Code Comprehension Report for Mark

a.prisco50@studenti.unisa.it 0522501976
c.ranieri7@studenti.unisa.it 0522501977

June 30, 2025

1 Version History

Versione	Data	Descrizione
0.1	25/06/2025	Stesura della struttura del documento.
1.0	25/06/2025	Prima stesura del documento.
1.1	26/06/2025	Correzione del paragrafo *Obiettivo del documento*.
1.2	26/06/2025	Modifica del paragrafo *Informazioni Relative ai Dati*.
1.3	26/06/2025	Modifica della sezione *Dettagli Tecnici*.

2 Introduzione

Di seguito sono riportate tutte le informazioni relative al sistema Mark sotto il punto di vista del codice messo a disposizione e dei dati utilizzati.

Nota. Oltre ad un'analisi statica del progetto, quest'ultimo è stato anche eseguito per studiarne il comportamento attraverso un'analisi dinamica.

Obiettivo del Documento L'obiettivo di questo documento è quello di individuare aspetti problematici nel codice, comprendere quali sono i dati che esso elabora e come questi sono organizzati.

Contesto del Progetto Mark è un tool scritto in Python che dato un progetto, tramite l'ausilio di una Knowledge Base e delle detection rules, può classificare progetti ML in tre diverse categorie: ML-Consumer, ML-Producer e ML-Producer e Consumer.

3 Dettagli Tecnici

Di seguito sono riportati tre paragrafi, un primo paragrafo in cui sono riportate le informazioni relative alla logica di classificazione utilizzata dal sistema Mark, un secondo paragrafo in cui sono riportate informazioni e problemi relativi al codice e un terzo paragrafo in cui sono riportate le informazioni relative ai dati e alla loro organizzazione.

Logica di Classificazione Di seguito sono riportate le logiche di classificazione per un ML producer, un ML consumer e un ML producer e consumer.

ML-Model Producer. Per un ML producer vengono analizzati i metodi API relativi all'addestramento dei modelli. Un progetto è etichettato come *ML-Model Producer* se contiene un'istruzione di importazione per una libreria ML produttrice e un'invocazione di un metodo di addestramento associato.

Sia F un file appartenente al progetto P, L l'insieme delle librerie di machine learning, e A l'insieme delle API di addestramento. Sulla base di queste informazioni sono definiti i seguenti predicati:

$$containsMLLibrary(F, l) = \begin{cases} \textit{True}, & \text{se } F \text{ contiene } l, \text{ con } l \in L \\ \textit{False}, & \text{altrimenti} \end{cases}$$

$$containsTrainingAPI(F, a) = \begin{cases} \textit{True}, & \text{se } F \text{ contiene } a, \text{ con } a \in A \\ \textit{False}, & \text{altrimenti} \end{cases}$$

Un progetto è classificato come *ML-Model Producer* se esiste almeno un file F tale che:

- **Rule #1:** $\exists l \in L : containsMLLibrary(F, l)$ è *True*
- **Rule #2:** $\exists a \in A : containsTrainingAPI(F, a)$ è *True*

In altri termini, se un file contiene almeno una libreria ML e almeno un'API di addestramento, il progetto viene etichettato come *ML-Model Producer*.

ML-Model Consumer. Per un ML consumer sono stati cercati i metodi API associati all'inferenza del modello. Sia I l'insieme delle API di inferenza e sia $T = \{\textit{test}, \textit{eval}, \textit{example}, \textit{validate}\}$ l'insieme dei termini legati al testing. Si definiscono i seguenti predicati:

$$containsInferenceAPI(F, i) = \begin{cases} \textit{True} & \text{se } F \text{ contiene } i, \text{ con } i \in I \\ \textit{False} & \text{altrimenti} \end{cases}$$

$$containsTesting(F) = \begin{cases} \textit{True} & \text{se il nome del file } F \text{ contiene una sottostringa in } T \\ \textit{False} & \text{altrimenti} \end{cases}$$

Un progetto è classificato come *ML-Model Consumer* se esiste almeno un file F tale che:

- **Rule #1:** $\exists l \in L : containsMLLibrary(F, l)$ è *True*
- **Rule #2:** $\exists i \in I : containsInferenceAPI(F, i)$ è *True*
- **Rule #3:** $\nexists a \in A : containsTrainingAPI(F, a)$ è *True*
- **Rule #4:** $containsTesting(F)$ è *False*

Nota. Le regole 3 e 4 sono state introdotte per evitare falsi positivi dovuti all'uso di API di inferenza durante il testing di modelli.

