimepiride", "acetohexamide", "glipizide", "glyburide", "tolbutamide", "piogl
           "rosiglitazone", "acarbose", "miglitol", "troglitazone", "tolazamide", "examide
           "insulin", "glyburide-metformin", "glipizide-metformin", "glimepiride-pioglitaz
           "metformin-rosiglitazone", "metformin-pioglitazone", "change", "diabetesMed", "

readmitted_values = [">30", "<30", "NO"]
insulin_variants = ["steady", "no", "up", "down"]  # sample insulin values with varied cas

# Generate records
records = []
for i in range(11308):
    record = [
        random.randint(1000000, 200000000),  # patient_nbr
        random.choice(["Male", "Female"]),  # gender
        random.choice(["[30-40)", "[50-60)", "[60-70)", "[70-80)", "[80-90)", "[90-100)"])
        random.choice(["[100-125)", "?"]),  # weight
        random.choice([1, 2, 3, 5, 6]),  # admission_type_id
        random.choice([1, 3, 6, 14, 28]),  # discharge_disposition_id
        random.choice([1, 7, 20]),  # admission_source_id
        random.randint(1, 14),  # time_in_hospital
        random.choice(["UN", "MC", "CP", "HM", "MD", "BC", "SP", "?"]),  # payer_code
        random.randint(1, 75),  # num_lab_procedures
        random.randint(0, 6),  # num_procedures
        random.randint(1, 50),  # num_medications
        random.randint(0, 5),  # number_outpatient
        random.randint(0, 5),  # number_emergency
        random.randint(0, 2),  # number_inpatient
        random.choice(["250.8", "707", "496", "491", "555", "428"]),  # diag_1
        random.choice(["250", "272", "403", "250.02", "496"]),  # diag_2
        random.choice(["682", "585", "276", "599", "401"]),  # diag_3
        random.randint(1, 9),  # number_diagnoses
        "No", "No", "No", "No", "No", "No", "No", "No", "No", "No", "No", "No", "No", "No"
        random.choice(insulin_variants).upper() if random.random() < 0.3 else random.choic
        "No", "No", "No", "No", "No",  # additional med features
        random.choice(["Ch", "Steady", "No", "Up", "Down"]),  # change
        "Yes",  # diabetesMed
        random.choice(readmitted_values)  # readmitted
    ]
    records.append(record)
```

Figura 4.3: Esempio di generazione da parte di gpt-4o di script per creare nuovi record con data smell.

Utilizzo di SDV

L'approccio utilizzato in definitiva per la creazione di dati sintetici è stato basato sul tool SDV[6] [39], descritto nella Sezione 3.3.2. In particolare, è stato scelto di utilizzare il sintetizzatore CTGAN[7], dati i vantaggi che apporta questo tipo di modello sui dati tabulari, come riportato nella Sezione 3.3.2. Questo modello è stato utilizzato per tutti i dataset, anche se per uno di essi viene fornito un *warning* relativo all'utilizzo di questo tipo di synthesizer su questo specifico dataset a causa della sua struttura che porta alla creazione di molte colonne, in modo da uniformare i risultati e le analisi.

Sono stati addestrati dei modelli per ogni dataset in esame, partendo dalle loro versioni già pre-processate, avendo quindi rimosso le feature che presentavano già data smell in esame o che rappresentavano id univoci, ma senza la fattorizzazione delle feature categoriche in valori numerici. Per l'addestramento del sintetizzatore per il dataset diabetic_data è stato utilizzato Cuda tramite Google Colab[10], per cui anche per la generazione di nuovi dati è stato necessario utilizzare questo approccio. Il tool SDV permette di condizionare la generazione dei dati solo fornendo degli specifici vincoli ai valori da assumere per alcune feature, definendo però dei valori esatti che debbano essere generati, tuttavia non permettendo valori del tipo NaN. Per tale motivo, l'iniezione dei data smell è stata realizzata manualmente tramite script python, presenti nel file "datasmells_injection.py", disponibile nella repository indicata al termine della Sezione 1.3. Per l'iniezione del data smell Extreme Value, è stato notato che il tool DSD valuta come tali i dati che hanno un valore di z-score maggiore di 3. Nella configurazione di default sono richiesti almeno il 10% di dati con questa caratteristica per identificare lo smell. Andando a calcolare i valori da iniettare necessari, non è risultato possibile trovare una combinazione di valori che permettesse di raggiungere tale obiettivo, in quanto l'iniezione di un tale numero di dati con valori estremi causa una importante deviazione della distribuzione, che quindi porta i valori a non raggiungere lo z-score necessario. Per tale motivo, è stato scelto di testare l'iniezione di questo data smell andando a impostare la soglia "Strict" di DSD, invece che quella standard, ovvero la soglia "Tolerant", facendo quindi individuare il data smell anche con un singolo valore estremo. Per questo

[10] `https://colab.research.google.com/`

esperimento la dimensione dei dati da generare, su cui poi iniettare questo data smell, è stata individuata tramite la formula 4.4.1, impostando però come numero di dati da generare:

$$numero\ dati\ da\ generare = ceiling((\#record\_df * 100)/99) - \#record\_df) \tag{4.4.2}$$

Tale formula è stata anche utilizzata per generare i dati di partenza per l'iniezione del data smell Floating Point Number As String, per via delle soglie di identificazione presenti in DSD. Con questa formula è possibile creare dei set di dati generati che sollevino il data smell Extreme Value in DSD, tuttavia l'approccio richiesto per questo data smell consiste nel sostituire completamente il dato iniettato dal sintetizzatore CTGAN, invece di modificarlo partendo da esso come avviene per gli altri data smell, andando a manipolare eccessivamente la coerenza dei record generati. Inoltre, anche imponendo come vincolo durante la generazione un valore fissato che avesse lo z-score richiesto, il processo non forniva alcun output, fornendo il seguente messaggio: *"ValueError: Unable to sample any rows for the given conditions."*, probabilmente a causa dei limiti delle tecniche di data augmentaiton Generation-based riportati anche da [37], ovvero che non sono in grado di generare dati completamente diversi rispetto a quelli appresi [37]. Per questi motivi, non è stata continuata la sperimentazione con questo data smell, divenendo quindi uno dei possibili sviluppi futuri.

Per la validazione sulla corretta iniezione dei data smell in questo modo, i dati generati sono stati uniti al dataset intero (avendo rimosso precedentemente le feature con data smell e rappresentanti id univoci), e questi sono stati valutati tramite DSD per verificare effettivamente la presenza del data smell desiderato. Per i dataset kdd-census e diabetic_data, i dati generati sono stati uniti a tutti i sotto-dataset in cui è necessario suddividerli per procedere con le analisi, sollevando il data smell richiesto in ogni condizione. In questi casi era però richiesto che la presenza dei data smell fosse relativa ai dataset interi, quindi è stato necessario verificare manualmente che la quantità di *faulty element* fosse abbastanza da poter identificare tale data smell anche nei dataset non suddivisi. Questo ha portato a dover cambiare la feature

scelta per l'iniezione del data smell nello scenario dei Suspect Sign, in quanto per mantenere la medesima dimensione dei dati generati, e quindi per non aggiungere una ulteriore variabile ai confronti tra le performance dei modelli, è importante agire su feature che non presentano molti valori "0", che quindi non possono essere modificati e vanno a diminuire la quantità totale di *faulty element*. In particolare, per il dataset diabetic_data la feature "discharge_disposition_id" (seconda feature per feature importance) invece di "number_inpatient". Per il dataset kdd-census invece è stata utilizzata la feature "age", la prima per feature importance, che però presenta una piccola quantità di valori "0" nei dati sintetici generati, non permettendo di raggiungere il numero di *faulty element* richiesti per essere il 10% del totale, ovvero 7762, bensì 7539, ovvero il 9,71% circa sul dataset combinato. Essendo una differenza leggera è stata mantenuta questa configurazione, in modo da mantenere costante tra i dataset la percentuale di dati generati sul totale, e per non utilizzare una feature con bassa feature importance ma che non presentasse valori "0", che in questa caso si sarebbe trattato della feature "year", 27ª su 36 feature per feature importance. Per la validazione del Suspect Sign smell nel dataset diabetic_data, è emerso come nel terzo sotto-dataset che lo compone, il data smell non viene individuato dal tool DSD, a causa della diversa distribuzione dei dati che porta ad un 25° percentile diverso dagli altri sotto-dataset. Tuttavia, analizzando questo valore per il dataset integrale unito ai dati sintetici con tali modifiche, risulta come il data smell dovrebbe essere correttamente individuato.

4.4.4 Elaborazione dei risultati

Per l'elaborazione dei risultati, i dataset sono divisi in train e test split come fatto per la versione baseline, e in seguito i dataset con i data smell iniettati sono aggiunti agli split di training. Gli split di training e di test così ottenuti sono poi pre-processati andando a convertire i dati categorici trasformandoli in dati numerici, come descritto nella Sezione 4.2.1. In questo modo è possibile valutare le metriche di accuracy e di performance dei modelli di classificazione, elencati nella Sezione 4.3, quando addestrati su dati sintetici in cui sono stati iniettati, o meno, diversi tipi di data smell, tramite il tool LazyPredict[4]. Tutti gli addestramenti, e quindi i risultati,

sono stati elaborati tramite Google Colab[10], utilizzando il runtime con TPU v2.8, e adottando come variabile target della classificazione l'ultima feature presente in ogni dataset. Questa scelta è stata necessaria in quanto per l'addestramento sul dataset di dimensioni maggiori sono richieste fino a 69.8 GB di RAM. I risultati e i diversi tipi di analisi condotte sono riportati ed analizzati nella Sezione 5.

CAPITOLO 5

Validazione

5.1 Approccio adottato

In seguito alla raccolta dei dati relativi alle metriche di valutazione ottenute dai modelli nelle diverse configurazioni dei dataset di training, questi sono stati elaborati in modo da poter trarre delle considerazioni con significatività statistica e poter rispondere alle Research Question definite nella Sezione 4.1.

5.1.1 Metriche utilizzate

Per poter rispondere alla RQ_1, sono state valutate le metriche di Accuracy, Balanced Accuracy e F1 Score calcolate sugli split di test, ampiamente utilizzate in letteratura, di seguito descritte nei casi di classificazioni binarie :

- F1 Score (o F Measure [58]): è la media armonica della Precision e della Recall, ovvero: $\frac{2*Precision*Recall}{Precision+Recall}$ [59], con Precision = $\frac{TP}{TP+FP}$ [60] e Recall = $\frac{TP}{TP+TN}$ [60]
- Accuracy: è il rapporto tra la somma degli elementi correttamente classificati sul totale degli elementi: $\frac{TP+TN}{TP+FP+FN+TN}$ [60]

- Balanced Accuracy: è data dalla seguente formula: $\frac{\frac{TP}{P}+\frac{TN}{N}}{2}$ [61], dove per P si intende la somma tra TP e FN, e per N la somma tra TN e FP [61], e viene utilizzata in contesti di dati sbilanciati, ovvero dove l'Accuracy può dare risultati fortemente influenzati dallo sbilanciamento delle classi nel dataset [61]

Per rispondere alla RQ_2 è stata invece scelta come metrica di performance il tempo impiegato sommando quello di addestramento, di testing e di calcolo delle metriche di valutazione per ogni modello, in termini di secondi impiegati, che corrisponde alla metrica "Time Taken" calcolata da LazyPredict[1], che fornisce anche le metriche di accuracy menzionate precedentemente utilizzando *sklearn.metrics*[2]. Per la metrica F1 Score, in contesti di classificazione non binaria, questa viene calcolata da LazyPredict[1] con il setting "$avearage = weighted$", descritto dalla documentazione di *sklearn.metrics*[3].

5.1.2 Confronti effettuati

Per poter rispondere alle RQ, valutando se è possibile o meno rigettare le ipotesi nulle, sono state svolte diverse analisi volte a comprendere l'impatto dei data smell a diversi livello di dettaglio. In particolare, i confronti sono stati fatti:

- tra le distribuzioni **delle medie** delle metriche di valutazione derivanti dai modelli di classificazione calcolate sui risultati ottenuti negli stessi scenari di iniezione **sui dataset coinvolti** (Sezione 5.2.1). Per questi confronti si utilizza quindi una sorta di "Macro Averaging" [62], andando a calcolare per ogni diverso modello di classificazione la media per ogni sua metrica ottenuta nei diversi dataset del medesimo scenario, e unendo i risultati così ottenuti per ogni modello per creare le distribuzioni relative ad ogni scenario.

- tra le distribuzioni delle metriche di valutazione ottenute dai modelli di classificazione derivanti dai risultati ottenuti negli stessi scenari di iniezione **sui dataset coinvolti** (Sezione 5.2.2).

[1] https://lazypredict.readthedocs.io/en/latest/

[2] https://scikit-learn.org/1.5/api/sklearn.metrics.html

[3] https://scikit-learn.org/1.5/modules/model_evaluation.html#from-binary-to-multiclass-and-multilabel

- tra le distribuzioni delle metriche di valutazione ottenute dai modelli di classificazione nei diversi scenari di iniezione **su ogni singolo dataset**, analizzandole separatamente (Sezione 5.2.3).
- tra le medie delle metriche di valutazione ottenute dai modelli nei diversi scenari di iniezione (Sezione 5.2.4).

Per i primi due tipi di confronti, sono stati valutati gli scenari in cui le distribuzioni baseline sono calcolate utilizzando solo i dataset sui quali sono stati effettivamente iniettati i diversi tipi di data smell. Questo ha quindi portato a definire due categorie di dataset:

- dataset con feature numeriche: kdd-census, diabetic_data, Firefighter_Promotion_Exam_Scores, tae - per i data smell Missing Value e Suspect Sign
- dataset con feature categoriche: kdd-census, diabetic_data, nursery, Firefighter_Promotion_Exam_Scores - per il data smell Casing

Il data smell Floating Point Number As String, poiché iniettato solo nel dataset Firefighter_Promotion_Exam_Scores, è stato analizzato nel terzo tipo di confronti, relativi ai singoli dataset.

