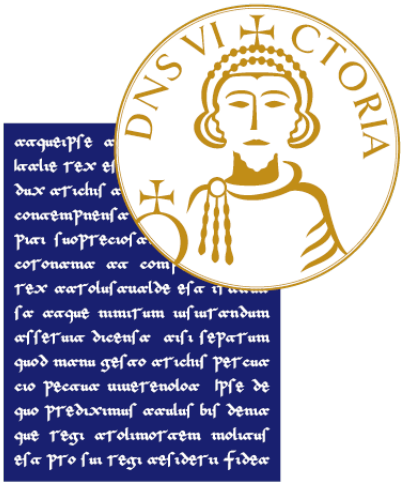
UNIVERSITÀ DEGLI STUDI DEL SANNIO
DIPARTIMENTO DI INGEGNERIA



CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

UNO STUDIO SULLE FASI DI UN MACHINE LEARNING WORKFLOW E UTILIZZO DI LIBRERIE

RELATORE:
CHIAR.MO PROF.
MASSIMILIANO DI PENTA

CORELATORE:
DOTT.SSA
VITTORIA NARDONE

CANDIDATO:
RAFFAELE MIGNONE
399000298

ANNO ACCADEMICO 2020-2021

INDICE

1 INTRODUZIONE 1
- 1.1 Obiettivi della tesi 2
- 1.2 Struttura della tesi 2

2 STATO DELL'ARTE 3
- 2.1 Confronto tra progetti di machine learning e progetti generici 3
- 2.2 Analisi in base al framework utilizzato 4
- 2.3 Analisi dei progetti di machine learning multi-linguaggio 6
- 2.4 Problematiche caratteristiche del machine learning 6
- 2.5 Studio di discussioni Stack Overflow riguardanti il ML 8
- 2.6 Entropia di un cambiamento 9

3 COSTRUZIONE DEL DATASET E METODOLOGIA 11
- 3.1 Research Questions 11
- 3.2 Selezione dei progetti 12
- 3.3 Fetch di issues e commit 12
- 3.4 Classificazione dei dati 13
  - 3.4.1 Classificazione delle issues 13
  - 3.4.2 Classificazione dei commit 15
- 3.5 Metodologia 16
  - 3.5.1 RQ1: come il ML e' distribuito sull'architettura dei progetti? 16
  - 3.5.2 RQ2: come sono distribuiti i bug sulle diverse fasi di ML? 17
  - 3.5.3 RQ3: esiste una differenza di entropy tra ML bug e altri bug? 18
  - 3.5.4 RQ4: come varia il livello di discussione tra ML bug e altri bug? 19
  - 3.5.5 RQ5: come varia il time-to-fix tra ML bug e altri bug? 19

4 RISULTATI 20
- 4.1 RQ1: come il ML e' distribuito sull'architettura dei progetti? 20
- 4.2 RQ2: come sono distribuiti i bug sulle diverse fasi di ML? 22
- 4.3 RQ3: esiste una differenza di entropy tra ML bug e altri bug? 22
- 4.4 RQ4: come varia il livello di discussione tra ML bug e altri bug? 23

4.5 RQ5: come varia il time-to-fix tra ML bug e altri bug? 25
4.6 Threats to validity 27
5 CONCLUSIONI 28
5.1 Sviluppi futuri 29
Bibliografia 30

LISTA DELLE FIGURE

Figura 3.1 Risultati della classificazione manuale delle issues 15
Figura 3.2 Risultato della classificazione dei commit 16
Figura 4.1 Percentuale di files e directories modificate in base al tipo di cambiamento 20
Figura 4.2 Percentuale di file che utilizzano librerie di ML 21
Figura 4.3 Istanze dei fix in base alla fase 22
Figura 4.4 Entropia in base al tipo di fix 23
Figura 4.5 Livello di discussione in base al tipo 24
Figura 4.6 Giorni necessari per il fix 26

ACRONIMI

AI	Artificial Intelligence
API	Application Program Interface
CERN	European Council for Nuclear Research
DL	Deep Learning
GPU	Graphics Processing Unit
ML	Machine Learning
NLP	Natural Language Processing
PR	Pull Request
RQ	Research Question
SATD	Self-Admitted Technical Debt
SO	Stack Overflow
VPS	Virtual Private Server

1

INTRODUZIONE

La storia dell'industria dello sviluppo software è caratterizzata da diversi cambiamenti rispetto alle applicazioni dominanti. Negli anni ottanta il paradigma dominante era quello dei personal computer, poi abbiamo avuto Internet a cui è seguita la nascita del Web al European Council for Nuclear Research (CERN). Nel 2007 con l'annuncio del primo iPhone è inizia l'era del *mobile computing* a cui è seguita quella del *cloud computing*. Negli ultimi anni l'industria non è stata a guardare, ma ha dato vita a sempre più prodotti che fanno uso di Artificial Intelligence (AI) e Machine Learning (ML). Gli strumenti e i software che fanno uso di queste tecnologie sono ormai parte della nostra vita quotidiana e pervadono i campi più disparati. Tra questi sicuramente possiamo annoverare: riconoscimento di immagini, diagnosi di malattie, Natural Language Processing (NLP), guida autonoma e riconoscimento vocale.

La crescente produzione di software basato sul ML ha generato un forte impulso anche per quanto riguarda la ricerca. L'attenzione non è stata puntata unicamente sullo studio di nuovi modelli e architetture, ma anche sul processo di sviluppo di questi prodotti per andare a valutare i vari problemi da un punto di vista ingegneristico. In letteratura non mancano studi atti ad evidenziare le differenze tra progetti di ML e progetti classici [1], né tanto meno confronti dei progetti rispetto alle dipendenze e alle librerie utilizzate [2].

Molti studi sono, invece, incentrati sulle problematiche legate allo sviluppo di applicazioni di ML. In alcuni casi l'analisi è stata svolta per librerie specifiche [3], in altri casi il focus è stato puntato sulle discussioni di Stack Overflow (SO) [4], [5]. In altri casi ancora l'attenzione è stata rivolta su problematiche specifiche come quella del Self-Admitted Technical Debt (SATD) [6].

Anche il seguente lavoro si concentra sui difetti riscontrati all'interno delle applicazioni di ML. In questo caso però la ricerca di differenze è legata agli interventi di *issue fixing* relativi al ML rispetto ad interventi di correzione generici.

1.1 OBIETTIVI DELLA TESI

Questo studio vuole verificare la presenza di differenze, all'interno di progetti di ML, rispetto a come sono trattate le *issues* legate a tematiche di ML e quelle generiche. In particolare si vuole capire come la risoluzione di queste problematiche va ad impattare sull'architettura, sia in termini di moduli modificati sia in termini di entropia generata. Si vuole anche scoprire se sono presenti delle fasi del processo di sviluppo che sono più critiche di altre. Infine si vuole capire se le *issues* sono trattate tutte allo stesso modo per quanto riguarda il livello di discussione e il tempo necessario alla loro risoluzione.

1.2 STRUTTURA DELLA TESI

Nella sezione 2 viene svolta una panoramica sullo stato dell'arte. Nella sezione 3 vengono presentate le Research Question (RQ), viene descritta la procedura utilizzata per la raccolta dei commit e delle issues e come queste sono state classificate. Inoltre viene illustrata la metodologia di analisi impiegata per lo studio di ogni *RQ*. I risultati delle analisi e una discussione qualitativa su alcuni *casi estremi* sono riportati nella sezione 4. Infine la sezione 5 chiude questa tesi.

2

STATO DELL'ARTE

In questo capitolo verranno presentati diversi lavori di ricerca alla base di questo studio. I lavori, sebbene tutti incentrati sul ML, vanno ad approfondire diversi aspetti. In alcuni casi l'attenzione principale è rivolta alle difficoltà e alle problematiche riscontrate dagli sviluppatori. In altri casi viene svolto un confronto tra progetti di ML e progetti generici o tra progetti che fanno uso di diversi framework di ML. Infine viene anche presentato un lavoro sulla complessità del processo di cambiamento del software e su i suoi effetti sull'introduzione di difetti.

2.1 CONFRONTO TRA PROGETTI DI MACHINE LEARNING E PROGETTI GENERICI

Nello studio di Gonzalez *et al.* [1] ci vengono presentate le principali differenze tra i repository di ML e i progetti classici. I dati per lo studio sono stati recuperati attraverso l'Application Program Interface (API) messa a disposizione di GitHub attraverso la quale è stato possible collezionare i dati associati a 9325 progetti open source così raggruppati:

- 5224 progetti legati all'AI e al ML divisi a loro volta in:
  - 700 tools e framework
  - 4524 applicazioni
- 4101 progetti generici

Gli aspetti considerati dallo studio sono molteplici e di varia natura. Una prima analisi è stata condotta rispetto alla nascita dei vari repositories. In questo modo è stato possibile individuare nel 2017 l'anno del *boom* dei repositori di AI & ML. Infatti questo è stato il primo anno in cui sono stati creati più progetti legati al ML rispetto a quelli generici.

Una seconda analisi ha permesso di capire come varia la partecipazione ai vari progetti. Per poter svolgere questa analisi i contributori sono stati divisi in:

- *esterni*: i loro contributi sono limitati ad aprire *issues* e commentare le discussioni.

- *interni*: oltre a svolgere i compiti precedentemente elencati devono anche aver chiuso delle issues o eseguito dei commit sul progetto.

In base a questa divisione si è visto come i tools di ML hanno un numero di contributori interni superiore rispetto ai progetti generici. Quest'ultimi però hanno una maggiore partecipazione esterna. Se invece l'analisi viene svolta considerando unicamente gli autori dei commit si scopre che i progetti generici mediamente hanno più *contributors*, ma i top 4 repositories con più committer sono tutti legati al mondo del ML.

