Relazione Progetto di Machine Learning

Francesca Stefano
Matricola 353310

Luglio 2024

1 Introduzione

Il presente progetto di Machine Learning si propone di ottimizzare un modello di Genetic Programming (GP) per migliorare le prestazioni e ridurre i tempi di esecuzione. Dopo l'ottimizzazione del codice e l'implementazione delle liste di adiacenza, il codice è stato testato su un dataset contenente dati ECG (Elettrocardiogramma). L'obiettivo finale è stato quindi valutare come la GP possa offrire prestazioni eccellenti nell'analisi dei dati ECG, un campo critico per la diagnosi e il monitoraggio delle patologie cardiache. Di seguito vengono illustrati i dettagli dell'ottimizzazione effettuata e i risultati ottenuti nell'applicazione della GP all'analisi del dataset ECG.

2 Ottimizzazione del Codice

L'obiettivo dell'ottimizzazione è stato ridurre i tempi di esecuzione del modello GP mantenendo la stessa accuratezza nelle predizioni.

2.1 GPmodular.py

Le principali modifiche apportate includono:

1. **Semplificazione delle importazioni e ottimizzazioni:**
   - Rimozione di moduli inutilizzati e ottimizzazione delle librerie importate.
   - Consolidamento delle operazioni di serializzazione utilizzando `dill` al posto di `pickle`.
2. **Ristrutturazione e ridefinizione delle funzioni:**

- Ottimizzazione delle funzioni per migliorare leggibilità ed efficienza.
- `EvalTrainingSet`: Rimozione di parametri globali inutilizzati e miglioramento della gestione delle eccezioni.
- `EvalTestSet`: Semplificazione della logica interna per migliorare la leggibilità.
- `EvalValidationSet`: Snellimento della funzione per eliminare duplicati.
- `GetIndividualsToKeep`: Introduzione di stampe di debug per una migliore tracciabilità.

3. **Eliminazione delle ridondanze:**
   - Rimozione di codice duplicato e semplificazione delle chiamate alle funzioni di selezione, mutazione e crossover degli individui.
4. **Aggiunta di controlli di errore:**
   - Introduzione di controlli per gestire eccezioni e errori comuni.
   - Utilizzo di `warnings` per ignorare avvisi non critici durante l'esecuzione, migliorando la pulizia del codice.
5. **Utilizzo delle funzioni lambda e delle primitive ephemeral constant:**
   - Integrazione delle `ephemeral constant` con le funzioni lambda per migliorare flessibilità e performance.

2.2 utlis_import_data.py

Gestione delle Importazioni

- **Codice di origine:** Utilizzo delle librerie `csv` e `numpy` per la gestione dei dati CSV e degli array.
- **Codice modificato:** Introduzione di `pandas` per la gestione dei dati CSV, mantenendo `numpy` per la manipolazione degli array.
- **Vantaggio di `pandas`:** Fornisce una gestione dei dati più efficiente e intuitiva rispetto a `csv`, migliorando la concisione e la leggibilità del codice.

Importazione dei Dati dal File CSV

- **Codice di origine:**
  - Lettura del file CSV riga per riga utilizzando `csv.reader`.
  - Costruzione manuale delle liste `header`, `labels`, e `data`.
  - Conversione manuale di ogni valore in `float`.
- **Codice modificato:**
  - Utilizzo di `pandas` per leggere il file CSV con `pd.read_csv`, semplificando il processo.
  - Estrazione diretta di `header`, `labels`, e `data` utilizzando metodi di `pandas`, migliorando la leggibilità e riducendo il codice ridondante.
- **Vantaggio di `pandas`:** Elimina la necessità di conversioni manuali tra formati dati, riducendo errori e migliorando l'efficienza.

Mescolamento degli Array

- **Codice di origine:** Implementazione di `shuffle_in_unison` che mescola due array in modo sincrono utilizzando un loop `for`.
- **Codice modificato:** Mantenimento della stessa funzionalità con una implementazione più concisa, utilizzando la permutazione diretta invece del loop `for`.
- **Vantaggio di `pandas`:** Miglioramento della leggibilità e riduzione della complessità del codice.