5.1.3 Grafici e test adottati

Per le analisi dei primi tre tipi di confronti, si è fatto utilizzo di boxplot ad intaglio e del test Kolmogorov-Smirnov [63], imponendo un *p-value* minore di 0.05 per determinare se le distribuzioni differissero con significatività statistica. Il valore da imporre per il p-value è stato scelto poiché utilizzato in letteratura [64]. Questo test è stato selezionato poiché permette di confrontare se due campioni provengono dalla stessa funzione di densità di probabilità senza fare assunzioni sulle distribuzioni dei campioni [63]. La scelta di questo test deriva principalmente dalla sua caratteristica di essere non parametrico, non richiedendo quindi di fare assunzioni sulle forme delle distribuzioni in esame. Dalla documentazione dell'implementazione di questo

test nella libreria SciPy[4] si evince come venga calcolata una *statistic_location*, uguale al valore dei due campioni confrontati che corrisponde alla *test statistic* ottenuta[4]. La *statistic_location* viene utilizzata per valutare lo *statistic_sign*, che sarà +1 se la funzione di distribuzione empirica del primo campione eccede quella del secondo campione in corrispondenza della *statistic_location*, -1 nel caso opposto[4]. Per questo motivo, volendo confrontare tra loro le metriche di valutazione dei modelli di classificazione, saranno considerate come meno performanti le distribuzioni indicate dallo *statistic_sign*, quindi sarà ritenuta peggiore la distribuzione del primo campione in caso di *statistic_sign* uguale a 1, e quella del secondo campione quando uguale a -1, nel caso di F1 Score, Accuracy e Balanced Accuracy. Nel caso del confronto tra i Tempi impiegati, invece sarà considerata come più performante la distribuzione del secondo campione in caso di *statistic_sign* uguale a 1, e quella del primo campione quando uguale a -1. In ogni caso, sarà riportata in grassetto, nelle Tabelle[5], la distribuzione più performante. Come specificato nella Sezione 4.4.4, i dati generati sono aggiunti solo agli split di training, lasciando invariati gli split di testing. Nella Sezione 5.2 sono riportati i risultati ottenuti nei 4 tipi di confronti effettuati.

5.2 Risultati ottenuti

I risultati mostrati in questa Sezione sono solo quelli per i quali è stato ottenuto un *p-value* minore di 0.05, ovvero i confronti nei quali è stata individuata una differenza tra le distribuzioni indicate con significatività statistica.
Per questo motivo, tutti gli altri confronti non riportati sono da intendersi con *p-value* ≥ 0.05. I valori riportati nelle Tabelle di questa Sezione non includono tutte la parti decimali ottenute e non sono arrotondati rispetto a quelli ottenuti dalle funzioni di SciPy[6] e Pandas[7]. Per i valori riportati invece nelle heatmap della Sezione 5.2.4, questi sono arrotondati alla seconda cifra della parte decimale per le metriche di accuracy,

[4] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ks_2samp.html`

[5] Come strumento di supporto alla creazione delle tabelle si è utilizzato il seguente tool online: `https://www.tablesgenerator.com/`

[6] `https://scipy.org/`

[7] `https://pandas.pydata.org/docs/index.html`

mentre alla prima cifra della parte decimale per i Tempi impiegati. I grafici restanti sono disponibili negli script della cartella "create_plot_scripts".

5.2.1 Distribuzioni delle medie delle metriche di valutazione ottenute sui dataset

I risultati significativi dei confronti tra le distribuzioni delle medie delle metriche di valutazione ottenute dai modelli di classificazione, addestrati sui diversi tipi di dataset, sono riportati nelle Figure 5.1 e 5.2, e nelle Tabelle 5.1 e 5.2.

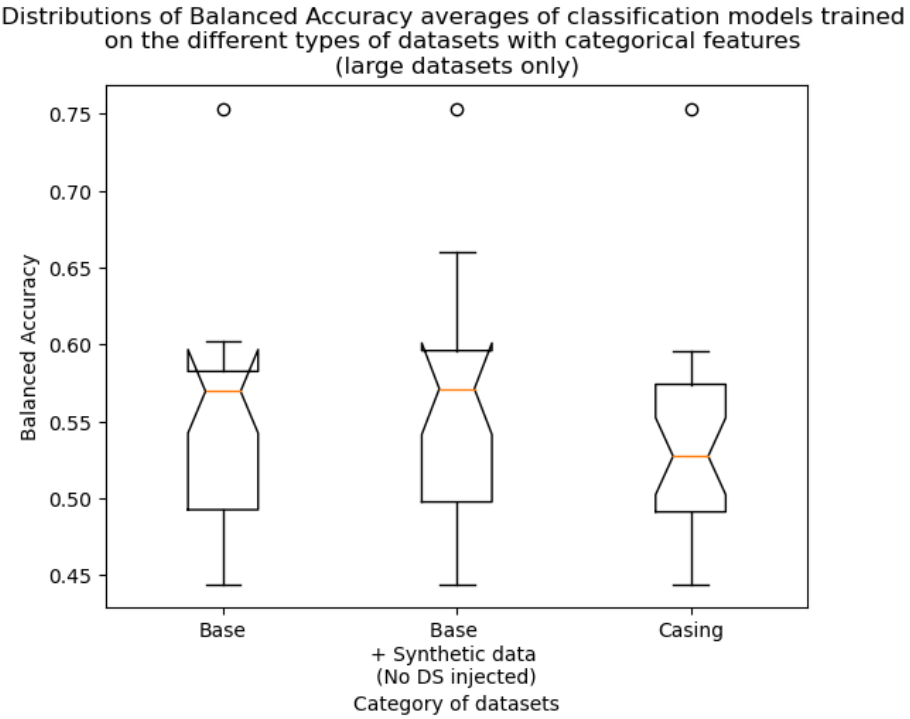


Figura 5.1: Confronto tra i boxplot delle distribuzioni delle medie della Balanced Accuracy - solo dataset large - dataset con feature categoriche

Categoria Dataset 1	Categoria Dataset 2	Metrica	Media distribuzione Dataset 1	Media distribuzione Dataset 2	Test statistic	Statistic location	Statistic sign	p-value
Base + Synthetic	Casing	Balanced Accuracy	**0.559**	0.534	0.370	0.568	-1	**0.048**

Tabella 5.1: Risultati significativi dei confronti tra le distribuzioni delle medie delle metriche di valutazione ottenute - solo dataset large - dataset con feature categoriche

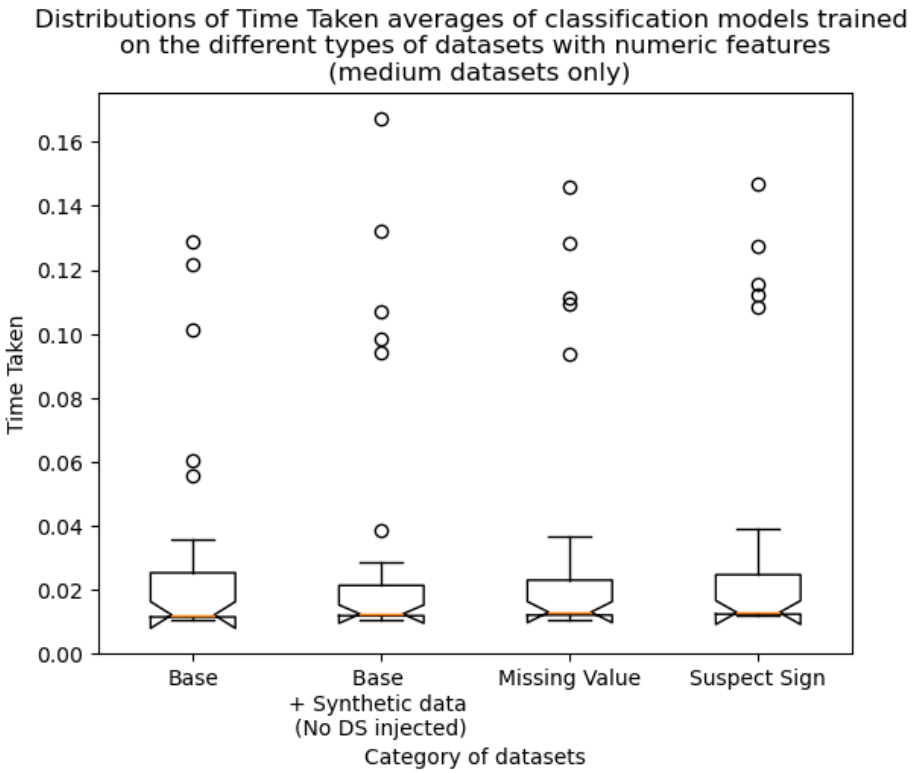


Figura 5.2: Confronto tra i boxplot delle distribuzioni delle medie del Tempo impiegato - solo dataset medium - dataset con feature numeriche

Categoria Dataset 1	Categoria Dataset 2	Metrica	Media distribuzione Dataset 1	Media distribuzione Dataset 2	Test statistic	Statistic location	Statistic sign	p-value
Base	Suspect Sign	Time Taken	**0.028**	0.034	0.370	0.012	1	**0.048**

Tabella 5.2: Risultati significativi dei confronti tra le distribuzioni delle medie delle metriche di valutazione ottenute - solo dataset medium - dataset con feature numeriche

Osservazioni

Da questi risultati ottenuti emerge in particolare come:

1. per i dataset aventi feature categoriche, le medie di Balanced Accuracy ottenute su dataset di training aventi il data smell **Casing**, limitatamente ai dataset large, siano in genere peggiori rispetto a quelle che si ottengono sui dataset della medesima dimensione, aventi dati sintetici senza la presenza del data smell (dataset "Base + Synthetic data").

2. per i dataset aventi feature numeriche, l'unica differenza significativa appare essere quella sulle medie dei tempi impiegati, analizzando solo i dataset medium, denotando un maggiore tempo per il dataset di training con dati iniettati aventi un data smell di tipo **Suspect Sign** rispetto alla baseline, non avente dati aggiuntivi.

5.2.2 Distribuzioni delle metriche di valutazione ottenute sui dataset

I risultati significativi dei confronti tra le distribuzioni delle metriche di valutazione ottenute dai modelli di classificazione, addestrati sui diversi tipi di dataset, sono riportati nella Figura 5.4 e nelle Tabelle 5.3 e 5.4, comprendenti anche le distribuzioni dei tempi impiegati utili ad interpretare i risultati dei boxplot (che si concentrano solo sull'intervallo dove risiedono la maggior parte dei valori assunti dai campioni) e delle altre Tabelle, poiché queste distribuzioni presentano diversi outlier.

Categoria Dataset 1	Categoria Dataset 2	Metrica	Media distribuzione Dataset 1	Media distribuzione Dataset 2	Test statistic	Statistic location	Statistic sign	p-value
Base	Suspect Sign	Time Taken	**0.028**	0.033	0.301	0.012	1	**0.015**
Base + Synthetic	Suspect Sign	Time Taken	**0.032**	0.033	0.264	0.012	1	**0.049**

Tabella 5.3: Risultati significativi dei confronti tra le distribuzioni delle metriche di valutazione ottenute - solo dataset medium - dataset con feature numeriche

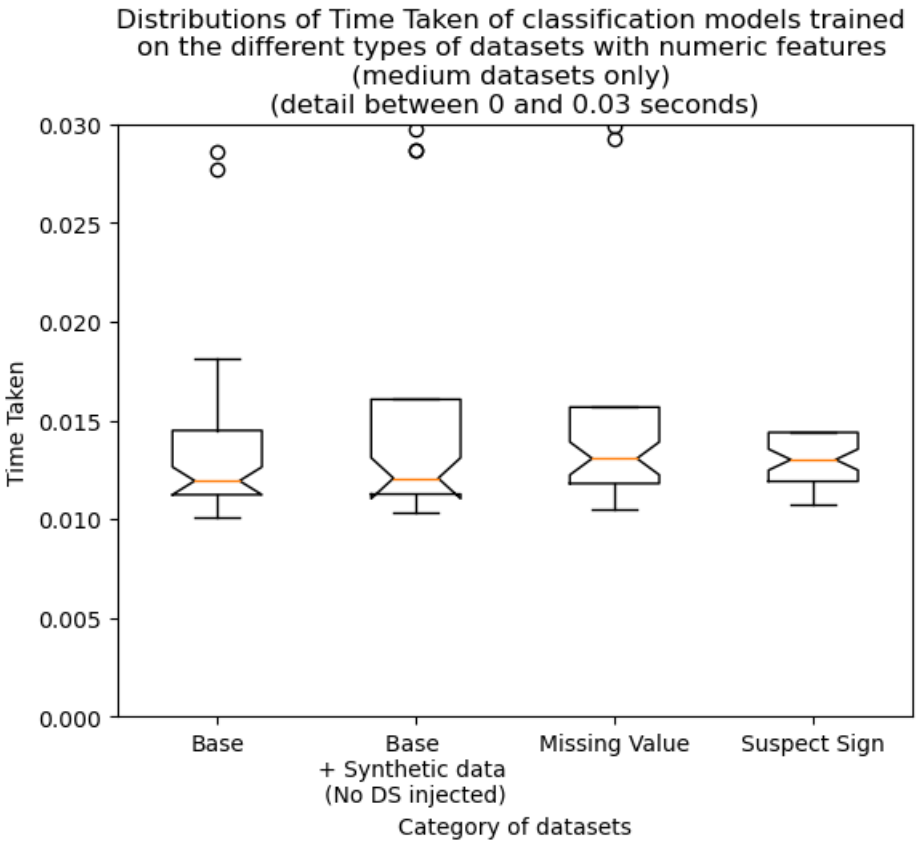


Figura 5.3: Confronto tra i boxplot delle distribuzioni del Tempo impiegato - solo dataset medium - dataset con feature numeriche

Dataset	Media	Deviazione Standard	Min	25%	50%	75%	Max
Base	0.028	0.036	0.010	0.011	0.011	0.014	0.148
Base + Synthetic	0.032	0.044	0.010	0.011	0.012	0.016	0.223
Suspect Sign	0.033	0.042	0.010	0.011	0.013	0.014	0.166

Tabella 5.4: Distribuzioni dei tempi impiegati (con differenze significative) - solo dataset medium - dataset con feature numeriche

Osservazioni

Da questi risultati ottenuti emergono delle osservazioni diverse da quelle denotate analizzando le distribuzioni delle medie delle metriche di valutazione ottenute dai modelli. Da queste analisi in particolare emerge come:

1. per i tempi impiegati nei dataset con feature numeriche di dimensioni medium si osservano gli stessi risultati denotati analizzando le medie, ma questa volta si evince anche dei maggiori tempi impiegati sui dati affetti da **Suspect Sign** smell rispetto a quelli ottenuti sui dataset di training uniti ai dati sintetici senza data smell.