Un'ulteriore analisi è stata svolta anche per quanto riguarda il linguaggio con cui sono stati realizzati i vari progetti. Sia nel caso delle applicazioni che nei tools di ML il linguaggio più popolare è Python, mentre la seconda posizione varia. Nel caso dei tools questa è occupata da C++, mentre nelle applicazioni dai Notebook Jupyter. Nei progetti generici invece Python occupa solo la terza posizione in quanto a popolarità e le prime due sono occupate da JavaScript e Java.

2.2 ANALISI IN BASE AL FRAMEWORK UTILIZZATO

Nello studio di Han *et al.* [2] sono stati considerati 1150 progetti GitHub così distribuiti:

- 708 progetti che dipendono da `TensorFlow`.
- 339 progetti che dipendono da `PyTorch`.
- 103 progetti che dipendono da `Theano`.

Per poter classificare i progetti manualmente gli autori hanno considerato il nome del progetto, la descrizione, le label e il contenuto del readme. La classificazione è avvenuta sia rispetto all'obiettivo del progetto sia rispetto al dominio applicativo. Come obiettivi dei progetti sono stati considerati:

- *Competitions*: progetti realizzati per la partecipazione a delle competizioni o sfide.
- *Learning & Teaching*: progetti realizzati per libri e/o tutorial o per esercitarsi.
- *Paper Experiments*: progetti realizzati al fine di ricerca.
- *Software Development*: comprende librerie, plug-in, tools ecc.a
- *Other*

La classifica delle librerie più utilizzate è rimasta sostanzialmente invariata per tutte le categorie; il primo posto è occupato da `TensorFlow` seguito da `PyTorch` e `Theano`. L'unica eccezione riguarda i progetti realizzati a fini di ricerca. In questo caso `TensorFlow` e `PyTorch` sono

in posizioni invertite. Anche per quanto riguarda la classificazione rispetto al dominio applicativo la situazione è costante. Infatti, indipendentemente dalla libreria utilizzata, i progetti più frequenti sono quelli che hanno a che fare con video e immagini e con il NLP.

Un'ulteriore RQ è andata a valutare il tipo di dipendenza, facendo distinzione tra dipendenze dirette e indirette. Per tutte è tre le librerie si è visto che è più probabile avere una dipendenza diretta che indiretta. `PyTorch` è la libreria che più frequentemente è importata direttamente, mentre `Theano` ha una probabilità di essere importata direttamente quasi uguale a quella di essere importata indirettamente.

Un'ulteriore analisi è stata condotta per individuare quanto frequentemente i progetti aggiornano le loro dipendenze o eseguono dei downgrade. In questo caso si è visto che i progetti basati su `TensorFlow` e `PyTorch` aggiornano le proprie dipendenze molto più frequentemente rispetto ai progetti basati su `Theano`. Mentre il tasso di downgrade è sostanzialmente equivalente. Nel caso dei progetti che dipendono da `TensorFlow` la maggior parte dei downgrade viene spiegata dalla volontà di non utilizzare la nuova API introdotta nella versione 2.0 della libreria. Sempre analizzando la versione della libreria utilizzata si è visto che i progetti basati su `Theano` sono quelli che utilizzano più frequentemente l'ultima versione disponibile della libreria.

In un altro lavoro di Han *et al.* [7] il focus si sposta sugli argomenti di discussione e su come questi variano in base al framework utilizzato. In questo caso all'interno dei dataset non sono rientrati unicamente i dati recuperati da GitHub, ma anche le discussioni su SO.

Questo studio ha permesso di evidenziare differenze e similitudini per quanto riguarda le discussioni che si generano intorno ai tre framework di interesse. In particolare emerge che le fasi più discusse sono quelle di *model training* e di *preliminary preparation*. Mentre la fase meno discussa è quella di *model tuning*. Per quanto riguarda le differenze, dallo studio, emerge che `TensorFlow` e `PyTorch` hanno topic di discussione totalmente confrontabili. Oltre ai topic citati precedentemente, per questi framework, si discute molto anche della *data preparation*. Mentre le discussioni riguardanti `Theano` sono quasi esclusivamente concentrate sul *model training*.

Da questi due studi si evince una forte somiglianza per quanto riguarda `TensorFlow` e `PyThorch`. La principale differenza viene riscontrata per quanto riguarda i campi di applicazione, con `TensorFlow` che viene generalmente preferito fatti salvi gli ambiti di ricerca. Mentre `Theano` presenta molte diversità sia per quanto riguarda gli impieghi che le discussioni.

2.3 ANALISI DEI PROGETTI DI MACHINE LEARNING MULTI-LINGUAGGIO

Lo studio di Grichi *et al.* [8] si concentra sui sistemi *multi-linguaggio*. In questo caso si vuole capire se i sistemi di ML sono più soggetti all'essere realizzati attraverso linguaggi diversi. Inoltre analizzando le Pull Request (PR) realizzate in più linguaggi si vuole capire se queste sono accettate con la stessa frequenza di quelle *mono-linguaggio* e se la presenza di difetti è equivalente.

L'analisi è stata svolta su 27 progetti open source hostati su GitHub. I progetti sono poi stati classificati in tre categorie:

- Cat I: include 10 sistemi di ML *multi-linguaggio*.
- Cat II: include 10 sistemi generici *multi-linguaggio*.
- Cat III: include 7 sistemi di ML *mono-linguaggio*.

Successivamente sono state scaricate le PR di ogni progetto considerato. Le PRs sono state categorizzate per individuare quelle accettate e quelle rifiutate. Inoltre le PR sono state categorizzate anche il base al numero di linguaggi utilizzati. In questo modo è stato possibile individuare le PR *mono-linguaggio* e quelle *multi-linguaggio*. Infine per ogni PR è stato individuato il tempo necessario alla sua accettazione o chiusura e i difetti introdotti dalla PR.

Per quanto riguarda la percentuale di linguaggi di programmazione utilizzati i progetti della categoria I e II sono comparabili. La principale differenza riguarda i tipi di linguaggi utilizzati. Nel caso dei progetti *multi-linguaggio* di ML l'accoppiata più comune è Python e C/C++. Mentre nel caso dei progetti generici la coppia più comune è data da Java e C/C++. I progetti della categoria I e II sono paragonabili anche rispetto al numero di PRs e PRs *multi-linguaggio*.

Lo studio ha evidenziato come all'interno dei progetti di ML le PR *mono-linguaggio* sono accettate molto più facilmente rispetto a quelle *multi-linguaggio*. Inoltre anche nel caso in cui queste vengano accettate, il tempo necessario alla loro accettazione è maggiore. Infine si è visto anche che rispetto alle PRs *multi-linguaggio* non esistono differenze in base all'introduzione di *bug* tra i progetti della categoria I e II. Mentre le PR che includono un singolo linguaggio sembrano essere più affette da *bug* nel caso dei sistemi di ML.

2.4 PROBLEMATICHE CARATTERISTICHE DEL MACHINE LEARNING

In letteratura sono presenti anche lavori che si concentrano sull'analisi delle problematiche e dei *bug* riscontrati all'interno di applicazioni di

ML. Nello studio di Zhang *et al.* [3] l'attenzione è rivolta unicamente alle problematiche correlate a `TensorFlow`. Per lo studio sono stati recuperati dei *bug* di `TensorFlow` sia da progetti su GitHub (88 elementi) sia da quesiti su SO (87 elementi).

Gli autori dello studio, per poter individuare la causa dei *bug* e i loro sintomi hanno dovuto analizzare manualmente gli elementi del dataset. Nel caso di *bug* discussi su SO le informazioni sono state recuperate dalla discussione. Mentre nel caso dei *bug* recuperati da GitHub le informazioni sono state recuperate tramite lo studio dell'intervento di *fix* e il messaggio associato ad esso.

In questo modo è stato possibile individuare tre sintomi:

- *Error*: durante l'esecuzione viene sollevato un errore riconducibile a `TensorFlow`.
- *Low Effectiveness*: il programma presenta dei valori di *accuracy*, *loss* ecc. estremamente scadenti.
- *Low Efficiency*: il programma viene eseguito troppo lentamente.

Per quanto riguarda le cause è stato possibile individuarne sei:

- *Incorrect Model Parameter or Structure*: il *bug* è riconducibile ad un cattivo utilizzo dei parametri del modello o alla sua struttura.
- *Unaligned Tensor*: si verifica ogni qual volta la *shape* dell'input non corrisponde con quella attesa.
- *Confusion with TensorFlow Computation Model*: in questo caso i *bug* si verificano quando gli utenti non sono pratici del modello computazionale utilizzato da `TensorFlow`.
- *TensorFlow API Change*: il *bug* dipende da un cambiamento nell'API di `TensorFlow`.
- *TensorFlow API Misuse*: in questo caso il *bug* è riconducibile ad un utilizzo scorretto dell'API di `TensorFlow`.
- *Structure Inefficiency*: questa categoria può essere vista come una versione più *soft* della prima categoria. Infatti in questo caso il problema nella struttura non genera un errore ma solo delle inefficienze.

Anche lo studio di Humbatova *et al.* [9] ha come obbiettivo l'analisi delle problematiche legate al ML. In questo caso però la visione è più ampia e non si limita ad una singola libreria. Inoltre in questo caso lo scopo ultimo del lavoro è la costruzione di una tassonomia per le problematiche di ML.