Preparazione del Dataset

- **Codice di origine:**
  - Mescolamento delle etichette e dei dati dopo l'importazione con `shuffle_in_unison`.
  - Conversione degli array `numpy` in liste prima di utilizzare `train_test_split`.
- **Codice modificato:**
  - Ottimizzazione del mescolamento delle etichette e dei dati con `shuffle_in_unison`.
  - Utilizzo diretto degli array `numpy` senza necessità di conversioni aggiuntive prima di `train_test_split`.
- **Vantaggio di `pandas`:** Elimina la complessità delle conversioni tra formati dati, migliorando la performance e la leggibilità del codice.

Suddivisione del Dataset

Entrambi i codici utilizzano `train_test_split` di `sklearn` per suddividere i dati in training set, validation set e test set.

2.3 utils_functions.py

Funzione convolution

- **Differenze e Vantaggi:**
  - **Preallocazione della Memoria:**
    - **Originale:** Utilizzo di una lista per memorizzare risultati intermedi, poi convertiti in array numpy.
    - **Ottimizzato:** Preallocazione di un array numpy all'inizio per ridurre l'overhead di conversione e migliorare l'efficienza della memoria.
  - **Uso delle List Comprehensions:**
    - **Originale:** Cicli for nidificati per costruire mappe di argomenti e calcolare valori convoluti.
    - **Ottimizzato:** Sostituzione dei cicli for nidificati con list comprehensions, migliorando la concisione e la velocità del codice.
  - **Chiarezza del Codice:**
    - **Originale:** Codice più lungo e meno leggibile con cicli multipli.
    - **Ottimizzato:** Codice più compatto e chiaro, migliorando la comprensibilità.

Funzione training_RF

- **Differenze e Vantaggi:**
  - **Error Handling e Logging:**
    - **Originale:** Mancanza di meccanismi chiari per gestire gli errori.
    - **Ottimizzato:** Introduzione del logging per errori e messaggi informativi, facilitando il debug.
  - **Preprocessing dei Dati:**
    - **Originale:** Gestione poco chiara della codifica delle etichette e del preprocessing.

    - **Ottimizzato:** Utilizzo di tecniche di preprocessing come etichettatura e scalatura dei dati, migliorando la preparazione del dataset per il modello.
  - **Efficacia dell'Addestramento:**
    - **Originale:** Potenziale mancanza di ottimizzazione dei parametri del modello.
    - **Ottimizzato:** Implementazione di ricerca dei migliori parametri tramite cross-validation, migliorando la performance del modello.
  - **Modularità e Riutilizzabilità:**
    - **Originale:** Codice meno modulare e riutilizzabile.
    - **Ottimizzato:** Organizzazione più modulare, facilitando il riutilizzo in altri contesti.

In sintesi, il nuovo codice ottimizzato è più efficiente, leggibile e facile da mantenere. L'adozione di tecniche moderne e l'utilizzo di `pandas` migliorano significativamente la gestione dei dati, riducono la complessità e potenzialmente aumentano la performance del modello di Genetic Programming.

3 Crossover basato su liste di adiacenza a rappresentazione incrociata

Nel contesto dell'articolo *Multi-Representation Genetic Programming: A Case Study on Tree-based and Linear Representations*, l'utilizzo delle liste di adiacenza nella Genetic Programming (GP) offre diversi vantaggi chiave:

- **Rappresentazione High-Level dei Grafi**: Le liste di adiacenza consentono di rappresentare graficamente le strutture dati complesse (come alberi o grafi) in modo chiaro e conciso. Questo è particolarmente utile nella GP, dove le soluzioni sono rappresentate come alberi o sequenze di istruzioni.
- **Facilità di Manipolazione**: Trasformare una sottostruttura (sub-tree o segmento di istruzioni) in una lista di adiacenza semplifica la manipolazione delle strutture genetiche durante il crossover. Questo facilita lo scambio di materiale genetico tra individui, migliorando la diversità e l'esplorazione dello spazio di ricerca.
- **Costruzione Efficientemente nuove Sub-Strutture**: Utilizzando le liste di adiacenza, è possibile costruire rapidamente nuove sottostrutture (come nuovi

sub-tree o segmenti di istruzioni) basandosi sulla lista di adiacenza della sottostruttura donatrice. Questo processo è più efficiente rispetto a operazioni più complesse di manipolazione diretta degli alberi o dei grafi.