5.2.3 Distribuzioni delle metriche di valutazione ottenute sui singoli dataset

I risultati significativi dei confronti tra le distribuzioni delle metriche di valutazione ottenute dai modelli di classificazione, addestrati sulle diverse versioni degli specifici dataset, sono riportati nelle Figure 5.4, 5.5, 5.6, 5.7 e 5.8 e nelle Tabelle 5.5, 5.6, 5.7 e 5.8. Anche in questo caso sono comprese le distribuzioni dei tempi impiegati per i confronti significativi.

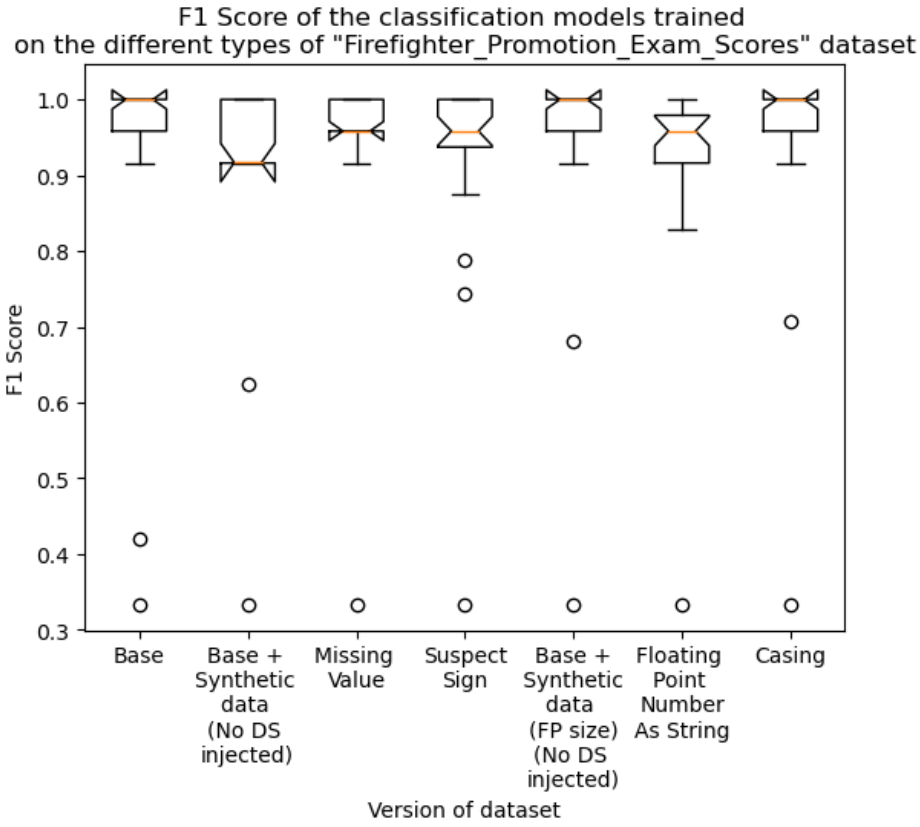


Figura 5.4: Confronto tra i boxplot delle distribuzioni di F1 Score - dataset Firefighter_Promotion_Exam_Scores

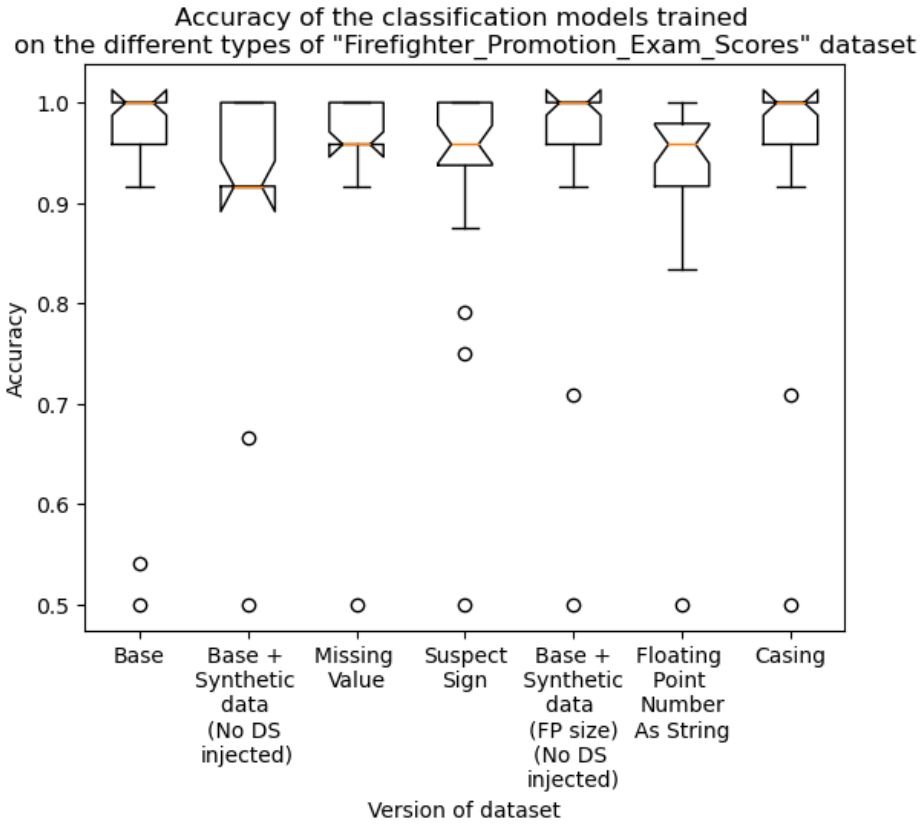


Figura 5.5: Confronto tra i boxplot delle distribuzioni di Accuracy - dataset Firefighter_Promotion_Exam_Scores

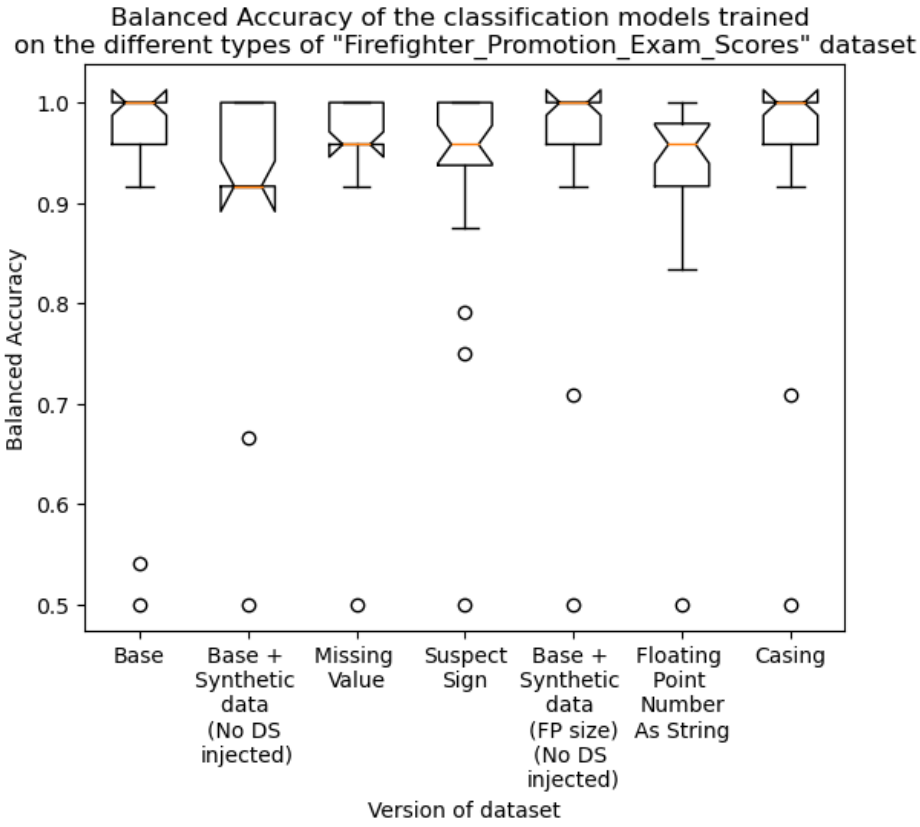


Figura 5.6: Confronto tra i boxplot delle distribuzioni di Balanced Accuracy - dataset Firefighter_Promotion_Exam_Scores

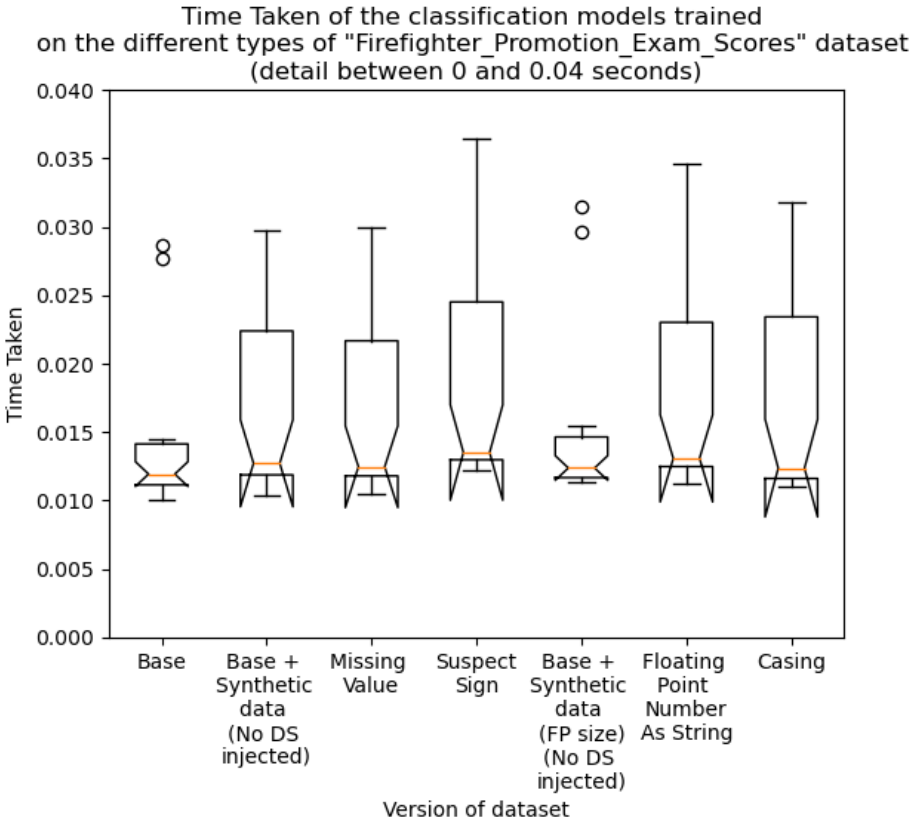


Figura 5.7: Confronto tra i boxplot delle distribuzioni del Tempo impiegato - dataset Firefighter_Promotion_Exam_Scores

Versione Dataset 1 (Firefighter _Promotion _Exam_Scores)	Versione Dataset 2 (Firefighter _Promotion _Exam_Scores)	Metrica	Media distribuzione Dataset 1	Media distribuzione Dataset 2	Test statistic	Statistic location	Statistic sign	p-value
Base	Floating Point Number As String	F1 Score	**0.936**	0.922	0.370	0.958	-1	**0.048**
Base	Floating Point Number As String	Accuracy	**0.947**	0.929	0.370	0.958	-1	**0.048**
Base	Floating Point Number As String	Balanced Accuracy	**0.947**	0.929	0.370	0.958	-1	**0.048**
Base	Suspect Sign	Time Taken	**0.024**	0.033	0.592	0.012	1	**9.833e-05**
Base + Synthetic	Suspect Sign	Time Taken	**0.031**	0.033	0.370	0.012	1	**0.048**
Base	Floating Point Number As String	Time Taken	**0.024**	0.045	0.407	0.012	1	**0.021**

Tabella 5.5: Risultati significativi dei confronti tra le distribuzioni delle metriche di valutazione ottenute - dataset Firefighter_Promotion_Exam_Scores

Dataset	Media	Deviazione Standard	Min	25%	50%	75%	Max
Base	0.024	0.030	0.010	0.011	0.011	0.014	0.121
Base + Synthetic	0.031	0.038	0.010	0.011	0.012	0.022	0.129
Floating Point Number As String	0.045	0.085	0.011	0.012	0.013	0.023	0.427
Suspect Sign	0.033	0.040	0.012	0.012	0.013	0.024	0.147

Tabella 5.6: Distribuzioni dei tempi impiegati (con differenze significative) - dataset Firefighter_Promotion_Exam_Scores

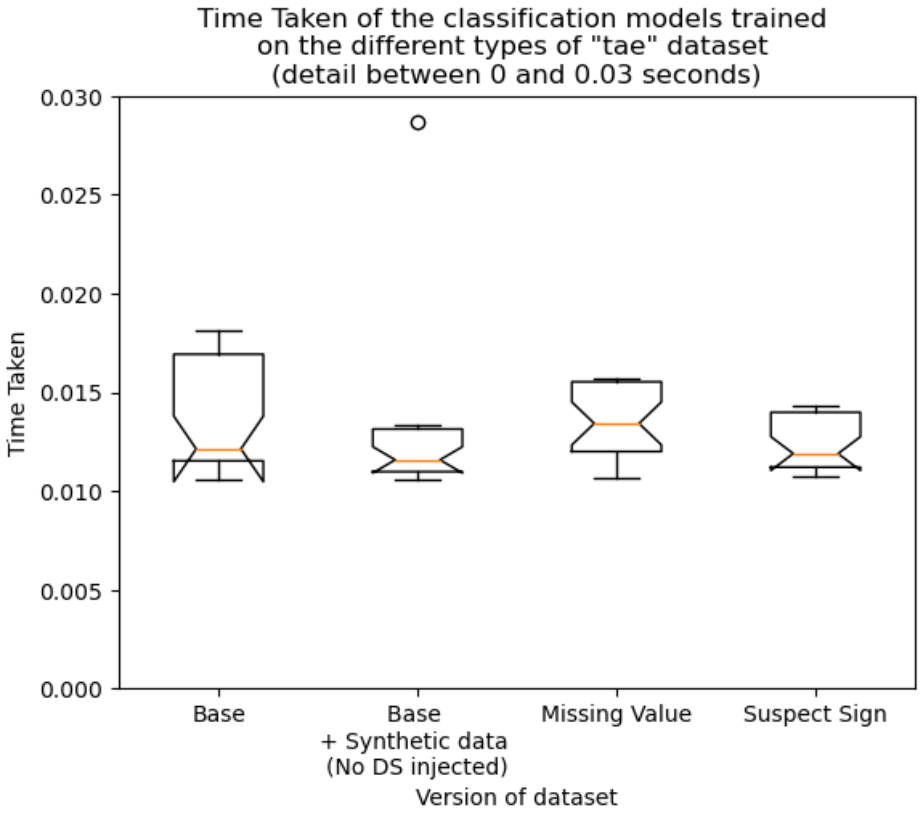


Figura 5.8: Confronto tra i boxplot delle distribuzioni del Tempo impiegato - dataset tae

Versione Dataset 1 (tae)	Versione Dataset 2 (tae)	Metrica	Media distribuzione Dataset 1	Media distribuzione Dataset 2	Test statistic	Statistic location	Statistic sign	p-value
Base + Synthetic	Missing Value	Time Taken	**0.032**	0.045	0.423	0.011	1	**0.018**

Tabella 5.7: Risultati significativi dei confronti tra le distribuzioni delle metriche di valutazione ottenute - dataset tae

Dataset	Media	Deviazione Standard	Min	25%	50%	75%	Max
Base + Synthetic	0.032	0.050	0.010	0.010	0.011	0.013	0.223
Missing Value	0.045	0.085	0.011	0.012	0.013	0.023	0.427

Tabella 5.8: Distribuzioni dei tempi impiegati (con differenze significative) - dataset tae

Osservazioni

Da questi risultati ottenuti emerge in particolare come, focalizzando l'analisi sui singoli dataset, non si notano differenze significative nei dataset large, mentre queste sono presenti nelle metriche di valutazione dei modelli di classificazione quando addestrati sulle diverse versioni dei dataset medium. In particolare, emerge che:

1. per il dataset **Firefighter_Promotion_Exam_Scores** si denota come tutte le metriche calcolate sulla versione del dataset base abbiano risultati migliori rispetto a quelle ottenute dai modelli addestrati sul dataset avente i dati iniettati con il **Floating Point Number As String** smell.
