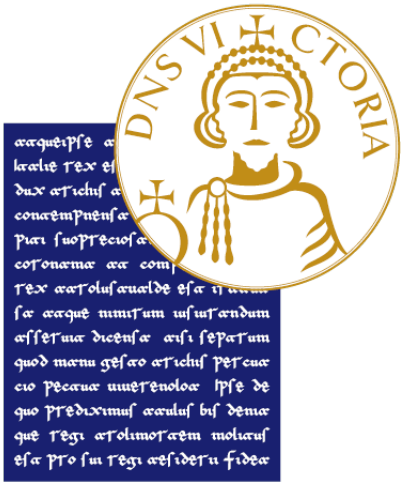
UNIVERSITÀ DEGLI STUDI DEL SANNIO
DIPARTIMENTO DI INGEGNERIA



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

[11 giugno 2021 at 10:40 –]

INDICE

LISTA DELLE FIGURE

ACRONIMI

AI Artificial Intelligence
API Application Program Interface
CERN European Council for Nuclear Research
DL Deep Learning
ML Machine Learning
NLP Natural Language Processing
SATD Self-Admitted Technical Debt
VPS Virtual Private Server

1 INTRODUZIONE

La storia dell'industria dello sviluppo software è caratterizzata da diversi cambiamenti rispetto alle applicazioni dominati. Negli anni ottanta il dominio dominante era quello dei personal computer, puoi 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 Machine Learning 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. Il seguente lavoro si muove proprio in questo solco e ha lo scopo di studiare come le componenti di ML sono distribuite sull'architettura dei sistemi, ma anche di capire se esistono delle differenze tra gli interventi di *issue fixing* doviti a problematiche di Machine Learning e quelli generici.

1.1 OPERE CORRELATE

In questo lavoro[1] sono stati studiati mediante tecniche di statistica descrittiva 9325 progetti. I progetti sono stati distinti tra sistemi di ML, a loro volta suddivisi in framework (700 elementi) e applicazioni (4524 elementi), e sistemi generici (4101 elementi). Gli aspetti considerati per l'analisi sono vari, si va dalla distribuzione dei contributi, a loro volta divisi tra interni ed esterni, fino all'analisi dei linguaggi più utilizzati, passando per un'analisi sulla popolarità dei vari repositories. Inoltre vengono valutate anche le differenze per quanto riguarda le interazioni collaborative (discussioni, review, ecc.).

In altri lavori[2] il focus è stato puntato sulla gestione delle dipendenze dei progetti di Machine Learning. In questo caso si va a valutare le eventuali differenze in base al framework utilizzato. Le librerie considerate nello studio sono: `TensorFlow`, `PyTorch` e `Theano`. Mentre eventuali differenze sono state ricercate rispetto agli obbiettivi del progetto (tutorial/libri, applicativi, ricerca), il dominio applicativo, la popolarità, la frequenza di aggiornamento delle dipendenze e i motivi degli aggiornamenti.

In altri casi[3] ancora l'attenzione è stata rivolta alla natura *multi-linguaggio* tipica delle soluzioni di Machine Learning e all'impatto che ciò ha sul sistema. In questo caso sono stati considerati, tra progetti mono-linguaggio e multi-linguaggio, 27 repository open source. Per quanto riguarda l'analisi sono stati presi in considerazione la percentuale di pull request accettate, il tempo necessario per accettare una pull request e la propensione nell'introdurre i *bug* all'interno delle pull request.

In letteratura sono presenti anche molti lavori che si concentrano sull'analisi delle problematiche e dei *bug* riscontrati all'interno di applicazioni di Machine Learning. In alcuni casi lo studio[4] è stato svolto in maniera specifica per una singola libreria. Nello specifico sono state considerate 87 domande postate su *StackOverflow* in relazione a bug di `TensorFlow` e 82 commit, selezionati da 11 progetti su *GitHub*. Lo studio aveva l'obiettivo di individuare i sintomi e le cause scatenati di questi difetti e individuare le sfide per l'individuazione e la localizzazione di questi.

In altri casi[5] l'attenzione non è stata rivolta ad una libreria specifica, ma si è cercato di definire una tassonomia delle problematiche che fosse però generale per tutti i framework e le applicazioni di ML. Anche in questo caso i dati per lo studio sono stati recuperati sia da *GitHub* che da *StackOverflow*. Altre volte ancora l'analisi[6] è stata mirata su alcuni aspetti specifici come l'introduzione e la rimozione di Self-Admitted Technical Debt (SATD) all'interno di progetti che fanno uso di Deep Learning (DL).