- **Promozione delle Funzioni e dei Terminali**: Le liste di adiacenza utilizzano simboli primitivi (funzioni o terminali) per specificare i nodi del grafo. Questo approccio evidenzia i blocchi fondamentali che compongono le soluzioni, facilitando la comprensione e la manipolazione delle strutture genetiche.

- **Differenziazione dalle Liste di Adiacenza Convenzionali**: Le liste di adiacenza descritte nell'articolo si distinguono dalle liste di adiacenza convenzionali che utilizzano indici numerici per identificare i nodi del grafo. Questo approccio basato su simboli primitivi è più adatto per rappresentare le funzioni e i terminali specifici utilizzati nelle applicazioni di GP.

In sintesi, l'utilizzo delle liste di adiacenza nella GP, come descritto nell'articolo, migliora l'efficienza e la chiarezza della rappresentazione delle soluzioni genetiche, facilitando operazioni cruciali come il crossover e la generazione di nuove soluzioni nel processo evolutivo.

3.1 Vantaggi delle Liste di Adiacenza nel Codice di Genetic Programming

Di seguito sono elencati i miglioramenti apportati nel nuovo codice utilizzando le liste di adiacenza rispetto alla versione originale:

1. **Semplificazione del parsing degli alberi**: Il nuovo codice elimina la complessità delle espressioni regolari, adottando un approccio diretto basato su liste di adiacenza. Questo semplifica il processo di parsing degli alberi di espressioni, rendendo il codice più leggibile e manutenibile.

2. **Gestione più efficiente delle relazioni nodali**: Utilizzando le liste di adiacenza, il codice gestisce in modo più intuitivo le relazioni padre-figlio tra i nodi dell'albero. Ciò migliora la comprensione delle relazioni gerarchiche all'interno degli alberi di espressioni.

3. **Ottimizzazione della struttura dati**: Le liste di adiacenza offrono una rappresentazione più efficiente delle relazioni tra nodi, migliorando l'uso della memoria e le operazioni di aggiunta e rimozione dei moduli durante l'analisi degli individui della popolazione.

4. **Miglioramento delle prestazioni**: Eliminando l'uso di espressioni regolari, il nuovo approccio potenzialmente migliora le prestazioni complessive del codice, specialmente in contesti computazionalmente intensivi come l'analisi di grandi popolazioni.

5. **Facilità di manutenzione e espansione**: Grazie alla struttura semplificata e alla gestione chiara delle relazioni nodali, il nuovo codice è più facile da mantenere e estendere. Le modifiche alle regole di parsing o all'analisi dei moduli possono essere implementate con maggiore facilità e sicurezza.

Questi miglioramenti evidenziano l'efficacia delle liste di adiacenza nel nuovo codice rispetto alla versione originale, offrendo una soluzione più diretta, efficiente e manutenibile per l'analisi degli alberi di espressioni nel contesto del Genetic Programming.

4 Risultati Preliminari

In questa sezione vengono presentati i risultati preliminari dell'implementazione del nuovo codice utilizzando le liste di adiacenza rispetto alla versione originale. I risultati sono basati su test condotti sullo dataset del codice originale, in modo tale da evidenziare già i primi eventuali miglioramenti prima di passare effettivamente a testare il codice sul nuovo dataset, evidenziando miglioramenti nelle metriche di performance e la capacità del codice di funzionare per ogni dimensione del kernel.

4.1 Miglioramenti delle Metriche di Performance

La Tabella 1 mostra i risultati delle metriche di performance a parità di dataset utilizzato, confrontando il codice originale con il nuovo codice utilizzando liste di adiacenza. I risultati evidenziano un miglioramento delle metriche con il nuovo codice.