   Inoltre, i tempi impiegati per addestrare i modelli sui dataset incrementati con dati aventi **Suspect Sign** smell sono peggiori rispetto alla baseline e rispetto ai tempi impiegati sulla versione del dataset incrementata con dati senza data smell.

2. per il dataset **tae** si denota come i tempi impiegati sulla versione del dataset Missing Value siano peggiori rispetto alla versione Base + Synthetic.

Occorre notare come nei confronti relativi alle metriche F1 Score, Accuracy e Balanced Accuracy per il dataset "Firefighter_Promotion_Exam_Scores" (Figure 5.4, 5.5 e 5.6 e Tabella 5.5) venga fornito il seguente warning dalla funzione per il calcolo del test statistico: "*RuntimeWarning: ks_2samp: Exact calculation unsuccessful. Switching to method=asymp*".

5.2.4 Medie delle metriche dei modelli

Dai precedenti confronti effettuati nelle Sezioni 5.2.1, 5.2.2 e 5.2.3, emerge come le principali differenze significative con i risultati baseline e con le versioni dei dataset Base + Synthetic, siano legate ai data smell Casing, per i dataset con feature categoriche, e Suspect Sign, per i dataset con feature numeriche.

Per questo motivo, sono ora confrontate le distribuzioni delle medie delle metriche ottenute dai singoli modelli di classificazione, in modo da comprendere meglio quali tipi di modelli impattino maggiormente sui risultati ottenuti, osservando le differenze nei vari scenari.

Essendo 27 i modelli di classificazione considerati, le heatmap sono state suddivise in tre sotto-figure. Le heatmap relative al dataset Firefighter_Promotion_Exam_Scores, prodotte per valutare il caso del data smell Floating Point Number As String, sono disponibili negli script della cartella "create_plot_scripts" nella repository Github indicata nella Sezione 1.3, oltre a queste di seguito riportate.

Medie delle metriche di accuracy dei modelli

Nelle Figure 5.9, 5.10, 5.11, 5.12, 5.13 e 5.14 sono riportate le heatmap delle medie delle metriche di accuracy dei modelli quando addestrati sui diversi tipi di dataset, sia nello scenario dei dataset con feature categoriche che in quello dei dataset con feature numeriche. Per questi confronti non stati separati i dataset in base alle loro dimensioni.

