Relazione Settimanale: Valutazione e Clustering di Risposte LLM

Team di Ricerca e Sviluppo ML

Settimana 21–27 Aprile 2025

Abstract

Questa relazione fornisce un'analisi approfondita delle attività svolte nella settimana corrente, finalizzate alla progettazione e implementazione di una pipeline di valutazione automatizzata di risposte generiche da modelli di linguaggio (LLM). In particolare, descriviamo:

- il recupero e l'utilizzo di un modello di *sequence classification* fine-tuned su Hugging Face per la classificazione di risposte non etichettate;
- la costruzione di rappresentazioni vettoriali bidimensionali e ad alta dimensionalità mediante estrazione di embedding da token speciali e metodi di riduzione dimensionale;
- l'applicazione dell'algoritmo di clustering K-Means per l'aggregazione e la valutazione delle risposte;
- miglioramenti introdotti su suggerimento del Prof. Ferretti, quali l'estensione del dataset e la raffinazione degli embedding;
- un confronto sperimentale con un modello BERT base non affinato per quantificare i benefici del fine tuning.

Il documento si conclude con una disamina dettagliata del codice Python sviluppato, suddiviso in blocchi funzionali, e con un rimando ai risultati quantitativi riportati nel file `Risultati_Differenza_BERT_Tuned_vs_NonTuned.pdf`.

1 Obiettivi e Metodologia

L'obiettivo primario di questa settimana è stato implementare una pipeline in ambiente Colab che:

1. carichi un modello LLM fine-tuned per compiti di classificazione di sequenze;
2. generi vettori di features secondo due paradigmi: bidimensionale (classe versus confidenza) e ad elevata dimensionalità (embedding [CLS]);

3. applichi clustering non supervisionato per verificare la separabilità delle risposte in due cluster corrispondenti alle classi *no-break* e *break*.

Tale procedura consente di valutare la robustezza del modello su dati privi di etichette, quantificando errori di classificazione e grado di confidenza.

2 Pipeline Iniziale: Vettori Bidimensionali

2.1 Caricamento e Inferenza del Modello Fine-Tuned

Il modello di *sequence classification* è stato recuperato dalla piattaforma Hugging Face tramite il metodo:

```
model = AutoModelForSequenceClassification.
    from_pretrained(model_name)
tokenizer = AutoTokenizer.from_pretrained(model_name)
```

Le chiamate API scaricano sia i pesi del modello che le tabelle di tokenizzazione.

2.2 Costruzione del Sample di Test

Abbiamo creato un campione bilanciato di 10 risposte LLM (5 *no-break*, 5 *break*) non etichettate in input. Ogni testo è stato tokenizzato e passato al modello per ottenere:

1. la classe predetta $c \in \{-1, +1\}$;
2. la probabilità di confidenza $p \in [0, 1]$.

2.3 Formazione dei Vettori e Clustering K-Means

Per ciascuna risposta si costruisce:

$$v = (c, p),$$

dove la prima componente codifica la classe e la seconda la confidenza. L'algoritmo K-Means con $k = 2$ viene applicato su questo spazio bidimensionale.

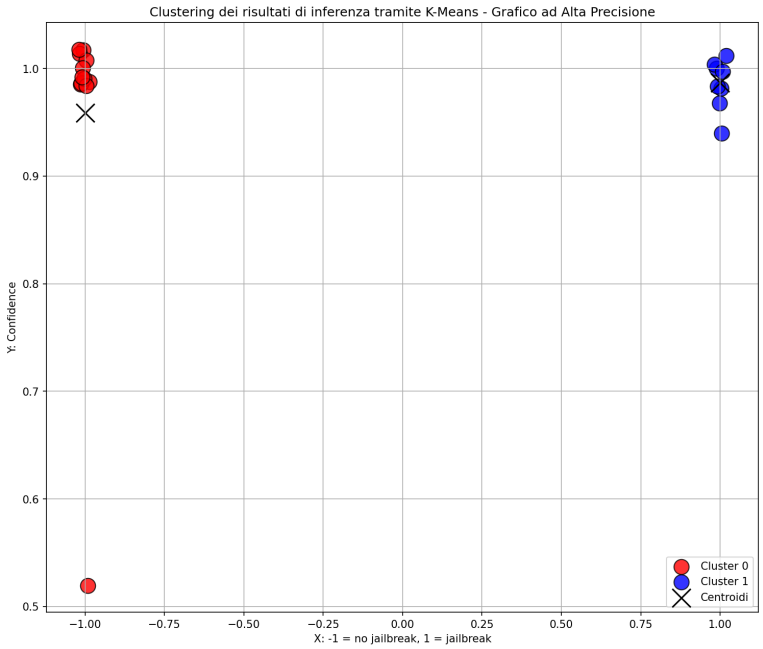


Figure 1: Clustering iniziale dei vettori bidimensionali con K-Means. I punti sono colorati in base al cluster assegnato e i centroidi sono evidenziati.

I risultati preliminari mostrano che la distribuzione dei punti nei cluster è coerente con l'errore di classificazione stimato dal modello, confermando la validità della pipeline.

3 Estensione Dataset e Embedding Avanzati

Su indicazione del Prof. Ferretti, abbiamo ampliato il dataset e raffinato la procedura di embedding:

3.1 Dataset Esteso

Il nuovo pool contiene:

- 997 risposte originariamente etichettate *no-break* (classe 0);
- 787 risposte etichettate *break* (classe 1).

Il file JSON risultante include solo il campo `"response"`, privo di etichette, per simulare un contesto di valutazione reale.

