Fine-tuning di un modello Transformer per il task di textual entailment (RTE) in italiano

Silvia Di Giampasquale, Chiara Segala
silvi.digiampasquale@studio.unibo.it
chiara.segala@studio.unibo.it

Giugno 2025

Sommario

Il presente progetto aveva come obiettivo il riadattamento dello script `run_glue_no_trainer.py`, fornito durante le esercitazioni a lezione, per eseguire il fine-tuning di un modello Transformer su un compito di classificazione testuale in lingua italiana. In particolare, ci siamo concentrate sul task di textual entailment, selezionando come benchmark il dataset e-RTE-3-ITA[1], corrispettivo in italiano del task RTE presente nel benchmark GLUE per l'inglese.

1 Introduzione

Il progetto ha riguardato l'adattamento del processo di fine-tuning di un modello Transformer - originariamente realizzato per i task GLUE in lingua inglese tramite lo script run_glue_no_trainer.py - a un caso di studio equivalente in italiano. A tal fine abbiamo scelto il benchmark e-RTE-3-IT (Recognizing Textual Entailment in italiano), una versione tradotta del dataset RTE3 messa a disposizione da GLUE per il task di Textual Entailment Recognition (o Natural Language Inference, NLI).

Il compito di Recognizing Textual Entailment consiste nel determinare se il contenuto di un frammento di testo ("ipotesi") è logicamente implicato, contraddetto o neutrale rispetto a un altro frammento ("premessa"). In pratica, il modello deve stabilire se esiste una relazione di derivazione logica fra i due enunciati.

Per questo progetto abbiamo utilizzato le risorse messe a disposizione dalla Textual Entailment Resource Pool - l'iniziativa di ACL per la raccolta di dataset RTE (`https://aclweb.org/aclwiki/Textual_Entailment_Resource_Pool`) - in particolare la campagna RTE3, di cui è disponibile una versione

[1] `https://github.com/andreazaninello/e-rte3-it`

italiana. Tale risorsa offre un numero consistente di esempi annotati, consentendo di addestrare e valutare in modo robusto il modello su un'ampia varietà di coppie premessa-ipotesi.

2 Pre-processing

Prima di adattare il processo di fine-tuning a un nuovo dataset, i file di e-RTE-3-IT sono stati convertiti dal formato XML a CSV. Questo passaggio è necessario perché lo script `run_glue_no_trainer.py` - fornito nella repository di Hugging Face - si basa su file tabellari (CSV o JSON) con colonne standardizzate (sentence1, sentence2, label), sui quali è possibile applicare operazioni di normalizzazione e splitting.

La conversione si è articolata nei seguenti passi:

1. Lettura e decodifica: I file XML sono stati aperti in modalità binaria e decodificati con l'encoding utf-8-sig, in modo da rimuovere eventuali BOM e sostituire automaticamente i caratteri non validi.

2. Parsing e costruzione del DataFrame: Con xml.etree.ElementTree si è navigata la struttura ad albero e, per ciascun elemento `<pair>`, si sono estratti:

   - il testo della premessa (`<t>`)
   - il testo dell'ipotesi (`<h>`)
   - l'attributo entailment ("yes", "no", "unknown")

   Questi dati sono stati raccolti in un pandas.DataFrame con le colonne sentence1, sentence2 e label.

3. Normalizzazione delle etichette: La colonna label è stata resa uniforme (tutto minuscolo, rimozione di spazi) e le tre possibili stringhe sono state mappate su interi:

   - "yes" $\rightarrow$ 0
   - "no" $\rightarrow$ 1
   - "unknown" $\rightarrow$ 2

4. Split stratificato: Il DataFrame derivato da DEV.xml è stato suddiviso in due sottoinsiemi (80% train, 20% validation) mantenendo la stessa distribuzione delle classi grazie allo stratify sulla colonna delle etichette.

5. Esportazione in CSV: Infine, train.csv e validation.csv sono stati creati con DataFrame.to_csv(index=False) e salvati nella directory di output, pronti per essere utilizzati dallo script di fine-tuning in formato tabellare.