Anche in questo caso i dati sono stati recuperati sia da SO che da GitHub. Inoltre per questo studio è stata anche svolta un'intervista a 20 persone tra ricercatori e sviluppatori nel campo del ML. Partendo da questi dati è stata costruita una tassonomia attraverso un approccio

bottom-up. La tassonomia si compone di 5 categorie *top-level*, 3 delle quali sono state divise in sotto categorie.

Tra le categorie di primo livello ci sono:

- *Model*: in questa categoria rientrano tutte le problematiche che riguardano la struttura e le proprietà del modello.
- *Tensors & Inputs*: rientrano in questa categoria tutti i problemi rispetto alla *shape* dei dati e al loro formato.
- *Training*: questa categoria è la più ampia della tassonomia. Rientrano in questa categoria la qualità e il preprocessing dei dati utilizzati per l'apprendimento, il *tuning* degli *hyperparametri*, la scelta della funzione di perdita più appropriata ecc.
- *GPU Usage*: in questa categoria ricadono tutti i problemi nell'uso della Graphics Processing Unit (GPU).
- *API*: rientrano in questa categoria tutti i problemi generati da un non corretto utilizzo dell'API del framework di ML.

Come si può notare, fatta salva la specificità del primo lavoro, esiste una forte similitudine tra le categorie di problemi individuate dai due studi.

2.5 STUDIO DI DISCUSSIONI STACK OVERFLOW RIGUARDANTI IL ML

Nello studio di Bangash *et al.* [10] viene svolta un'analisi degli argomenti di ML discussi più frequentemente dagli sviluppatori. In questo caso, a differenza dello studio di Han *et al.* [7] discusso precedentemente, non viene svolta alcuna distinzione in base alla libreria utilizzata. Inoltre questo studio utilizza unicamente informazioni recuperate da SO, mentre l'altro lavoro univa le domande di SO alla discussione generata all'interno dei repositories di GitHub.

In questo caso il topic più frequentemente discusso riguarda la presenza di errori all'interno del codice. Seguono discussioni rispetto agli algoritmi di apprendimento e al training dei dati. Lo studio ha evidenziato anche come molte discussioni riguardano librerie e framework di ML come ad esempio `numpy`, `pandas`, `keras`, `Scikit-Learn`, ecc. Tutte queste discussioni sono state inserite nel topic *framework*.

Anche nel lavoro di Alshangiti *et al.* [11] vengono analizzate le domande presenti sulla piattaforma SO. In questo caso però oltre ad un analisi qualitativa rispetto al contenuto di queste discussioni è stata eseguita anche un'analisi comparativa tra le discussioni inerenti al ML e le altre.

Per svolgere questa analisi gli autori sono partiti dal dump del database di SO e hanno individuato tre campioni:

- *Quantitative Study Sample*: si compone di 86983 domande inerenti al ML, con le relative risposte. L'individuazione dei post è avvenuta attraverso la definizione di una lista contente 50 tag utilizzate su SO per le domande di ML.
- *Qualitative Study Sample*: contiene 684 post realizzati da 50 utenti. Questo campione è stato ottenuto eseguendo un ulteriore campionamento sul campione discusso al punto precedente.
- *Baseline Sample*: si compone di post che non riguardano il ML. Questo campione viene utilizzato per comparare le domande di ML con quelle generiche.

La prima *RQ* dello studio vuole verificare se rispondere ad una domanda inerente al ML sia più complicato. Per valutare la complessità di risposta sono state contate le domande che non presentano alcuna risposta, le domande che non presentano risposte accettate e la mediana del tempo necessario affinché una domanda abbia una risposta accettata. Dal confronto tra il primo e il terzo sample rispetto a queste metriche è emerso che i post inerenti al ML hanno una maggiore probabilità di non avere risposte/risposte accettate. Inoltre si è visto come mediamente le domande di ML necessitano di un tempo dieci volte maggiore per poter avere una risposta accettata. Una spiegazione a questo fenomeno ci viene fornita dalla seconda *RQ* in cui viene evidenziato che all'interno della community di SO c'è una carenza di esperti di ML.[1]

Lo studio è stato in grado anche di individuare le fasi in cui gli sviluppatori riscontrano maggiori problematiche. In generale le maggiori difficoltà sono state riscontrate nel *preprocessing dei dati*, nella configurazione dell'ambiente di sviluppo e nel deployment del modello. Per quanto riguarda i task specifici del Deep Learning (DL) le maggiori problematiche riguardano applicazioni di NLP e riconoscimento degli oggetti. Infine lo studio ha mostrato come, nonostante la vasta adozione, molti utenti riscontrano problemi nell'utilizzo dell'API di `TensorFlow`.

2.6 ENTROPIA DI UN CAMBIAMENTO

Nello studio di Hassan [4] si vuole capire in che modo la complessità del processo del cambiamento del software vada ad impattare

1 L'individuazione degli esperti è avvenuta secondo l'approccio *ExpertiseRank*. Questo approccio crea un grafo diretto, in cui gli utenti sono rappresentati dai nodi e gli archi rappresentano una relazione di aiuto, attraverso il quale è possibile determinare l'esperienza degli utenti. Per esempio considerando un caso in cui l'utente B ha aiutato l'utente A avremo che l'esperienza di B è superiore a quella di A. Se l'utente C risponde ad una domanda di B, allora questo avrà una esperienza superiore sia ad A che a B, in quanto è stato in grado di aiutare un utente (B) che aveva dimostrato a sua volta di essere esperto (rispondendo ad A).

sull'introduzione di difetti all'interno della codebase. Per valutare la complessità del processo di cambiamento è stato *preso in prestito* il concetto di entropia [5] utilizzato nella teoria della comunicazione.

Lo studio è stato condotto su sei progetti open source di grandi dimensioni. Attraverso i sistemi di *version control* e all'analisi lessicale dei messaggi di cambiamento sono stati individuate tre tipologie di cambiamento.

- *Fault Repairing modification*: include i cambiamenti attuati per risolvere un difetto nel prodotto software. Questa categoria di modifiche non è stata utilizzata per il calcolo dell'entropia, ma per validare lo studio.
- *General Maintenance modification*: include cambiamenti di mantenimento che non vanno ad influenzare il comportamento del codice. Rientrano in questa categoria la re-indentazione del codice, cambiamenti alla nota del copyright ecc. Questi cambiamenti sono stati esclusi dallo studio.
- *Feature Introduction modification*: include tutti i cambiamenti che vanno ad alterare il comportamento del codice. Questi cambiamenti sono stati individuati per esclusione e sono stati utilizzati per il calcolo dell'entropia.

All'interno dello studio vengono definiti tre modelli che permettono di calcolare la complessità del processo di cambiamento software.

- *Basic Code Change model*: è il primo modello presentato, assume un periodo costante per il calcolo dell'entropia e considera costante il numero di file presenti all'interno del progetto.
- *Extend Code Change model*: è un'evoluzione del modello di base che lo rende più flessibile.
- *File Code Change model*: i modelli illustrati precedentemente forniscono un valore complessivo di entropia per l'intero progetto. Questo modello permette di valutare l'entropia in modo distinto per ogni file.

Lo studio ha dimostrato che nel caso di sistemi di grandi dimensioni, la complessità del processo di cambiamento è in grado di predire l'occorrenza di fault. Inoltre viene anche mostrato come la predizione basata sulla complessità del processo sia più precisa rispetto alla predizione basata sulla complessità del codice.

3

COSTRUZIONE DEL DATASET E METODOLOGIA

L'obiettivo di questa tesi è verificare la presenza di differenza all'interno di progetti di ML rispetto a come sono trattati gli interventi di *issue fixing* legati al ML e quelli generici. L'attenzione è rivolta all'impatto degli interventi sull'architettura del sistema, alle tempistiche necessarie alla risoluzione e al livello di discussione di questi difetti. Inoltre si vuole anche capire se esistono delle fasi del processo di sviluppo che sono più critiche di altre.

3.1 RESEARCH QUESTIONS

Gli obiettivi di questa tesi sono stati racchiusi in cinque RQ di seguito elencate.

- **RQ1**: *come il ML e' distribuito sull'architettura dei progetti?*

  In questa *RQ* si vuole investigare l'architettura dei progetti. In particolare l'attenzione viene concentrata sui files e sulle directories modificate durante interventi di *issues fixing*. Obiettivo di questa domanda è anche individuare la percentuale di files che utilizzano import riconducibili a librerie e framework di ML.

- **RQ2**: *come sono distribuiti i bug sulle diverse fasi di ML?*

  Il workflow tipico per lo sviluppo di un'applicazione di ML si compone di più fasi. L'obiettivo di questa *RQ* è quello di individuare le fasi più critiche per quanto riguarda l'introduzione di difetti all'interno del prodotto software.

- **RQ3**: *esiste una differenza di entropy tra ML bug e altri bug?*

  A partire dai lavori precedenti svolti sull'entropia di un cambiamento, si vuole capire se esiste una differenza in termini di entropia generata tra le correzioni dei difetti ascrivibili al ML e gli altri difetti.

- **RQ4**: *come varia il livello di discussione tra ML bug e altri bug?*

  Questa *RQ* riguarda il livello di discussione dei *bug*. In particolare si vuole capire se, all'interno dei progetti di ML, i bug generici sono discussi con lo stesso livello di approfondimento di quelli specifici del ML.

- **RQ5**: *come varia il time-to-fix tra ML bug e altri bug?*

  Un altro aspetto caratteristico di un *fix* è il tempo necessario per poter essere attuato. Questa *RQ* ha lo scopo di verificare l'esistenza di differenze tra i *bug* generici e quelli di ML.