Noi lavori precedentemente discussi l'analisi è stata svolta su dati recuperati dalla storia dei repositories e in alcuni casi recuperando informazioni aggiuntive da piattaforme di discussione online. Altri lavori[7]–[9] invece hanno analizzato unicamente le discussioni su *StackOverflow* per andare a capirne il contenuto. Lo scopo di questi studi è quello di individuare le fasi più critiche del processo di sviluppo e capire quali sono gli argomenti che gli sviluppatori discutono più frequentemente.

Altri studi[10] ancora hanno traslando il concetto di entropia[11] utilizzato nella teoria della comunicazione per andare a valutare la complessità del processo di cambiamento del software. Andando,

inoltre, ad evidenziare come la complessità del processo possa essere utilizzata per predire i *faults* all'interno dei prodotti software con risultati migliori rispetto alle metriche di complessità del software.

1.2 RESEARCH QUESTIONS

Questo studio ha l'obiettivo di dare una risposta a queste cinque domande:

- **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 concentratala sui files e sulle directories modificate durante interventi di *issues fixing*. Obbiettivo di questa domanda è anche individuare la percentuale di files che utilizzano import riconducibili a librerie e framework di Machine Learning.

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

  Il workflow tipico per lo sviluppo di un'applicazione di Machine Learning 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 Machine Learning 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 Machine Learning, 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 Machine Learning.

2

COLLEZIONE DEI DATI E ANALISI PRELIMINARE

2.1 SELEZIONE DEI PROGETTI

L'individuazione dei progetti da analizzare è avvenuta mediate l'ausilio dell'Application Program Interface (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.

2.2 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 uno 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.

2.3 CLASSIFICAZIONE DEI DATI

2.3.1 *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. 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. Il numero elevato di elementi non rende praticabile una classificazione manuale per cui si è optato per una classificazione automatica. 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 mediate il classificatore statico non necessita di un *labeling* manuale dei dati, ma richiede la definizione dei vocaboli tipici del ML. Lista dei termini caratteristici del Machine Learning non è stata costruita da zero, ma è basata su lavori precedenti[5]. In questo modo tutte le issues che utilizzavano almeno un vocabolo tipico del Machine Learning 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. Durante il labeling si scelto di classificare ulteriormente le issue di ML al fine di individuare anche

la fase in cui il problema si è palesato. La definizioni delle varie fasi è avvenuta partendo da un lavoro di *Microsoft*[16].

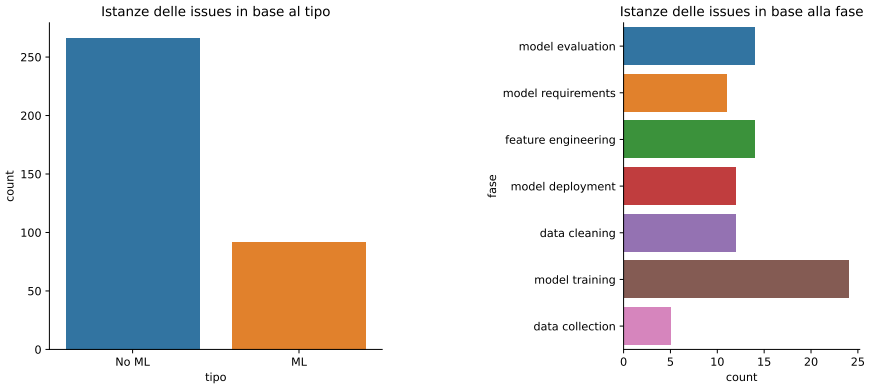
Le fasi considerate sono:

- *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 monitora al fini 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. 2.1, il modello basato sulla lista di vocaboli è leggermente più preciso del modello bayesiano, ma presenta una recall decisamente più bassa. Dalla fig. 2.1a si evince la natura minoritaria delle issues di

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 2.1: Risultati della classificazione manuale delle issues

ML rispetto alle issues generiche, per questo motivo si è scelto di preferire il modello naïve Bayes in modo da perdere quante meno istanze possibili anche a costo di sacrificare leggermente la precisione.

Tabella 2.1: Confronto dei due modelli per la classificazione delle issues.

	Classificatore statico	naïve Bayes
precision	0.46	0.41
recall	0.74	0.94

2.3.2 *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. 2.2.

A questo punto è stato possibile separare i *fix* di Machine Learning 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.