Osservazioni Da questi risultati ottenuti emerge principalmente che per i dataset con feature categoriche, tra i diversi modelli, alcuni presentano differenze maggiori tra gli scenari analizzati, in particolare SVC e KNeighborsClassifier hanno risultati di F1 Score, Accuracy e Balanced Accuracy peggiori rispetto alla baseline sui dataset Casing, PassiveAggressiveClassifier denota lo stesso comportamento sulle metriche di Accuracy e Balanced Accuracy, come anche GaussianNB sulla sola Balanced Accuracy e BernoulliNB su F1 Score. Alcuni modelli mostrano anche dei risultati opposti su alcune di queste tre metriche, come Perceptron, QuadraticDiscriminantAnalysis, BernoulliNB e lo stesso GaussianNB, anche se le differenze sono leggere, in particolare per la metrica Balanced Accuracy.
Anche per i dataset con feature numeriche ci sono modelli che presentano differenze maggiori tra i diversi scenari dei dataset, in particolare SDGClassifier, Perceptron e BernoulliNB hanno risultati di F1 Score, Accuracy e Balanced Accuracy peggiori nello scenario Suspect Sign rispetto alla baseline. Anche PassiveAggressiveClassifier ha il medesimo comportamento ma limitato alla metrica Accuracy. I modelli GaussianNB e QuadraticDiscriminantAnalysis sembrano invece avere risultati generalmente migliori, avendo quindi un comportamento opposto rispetto ai modelli precedentemente elencati.

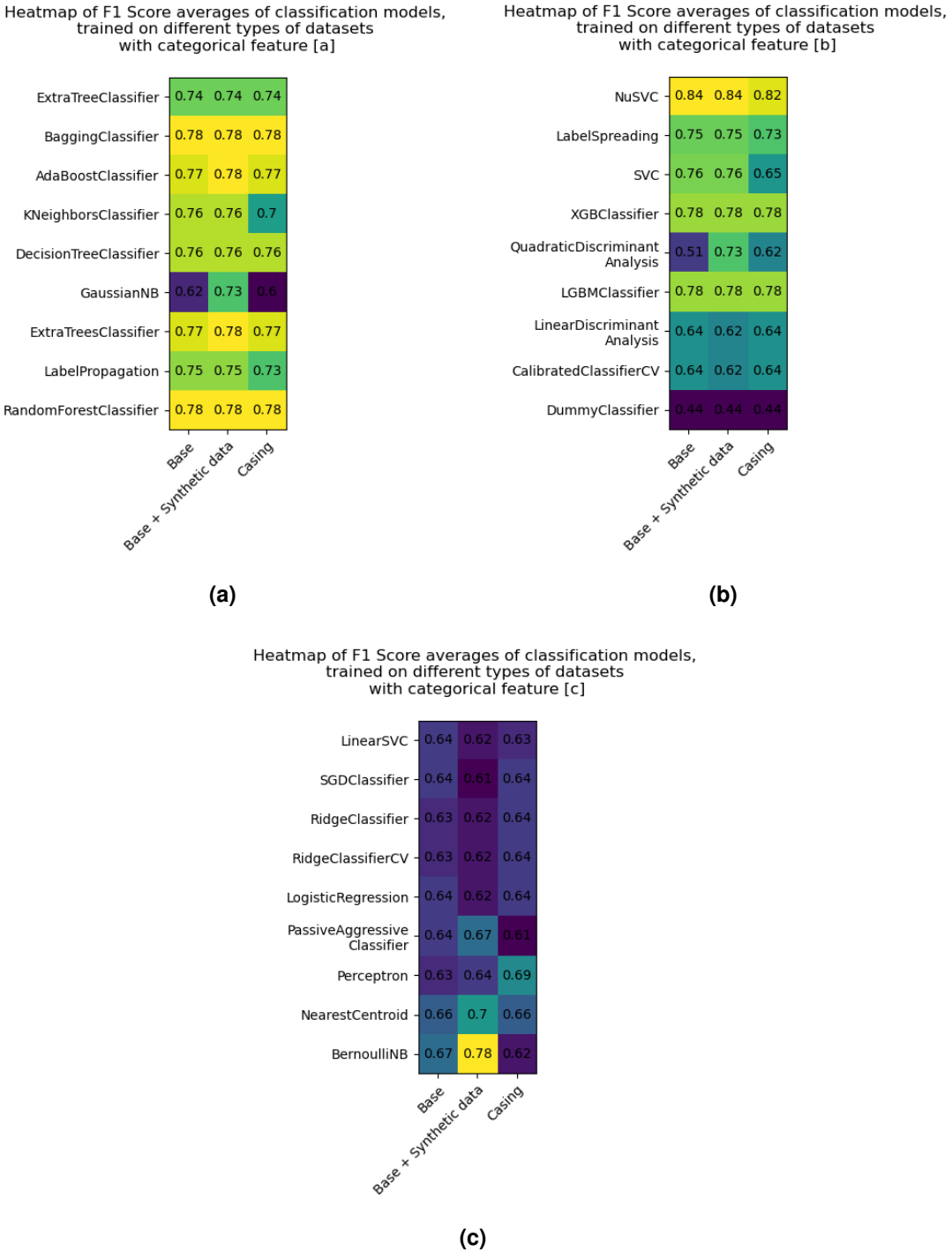


Figura 5.9: Heatmap delle medie di F1 Score dei modelli di classificazione, addestrati sui diversi tipi di dataset con feature categoriche

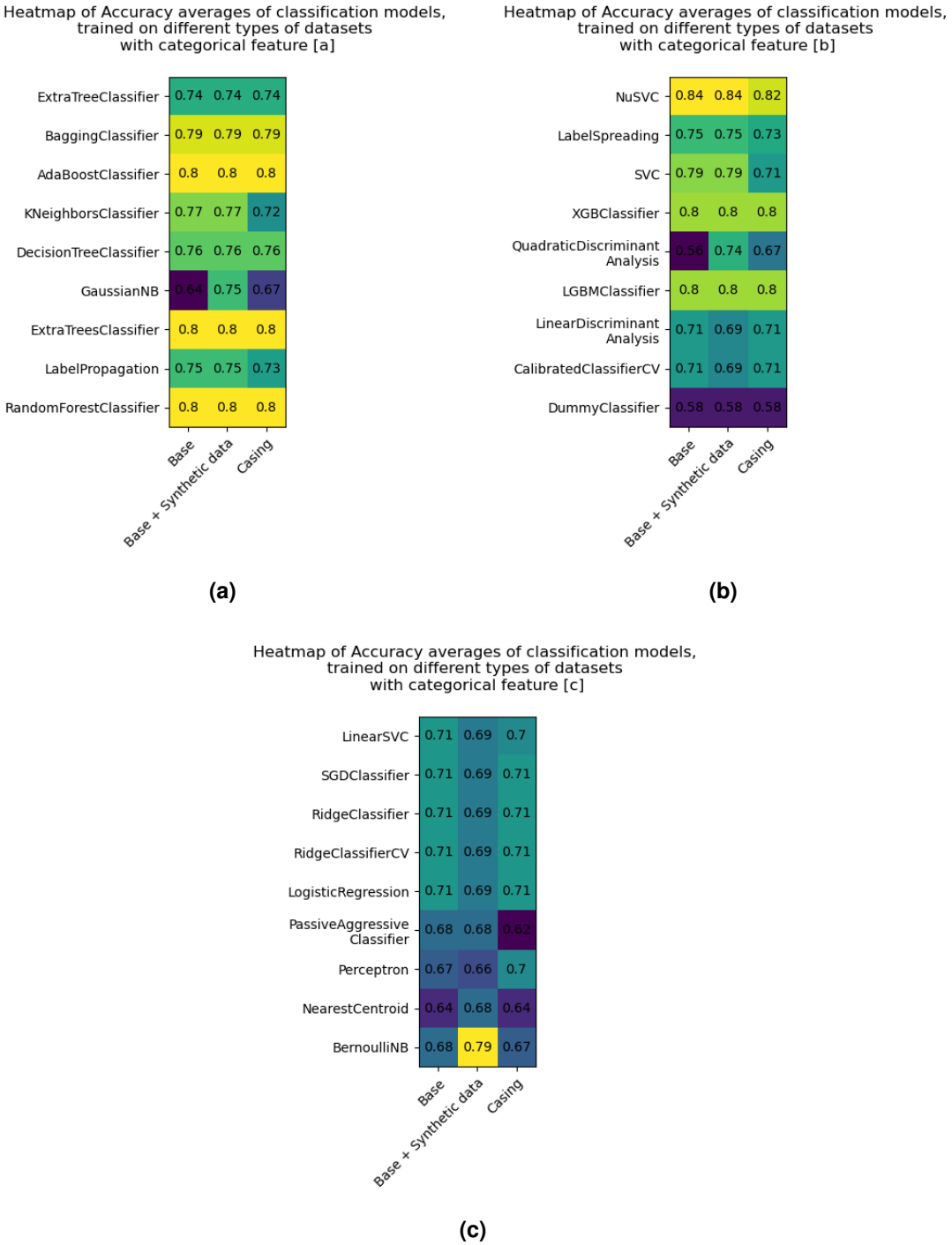


Figura 5.10: Heatmap delle medie di Accuracy dei modelli di classificazione, addestrati sui diversi tipi di dataset con feature categoriche

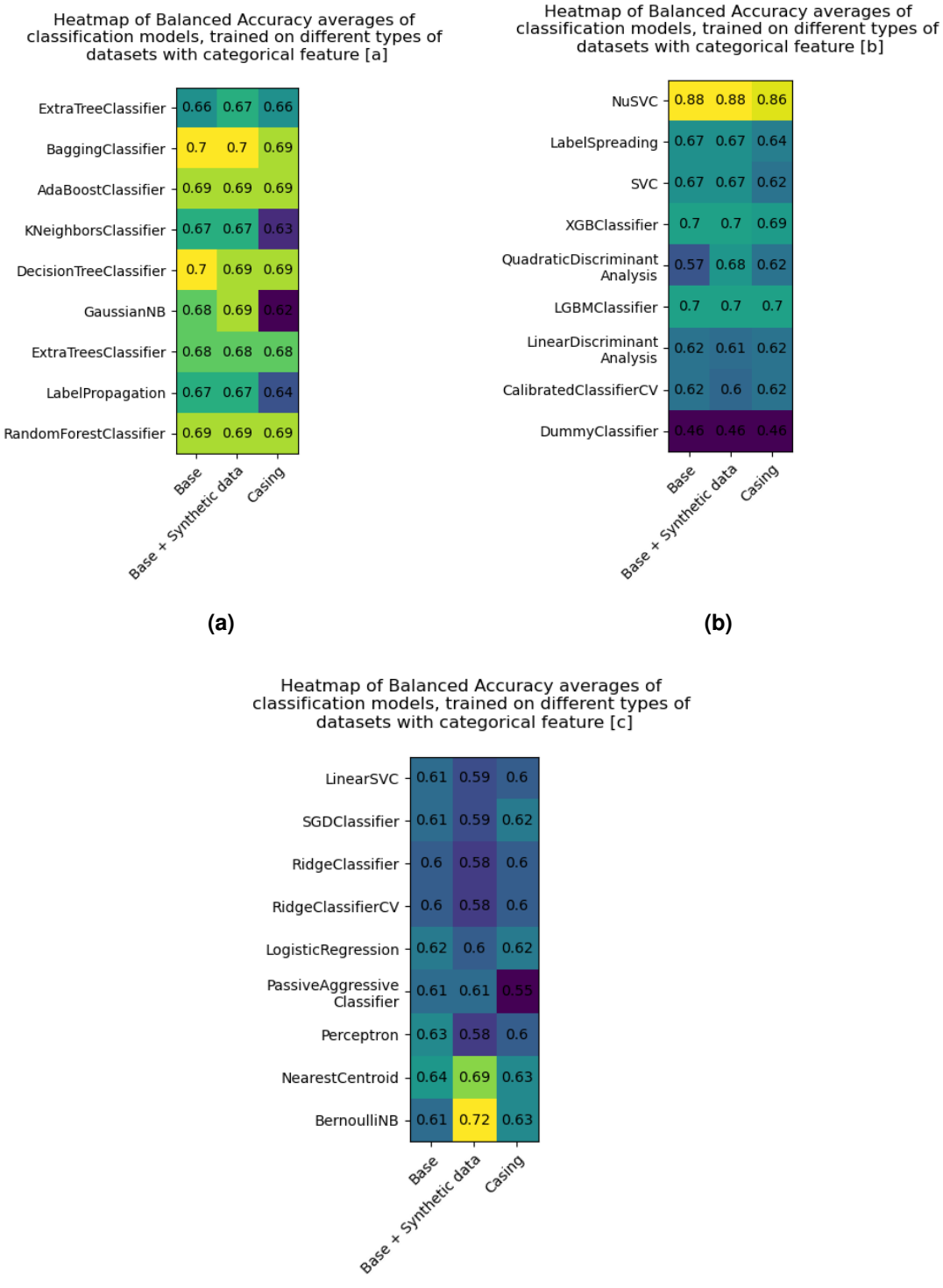


Figura 5.11: Heatmap delle medie di Balanced Accuracy dei modelli di classificazione, addestrati sui diversi tipi di dataset con feature categoriche

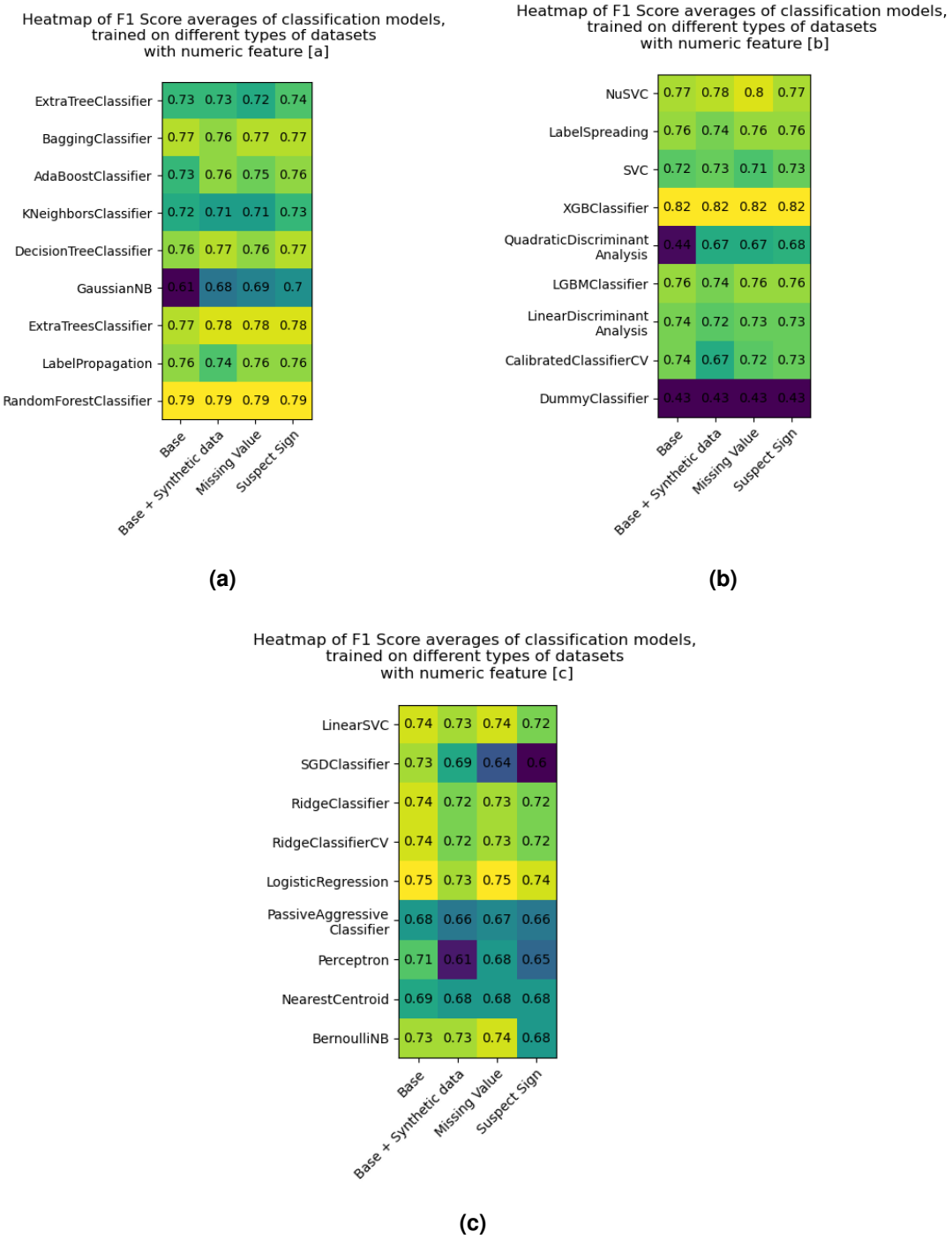


Figura 5.12: Heatmap delle medie di F1 Score dei modelli di classificazione, addestrati sui diversi tipi di dataset con feature numeriche

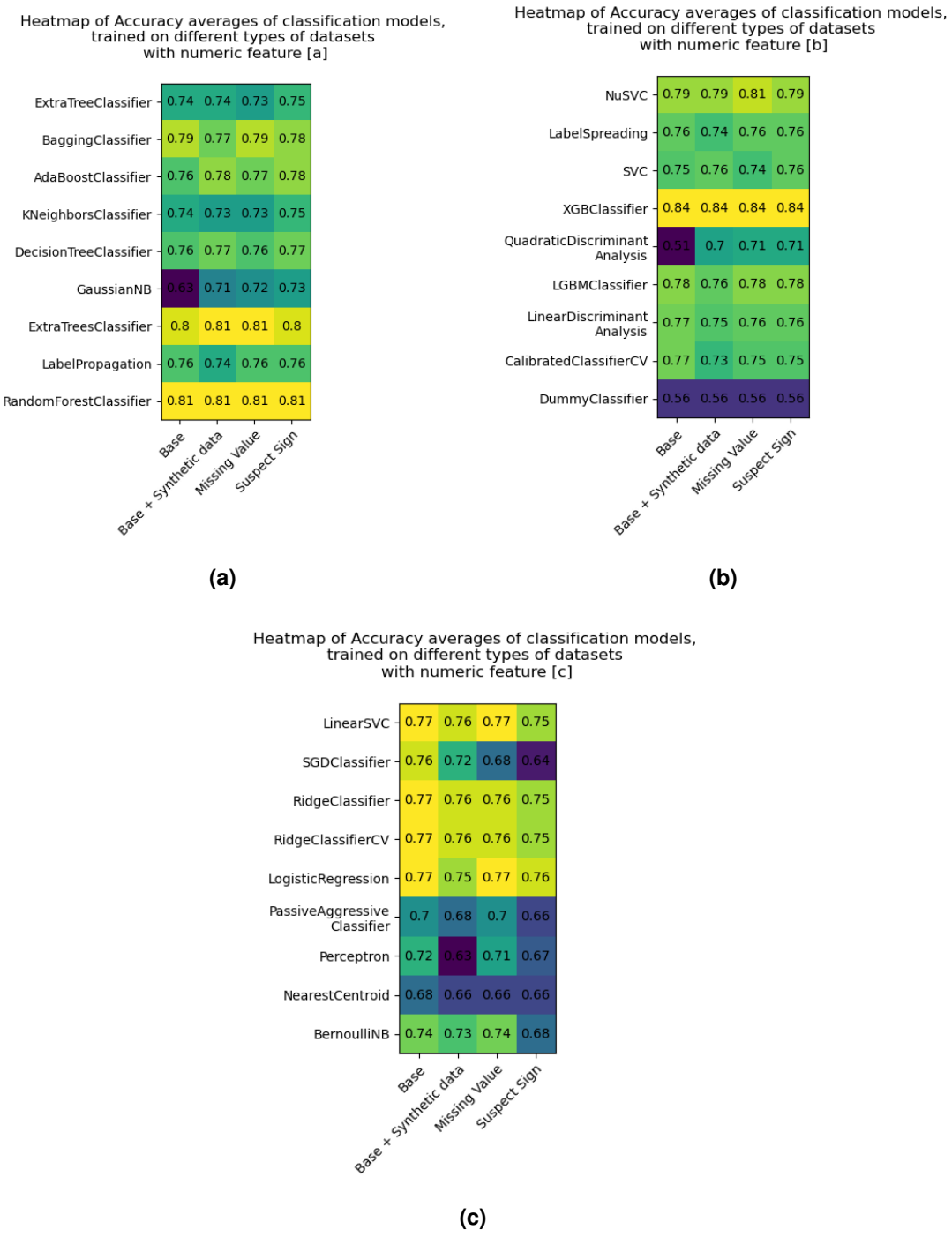


Figura 5.13: Heatmap delle medie di Accuracy dei modelli di classificazione, addestrati sui diversi tipi di dataset con feature numeriche

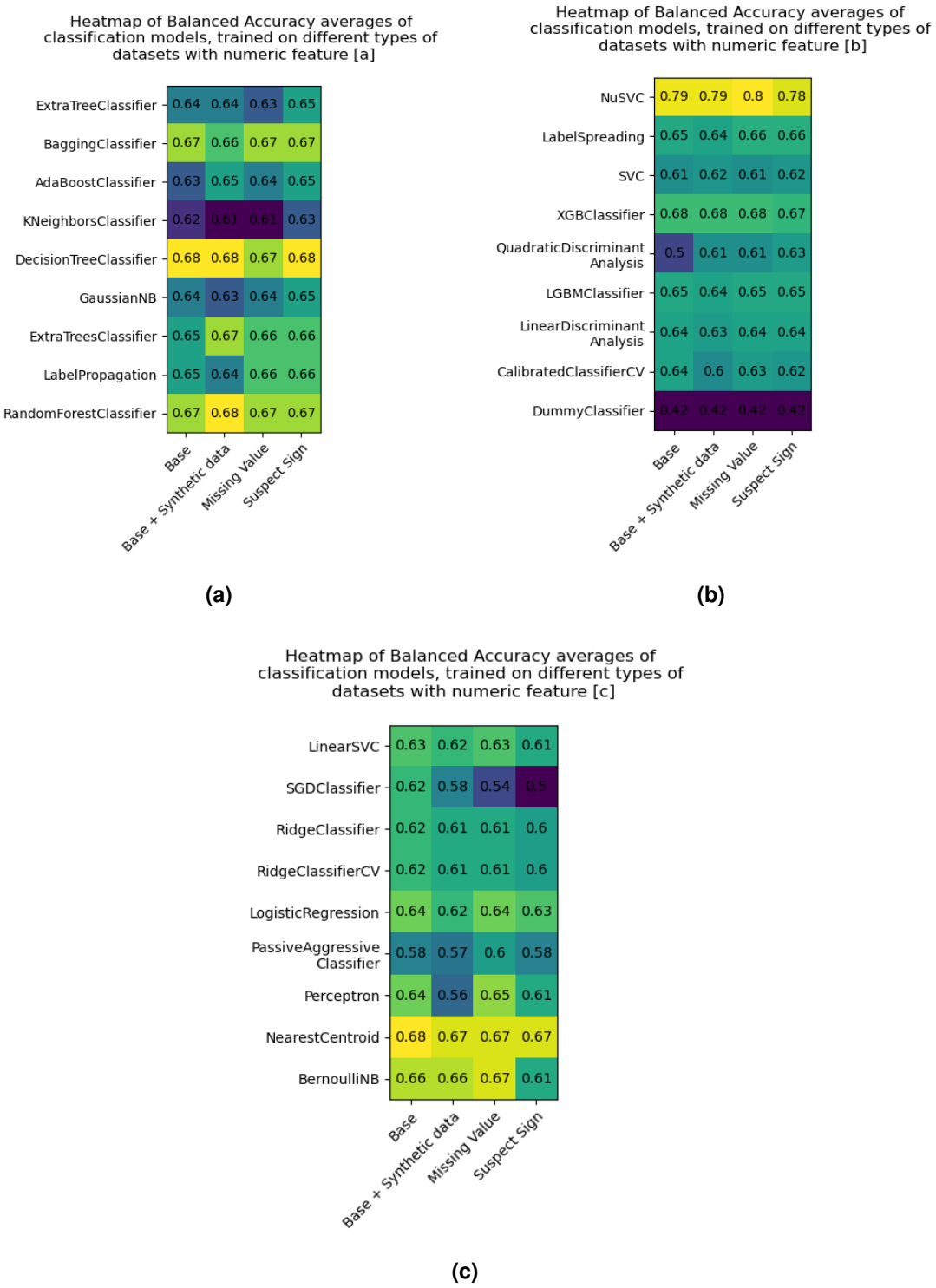


Figura 5.14: Heatmap delle medie di Balanced Accuracy dei modelli di classificazione, addestrati sui diversi tipi di dataset con feature numeriche

Medie delle metriche di valutazione ottenute dai modelli

Di seguito, nelle Figure 5.15 e 5.16 sono riportate invece le heatmap delle medie dei tempi impiegati dai modelli, calcolate quando addestrati negli stessi scenari sui diversi tipi di dataset, separando i casi dei dataset con feature categoriche e quelli dei dataset con feature numeriche.