3.2 SELEZIONE DEI PROGETTI

L'individuazione dei progetti da analizzare è avvenuta mediate l'ausilio dell'API messa a disposizione da GitHub. In particolare è stata eseguita una query per ottenere una lista di repository che fanno uso di librerie e framework di ML come `TensorFlow`, `Pytorch` e `scikit-learn`. In questo modo è stato possibile ottenere una lista di 26758 repository che è stata successivamente filtrata per individuare solo i progetti d'interesse per il seguente studio.

L'operazione di filtraggio è avvenuta attraverso due fasi; una prima automatica e una seconda manuale. La prima fase ha avuto l'obiettivo di selezionare unicamente i repository *popolari*. Nella maggior parte dei casi viene utilizzato il numero di stelle come indice della popolarità di un progetto [12], ma per questo lavoro si è preferito dare maggiore rilevanza ad altri aspetti, come il numero di fork, il numero di *contributors* e il numero di issues chiuse. Questa scelta è stata dettata dall'esigenza di selezionare non solo repository popolari, ma anche caratterizzati da una forte partecipazione della community.

I progetti che hanno superato questa prima selezione dovevano:

- essere lavori originali, per cui sono stati esclusi tutti i fork.
- avere almeno cento issues chiuse.
- avere almeno dieci contributors.
- avere almeno venticinque fork.

Alla fine di questa prima selezione il numero di repository si è ridotto a sessantasei e sono stati analizzati manualmente per rimuovere listati associati a libri e/o tutorial, progetti non in lingua inglese e librerie. Alla fine di questa seconda fase il numero di progetti è sceso a trenta.

3.3 FETCH DI ISSUES E COMMIT

Una volta individuati i progetti da analizzare si è reso necessario recuperare l'intera storia dei progetti e le issues ad essi associate. Per entrambe le operazioni è stato utilizzato il tool *perceval* [13]. Nel caso delle issues, essendo queste informazioni non direttamente contenute all'interno del repository `git`, è stato necessario utilizzare nuovamente l'API di GitHub. Poiché le chiamate associate ad un singolo *token*

sono limitate nel tempo si è scelto di configurare *perseval* in modo tale da introdurre in automatico un ritardo ogni qualvolta veniva raggiunto il limite. Inoltre il codice è stato dispiegato su un Virtual Private Server (VPS) in modo da poter eseguire il fetch senza che fosse necessario mantenere attiva una macchina fisica.

Con il processo precedentemente illustrato è stato possibile recuperare:

- 34180 commit.
- 15267 tra issues e pull request.

3.4 CLASSIFICAZIONE DEI DATI

3.4.1 *Classificazione delle issues*

Al fine di poter eseguire un confronto tra i *fix* di ML e quelli *generici* è stato necessario classificare sia le issues che i commit. Il numero elevato di elementi non rende praticabile una classificazione manuale per cui si è optato per una classificazione automatica. Per quanto riguarda i primi si è scelto di attuare una classificazione basata sul testo, in particolare considerando il titolo e il corpo della issue, ma escludendo i commenti di risposta in modo da non rendere i dati troppo rumorosi. A tal fine sono stati implementati ed analizzati due classificatori, uno supervisionato e uno non supervisionato.

I due modelli considerati sono:

- un classificatore statico basato su una lista di vocaboli tipici del ML.
- un modello *naïve Bayes* [14], [15].

La classificazione mediante il classificatore statico non necessita di un *labeling* manuale dei dati, ma richiede la definizione dei vocaboli tipici del ML. La lista dei termini caratteristici del ML non è stata costruita da zero, ma è basata sul lavoro di Humbatova *et al.* [9]. In questo modo tutte le issues che utilizzavano almeno un vocabolo tipico del ML sono state classificate come issues di ML.

Nel caso del modello *naïve Bayes*, essendo questo un algoritmo di apprendimento supervisionato, si è resa necessaria una classificazione manuale delle issues. A tal scopo è stato eseguito un campionamento stratificato in base al progetto di provenienza di 376 issues che sono state divise tra due lettori e labellate. La label delle *issues* è stata determinata andando ad analizzare il titolo, il corpo e i commenti associati alla *issue*. Durante il labeling si è scelto di classificare ulteriormente le issue di ML al fine di individuare anche la fase in cui il problema

si è palesato. La definizione delle varie fasi è avvenuta partendo dal lavoro di Amershi *et al.* [16] realizzato nei laboratori di *Microsoft*.

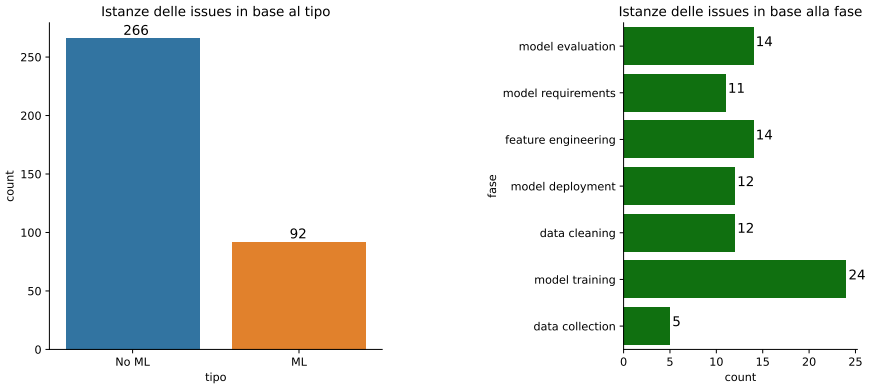
Le fasi considerate sono:

- *Model Requirements*: questa fase comprende tutte le discussioni rispetto all'individuazione del modello più adatto, le funzionalità che questo deve esporre e come adattare un modello esistente per eseguire una diversa funzionalità.
- *Data Collection*: comprende le operazioni volte alla definizione di un dataset. Rientrano in questa fase sia la ricerca di dataset già esistenti che la costruzione di nuovi dataset.
- *Data Labeling*: questa fase si rende necessaria ogni qual volta si opera con modelli basati su apprendimento supervisionato.
- *Data cleaning*: in questa fase non rientrano soltanto le operazioni strettamente di pulizia dei dati come ad esempio rimozione di record rumorosi o incompleti, ma tutte le trasformazioni eseguite sui dati, quindi anche operazioni di standardizzazione, flip di immagini ecc.
- *Feature Engineering*: questa fase serve per identificare le trasformazioni da attuare sui dati e le migliori configurazioni degli *hyperparametri* al fine di migliorare il modello.
- *Model Training*: questa fase racchiude il training vero e proprio del modello.
- *Model Evaluation*: in questa fase vengono valutate le performance del modello utilizzando metriche standard come *precision* e *recall*, ma anche andando a confrontare i risultati ottenuti rispetto a quelli generati da altri modelli o rispetto all'esperienza[1].
- *Model Deployment*: questa fase riguarda il dispiegamento del modello sul dispositivo target.
- *Model Monitoring*: una volta dispiegato il modello deve essere continuamente monitorato al fine di assicurasi un corretto comportamento anche sui dati reali.

A partire dal dataset *labellato* è stato possibile costruire un training e un test set, mediante i quali è stato possibile allenare e valutare le performance del modello bayesiano. Mentre le performance del primo modello sono state valutate sull'intero dataset.

Al fine di poter confrontare i due modelli sono state utilizzate le metriche di *precision* e *recall*. Com'è possibile notare dai valori riportati in tbl. 3.1, il modello basato sulla lista di vocaboli è leggermente più preciso del modello bayesiano, ma presenta una *recall* decisamente più bassa. Dalla fig. 3.1a si evince la natura minoritaria delle issues di ML rispetto alle issues generiche, per questo motivo si è preferito il

1 Non sempre è possibile valutare un modello in modo oggettivo, ci sono determinati contesti, come ad esempio la generazione di *deep fakes*, in cui è comunque necessaria una valutazione umana per determinare la qualità del risultato.



(a) Numero di issues rispetto al tipo

(b) Numero di issues rispetto alla fase

Figura 3.1: Risultati della classificazione manuale delle issues

modello naïve Bayes in modo da perdere quante meno istanze possibili anche a costo di sacrificare leggermente la precisione.

Tabella 3.1: Confronto dei due modelli per la classificazione delle issues.

	Classificatore statico	naïve Bayes
precision	0.46	0.41
recall	0.74	0.94

3.4.2 *Classificazione dei commit*

Prima di poter classificare i commit si è reso necessaria un'ulteriore fase di filtraggio in modo da poter separare i commit di *issue fixing* da quelli generici. Sono stati considerati come commit di *fix* tutti quei commit al cui interno veniva fatto riferimento a delle *issues* attraverso la notazione *"#"*. Questa operazione ha ridotto il dataset dei commit a 3321 unità la cui distribuzione in base al tipo è riportata in fig. 3.2.

Da ogni commit sono state estratte le informazioni rilevanti per le analisi. In particolare è stato conservato:

- Il progetto di appartenenza.
- L'hash del commit.
- La data del commit.
- L'autore del commit.
- La lista dei files modificati.
- Le linee modificate.
- La lista delle *issues* citate.

A questo punto è stato possibile separare i *fix* di ML da quelli generici. La classificazione è avvenuta attraverso la lista delle issues citate all'interno del *commit message* e sono stati considerati come commit di ML tutti quei commit che facevano riferimento ad almeno una issue di ML.