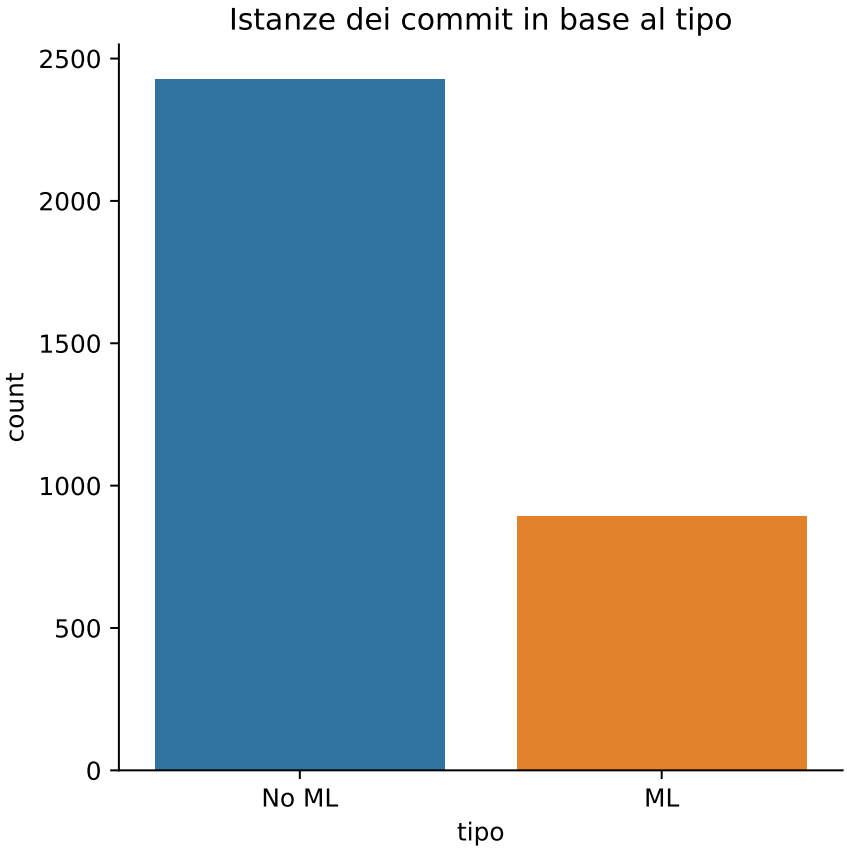


Figura 2.2: Risultato della classificazione dei commit

3

ANALISI

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

In questa prima analisi si è andato a verificare l'esistenza di una differenza nei files e nelle directories modificate in base al tipo di cambiamento. Per poter svolgere questa analisi è stato necessario individuare il numero totale di file modificati per *fix* generici e per i *fix* specifici del Machine Learning. A tal fine i commit sono stati raggruppati rispetto al progetto e al tipo di cambiamento (ML, no ML) e per ogni istanza di questo raggruppamento si è eseguito l'union set dei files modificati. Come output di questa fase si è generato 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 ottenere l'insieme totale dei files modificati durante i *fix*. In questo modo è stato possibile andare a valutare la percentuale di files modificati in relazione al tipo di cambiamento. Attraverso la funzione di libreria Python `os.path.dirname` sono stati ottenuti i tre insiemi sopra citati anche per quanto riguarda le directories.

Dalla fig. 3.1 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 Machine Learning è caratterizzato da una maggiore varianza.

Un'ulteriore analisi rispetto all'architettura dei progetti è stata svolta mediante gli import. Attraverso uno script sono stati estratti, per ogni file, gli import utilizzati all'interno del file stesso. A questo punto sono stati individuati i files di Machine Learning in base agli import utilizzati. La classificazione è avvenuta utilizzando due livelli di severità; in un primo (severità *strict*) caso sono stati considerati come import di Machine Learning solo delle librerie strettamente di ML come ad esempio `keras`, `TernsorFlow`, `PyTorch`, ecc. Mentre in un secondo caso (severità *base*) sono state incluse anche librerie utilizzate

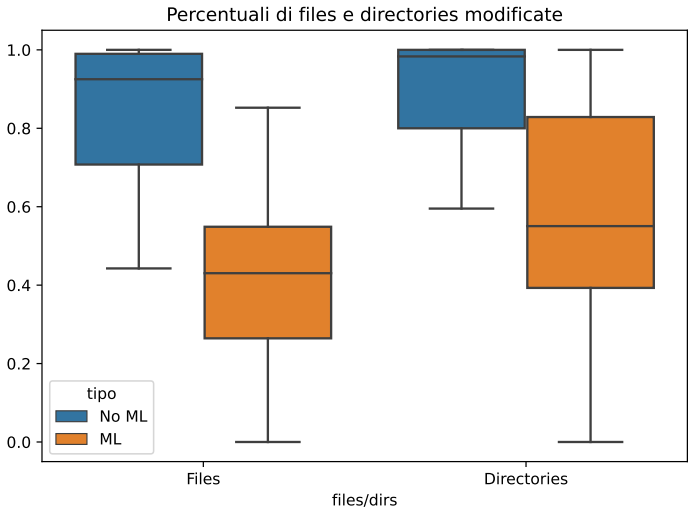


Figura 3.1: Percentuale di files e directories modificate in base al tipo di cambiamento

spesso in ambito ML, ma anche in altri ambiti, come ad esempio `pandas`, `numpy` e `scipy`.