3 Codice

Gli step del codice sono elencati di seguito in ordine di esecuzione:

1. Montaggio di Google Drive in Colab utile per:
   - Caricare lo script `run_glue_no_trainer_italiano.py`;
   - Salvare i file CSV di training e validation;
   - Conservare i checkpoint del modello e tutti i risultati del fine-tuning sul development set.
2. Installazione delle librerie necessarie:
   - transformers (4.52.4): libreria Hugging Face per modelli Transformer;
   - datasets (3.6.0): toolkit Hugging Face per gestire e preprocessare dataset NLP;
   - accelerate (1.8.1): supporto a training distribuito e uso di GPU/TPU in PyTorch;
   - evaluate (0.4.4): libreria Hugging Face per il calcolo di metriche (ad es. accuracy);
   - lxml (5.4.0): parser XML/HTML (per leggere i file .xml di e-RTE-3-IT);
   - pandas (2.3.0): strumento per manipolare tabelle di dati (DataFrame);
   - scikit-learn (1.7.0): fornisce funzioni di utilità, in particolare `train_test_split`.
3. Importazione delle librerie per la conversione XML→CSV:
   - xml.etree.ElementTree (versione 1.3.0): per leggere e navigare l'albero dei file XML;
   - pandas (2.3.0): per trasformare i dizionari estratti dall'XML in DataFrame tabellari e normalizzare le etichette;
   - csv (versione integrata in Python 3.11.13): per eventuali operazioni manuali di input/output su file CSV;
   - os (versione integrata in Python 3.11.13): per costruire e verificare percorsi, creare directory di output, ecc.
4. Definizione dei percorsi di lavoro:
   - drive_root: variabile che memorizza il percorso di base della cartella di progetto in Google Drive;
   - dataset_folder: percorso completo alla sottocartella contenente i file XML di e-RTE-3-IT (DEV.xml, TEST.xml);

- script_path: percorso allo script di fine-tuning `run_glue_no_trainer_italiano.py` che verrà eseguito in Colab.

5. Pre-processing del dataset e-RTE-3-IT:
   - `parse_rte_xml`: definizione della funzione, utile per leggere, decodificare e convertire un file .xml in DataFrame, estraendo le coppie Testo-Ipotesi (T-H) e le relative etichette di inferenza;
   - dev_df: variabile contenente il DataFrame ottenuto applicando la funzione `parse_rte_xml` al file DEV.xml;
   - Normalizzazione delle etichette in dev_df: rimozione di spazi e conversione in minuscolo;
   - Stampa di controllo per visualizzare la distribuzione delle etichette nel DataFrame;
   - Mappatura delle etichette testuali in valori numerici, in particolare: yes $\rightarrow$ 0, no $\rightarrow$ 1, unknown $\rightarrow$ 2;
   - Stampa di controllo post-mappatura per verificare la correttezza e la coerenza della distribuzione delle etichette;
   - Suddivisione del DataFrame dev_df in due sottoinsiemi, train e validation, con proporzione 80-20 e stratificazione basata sulla distribuzione delle etichette;
   - Conversione dei DataFrame ottenuti in file csv;
   - Creazione della cartella di output "e-rte3-it-model", se non già esistente;
   - Copia dei file train.csv e validation.csv all'interno della cartella di output tramite comandi di shell !cp.

3.1 Fase di addestramento del modello

L'addestramento del modello è stato eseguito utilizzando lo script Python `run_glue_no_trainer_italiano.py`, una versione modificata dello script `run_glue_no_trainer.py`, presentato a lezione. In particolare, sono state rimosse le parti relative alla gestione del parametro `task_name`, non rilevante in questo contesto, poiché l'unico task considerato è RTE. Inoltre, sono state aggiunte funzionalità specifiche per il progetto, come la possibilità di salvare automaticamente i checkpoint al termine di ogni epoca e il modello con le migliori prestazioni su validation nella cartella `best_model`. Lo script fa uso della libreria `Transformers` di Hugging Face per la gestione del modello pre-addestrato e del ciclo di addestramento. Il modello selezionato, dopo una fase esplorativa su più alternative, è `dbmdz/bert-base-italian-xxl-cased`, una versione di BERT specificamente ottimizzata per la lingua italiana e case-sensitive. Come file di input vengono forniti train.csv e validation.csv, contenenti rispettivamente il training set e il validation set.