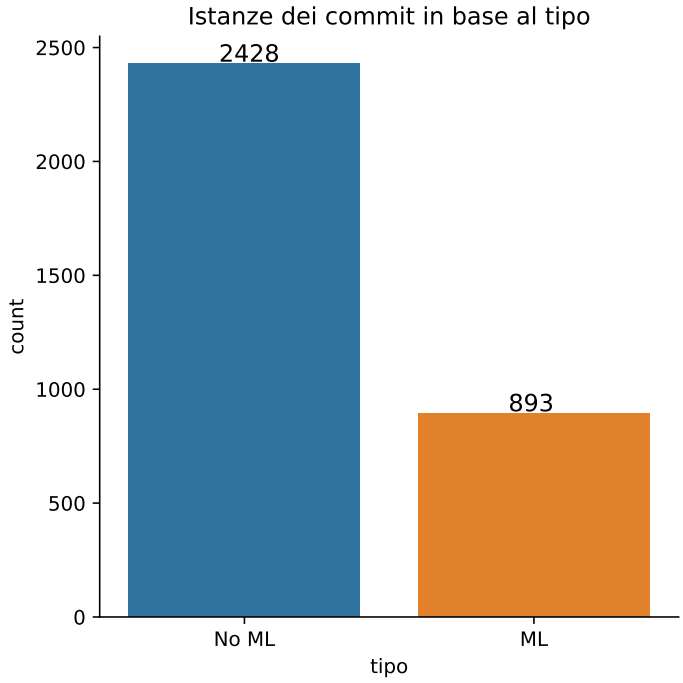


Figura 3.2: Risultato della classificazione dei commit

3.5 METODOLOGIA

3.5.1 *RQ1: come il ML e' distribuito sull'architettura dei progetti?*

In questa prima domanda si vuole andare a capire quant'è ampia la *superficie* del progetto che viene modificata durante gli interventi di *fix*, facendo distinzione tra le correzioni che riguardano il ML e quelle generiche. Inoltre si vuole anche capire quanti file importano librerie tipiche del ML.

Per poter svolgere la prima analisi è stato necessario individuare il numero totale di file modificati per *fix* generici e per i *fix* specifici del ML. A tal fine i commit sono stati raggruppati rispetto al progetto e al tipo di cambiamento (ML, no ML). All'interno di ogni raggruppamento si è eseguita la concatenazione della lista dei file modificati. Poiché non si è interessati al numero di modifiche che ha subito ogni file le liste sono state trasformate in insiemi per eliminare le ripetizioni. Come output di questa fase si è ottenuto per ogni progetto:

- l'insieme dei file modificati per *fix* di ML
- l'insieme dei file modificati per fix generici

Infine eseguendo l'union set tra questi due insiemi si è ottenuto l'insieme totale dei files modificati durante i *fix*. A questo punto per ogni

progetto si è calcolata la percentuale di file modificati durante interventi di *fix* di ML (`ml_file_ratio`) e la percentuale di file modificati durante *fix* generici (`no_ml_file_ratio`).

Attraverso la funzione di libreria Python `os.path.dirname` sono stati ottenuti i tre insiemi sopra citati anche per quanto riguarda le directories. E in modo analogo si è calcolata la percentuale di directories modificate durante interventi di ML (`ml_dirs_ratio`) e interventi generici (`no_ml_dirs_ratio`). Queste distribuzioni sono state analizzate graficamente attraverso l'ausilio di boxplot.

Per la seconda analisi si è reso necessario conoscere per ogni file la lista degli import utilizzati. Questa informazione è stata recuperata attraverso uno script, che dato in input un progetto restituisce la lista dei files affiancati dalla lista degli import utilizzati all'interno del file stesso. L'individuazione dei file di ML è avvenuta mediante la definizione di due gruppi di librerie tipiche del ML.

- Gruppo 1: librerie specifiche del ML come ad esempio `keras`, `TensorFlow` e `Pytorch`.
- Gruppo 2: librerie utilizzate in ambito ML, ma anche in altri contesti. Appartengono a questo gruppo librerie come `numpy`, `scipy` e `pandas`.

Ogni file è stato classificato come di ML o meno in base a due livelli. Nel primo caso, indicato con *all*, per rientrare all'interno dei file che fanno uso di librerie di ML bastava importare almeno una libreria contenuta in uno dei due gruppi precedentemente descritti. Mentre nel secondo caso, indicato con *wo_pandas_numpy_scipy*, era necessario importare almeno una libreria presente nel primo gruppo.

Per entrambe le classificazioni si è andato a valutare a quanto ammontava la percentuale di file di ML appartenenti ad ogni progetto. Anche in questo caso le distribuzioni sono state analizzate attraverso l'ausilio di un boxplot.

3.5.2 *RQ2: come sono distribuiti i bug sulle diverse fasi di ML?*

Come illustrato nella sec. 3.4.2 per poter determinare la natura di un *issue fix* si è fatto ricorso alla classificazione delle *issues* ad esso associate. La maggior parte delle *issues* è stata classificata automaticamente, ma è stato comunque necessario classificarne una porzione in modo manuale per poter avere un train/test set. Come detto precedentemente, nel caso delle *issues* classificate a mano, oltre all'individuazione della tipologia (ML, non ML) è stata individuata anche la fase in cui il problema si palesava (si veda sec. 3.4.1). In questa *RQ* si vuole andare

a valutare come questo dato aggiuntivo sulle fasi viene *proiettato* sui commit di *fix*.

Per poter svolgere questa analisi è necessario incrociare i dati sui commit di *fix* con la classificazione delle *issues*. A partire dal dataset delle *issues* è stato creato per ogni progetto un dizionario *issue* $\rightarrow$ fase. Quindi per ogni commit si è individuata la fase attraverso questo dizionario ausiliario.

In particolare un commit poteva citare:

- nessuna *issues* inclusa nel dizionario. In questo caso non è possibile individuare la fase del commit.
- una *issues* presente nel dizionario. In questo caso al commit viene assegnata la fase della *issue*.
- più di una *issues* presente nel dizionario. In questo caso al commit venivano associate più fasi[2].

L'analisi quantitativa è avvenuta attraverso un barplot in cui venivano riportati unicamente i commit a cui è stato possibile assegnare almeno una fase.

3.5.3 *RQ3: esiste una differenza di entropy tra ML bug e altri bug?*

La successiva analisi aveva lo scopo di verificare l'esistenza di una differenza tra l'entropia del *fix* rispetto alla natura di questi. Il lavoro di questa analisi è basato sul modello *BCC* discusso nella sec. 2.6. L'analisi è stata svolta sia a livello di file, sia a livello di linee quindi per ogni commit del dataset è stato necessario individuare sia il numero di file che hanno subito delle modifiche, sia il numero di linee alterate, considerando in questo modo sia le aggiunte che le rimozioni. Il dato rispetto alle linee modificate è già presente nel dataset di partenza (si veda sec. 3.4.2), mentre il numero di file modificati può essere ricavato dalla lista dei files modificati nel commit.

Inoltre per poter calcolare la probabilità di un cambiamento è stato necessario conoscere anche il numero totale di file e di linee di ogni progetto. Questi valori sono stati calcolati attraverso la storia `git` del branch `master`[3]. Per ogni commit sono stati individuati i file aggiunti ($+1$) e rimossi (-1) in modo tale da poter calcolare il delta-cambiamento del commit. Eseguendo la somma di questo delta su tutti i commit si è ottenuto il numero totale di file del progetto. In modo analogo si è proceduto anche per quanto riguarda le linee.

2 Nessun commit di *fix* presente nel dataset utilizzato è rientrato in questa categoria.

3 Oltre al branch `master` è stato considerato anche il branch `main` diventato molto comune dopo le proteste del movimento Black Lives Matter e il branch `master-V2` unico branch utilizzato da un progetto.

Le due distribuzioni sono state valutate graficamente attraverso un boxplot. Inoltre sono stati svolti dei test statistici (*ranksum* e *Cliff's delta*) per verificare la rilevanza di queste differenze.

3.5.4 *RQ4: come varia il livello di discussione tra ML bug e altri bug?*

Per rispondere a questa domanda è stato necessario andare a valutare il numero di commenti presenti all'interno di ogni issues. Questo dato non è presente nel dataset dei commit generato inizialmente (si veda sec. 3.4.2), ma può essere ricavato a partire dalla lista delle *issues* citate. Dato un commit si è considerata la lista delle *issues* citate, e per ogni *issue* citata si è calcolato il numero di commenti. Poiché un singolo commit può far riferimento a più *issues* è stato necessario anche calcolare il numero di commenti medi.

Il livello della discussione non viene determinato solo dal numero di commenti, ma anche dalla lunghezza di questi. Quindi per ogni *issue* è stato calcolato anche il numero medio di parole presenti all'interno di un commento.

I dati per entrambe le distribuzioni sono stati valutati graficamente attraverso l'ausilio di un boxplot e attraverso i test statistici illustrati precedentemente.

3.5.5 *RQ5: come varia il time-to-fix tra ML bug e altri bug?*

In quest'ultima analisi si vuole andare a valutare se c'è differenza nel tempo necessario per eseguire il *fix*. Anche in questo caso, per poter rispondere alla domanda, è necessario incrociare i dati dei commit con quelli delle *issues* attraverso la lista delle *issues* citate. Dato una *issue* sono stati individuate la data di apertura e di chiusura. Nel caso in cui ad un commit sono associate più *issues* è stata presa come data di apertura il minimo tra tutte le date di apertura delle *issues* e, in modo analogo, si è proceduto anche per la data di chiusura con la differenza che i dati sono stati aggregati attraverso la funzione `max`.

Una volta noto il momento di apertura e di chiusura della problematica è stato possibile calcolare il numero di giorni intercorsi tra questi due istanti temporali. Le distribuzioni così ottenute sono state analizzate ancora una volta mediante un *boxplot*, il test *ranksum* e il test *Cliff's delta*.