ML-Model Producer & Consumer. È importante notare che le regole di classificazione per *Producer* e *Consumer* non sono mutualmente esclusive. Se un file F soddisfa tutte le regole da 1 a 2 per i *Producer*, e un altro file F' dello stesso progetto soddisfa tutte le regole da 1 a 4 per i *Consumer*, il progetto viene etichettato come *ML-Model Producer & Consumer*. Tuttavia, per design, MARK non assegna mai entrambi i ruoli allo stesso file. Questa scelta si basa sull'assunzione che ogni file abbia un obiettivo coerente: se un file contiene sia API di addestramento che di inferenza, si assume che le API di inferenza vengano utilizzate per la valutazione e non per l'inferenza in produzione.

Informazioni Relative al Codice Il progetto si compone di un totale di **9** moduli, elencati di seguito:

- **cloner.py**
- **cloning_check.py**
- **exec_analysis.py**
- **producer_classifier_by_dict.py**
- **consumer_classifier_by_dict.py**
- **library_extractor.py**
- **notebook_converter.py**
- **merge.py**
- **Results_Analysis.py**

Il flusso di esecuzione del sistema prevede inizialmente il recupero dei dati di input a partire da un file contenente una lista di repository GitHub. Tale lista viene utilizzata dal modulo *cloner.py* per effettuare la clonazione locale delle repository e per generare un file di log con l'elenco delle repository che non è stato possibile clonare. Successivamente, il modulo *cloning_check.py* verifica quali repository siano state effettivamente clonate. Dopo tale controllo, lo script *notebook_converter.py* converte eventuali file *.ipynb* in file *.py*, operazione fondamentale poiché i file *.ipynb* non possono essere analizzati dai moduli successivi. In assenza di questa fase, alcuni file non verrebbero considerati per la classificazione, compromettendo la correttezza del risultato.

Terminata la fase di preparazione, viene eseguito il modulo *exec_analysis.py*, che controlla l'esistenza dei file contenenti i dizionari delle librerie da attenzionare e definisce le regole di classificazione per identificare ML producers e ML consumers. Questo modulo istanzia le classi contenute nei moduli *producer_classifier_by_dict.py* e *consumer_classifier_by_dict.py*, i quali classificano rispettivamente i moduli che istanziano modelli di machine learning (producers) e quelli che li utilizzano (consumers). Entrambi i moduli si appoggiano a *library_extractor.py*, che si occupa di identificare le librerie utilizzate in ciascun file *.py*, estraendole tramite un'analisi degli import e confrontandole con i dizionari predefiniti.

È importante sottolineare che le classi *MLConsumerAnalyzer* e *MLProducerAnalyzer* condividono metodi con funzionalità simili o duplicati, come *build_regex_pattern*, *baseline_check*, *load_producer_library_dict*, nonché metodi dal comportamento analogo ma con nomi differenti come *init_consumer_analysis_folder* e *init_producer_analysis_folder*, oppure *check_training_method* e *check_for_training_method*. Inoltre, le due classi adottano approcci differenti all'interno della funzione *check_ml_library_usage*: la versione usata da *MLConsumerAnalyzer* è ridefinita localmente nella classe, mentre quella di *MLProducerAnalyzer* utilizza l'implementazione contenuta in *library_extractor.py*. Il modulo *consumer_classifier_by_dict.py*, inoltre, utilizza anche i dizionari dei producers per ridurre i falsi positivi nella regola 3 di classificazione.

Completata la classificazione, i risultati vengono salvati in file .csv attraverso *Results_Analysis.py*, che presenta una sequenza di blocchi di codice ripetuti per generare i file *result_producer.csv* e *result_consumer.csv*. Tali risultati sono successivamente utilizzati dal modulo *merge.py*, il quale effettua nuovamente la costruzione di tali file e calcola le metriche di performance, quali *precision*, *recall*, *accuracy* e *F1-score*.