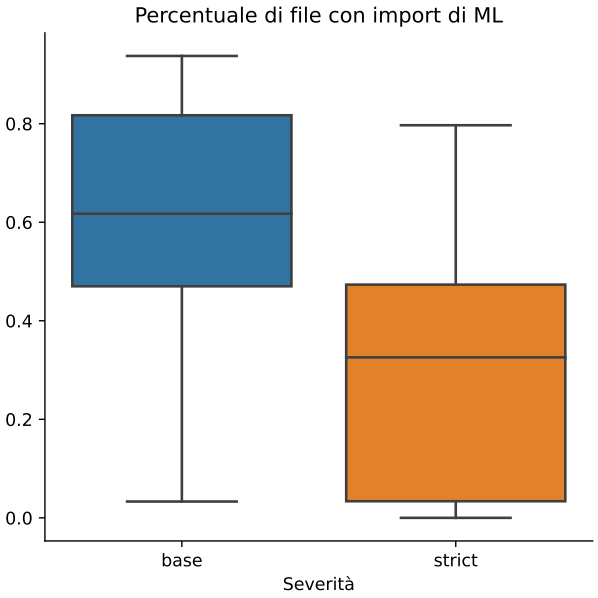


Figura 3.2: Percentuale di file che utilizzano librerie di ML

Dal boxplot riportato in fig. 3.2 si può notare che, indipendentemente dalla severità dell'analisi, la percentuale di file che utilizzano librerie di Machine Learning 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%.

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

Come illustrato nella sec. 2.3.2 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. 2.3.1). Questo dato aggiuntivo presente su alcune issues è stato *proiettato* anche sulla classificazione dei commit di *fix* per andare a valutare come questi sono distribuiti sulle varie fasi. I risultati di questa analisi sono riportati in fig. 3.3.

Figura 3.3: Istanze dei fix in base alla fase

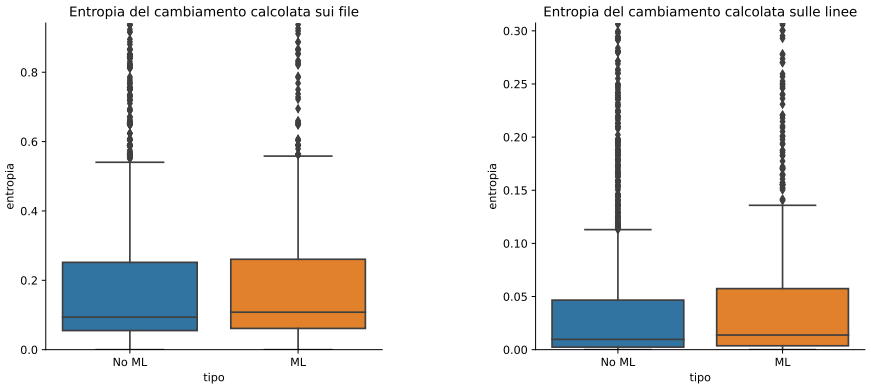
Rispetto alla distribuzione sulle issues (fig. 2.1b) è possibile notare la scomparsa della fase *data collection*, inoltre è evidente anche la riduzione delle occorrenze di *model training* e una crescita d'importanza per quanto riguarda le fasi di *model requirements* e *model deployment*. Sfortunatamente 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.

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

La successiva analisi avevo lo scopo di verificare l'esistenza di una differenza tra l'entropia del *fix* rispetto alla natura di questi. 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.

Inoltre per poter valutare l'entità del 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`[1]. 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.

Una volta note queste informazioni preliminari è stato possibile calcolare l'entropia dei *fix* che è stata riportata nei boxplot[2] in fig. 3.4. Dal boxplot in fig. 3.4a è possibile notare una distribuzione equivalente per le due tipologie di fix. Una situazione analoga si riscontra anche nell'analisi sulle linee (fig. 3.4b) anche se in questo caso è possibile notare che i valori di entropia associati ai fix di ML sono shiftati leggermente verso l'alto.



(a) Entropia calcolata sui files

(b) Entropia calcolata sulle linee

Figura 3.4: Entropia in base al tipo di fix

1 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.