4

RISULTATI

4.1 RQ1: COME IL ML E' DISTRIBUITO SULL'ARCHITETTURA DEI PROGETTI?

Dalla fig. 4.1 si può notare che i cambiamenti generici vanno ad impattare su una superficie maggiore del sistema, sia che l'analisi sia svolta al livello di files che di directories. Un'ulteriore aspetto interessante riguarda la varianza delle distribuzioni, infatti, indipendentemente dalla granularità dell'analisi, il dato riguardante i cambiamenti di ML è caratterizzato da una maggiore varianza.

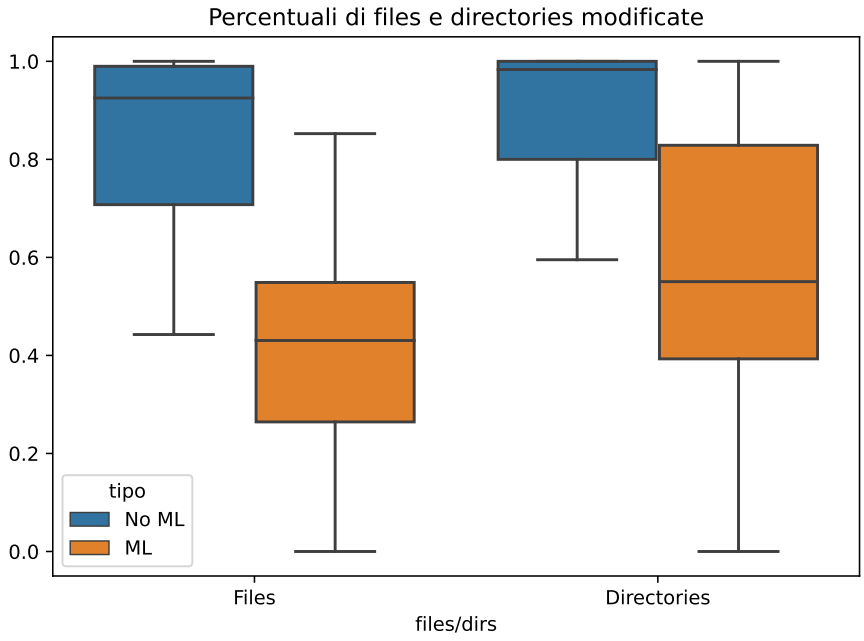


Figura 4.1: Percentuale di files e directories modificate in base al tipo di cambiamento

Nel boxplot in fig. 4.2 sono invece riportati i risultati per quanto riguarda l'utilizzo di import di ML. Si può notare che, indipendentemente dal livello di analisi, la percentuale di file che utilizzano librerie di ML è caratterizzata da una forte varianza. Ciò indica che i progetti inclusi all'interno dello studio sono di varia natura e che alcuni sono più incentrati sul ML rispetto ad altri. Inoltre, considerando l'analisi *strict*, è possibile osservare come solo un 25% dei progetti abbia una percentuale di files di ML superiore al 45%.

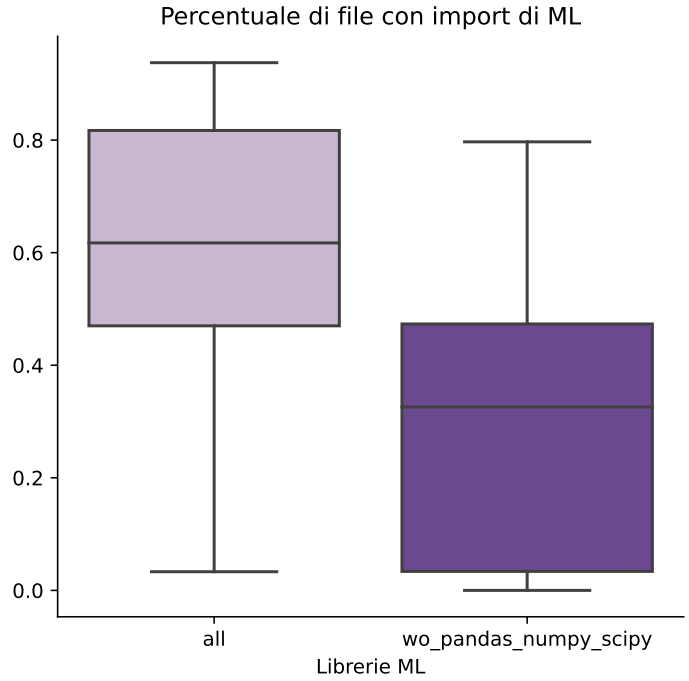


Figura 4.2: Percentuale di file che utilizzano librerie di ML

In relazione all'analisi *wo_pandas_numpy_scipy* sono stati poi analizzati i cinque progetti più ML *intensive* per valutare eventuali caratteristiche comuni rispetto al dominio applicativo. Com'è possibile notare dalla tbl. 4.1 i vari progetti si occupano di problematiche diverse, ma in quasi tutti i casi è prevista l'estrapolazione di informazioni da immagini. L'unica eccezione è data dal progetto *jdb78/pytorch-forecasting* che si occupa del *forecasting* di serie temporali.

Tabella 4.1: Dominio applicativo dei progetti con maggior uso di librerie di ML

Progetto	Dominio Applicativo
davidsandberg/facenet	Riconoscimento facciale
jdb78/pytorch-forecasting	Time series forecasting
tianzhi0549/FCOS	Riconoscimento di oggetti
emedvedev/attention-ocr	Riconoscimento del testo
Tianxiaomo/pytorch-YOLOv4	Riconoscimento di oggetti

Sia nel caso in cui l'analisi sia svolta sui file modificati, sia nel caso in cui sia svolta sugli import, il dato riguardante il ML è caratterizzato da una forte varianza. Questo vuol dire che la diversa natura dei progetti considerati nello studio genera delle caratteristiche diverse per quanto riguarda l'architettura.

4.2 RQ2: COME SONO DISTRIBUITI I BUG SULLE DIVERSE FASI DI ML?

Andando a confrontare la distribuzione delle fasi sui commit (fig. 4.3) rispetto alla distribuzione sulle issues (fig. 3.1b) è possibile notare la scomparsa della fase *data collection*.

Inoltre è evidente anche la riduzione delle occorrenze di *model training* e una crescita d'importanza per quanto riguarda le fasi di *model requirements* e *model deployment*. Sfortunatamente i dati disponibili per questa analisi sono molto limitati (è stato possibile ricavare la fase solo per quaranta *fix*), per cui non è stato possibile effettuare delle analisi più approfondite.

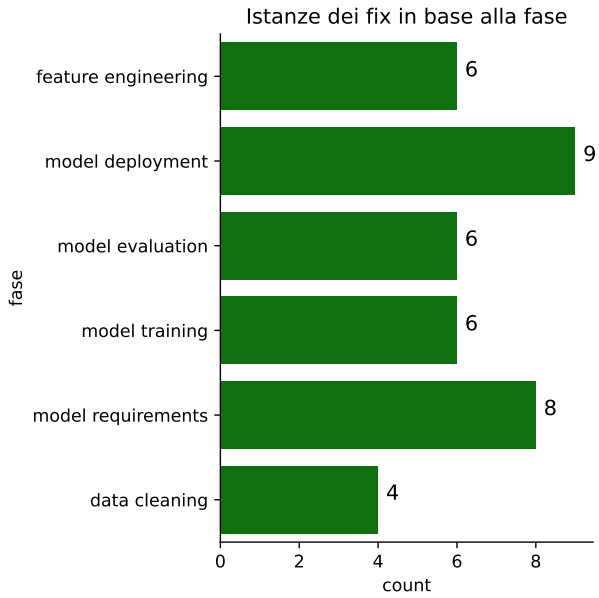


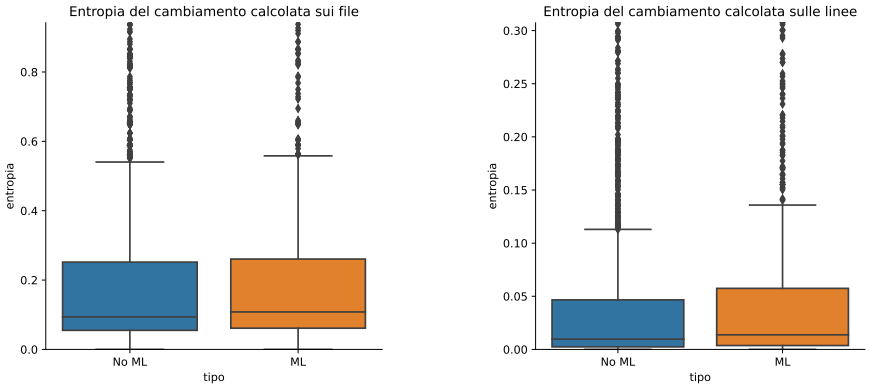
Figura 4.3: Istanze dei fix in base alla fase

4.3 RQ3: ESISTE UNA DIFFERENZA DI ENTROPY TRA ML BUG E ALTRI BUG?

Dal boxplot[1] in fig. 4.4a è possibile notare una distribuzione equivalente per le due tipologie di fix. Una situazione analoga si riscontra anche nell'analisi sulle linee (fig. 4.4b) anche se in questo caso è possibile notare che i valori di entropia associati ai fix di ML sono shiftati leggermente verso l'alto.

Per verificare la rilevanza statistica di questa diversità sono stati svolti il *ranksum* test e il *Cliff's delta* i cui risultati sono riportati nella tbl. 4.2.