Gli iperparametri utilizzati durante l'addestramento sono i seguenti:

- `per_device_train_batch_size` e `per_device_eval_batch_size`: entrambi fissati a 8, in quanto rappresentano un buon compromesso tra stabilità e utilizzo della memoria GPU disponibile;
- `learning_rate`: impostato a 1×10^{-5}. Sono stati effettuati tentativi anche con 2×10^{-5} e 5×10^{-6}: il primo portava a overfitting già dopo poche epoche, mentre il secondo risultava troppo lento, impedendo un apprendimento efficace;
- `num_train_epochs`: fissato a 10. Sono stati inizialmente testati valori più bassi (4 e 6 epoche), ma 10 è risultato il valore con cui si sono ottenute le migliori prestazioni;
- `max_length` e `pad_to_max_length`: le sequenze vengono troncate o completate tramite padding fino a un massimo di 128 token. È stato testato anche il valore 256, ma non ha portato a miglioramenti significativi in termini di accuratezza, a fronte di un maggiore consumo di memoria e tempo;
- `seed`: impostato a 31 per garantire la riproducibilità degli esperimenti.

Durante l'addestramento, il modello salva automaticamente un checkpoint al termine di ogni epoca (`checkpointing_steps = epoch`), e al termine del processo viene ricaricata la versione migliore sulla base della metrica di accuratezza (`metric_for_best_model = accuracy`, `load_best_model_at_end = True`, `greater_is_better = True`). I risultati e i file di modello vengono salvati nella directory `output_dir`, precedentemente creata.

3.2 Test del modello

Prima di procedere alla fase di inferenza sul test set per la valutazione finale del modello, è stato necessario applicare sul file TEST.xml lo stesso preprocessing effettuato sul development set. In particolare, si è utilizzata la funzione `parse_rte_xml` per convertire il file XML in un DataFrame, dopodiché si è proceduto alla normalizzazione e alla mappatura delle etichette testuali nei corrispondenti valori numerici, secondo la stessa logica già applicata in fase di training. Sono stati inoltre eseguiti dei print di controllo per verificare la distribuzione delle classi prima e dopo la mappatura, al fine di assicurarsi della correttezza del processo. Infine, il test set così elaborato è stato salvato in formato CSV e copiato all'interno della cartella "e-rte3-it-model".

3.2.1 Inferenza e valutazione sul test set

Per la valutazione finale delle prestazioni del modello addestrato, è stata eseguita l'inferenza sul test set utilizzando il modello salvato nella directory "best_model", corrispondente alla versione con le migliori performance ottenute sul validation set durante il training. Il modello è stato caricato insieme al

corrispondente tokenizer tramite la funzione `from_pretrained` dalla libreria `Transformers`.

Il test set, precedentemente salvato in formato CSV, è stato caricato utilizzando la funzione `load_dataset` della libreria `datasets`. Successivamente, è stata eseguita la tokenizzazione delle coppie di frasi tramite una funzione di pre-processing, che applica troncamento e padding fino a un massimo di 128 token, per mantenere coerenza con la fase di addestramento.

I dati preprocessati sono stati poi convertiti in tensori PyTorch e caricati tramite un `DataLoader` con `batch_size` pari a 32 per ottimizzare l'elaborazione in batch durante l'inferenza. In fase di predizione, il modello è stato impostato in modalità `eval()` e sono stati disabilitati i gradienti con `torch.no_grad()`, per velocizzare l'inferenza e ridurre il consumo di memoria.

Le predizioni sono state ottenute selezionando, per ciascun esempio, la classe con il logit massimo. Infine, l'accuratezza è stata calcolata confrontando le etichette predette con quelle reali, utilizzando la metrica standard `accuracy` fornita dal modulo `evaluate`.

Il modello ha raggiunto un'accuratezza finale pari a **0.64375** sul test set, superando sia la baseline casuale (pari a circa 0.3333, nel caso di una distribuzione uniforme su tre classi), sia una strategia basata sulla predizione costante della classe più frequente, che avrebbe prodotto un'accuratezza di circa 0.485.

4 Conclusioni

Al termine del progetto è stato portato a termine il fine-tuning di un modello Transformer sul task di Recognizing Textual Entailment in italiano, utilizzando il dataset e-RTE-3-ITA. Lo script `run_glue_no_trainer.py` è stato modificato per adattarsi al task specifico, rimuovendo le componenti superflue e aggiungendo funzionalità per il salvataggio dei checkpoint e del modello ottimale. Sono inoltre state implementate operazioni di pre-processing per convertire il formato del dataset in quello compatibile con GLUE. In fase di valutazione, il modello ha ottenuto prestazioni superiori rispetto alle baseline considerate.