Nota. Quasi tutte le funzioni dei moduli definiti sopra non eseguono controlli sulla correttezza dei path utilizzati come input o output.

Informazioni Relative ai Dati In questo paragrafo l'obiettivo posto è quello di descrivere come sono organizzati i dati utilizzati dal sistema, come vengono utilizzati e cosa rappresentano. Tali informazioni sono organizzate nella seguente tabella.

Nome	**Descrizione**	**Problema**
repos2	Cartella che contiene tutte le repository clonate dal modulo *cloner.py*.	Nessuno
selected_projects.csv	File che contiene i nomi delle repository da clonare.	Nessuno
errors.csv	File che contiene i nomi delle repository che il modulo *cloner.py* non è riuscito a clonare ed il relativo motivo.	Nessuno
applied_projects.csv	File che contiene i nomi tutte le repository da controllare che siano state clonate.	Nessuno
effective_repos.csv	File che contiene i nomi di tutte le repository clonate, ovvero quelle contenute nella cartella repos2.	Nessuno
not_cloned.csv	File che contiene i nomi di tutte le repository che non sono state clonate.	Nessuno
library_dict_consumers_*X*.csv	File che contiene i nomi delle librerie e relative keywords che esse utilizzano. Questo file serve al classificatore dei consumers.	Nessuno
library_dict_producers_*X*.csv	File che contiene i nomi delle librerie e relative keywords che esse utilizzano. Questo file serve al classificatore dei producers.	Nessuno
*nome_repo*_ml_producer.csv	File che contiene per ogni file .py classificato come producer le informazioni relative a librerie ed API che hanno portato a questa classificazione oltre alla riga di codice dove si trovano tali librerie ed API.	Nessuno
*nome_repo*_ml_consumer.csv	File che contiene per ogni file .py classificato come consumer le informazioni relative a librerie ed API che hanno portato a questa classificazione oltre alla riga di codice dove si trovano tali librerie ed API.	Nessuno

Nome	Descrizione	Problema
results_first_step.csv	File che contiene i risultati dell'analisi su più repository e per ogni repository riporta il risultato della classificazione per ogni file classificato come producer. Se non è definito un path di output viene creato nella cartella *Producers_Final*.	Nessuno
results_consumer.csv	File che contiene i risultati dell'analisi su più repository e per ogni repository riporta il risultato della classificazione per ogni file classificato come consumer. Se non è definito un path di output viene creato nella cartella *Consumers_Final*.	Nessuno
result_consumer_*X*.csv	File che è composto da tre colonne che rappresentano rispettivamente il nome della repository, se è realmente un consumer e come il sistema lo ha classificato.	Nessuno
result_producer_*X*.csv	File che è composto da tre colonne che rappresentano rispettivamente il nome della repository, se è realmente un producer e come il sistema lo ha classificato.	Nessuno
oracle_consumer.csv	File che contiene per le repository contenute nel file *selected_projects.csv* la corretta label di classificazione relativa ai consumers.	Nessuno
oracle_producer.csv	File che contiene per le repository contenute nel file *selected_projects.csv* la corretta label di classificazione relativa ai producers.	Nessuno
producer_verification.csv	File che contiene per ogni repository contenuta nel file *oracle_producer.csv* il valore reale di classificazione e la label associata dal classificatore.	Le informazioni contenute in questo file sono le stesse di quelle contenute nei *result_producer_X.csv*.
consumer_verification.csv	File che contiene per ogni repository contenuta nel file *oracle_consumer.csv* il valore reale di classificazione e la label associata dal classificatore.	Le informazioni contenute in questo file sono le stesse di quelle contenute nei *result_consumer_X.csv*.
producer_false_positive.csv	File che riporta i producers che non sono tali ma che sono stati classificati come tali.	Nessuno
producer_false_negative.csv	File che riporta i producers che sono tali ma che non sono stati classificati come tali.	Nessuno

Nome	Descrizione	Problema
consumer_false_positive.csv	File che riporta i consumers che non sono tali ma che sono stati classificati come tali.	Nessuno
consumer_false_negative.csv	File che riporta i consumers che sono tali ma che non sono stati classificati come tali.	Nessuno