1 Per ragioni di visualizzazione è stato scelto il 95-*esimo* quantile come limite superiore di entrambi i grafici.



(a) Entropia calcolata sui files (b) Entropia calcolata sulle linee

Figura 4.4: Entropia in base al tipo di fix

Nel caso dell'entropia sui file possiamo dire che la differenza è marginale poiché il *p-value* è prossimo a 0.05, mentre nel caso dell'entropia calcolato sulle linee la differenza viene confermata dal test. In entrambi i casi, però, l'*effect size* è trascurabile segno che la complessità dell'intervento non varia in base al tipo di intervento.

Tabella 4.2: Risultati dei test statistici per quanto riguarda l'entropia

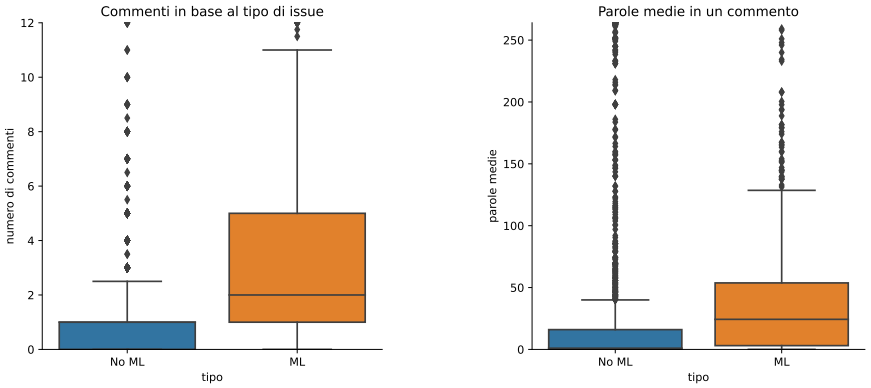
	ranksum p-values	Cliff's delta
file entropy	0.059	0.044
line entropy	5.932e-06	0.105

Non sono emerse differenze statisticamente rilevanti per quanto riguarda la complessità del processo di cambiamento.

4.4 RQ4: COME VARIA IL LIVELLO DI DISCUSSIONE TRA ML BUG E ALTRI BUG?

Osservando invece il boxplot[2] in fig. 4.5a si evince una differenza molto più marcata tra le due distribuzioni. In particolare è possibile notare che le *issue fix* di ML presentano una maggiore discussione e anche una maggiore varianza. Se consideriamo la differenza interquartile, in modo da escludere completamente eventuali outlier, possiamo osservare che nei *fix* generici questa varia tra zero e uno. Ciò vuol dire che il 50% interno delle issues o non presenta commenti o ne presenta uno solo. Mentre la differenza interquartile dei *fix* di ML è compreso tra uno e cinque, quindi nel 50% interno tutte le issues hanno almeno un commento di risposta.

2 In questo caso il limite superiore è pari al 97-*esimo* quantile.



(a) Numero di commenti medi

(b) Numero di parole medie per commento

Figura 4.5: Livello di discussione in base al tipo

I risultati dell'analisi rispetto alle parole medie contenute in un commento sono riportati in fig. 4.5b. Anche in questo caso si può vedere che nel caso di ML *fix* la distribuzione presenta valori più elevati e maggiore varianza. Per cui non solo nei *fix* di ML c'è maggiore discussione, ma la discussione è anche più *densa*.

Anche in questo caso sono stati svolti i test statistici. In tbl. 4.3 è possibile vedere come per entrambe le metriche considerate il *p-value* sia abbondantemente inferiore alla soglia di 0.05 quindi abbiamo una conferma della diversità delle due distribuzioni riscontrata dal boxplot. Inoltre, per entrambe le metriche, abbiamo un *effect size* medio.

Tabella 4.3: Risultati dei test statistici per quanto riguarda il livello di discussione

	ranksum p-values	Cliff's delta
commenti medi	9.053e-75	0.425
parole per commento	2.889e-59	0.377

Infine, per entrambe le metriche, sono stati analizzati alcuni casi estremi. Nel caso della issue numero 96 del progetto *BrikerMan/Kashgari* la problematica riguarda un drastico calo di performance quando il fit viene eseguito con un metodo piuttosto che con un altro. All'interno dei commenti, diversi *contributors* del progetto, si scambiano possibili architetture, *snippet* di codice e metriche per confrontare i diversi modelli generati. In questo caso l'ampiezza della discussione è sicuramente dovuta alla difficoltà di individuare la problematica.

La issue numero 27 del progetto *pyswarms/issues* è una richiesta di aiuto da parte dell'autore per migliorare l'implementazione della ricerca per il tuning degli hyperparametri. In questo caso la discussione si

protrae per oltre trenta commenti ed è incentrata sui requisiti dell'implementazione e come implementarla nel rispetto delle linee guida del progetto. Quest'intervento di modifica è stato il primo contributo dell'utente non solo su questo progetto, ma sull'intera community di GitHub. Questa inesperienza può aver contribuito ad ampliare la discussione.

La stessa analisi è stata svolta anche per le issues che presentano un alto numero di parole medie per commento. In questo caso un valore molto elevato della metrica è spesso riconducibile alla condivisione di blocchi di codice. Ne sono un esempio la issue tratta precedentemente nel caso dei commenti, ma anche la issue 125 sempre del progetto *BrikerMan/Kashgari*. Altri fattori che contribuiscono a spiegare questo dato sono la presenza di blocchi di errori (*mitta-gessen/kraken/206*) o messaggi di log utili ad inquadrare l'origine del problema (*robertmartin8/PyPortfolioOpt/177*).

> Le *issues* di ML sono caratterizzata da una maggiore discussione. Un valore molto elevato di parole per commento può indicare uno scambio massiccio all'interno della discussione di *snippet* di codice, di log d'errore e configurazioni dell'ambiente.

4.5 RQ5: COME VARIA IL TIME-TO-FIX TRA ML BUG E ALTRI BUG?

Anche in questo caso, osservando la fig. 4.6, è possibile notare una netta differenza tra i *fix* di ML e gli altri. In particolare i bug di ML necessitano, mediamente, di maggior tempo per essere risolti e sono caratterizzati da una varianza maggiore. Inoltre è possibile vedere come la mediana non sia centrata, bensì spostata verso il basso. Questo vuol dire che il 50% basso dei *bug* di ML viene comunque risolto in tempi brevi (due giorni circa), mentre l'altro 50% può richiedere una quantità di tempo decisamente superiore.

Un'ulteriore testimonianza del maggior tempo necessario per risolvere le problematiche legate al ML ci viene data dagli outlier. Nel caso di un problema generico, questo, viene considerato come *anomalo* se per essere risolto necessita di un tempo superiore ai cinque giorni. Mentre nel caso dei *fix* di ML per essere considerato outlier una *issue*, necessaria di un *time-to-fix* superiore ai trentacinque giorni.

Il maggior tempo necessario ad attuare la correzione indica che i *bug* di ML sono più difficili da individuare e correggere rispetto a quelli generici. Inoltre questo risultato contribuisce a spiegare il dato emerso dalla sezione precedente, in quanto per individuare la

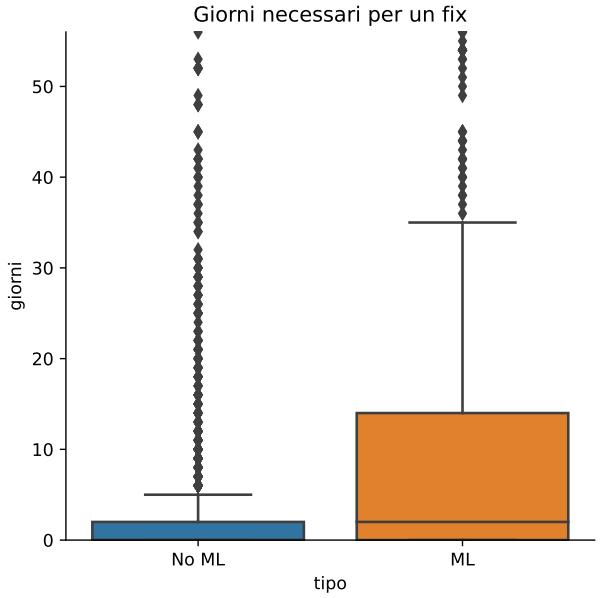


Figura 4.6: Giorni necessari per il fix

fonte del problema sembrerebbe essere necessaria una discussione più approfondita.

Per quanto riguarda i *fix* che hanno richiesto un tempo estremamente lungo la causa può dipendere anche da ulteriori fattori. Nel caso del progetto *CamDavidsonPilon/lifelines* la *issue* numero 507 segnala una problematica di *overflow* durante le operazioni sul dataset. Per stessa ammissione dell'autore del progetto la problematica è banale da risolvere, ma è stato comunque necessario attendere un paio di mesi affinché la correzione venisse portata sul branch principale.

Altre issues invece hanno necessitato di molto tempo per essere risolte in quanto venivano considerate a bassa priorità. In questi casi generalmente viene fornito un *work around* che permette di tamponare la problematica. La presenza di questo *work around* probabilmente riduce ulteriormente la priorità data alla *issue* il che dilata ulteriormente i tempi. Un esempio di questo comportamento ci viene dato dalla *issue* 135 del progetto *robertmartin8/PyPortfolioOpt* che ha richiesto circa sette mesi per essere risolta o dalla *issue* 98 del progetto *mittagessen/kraken* che invece ha necessitato di quasi due anni.