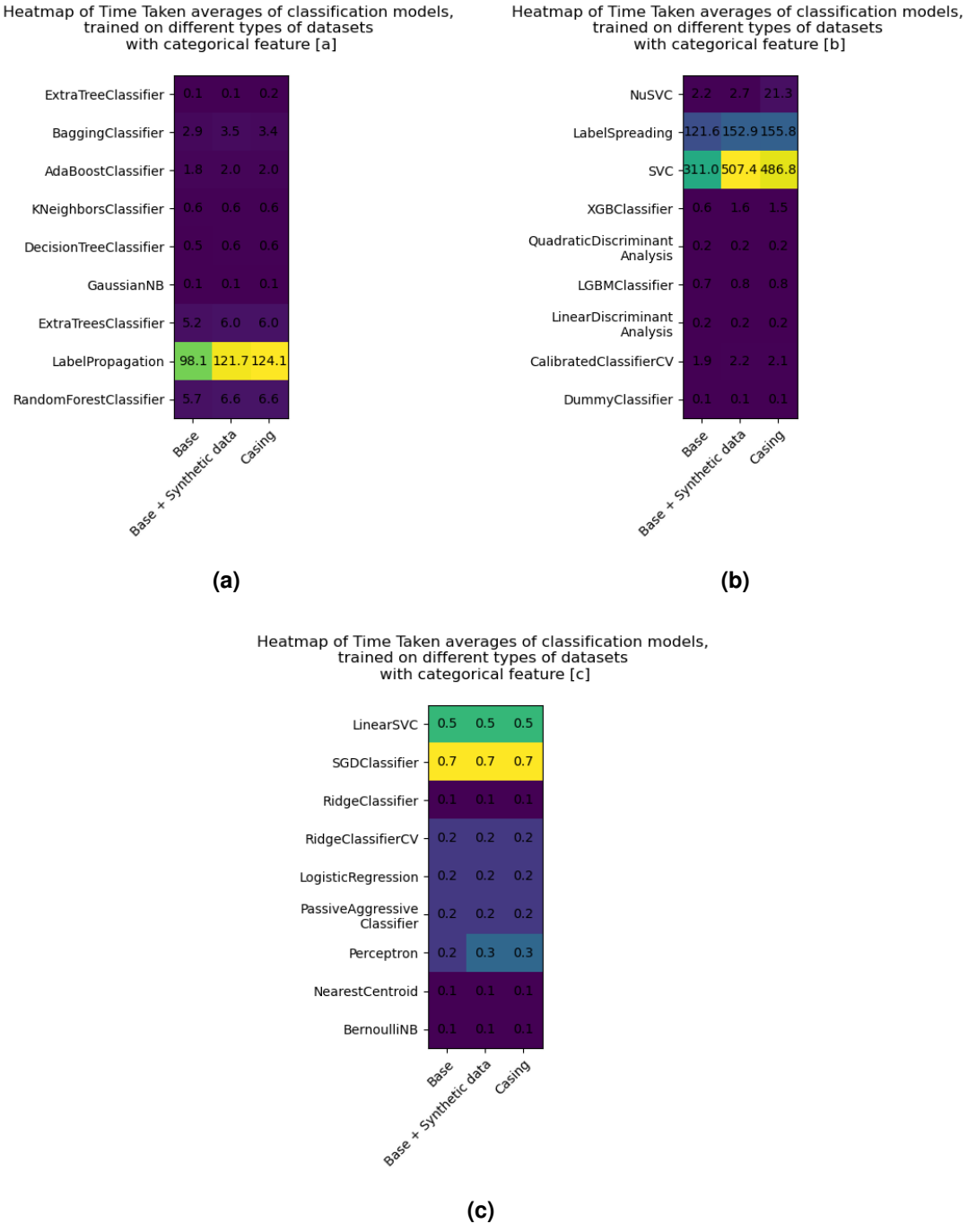


Figura 5.15: Heatmap delle medie dei Tempi impiegati dai modelli di classificazione, addestrati sui diversi tipi di dataset con feature categoriche

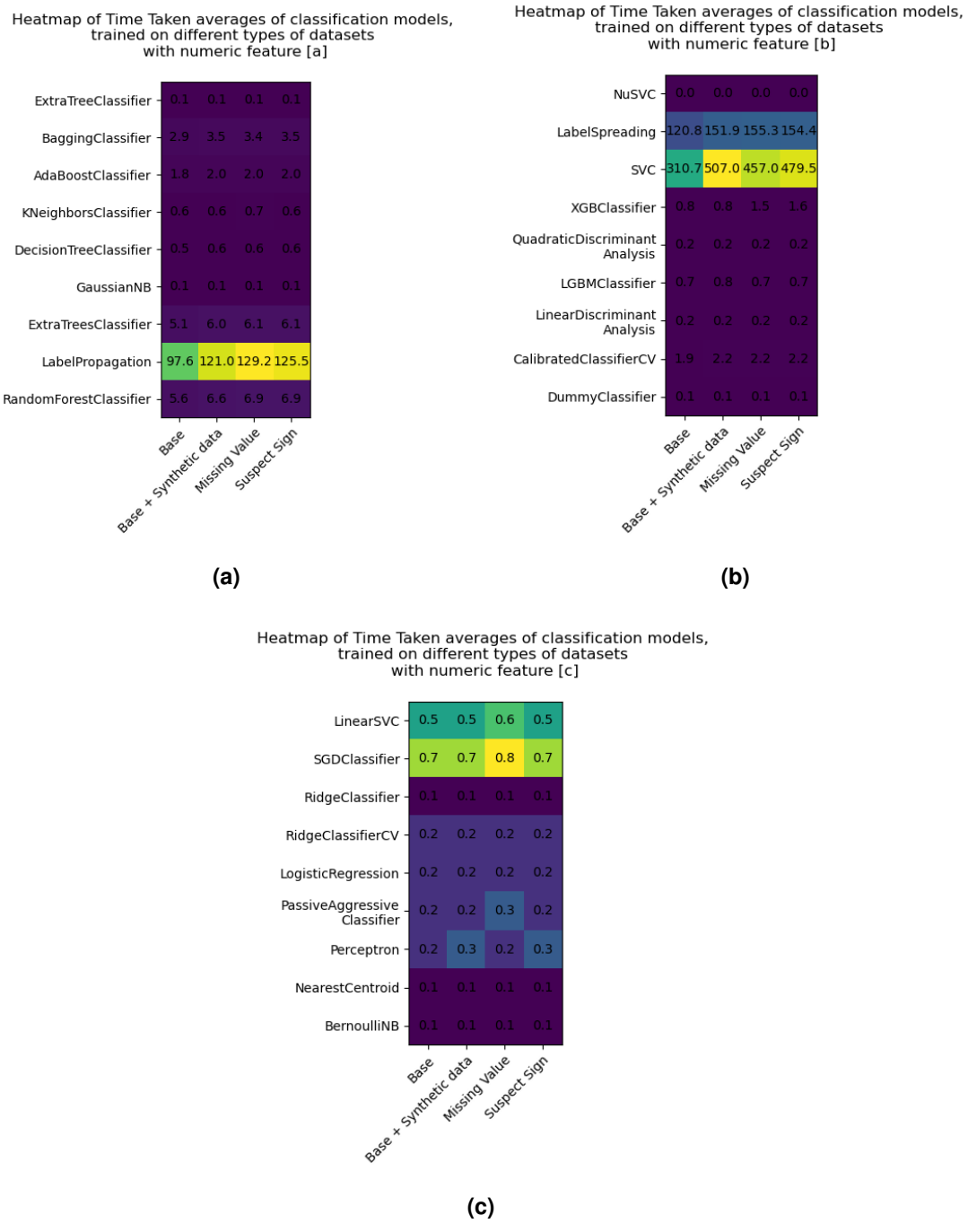


Figura 5.16: Heatmap delle medie dei Tempi impiegati dai modelli di classificazione, addestrati sui diversi tipi di dataset con feature numeriche

Osservazioni Da questi confronti è possibile notare come nello scenario Base + Synthetic i risultati in termini di tempi impiegati siano generalmente peggiori, mentre i modelli LabelPropagation, SVC, LabelSpreading e NuSVC sono quelli con i peggioramenti più incisivi derivanti dall'introduzione di dati con il Casing smell nello split di training. Anche in altri modelli si denotano delle differenze relative notevoli, come in XGBClassifier e BaggingClassifier. Analizzando invece i dataset con feature numeriche, si denota come anche in questo caso i modelli Label Propagation, SVC e LabelSpreading abbiano dei notevoli peggioramenti rispetto alla baseline quando addestrati su dati aventi dei Suspect Sign smell. Anche per questi confronti si evincono delle differenze relative notevoli in altri modelli, come in XGBClassifier e BaggingClassifier.

5.3 Risposte alle Research Question

Dai risultati riportati nella Sezione 5.2 è possibile rispondere alle due Research Question poste nella Sezione 4.1, rigettando le ipotesi nulle osservando i confronti che hanno denotato differenze significative, andando a sintetizzare i risultati principali ed escludendo però quelli relativi ai data smell Floating Point As Number e Missing Value poiché derivanti da osservazioni su un singolo dataset:

👉 **Answer to RQ_1.** Per i dataset aventi feature categoriche, le medie di Balanced Accuracy ottenute dai modelli di classificazione addestrati su dataset di dimensioni large arricchiti con dati sintetici aventi il data smell **Casing** sono in genere peggiori rispetto a quelle che si ottengono quando addestrati su un dataset della medesima dimensione con dati sintetici che non presentano data smell.

👉 **Answer to RQ_2.** Per i dataset aventi feature numeriche, si ravvisano differenze significative sulle distribuzioni dei tempi impiegati, analizzando solo i dataset medium, denotando un maggiore tempo impiegato dai modelli addestrati su dataset arricchiti con dati sintetici aventi il data smell di tipo **Suspect Sign** rispetto alla baseline senza dati aggiuntivi, e rispetto a quando addestrati sulla versione del dataset di partenza avente dati sintetici iniettati senza data smell.