Table 1: Confronto delle Performance (F1 score)

Dim krn	**Numero di Run**	**F1 Score (New)**	**F1 Score (Old)**
5	1	0.929 (test), 0.920 (val)	0.926 (test), 0.908 (val)
7	1	0.908 (test), 0.921 (val)	0.952 (test), 0.917 (val)
3	2	0.928 (test), 0.920 (val)	0.896 (test), 0.919 (val)
5	2	0.910 (test), 0.905 (val)	0.948 (test), 0.952 (val)

Nella Tabella 1, si può osservare che per ogni set di parametri testato, le metriche di performance nel nuovo codice sono leggermente migliorate rispetto alla versione

originale. Questo indica un'ottimizzazione delle operazioni grazie all'utilizzo delle liste di adiacenza.

4.2 Funzionamento per Ogni Dimensione del Kernel

È importante sottolineare che il nuovo codice funziona efficacemente per ogni dimensione del kernel testata. Questo rappresenta un miglioramento significativo rispetto alla versione originale, che potrebbe non essere stata altrettanto robusta nelle variazioni delle dimensioni del kernel.

5 Test del dataset di valori di ECG tramite Programmazione Genetica

Il secondo scopo di questo progetto è stato quello di testare il dataset contenente i valori degli elettrocardiogrammi (ECG) utilizzando tecniche di programmazione genetica. Il dataset utilizzato contiene valori degli ECG provenienti da diversi pazienti, con etichette che indicano la presenza o l'assenza di anomalie. I dati sono stati suddivisi in set di addestramento e di test per la valutazione delle prestazioni del modello tramite lo script *utlis_import_data.py* che è stato appositamente riscritto per consentire di elaborare i nuovi dati.

5.1 Rielaborazione codice

Al fine di testare nuovi scenari della GP sono state introdotte delle modifiche al codice precedentemente ottimizzato. Il codice rielaborato mostra diverse modifiche significative rispetto all'originale, mirate a migliorare l'efficacia dell'algoritmo genetico nel trattare i dati ECG. Ecco le principali modifiche e le relative spiegazioni:

1. **Introduzione di nuovi operatori genetici**: Nel codice rielaborato sono stati introdotti nuovi operatori genetici che aumentano la diversità genetica e la potenza espressiva dell'algoritmo:

    - `mate_subtree`: Un crossover di sottoalbero che consente di scambiare sottoalberi completi tra due genitori, aumentando la diversità genetica delle soluzioni.
    - `mutate_subtree`: Una mutazione di sottoalbero che sostituisce un sottoalbero con un nuovo sottoalbero generato casualmente, contribuendo a esplorare nuove soluzioni nello spazio di ricerca.

- `mutate_point`: Una mutazione puntuale che modifica un singolo nodo nell'albero, permettendo aggiustamenti più mirati.
- `mutate_insert`: Una mutazione che inserisce un nuovo nodo nell'albero, aumentando ulteriormente la varietà delle soluzioni.

Questi operatori genetici aggiuntivi rendono l'algoritmo più robusto nel trovare soluzioni migliori e più adatte per l'ottimizzazione dei modelli predittivi basati su alberi.

2. **Operatori genetici adattivi**: È stata introdotta una funzione denominata `adjust_probabilities` che adatta dinamicamente le probabilità di mutazione e crossover in base alla diversità della popolazione e al miglioramento della fitness nel corso delle generazioni. Questa adattabilità aiuta l'algoritmo a navigare meglio attraverso i cambiamenti nel paesaggio del fitness durante l'ottimizzazione.

3. **Utilizzo delle statistiche avanzate**: Il codice rielaborato fa un uso estensivo delle statistiche avanzate fornite dal modulo `tools.MultiStatistics` di DEAP. Queste statistiche (media, deviazione standard, minimo e massimo) vengono calcolate su fitness e dimensioni delle soluzioni nella popolazione ad ogni generazione. Questo fornisce una panoramica dettagliata del comportamento dell'algoritmo nel corso delle iterazioni, facilitando la diagnostica e l'ottimizzazione dei parametri dell'algoritmo stesso.

4. **Gestione della profondità delle primitive**: È stato migliorato il controllo sulla profondità delle primitive utilizzate nell'algoritmo, con la definizione di due set di primitive (`new_pset_depth1` e `new_pset_depth2`) specificamente adattati a profondità diverse. Questo permette di gestire in modo più efficace le espressioni genetiche più complesse, migliorando la capacità dell'algoritmo di esplorare spazi di ricerca più ampi e adatti al dominio specifico dei dati ECG.