Anche per quest'ultima *RQ* sono stati svolti i test statistici illustrati precedentemente. Dai risultati riportati in tbl. 4.4 è possibile notare un *p-value* inferiore a 0.05 e un *effect size* medio. Questi risultati non solo confermano la differenza osservata nel boxplot, ma ci confermano che l'impatto sulla metrica non è trascurabile.

Tabella 4.4: Risultati dei test statistici per quanto riguarda il time-to-fix

	ranksum p-values	Cliff's delta
day-to-fix	7.354e-53	0.355

Le problematiche di ML richiedono più tempo per essere risolte. La bassa priorità di una *issue* e la presenza di *work around* sono fattori che contribuiscono a ritardare l'intervento di *fix*.

4.6 THREATS TO VALIDITY

La *threats to validity* più critica per il lavoro svolto è di tipo *construct* e riguarda la classificazione delle *issues*. La classificazione è avvenuta in modo automatico attraverso un modello *naïve Bayes*. Il classificatore, sebbene sia caratterizzato da una *recall* molto elevata, presenta una *precision* discreta per cui è molto probabile che all'interno tra le *issues* di ML siano state incluse anche *issues* generiche. Inoltre, poiché la classificazione degli interventi di *issue fixing* dipende dalla classificazione degli *issues*, gli eventi di *misclassification* sono stati propagati anche su questa seconda classificazione.

Per quanto riguarda le *threat to validity* interne bisogna segnalare l'interpretazione data al *time-to-fix*. Infatti in questo lavoro il dato del *time-to-fix* è stato calcolato come la differenza tra l'istante di chiusura e di apertura della *issue*. Questa approssimazione è sicuramente semplicistica in quanto comprende altri sotto intervalli come *time-to-response*, *time-to-assign*, ecc. Mentre per quanto riguarda le *threat to validity* esterne va sicuramente segnalato che i risultati di questo lavoro si generalizzano unicamente per i trenta progetti inclusi nel dataset.

5

CONCLUSIONI

La *RQ1* (sec. 4.1) ci ha permesso di inquadrare la natura dei progetti considerati per questo studio. Attraverso l'analisi degli import si è mostrato come l'utilizzo di librerie di ML vari a seconda del progetto. Da questo dato si può capire che i progetti all'interno del dataset sono diversi tra di loro e che alcuni sono più incentrati sul ML rispetto ad altri. Si è anche visto che la percentuale di progetti con un numero di *source files* di ML superiore al 45% sia molto limitata. Inoltre andando ad analizzare la porzione di sistema impattata dai cambiamenti si è visto come anche in questo caso il dato sia caratterizzato da una forte variabilità.

Le *RQ3*, *RQ4* e *RQ5* (da sec. 4.3) sono andate a valutare nello specifico le differenze in termini di entropia, discussione e *time-to-fix* tra gli interventi di *issue fixing* generici e quelli specifici del ML. Da queste analisi si evince che tra i due tipi di interventi ci sono sia similitudini che differenze. Nel caso dell'entropia e della complessità del processo di cambiamento del software non sono emerse differenze rilevanti. Questo ci porta a pensare che il processo di cambiamento non varia in base al tipo di intervento, ma sia costante.

Nel caso del livello di discussione e del *time-to-fix* sono emerse delle differenze confermate anche dai test statistici effettuati. In entrambi i casi l'essere un *fix* legato al ML ha spinto la metrica verso l'alto. Nel caso dei messaggi scambiati non solo si è riscontrato un numero medio di messaggi più elevato, ma si è visto anche che questi mediamente sono più lunghi. Questo dato potrebbe dipendere sia dal maggiore tempo richiesto per d'individuazione e correzione delle problematiche legate al ML, sia da un maggiore interesse per queste problematiche rispetto alle altre.

In sintesi questo lavoro ha fatto emergere sia delle similitudini che delle differenze per quanto riguarda gli interventi di *fix* all'interno di progetti di ML. Le principali differenze sono state riscontrate per quanto riguarda il livello di discussione, decisamente più alto nel caso di *issues* di ML, e il tempo necessario alla correzione dei difetti, anche in questo caso maggiore nel caso del ML. Non sono emerse differenze rilevanti invece per quanto riguarda l'entropia generata dai cambiamenti. Infine si è visto come l'impatto delle componenti di ML sull'architettura vada a riflettere la natura dei progetti.

5.1 SVILUPPI FUTURI

Nella *RQ2* sfortunatamente non è stato possibile svolgere un'analisi più approfondita per la carenza di dati. Un possibile sviluppo futuro potrebbe consistere nella realizzazione di un classificatore *multi-label* in grado di individuare la fase in cui il problema si è manifestato. In questo modo non solo sarebbe possibile conoscere la fase per ogni intervento di *fix*, ma anche definire delle nuove analisi. Per esempio si potrebbe andare a ricercare differenze in termini di entropia, discussione e *time-to-fix* in base alla fase in cui si è presentata la *issue*.

Per quanto riguarda la valutazione dell'entropia si è scelto come intervallo temporale di riferimento il singolo commit. Utilizzando questa configurazione non si è riscontrata nessuna differenza degna di nota. Un possibile sviluppo futuro potrebbe consistere nell'andare a valutare l'entropia considerando dei riferimenti temporali più ampi e verificare in questo caso la presenza di differenze.

Infine un aspetto non considerato in questo lavoro riguarda i *contributors*. Una prima analisi potrebbe andare a valutare se esiste una sovrapposizione o meno tra chi effettua interventi di *fix* generici e chi si occupa di quelli legati al ML. Inoltre si potrebbero andare a ricercare anche differenze in base al tipo di contributore (interno, esterno).

BIBLIOGRAFIA

[1] D. Gonzalez, T. Zimmermann, e N. Nagappan, «The State of the ML-Universe: 10 Years of Artificial Intelligence & Machine Learning Software Development on GitHub», in *Proceedings of the 17th International Conference on Mining Software Repositories*, giu. 2020, pagg. 431–442, doi: 10.1145/3379597.3387473.

[2] J. Han, S. Deng, D. Lo, C. Zhi, J. Yin, e X. Xia, «An Empirical Study of the Dependency Networks of Deep Learning Libraries», in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, set. 2020, pagg. 868–878, doi: 10.1109/ICSME46990.2020.00116.

[3] Y. Zhang, Y. Chen, S.-C. Cheung, Y. Xiong, e L. Zhang, «An Empirical Study on TensorFlow Program Bugs», in *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, lug. 2018, pagg. 129–140, doi: 10.1145/3213846.3213866.

[4] A. E. Hassan, «Predicting Faults Using the Complexity of Code Changes», in *2009 IEEE 31st International Conference on Software Engineering*, 2009, pagg. 78–88, doi: 10.1109/ICSE.2009.5070510.

[5] C. E. Shannon, «A Mathematical Theory of Communication», *The Bell System Technical Journal*, vol. 27, n. 3, pagg. 379–423, lug. 1948, doi: 10.1002/j.1538-7305.1948.tb01338.x.

[6] J. Liu, Q. Huang, X. Xia, E. Shihab, D. Lo, e S. Li, «An Exploratory Study on the Introduction and Removal of Different Types of Technical Debt», *Empir Software Eng*, vol. 26, n. 2, pag. 16, mar. 2021, doi: 10.1007/s10664-020-09917-5.

[7] J. Han, E. Shihab, Z. Wan, S. Deng, e X. Xia, «What Do Programmers Discuss about Deep Learning Frameworks», *Empir Software Eng*, vol. 25, n. 4, pagg. 2694–2747, lug. 2020, doi: 10.1007/s10664-020-09819-6.

[8] M. Grichi, E. E. Eghan, e B. Adams, «On the Impact of Multi-Language Development in Machine Learning Frameworks», in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, set. 2020, pagg. 546–556, doi: 10.1109/ICSME46990.2020.00058.

[9] N. Humbatova, G. Jahangirova, G. Bavota, V. Riccio, A. Stocco, e P. Tonella, «Taxonomy of Real Faults in Deep Learning Systems», nov. 07, 2019. http://arxiv.org/abs/1910.11015 (consultato mar. 17, 2021).

[10] A. A. Bangash, H. Sahar, S. Chowdhury, A. W. Wong, A. Hindle, e K. Ali, «What Do Developers Know About Machine Learning: A Study of ML Discussions on StackOverflow», in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, mag. 2019, pagg. 260–264, doi: 10.1109/MSR.2019.00052.

[11] M. Alshangiti, H. Sapkota, P. K. Murukannaiah, X. Liu, e Q. Yu, «Why Is Developing Machine Learning Applications Challenging? A Study on Stack Overflow Posts», in *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, set. 2019, pagg. 1–11, doi: 10.1109/ESEM.2019.8870187.

[12] H. Borges, A. Hora, e M. T. Valente, «Understanding the Factors That Impact the Popularity of GitHub Repositories», *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pagg. 334–344, ott. 2016, doi: 10.1109/ICSME.2016.31.

[13] S. Dueñas, V. Cosentino, G. Robles, e J. M. Gonzalez-Barahona, «Perceval: Software Project Data at Your Will», in *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, mag. 2018, pagg. 1–4, doi: 10.1145/3183440.3183475.

[14] «Naive Bayes Classifier», *Wikipedia*. mag. 21, 2021, Consultato: giu. 03, 2021. [Online]. Disponibile su: https://en.wikipedia.org/w/index.php?title=Naive_Bayes_classifier&oldid=1024247473.

[15] P. Harrington, *Machine Learning in Action*. Manning Publications, 2012.

[16] S. Amershi *et al.*, «Software Engineering for Machine Learning: A Case Study», in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, mag. 2019, pagg. 291–300, doi: 10.1109/ICSE-SEIP.2019.00042.