5.4 Minacce alla validità

Di seguito sono riportate le minacce alla validità relative alla sperimentazione realizzata.

5.4.1 Minacce alla Conclusion validity

Una delle minacce relative alla Conclusion validity considerate in questo lavoro è la bassa potenza statistica. Per assicurare una significatività statistica è stato difatti utilizzato il test Kolmogorov-Smirnov, un test non parametrico, richiedendo un *p-value* minore di 0.05 per poter definire due distribuzioni come differenti. Questa scelta sull'imposizione del p-value deriva dalla letteratura [64]. Per limitare problematiche relative alla affidabilità delle misure, nella valutazione delle metriche di accuracy sono stati effettuati confronti su misure definite in letteratura per valutare i modelli di classificazione, ovvero l'F1 Score [59], l'Accuracy [60], la Balanced Accuracy [61]. Per valutare le performance è stato invece utilizzato il tempo impiegato sommando quello di addestramento, testing e calcolo delle metriche. Per l'individuazione dei data smell è stato utilizzato il tool proposto da Foidl et al. [3], ovvero DSD, già utilizzato per lo stesso scopo da Recupito et al. [5]. In aggiunta, i risultati potrebbero essere influenzati anche dai diversi problemi di classificazione che sono affrontati utilizzando i dataset scelti, che sono infatti di classificazione binaria per 3 dei 5 dataset, e multi-class per i restanti. Le conclusioni ottenute tuttavia non considerano questo aspetto, e quindi neanche il diverso modo in cui sono calcolate le metriche di accuracy in questi due scenari da LazyPredict[1], potendo quindi essere una minaccia alle stesse. Un'altra minaccia che potrebbe limitare i risultati è il non aver adottato una tecnica di cross-validation, a causa del costo per il training dei modelli.

5.4.2 Minacce alla Construct validity

Lo studio effettuato si limita solo ad alcuni specifici tipi di data smell, in particolare a 4 dei più frequenti individuati da Recupito et al. [5]. Per questo motivo, non sono stati considerati gli effetti di altri data smell rilevati, come ad esempio Duplicated Value, sui risultati. Analizzando i data smell presenti nei dataset con configurazione

"Strict" del tool DSD, emergono altri data smell in alcuni dataset, tuttavia questi non sono considerati come tali poiché viene preferita la configurazione "Tolerant" per poter identificare queste problematiche, ma ciò potrebbe comunque influire sui risultati. Inoltre, sempre utilizzando la configurazione "Strict", anche la generazione di dati sintetici tramite CTGAN ha aggiunto degli Extreme Value smell, quando si valuta il dataset "diabetic_data" se analizzato su due dei sotto-dataset che lo compongono. Anche questi aspetti potrebbero aver influenzato i risultati ottenuti. Oltre a queste minacce alla validità, anche il modo in cui sono stati aggiunti i data smell influenza i risultati e la relazione con le osservazioni: questi infatti sono stati inseriti manualmente tramite script Python, manipolando i dati sintetici generati dalla CTGAN. Se per alcuni data smell si tratta di manipolazioni semplici, come la rimozione del valore per Missing Value o l'inversione del segno per Suspect Sign, la manipolazione richiesta per generare consistentemente, e in ogni contesto, dei Casing smell risulta più complessa delle altre, aggiungendo anche un elemento di casualità ai dati, che non solo potrebbe rendere più complessa la gestione di questi tipi di dataset per i modelli di classificazione, ma potrebbe non rispecchiare una situazione reale di presenza di Casing smell, potendo quindi limitare la validità dei risultati. Anche in relazione alla scelta delle feature ci sono delle minacce da tenere in considerazione: difatti cercando di utilizzare le feature classificate al primo posto possibile dal metodo di feature selection *ReliefFAttributeEval*, non è sicuro queste siano le feature con il maggior impatto sulle prestazioni. Inoltre, questa classificazione di feature ha portato alle problematiche descritte nella Sezione 4.4.3, relative alla percentuale di dati affetti da Suspect Smell, che in un dataset è minore del 10%, e all'effettiva scelta della feature in cui iniettare il data smell, poiché per un altro dataset non è stata utilizzata la prima feature individuata da *ReliefFAttributeEval*. Per limitare l'impatto del numero di record generati, avendo dataset di dimensioni diverse, per ogni data smell sono stati iniettati un numero necessario di dati sintetici per rilevare il data smell, proporzionato alla dimensione del dataset stesso. Anche il modo in cui sono stati pre-processati i dataset per poter essere elaborati dai modelli di classificazione, descritto nella Sezione 4.2.1, può aver influito sui risultati, in particolare il modo in cui le variabili categoriche sono state codificate in variabili ordinali numeriche, senza tenere conto della effettiva relazione tra i valori, e imponendo quindi un nuovo ordine anche alle feature che

erano di tipo nominale. Tuttavia, questo tipo di pre-processing è stato effettuato per tutti i dataset considerati ed in tutti gli scenari. Un'ulteriore minaccia alla validità dei risultati potrebbe essere il modo in cui LazyPredict[1] effettua la selezione delle feature per addestrare i modelli di classificazione, che non viene analizzato nella fase di analisi dei risultati. Inoltre, potrebbe aver influito sui risultati anche la scelta di iniettare i data smell su dati sintetici invece che sui dati originali dei dataset, e anche la scelta di aver aggiunto i dati affetti da data smell sono sugli split di training invece che anche in quelli di testing, relativamente agli addestramenti dei modelli. Questi diversi trattamenti, e la loro interpolazione, possono aver limitato i risultati e le osservazioni ottenuti.

5.4.3 Minacce alla External validity

La scelta dei dataset potrebbe restringere la generalizzazione dei risultati. Per limitare questa minaccia, seppur modificandoli, sono stati scelti dei dataset utilizzati anche in letteratura, e in particolare anche da Recupito et al. [5], includendo sia dataset di dimensioni large (maggiori di 1000 record) che medium (tra 100 e 1000) [51], che presentassero dei tipi di feature diverse, comprendendo dataset con sia feature numeriche che categoriche, sia solo numeriche e sia solo categoriche, e che permettessero di valutare classificazioni sia binarie che multi-class. Per limitare anche l'impatto delle differenze di taglie e di caratteristiche sulla generalizzazione dei risultati, le analisi condotte sono state sezionate in base alla taglia (tranne quelle riportate nella Sezione 5.2.4) e ai tipi delle feature che compongono i dataset, in modo da specificare l'ambito in cui sono stati ottenuti i risultati, ma non in base al tipo di problema di classificazione, senza quindi separare i casi di classificazione binaria e multi-class.

CAPITOLO 6

Conclusioni

6.1 Risultati ottenuti e Osservazioni

Dalle risposte alle Research Question fornite nella Sezione 5.3, emerge come:

Finding 1. Tra i data smell analizzati, **Casing** porta a peggioramenti sulle medie della metrica di Balanced Accuracy dei modelli di classificazione quando addestrati su dataset di dimensioni large arricchiti da dati sintetici che presentano questo data smell, rispetto a quando addestrati sulla versione del dataset avente dati sintetici senza i data smell analizzati.

Finding 2. Tra i data smell analizzati, **Suspect Sign** porta a peggioramenti sulle performance dei modelli di classificazione quando addestrati su dataset arricchiti da dati sintetici che presentano tale data smell, seppur questo impatto sia limitato solo ai dataset di dimensioni medium, rispetto alle baseline considerate.

Take-away Message #1. I modelli di classificazione che sono maggiormente impattati negativamente nelle loro metriche di accuracy dall'iniezione di **Casing** smell, nei dataset con feature categoriche arricchiti da dati sintetici su cui sono addestrati, sono **SVC** e **KNeighborsClassifier**.

💬 **Take-away Message #2.** I modelli di classificazione che sono maggiormente impattati negativamente nelle loro performance dall'iniezione di **Suspect Sign** smell, oppure anche da quella di **Casing** smell, nei dataset dei rispettivi scenari arricchiti da dati sintetici su cui sono addestrati, sono **Label Propagation**, **SVC** e **Label-Spreading**.

Da quanto osservato a seguito di queste sperimentazioni, emerge quindi come sarebbe necessario porre maggiore attenzione alla presenza dei data smell sui dataset di addestramento dei modelli di classificazione, ad esempio adottando maggiormente tecniche di gestione della qualità dei dati, soprattutto se questi sono di grande rilevanza, in quanto la loro presenza potrebbe portare a peggioramenti nelle metriche di accuracy e nelle performance degli stessi.

6.2 Possibili sviluppi futuri

Come possibili sviluppi, potrebbero essere estese le analisi qui condotte, in modo da verificarne la generalizzabilità. In particolare, potrebbero essere inclusi più dataset alle analisi e con diverse caratteristiche, ad esempio includendo anche dataset di taglia small. Inoltre, questo potrebbe portare a più risultati relativamente ai data smell "Floating Point Number As String" e "Missing Value", che potrebbero unirsi a quelli trovati in questo lavoro e riportati nella Sezione 5.2. Potrebbe inoltre essere posta maggiore ricerca alle tecniche di generazione di dati sintetici che presentino "Extreme Value Smell", per poter quindi valutare anche quello scenario, oppure estendere le analisi anche ad altri tipi di data smell. In aggiunta, potrebbero essere svolte sperimentazioni anche senza l'aggiunta di dati sintetici, oppure sperimentando altre tecniche per la loro generazione e per l'iniezione di data smell. Infine, si potrebbe verificare la robustezza dei risultati ottenuti applicando una cross-validation al processo di sperimentazione.

Bibliografia

[1] OECD, *Artificial Intelligence in Society*, 2019. [Online]. Available: https://www.oecd-ilibrary.org/content/publication/eedfee77-en (Citato a pagina 1)

[2] H. B. Braiek and F. Khomh, "On testing machine learning programs," 2018. (Citato alle pagine 1 e 7)

[3] H. Foidl, M. Felderer, and R. Ramler, "Data smells: Categories, causes and consequences, and detection of suspicious data in ai-based systems," 2022. (Citato alle pagine 1, 7, 8, 9, 10, 12, 13, 17, 31, 39, 40 e 77)

[4] W. Cunningham, "The wycash portfolio management system," in *Addendum to the Proceedings on Object-Oriented Programming Systems, Languages, and Applications (Addendum)*, ser. OOPSLA '92. New York, NY, USA: Association for Computing Machinery, 1992, p. 29–30. [Online]. Available: https://doi.org/10.1145/157709.157715 (Citato alle pagine 1 e 4)

[5] G. Recupito, R. Rapacciuolo, D. Di Nucci, and F. Palomba, "Unmasking Data Secrets: An Empirical Investigation into Data Smells and Their Impact on Data Quality," 2 2024. [Online]. Available: https://figshare.com/articles/media/Unmasking_Data_Secrets_An_Empirical_Investigation_into_Data_Smells_and_Their_Impact_on_Data_Quality/24598851 (Citato alle pagine 1, 4, 5, 8, 12, 13, 15, 16, 17, 32, 33, 34, 39, 77 e 79)

[6] Z. Li, P. Avgeriou, and P. Liang, "A systematic mapping study on technical debt and its management," *Journal of Systems and Software*, 12 2014. (Citato a pagina 4)

[7] V. Lenarduzzi, T. Besker, D. Taibi, A. Martini, and F. Arcelli Fontana, "A systematic literature review on technical debt prioritization: Strategies, processes, factors, and tools," *Journal of Systems and Software*, vol. 171, p. 110827, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016412122030220X (Citato a pagina 4)

[8] F. Palomba, G. Bavota, M. D. Penta, R. Oliveto, D. Poshyvanyk, and A. De Lucia, "Mining version histories for detecting code smells," *IEEE Transactions on Software Engineering*, vol. 41, no. 5, pp. 462–489, 2015. (Citato a pagina 4)

[9] F. Palomba, A. Panichella, A. De Lucia, R. Oliveto, and A. Zaidman, "A textual-based technique for smell detection," in *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*, 2016, pp. 1–10. (Citato a pagina 4)

[10] N. Moha, Y.-G. Gueheneuc, L. Duchien, and A.-F. Le Meur, "Decor: A method for the specification and detection of code and design smells," *IEEE Transactions on Software Engineering*, vol. 36, no. 1, pp. 20–36, 2010. (Citato a pagina 4)

[11] R. Alfayez, W. Alwehaibi, R. Winn, E. Venson, and B. Boehm, "A systematic literature review of technical debt prioritization," in *Proceedings of the 3rd international conference on technical debt*, 2020, pp. 1–10. (Citato a pagina 4)

[12] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison, "Hidden technical debt in machine learning systems," in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28. Curran Associates, Inc., 2015. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2015/file/86df7dcfd896fcaf2674f757a2463eba-Paper.pdf (Citato a pagina 5)