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

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. 3.1. 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, l'*effect size* è trascurabile.

Tabella 3.1: 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

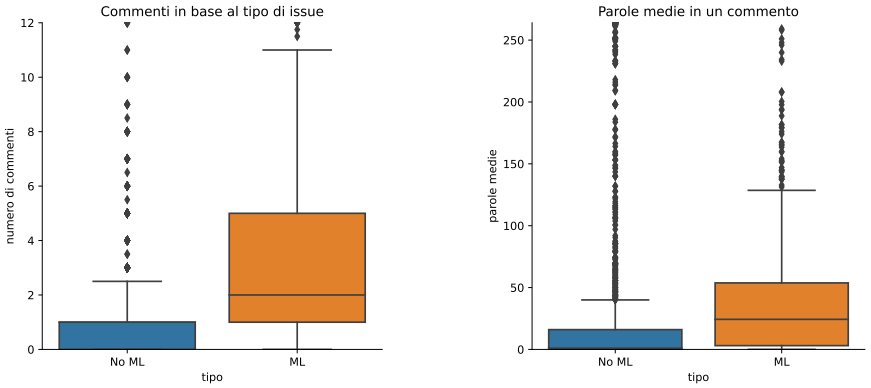
3.4 RQ4: COME VARIA IL LIVELLO DI DISCUSSIONE TRA ML BUG E ALTRI BUG?

Per rispondere a questa domanda è stato necessario andare a valutare il numero di commenti presenti all'interno di ogni issues. Poiché un singolo commit può far riferimento a più issues è stato considerato il numero di commenti medi. I risultati ottenuti sono stati riportati nel boxplot[3] in fig. 3.5a.

In questo caso 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 Machine Learning è compreso tra uno e cinque quindi nel 50% interno tutte le issues hanno almeno un commento di risposta.

A questo punto si è cercato di capire se al maggior numero di commenti è associata effettivamente una maggiore quantità di informazioni scambiate. Per svolgere questa analisi si è partiti dal presupposto che la quantità di informazioni scambiate sia proporzionale al numero di parole utilizzate nel commento. Quindi per ogni *issue* è stato calcolato il numero medio di parole presenti all'interno di un commento. I risultati di quest'ulteriore analisi sono riportati in fig. 3.5b. 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 Machine Learning c'è maggiore discussione, ma la discussione è anche più *densa*.

3 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 3.5: Livello di discussione in base al tipo

Anche in questo caso sono stati svolti i test statistici. In tbl. 3.2 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 3.2: 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

3.5 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*. Per valutare questo parametro è stato necessario estrarre da ogni *issue* la data di apertura e di chiusura e calcolare i giorni che intercorrono tra queste. I risultati così ottenuti sono stati riportati in fig. 3.6.

Anche in questo caso è possibile notare una netta differenza tra i *fix* di ML e gli altri. In particolare i bug di Machine Learning 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.

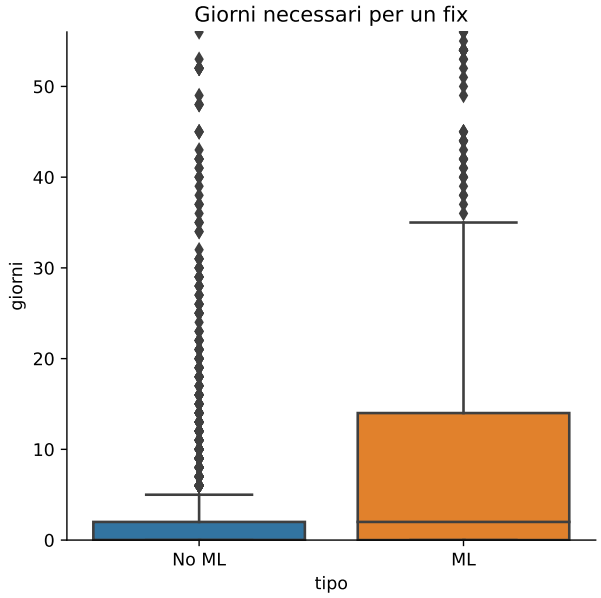


Figura 3.6: Giorni necessari per il fix

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 Machine Learning per essere considerato outlier un *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 fonte del problema sembrerebbe essere necessaria una discussione più approfondita.

Anche per quest'ultima *RQ* sono stati svolti i test statistici illustrati precedentemente. Dai risultati riportati in tbl. 3.3 è possibile notare un *p-value* inferiore a 0.05 e un *effect size* medio.

Tabella 3.3: 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

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] 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.

[4] 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.

[5] 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).

[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] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[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.