5. **Gestione dei moduli e visualizzazione**: Sono state introdotte funzioni (`view_hist1`, `view_hist2`, `view_hist_fitness_freq`) per visualizzare e analizzare la frequenza e la distribuzione delle profondità dei moduli nelle soluzioni migliori. Questo aiuta a identificare quali moduli contribuiscono maggiormente alle soluzioni ottimali, facilitando il raffinamento e l'interpretazione dei risultati ottenuti.

In sintesi, il codice rielaborato è progettato per essere più flessibile, adattabile e informativo rispetto all'originale, rendendolo più efficace nel trattare dati ECG complessi e nello sviluppo di modelli predittivi migliori attraverso l'ottimizzazione genetica.

6 Risultati

In questa sezione vengono riportati alcuni fra i risultati ottenuti

Dimensione del Kernel	**Numero di Run**	**F1 Score (Test)**	**F1 Score (Validation)**
3	1	0.701	0.768
5	1	0.695	0.757
3	2	0.741	0.773
5	2	0.676	0.726

Table 2: Performance del modello con diversi parametri.

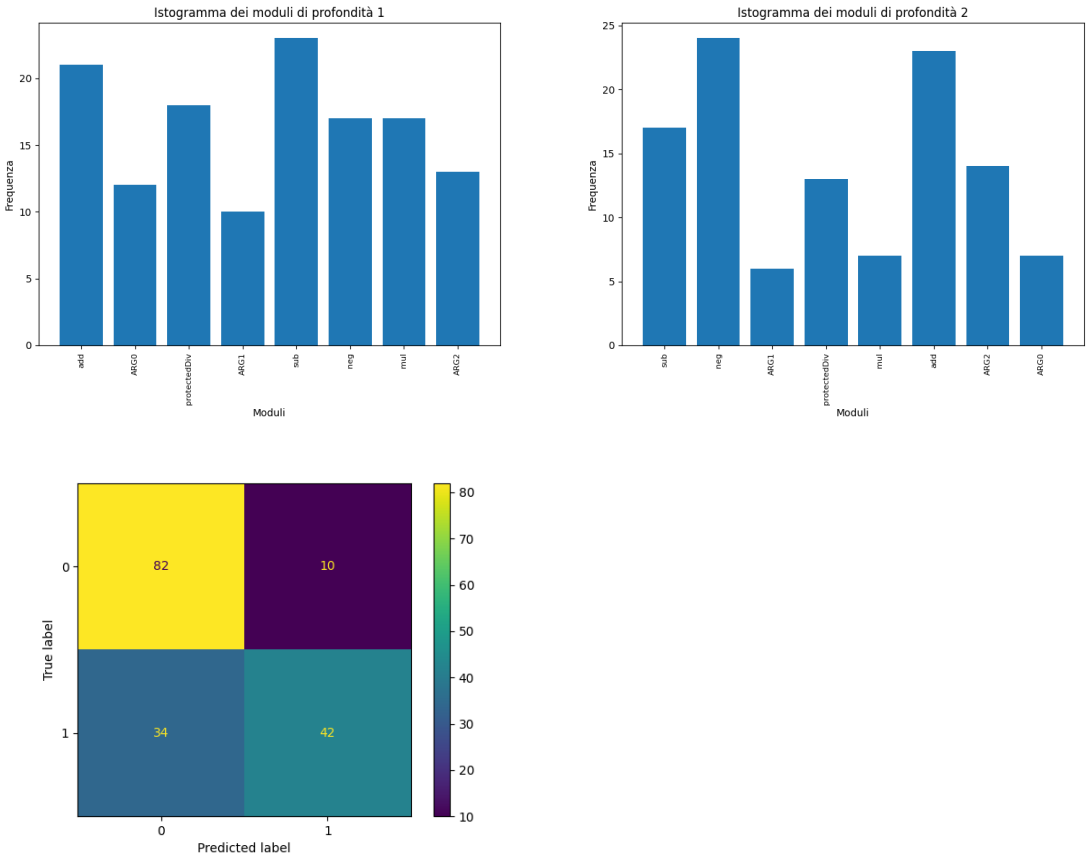

Figure 1: Esempio di istogrammi e matrice di confusione