[13] J. Bogner, R. Verdecchia, and I. Gerostathopoulos, "Characterizing technical debt and antipatterns in ai-based systems: A systematic mapping study," in *2021 IEEE/ACM International Conference on Technical Debt (TechDebt)*. IEEE, May

2021. [Online]. Available: http://dx.doi.org/10.1109/TechDebt52882.2021.00016 (Citato a pagina 5)

[14] M. F. Bosu and S. G. MacDonell, "Data quality in empirical software engineering: a targeted review," in *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE '13. ACM, Apr. 2013. [Online]. Available: http://dx.doi.org/10.1145/2460999.2461024 (Citato a pagina 5)

[15] G. Lacerda, F. Petrillo, M. Pimenta, and Y. G. Guéhéneuc, "Code smells and refactoring: A tertiary systematic review of challenges and observations," *Journal of Systems and Software*, vol. 167, p. 110610, 2020. (Citato alle pagine 5, 6 e 7)

[16] M. V. Mäntylä and C. Lassenius, "Subjective evaluation of software evolvability using code smells: An empirical study," *Empirical Software Engineering*, vol. 11, pp. 395–431, 2006. (Citato a pagina 6)

[17] A. Shome, L. Cruz, and A. van Deursen, "Data smells in public datasets," in *Proceedings of the 1st International Conference on AI Engineering: Software Engineering for AI*, ser. CAIN '22. ACM, May 2022. [Online]. Available: http://dx.doi.org/10.1145/3522664.3528621 (Citato alle pagine 7, 13, 14 e 15)

[18] M. Ge and M. Helfert, "A review of information quality research - develop a research agenda." 01 2007, pp. 76–91. (Citato a pagina 8)

[19] N. Polyzotis, M. Zinkevich, S. Roy, E. Breck, and S. Whang, "Data validation for machine learning," in *Proceedings of Machine Learning and Systems*, A. Talwalkar, V. Smith, and M. Zaharia, Eds., vol. 1, 2019, pp. 334–347. [Online]. Available: https://proceedings.mlsys.org/paper_files/paper/2019/file/928f1160e52192e3e0017fb63ab65391-Paper.pdf (Citato a pagina 10)

[20] E. Breck, N. Polyzotis, S. Roy, S. E. Whang, and M. Zinkevich, "Data infrastructure for machine learning," in *SysML conference*, 2018. (Citato a pagina 10)

[21] D. Baylor, E. Breck, H.-T. Cheng, N. Fiedel, C. Y. Foo, Z. Haque, S. Haykal, M. Ispir, V. Jain, L. Koc, C. Y. Koo, L. Lew, C. Mewald, A. N. Modi, N. Polyzotis,

S. Ramesh, S. Roy, S. E. Whang, M. Wicke, J. Wilkiewicz, X. Zhang, and M. Zinkevich, "Tfx: A tensorflow-based production-scale machine learning platform," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1387–1395. [Online]. Available: https://doi.org/10.1145/3097983.3098021 (Citato a pagina 10)

[22] S. Schelter, P. Schmidt, T. Rukat, M. Kiessling, A. Taptunov, F. Biessmann, and D. Lange, "Deequ - data quality validation for machine learning pipelines," in *NeurIPS 2018*, 2018. [Online]. Available: https://www.amazon.science/publications/deequ-data-quality-validation-for-machine-learning-pipelines (Citato a pagina 11)

[23] S. Schelter, D. Lange, P. Schmidt, M. Celikel, F. Biessmann, and A. Grafberger, "Automating large-scale data quality verification," *Proceedings of the VLDB Endowment*, vol. 11, no. 12, pp. 1781–1794, 2018. (Citato a pagina 11)

[24] E. Caveness, P. S. GC, Z. Peng, N. Polyzotis, S. Roy, and M. Zinkevich, "Tensorflow data validation: Data analysis and validation in continuous ml pipelines," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 2793–2796. (Citato a pagina 11)

[25] A. Swami, S. Vasudevan, and J. Huyn, "Data sentinel: A declarative production-scale data validation platform," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, 2020, pp. 1579–1590. (Citato a pagina 11)

[26] A. Rolland, "Mobydq," 2022. [Online]. Available: https://ubisoft.github.io/mobydq/ (Citato a pagina 11)

[27] N. Gupta, H. Patel, S. Afzal, N. Panwar, R. S. Mittal, S. Guttula, A. Jain, L. Nagalapatti, S. Mehta, S. Hans, P. Lohia, A. Aggarwal, and D. Saha, "Data quality toolkit: Automatic assessment of data quality and remediation for machine learning datasets," 2021. (Citato alle pagine 11 e 12)

[28] S. Afzal, R. C, M. Kesarwani, S. Mehta, and H. Patel, "Data readiness report," 2020. (Citato a pagina 12)

[29] S. Shrivastava, D. Patel, A. Bhamidipaty, W. M. Gifford, S. A. Siegel, V. S. Ganapavarapu, and J. R. Kalagnanam, "Dqa: Scalable, automated and interactive data quality advisor," in *2019 IEEE International Conference on Big Data (Big Data)*, 2019, pp. 2913–2922. (Citato a pagina 12)

[30] "Great expectations." [Online]. Available: https://greatexpectations.io/ (Citato a pagina 12)

[31] S. Krishnan, J. Wang, E. Wu, M. J. Franklin, and K. Goldberg, "Activeclean: interactive data cleaning for statistical modeling," *Proc. VLDB Endow.*, vol. 9, no. 12, p. 948–959, aug 2016. [Online]. Available: https://doi.org/10.14778/2994509.2994514 (Citato a pagina 12)

[32] N. Polyzotis and M. Zaharia, "What can data-centric ai learn from data and ml engineering?" 2021. (Citato a pagina 12)

[33] H. Cao, R. Ma, H. Ren, and S. S. Ge, "Data-defect inspection with kernel-neighbor-density-change outlier factor," *IEEE Transactions on Automation Science and Engineering*, vol. 15, no. 1, pp. 225–238, 2018. (Citato a pagina 15)

[34] T. Le Quy, A. Roy, V. Iosifidis, W. Zhang, and E. Ntoutsi, "A survey on datasets for fairness-aware machine learning," *WIREs Data Mining and Knowledge Discovery*, vol. 12, no. 3, Mar. 2022. [Online]. Available: http://dx.doi.org/10.1002/widm.1452 (Citato alle pagine 15 e 34)

[35] W. Elouataoui, I. El Alaoui, S. el Mendili, and G. Youssef, "An advanced big data quality framework based on weighted metrics," *Big Data and Cognitive Computing*, vol. 13, 12 2022. (Citato a pagina 15)

[36] A. Mumuni and F. Mumuni, "Data augmentation: A comprehensive survey of modern approaches," *Array*, vol. 16, p. 100258, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2590005622000911 (Citato alle pagine 17 e 18)

[37] L. Cui, H. Li, K. Chen, L. Shou, and G. Chen, "Tabular data augmentation for machine learning: Progress and prospects of embracing generative ai," 2024. [Online]. Available: https://arxiv.org/abs/2407.21523 (Citato alle pagine 18, 19, 20, 23, 24 e 47)

[38] L. Xu, M. Skoularidou, A. Cuesta-Infante, and K. Veeramachaneni, "Modeling tabular data using conditional gan," 2019. [Online]. Available: https://arxiv.org/abs/1907.00503 (Citato alle pagine 20, 23 e 24)

[39] N. Patki, R. Wedge, and K. Veeramachaneni, "The synthetic data vault," in *IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, Oct 2016, pp. 399–410. (Citato alle pagine 20, 21, 22, 24 e 46)

[40] Z. Li, H. Zhu, Z. Lu, and M. Yin, "Synthetic data generation with large language models for text classification: Potential and limitations," 2023. [Online]. Available: https://arxiv.org/abs/2310.07849 (Citato alle pagine 24, 29 e 30)

[41] G. Sahu, P. Rodriguez, I. H. Laradji, P. Atighehchian, D. Vazquez, and D. Bahdanau, "Data augmentation for intent classification with off-the-shelf large language models," 2022. [Online]. Available: https://arxiv.org/abs/2204.01959 (Citato alle pagine 24 e 25)

[42] T. Hartvigsen, S. Gabriel, H. Palangi, M. Sap, D. Ray, and E. Kamar, "Toxigen: A large-scale machine-generated dataset for adversarial and implicit hate speech detection," 2022. [Online]. Available: https://arxiv.org/abs/2203.09509 (Citato alle pagine 25 e 26)

[43] K. M. Yoo, D. Park, J. Kang, S.-W. Lee, and W. Park, "Gpt3mix: Leveraging large-scale language models for text augmentation," 2021. [Online]. Available: https://arxiv.org/abs/2104.08826 (Citato a pagina 26)

[44] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," 2018. [Online]. Available: https://arxiv.org/abs/1710.09412 (Citato a pagina 26)

[45] V. Kumar, A. Choudhary, and E. Cho, "Data augmentation using pre-trained transformer models," 2021. [Online]. Available: https://arxiv.org/abs/2003.02245 (Citato alle pagine 26 e 27)

[46] J. Ye, J. Gao, Q. Li, H. Xu, J. Feng, Z. Wu, T. Yu, and L. Kong, "Zerogen: Efficient zero-shot learning via dataset generation," 2022. [Online]. Available: https://arxiv.org/abs/2202.07922 (Citato alle pagine 27 e 28)

[47] J. Gao, R. Pi, Y. Lin, H. Xu, J. Ye, Z. Wu, W. Zhang, X. Liang, Z. Li, and L. Kong, "Self-guided noise-free data generation for efficient zero-shot learning," 2023. [Online]. Available: https://arxiv.org/abs/2205.12679 (Citato a pagina 28)

[48] R. Tang, X. Han, X. Jiang, and X. Hu, "Does synthetic data generation of llms help clinical text mining?" 2023. [Online]. Available: https://arxiv.org/abs/2303.04360 (Citato alle pagine 28 e 29)

[49] Z. Wang, A. W. Yu, O. Firat, and Y. Cao, "Towards zero-label language learning," 2021. [Online]. Available: https://arxiv.org/abs/2109.09193 (Citato a pagina 29)

[50] V. R. Basili, G. Caldiera, and H. D. Rombach, "The goal question metric approach," 1994. [Online]. Available: https://api.semanticscholar.org/CorpusID:13884048 (Citato a pagina 31)

[51] D. Bouget, M. Allan, D. Stoyanov, and P. Jannin, "Vision-based and marker-less surgical tool detection and tracking: a review of the literature," *Medical image analysis*, vol. 35, pp. 633–654, 2017. (Citato alle pagine 33 e 79)

[52] M. Hirzel and M. Feffer, "A suite of fairness datasets for tabular classification," 2023. [Online]. Available: https://arxiv.org/abs/2308.00133 (Citato a pagina 34)

[53] T. Le Quy, A. Roy, V. Iosifidis, W. Zhang, and E. Ntoutsi, "A survey on datasets for fairness-aware machine learning," *WIREs Data Mining and Knowledge Discovery*, vol. 12, no. 3, p. e1452, 2022. [Online]. Available: https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/widm.1452 (Citato a pagina 34)

[54] A. Gholamy, V. Kreinovich, and O. Kosheleva, "Why 70/30 or 80/20 relation between training and testing sets: A pedagogical explanation," *Int. J. Intell. Technol. Appl. Stat*, vol. 11, no. 2, pp. 105–111, 2018. (Citato a pagina 36)

[55] R. J. Urbanowicz, M. Meeker, W. La Cava, R. S. Olson, and J. H. Moore, "Relief-based feature selection: Introduction and review," *Journal of Biomedical Informatics*, vol. 85, pp. 189–203, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1532046418301400 (Citato a pagina 38)

[56] D. Sukhobok, N. Nikolov, and D. Roman, "Tabular data anomaly patterns," 08 2017, pp. 25–34. (Citato a pagina 39)

[57] Y. Zhang, Y. Yuan, and A. C.-C. Yao, "Meta prompting for ai systems," 2024. [Online]. Available: https://arxiv.org/abs/2311.11482 (Citato a pagina 41)

[58] T. A. Olukoga and Y. Feng, "A case study on the classification of lost circulation events during drilling using machine learning techniques on an imbalanced large dataset," *arXiv preprint arXiv:2209.01607*, 2022. (Citato a pagina 50)

[59] Y. Sasaki, "The truth of the f-measure," *Teach Tutor Mater*, 01 2007. (Citato alle pagine 50 e 77)

[60] M. Hossin and S. M.N, "A review on evaluation metrics for data classification evaluations," *International Journal of Data Mining & Knowledge Management Process*, vol. 5, pp. 01–11, 03 2015. (Citato alle pagine 50 e 77)

[61] K. H. Brodersen, C. S. Ong, K. E. Stephan, and J. M. Buhmann, "The balanced accuracy and its posterior distribution," in *2010 20th International Conference on Pattern Recognition*, 2010, pp. 3121–3124. (Citato alle pagine 51 e 77)

[62] K. Takahashi, K. Yamamoto, A. Kuchiba, and T. Koyama, "Confidence interval for micro-averaged f1 and macro-averaged f1 scores," *Applied Intelligence*, vol. 52, 03 2022. (Citato a pagina 51)

[63] I. T. Young, "Proof without prejudice: use of the kolmogorov-smirnov test for the analysis of histograms from flow systems and other sources." *Journal*

of Histochemistry & Cytochemistry, vol. 25, no. 7, pp. 935–941, 1977. (Citato a pagina 52)

[64] G. Di Leo and F. Sardanelli, "Statistical significance: p value, 0.05 threshold, and applications to radiomics—reasons for a conservative approach," *European radiology experimental*, vol. 4, pp. 1–8, 2020. (Citato alle pagine 52 e 77)

Ringraziamenti

In conclusione di questo percorso di laurea magistrale, vorrei ringraziare i Docenti per avermi trasmesso la loro competenza e passione per la materia, ed in particolare il Professore Fabio Palomba per la disponibilità e la dedizione, unitamente al Dottor Gilberto Recupito.

Ringrazio i miei genitori, Sara e Nello, e mio fratello Alessandro, per essere sempre un porto sicuro per me, insieme ai miei nonni e i miei zii.

È doveroso menzionare tutti i colleghi, divenuti amici, che ho potuto conoscere lungo questo percorso, che ringrazio per aver arricchito umanamente questa esperienza di vita.

Infine ringrazio gli amici con cui non ho condiviso i corsi universitari, in particolare Maria Martina, Miriam, Rebecca, Mirko, Emanuele e Fabrizio, per avermi incoraggiato lungo questo percorso.