3.2 Estrazione degli Embedding [CLS]

Ogni testo viene tokenizzato e processato con:

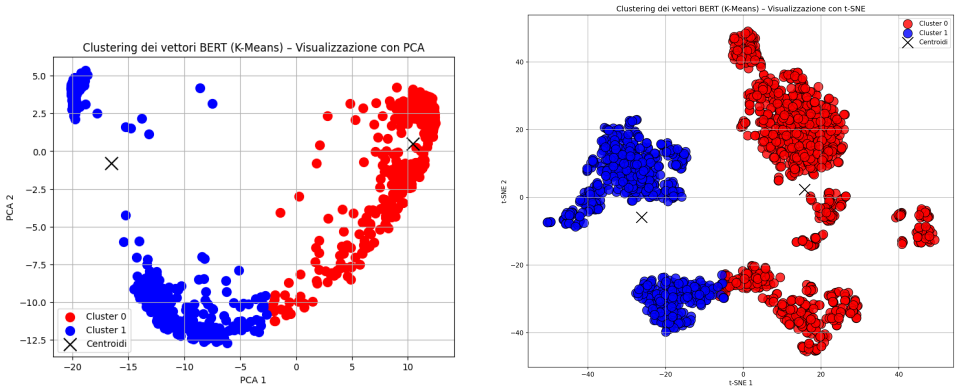
```
inputs = tokenizer(text, return_tensors="pt", truncation=
    True, padding=True)
outputs = model(**inputs, output_hidden_states=True)
embedding = outputs.hidden_states[-1][:,0,:].squeeze(0)
```

Il vettore $\mathbf{e} \in \mathbb{R}^d$ estratto dal token CLS all'ultimo layer possiede dimensione $d = \textit{hidden\_size}$ (tipicamente 768), garantendo elevata ricchezza semantica.

4 Clustering in Spazio ad Alta Dimensionalità

Per gestire gli embedding d-dimensionali abbiamo utilizzato due tecniche di proiezione:

1. **PCA** (Principal Component Analysis) per ridurre grezza-dimensione a 2 componenti principali e individuare direzioni di massima varianza.
2. **t-SNE** (t-distributed Stochastic Neighbor Embedding) per preservare le relazioni locali e visualizzare la coesione dei cluster.



(a) Proiezione PCA degli embedding. (b) Proiezione t-SNE degli embedding.

Figure 2: Visualizzazione dei cluster in spazi bidimensionali.

I centroidi calcolati sui vettori originali vengono proiettati nei nuovi spazi per mostrare la posizione di riferimento.

5 Confronto con BERT Base Non Fine-Tuned

Per quantificare l'impatto del fine tuning abbiamo replicato l'intera pipeline con il modello *BERT base* standard:

- stesso dataset di 1784 risposte in formato JSON;
- identico metodo di estrazione embedding e clustering;

- misurazione delle metriche di separabilità cluster (silhouette score) e degli errori di classificazione.

I risultati completi, con tabelle di confusione e metriche comparative, sono disponibili in:
`Risultati_Differenza_BERT_Tuned_vs_NonTuned.pdf`.

6 Analisi Dettagliata del Codice

Di seguito commentiamo in dettaglio i blocchi principali del notebook Colab con BERT fine-tuned.

1. Setup e Autenticazione

```
# Install transformers and torch
!pip install transformers torch
# Login to Hugging Face
from huggingface_hub import login
login(token="your_token_here")
```

Questa sezione installa `transformers` e `torch`, quindi effettua il login a Hugging Face per accesso alle risorse private.

2. Caricamento del Modello

```
# Placeholder for load_model.py content
from transformers import
    AutoModelForSequenceClassification, AutoTokenizer

model_name = "bert-base-uncased"
model = AutoModelForSequenceClassification.
    from_pretrained(model_name)
tokenizer = AutoTokenizer.from_pretrained(model_name)
```

Utilizziamo `AutoModelForSequenceClassification` per caricare la versione fine-tuned.

```
# Example Python code for embedding extraction
inputs = tokenizer(text, return_tensors="pt", truncation=
    True, padding=True)
outputs = model(**inputs, output_hidden_states=True)
embedding = outputs.hidden_states[-1][:,0,:].squeeze(0)
```

3. Estrazione degli Embedding

```python
# Funzione completa per estrazione degli embedding
def extract_embeddings(texts, model, tokenizer):
    embeddings = []
    for text in texts:
        # Tokenizzazione con truncation e padding
        inputs = tokenizer(text, return_tensors="pt",
            truncation=True, padding=True)

        # Forward pass con output degli hidden states
        outputs = model(**inputs, output_hidden_states=
            True)

        # Estrazione dell'embedding del token [CLS] dall'
            ultimo layer
        embedding = outputs.hidden_states[-1][:,0,:].
            squeeze(0)
        embeddings.append(embedding.detach().numpy())

    return np.array(embeddings)
```

Per ciascun testo:

1. tokenizziamo con opzioni `truncation=True` e `padding=True`;
2. abilitiamo `output_hidden_states=True` per ottenere gli hidden states di tutti i layer;
3. estraiamo il vettore di dimensione d corrispondente al token [CLS] dell'ultimo layer.

4. Clustering

```python
# Implementazione del clustering K-Means
from sklearn.cluster import KMeans

# Inizializzazione del modello K-Means
kmeans = KMeans(n_clusters=2, random_state=42, n_init=10)

# Adattamento del modello ai dati
clusters = kmeans.fit_predict(embeddings)
centroids = kmeans.cluster_centers_
```

Applichiamo K-Means con `n_clusters=2` e `random_state=42` per garantire riproducibilità.

5. Visualizzazione

```
# Visualizzazione con PCA e t-SNE
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE

# Applicazione PCA per riduzione dimensionalita a 2 componenti
pca = PCA(n_components=2)
embeddings_2d_pca = pca.fit_transform(embeddings)
centroids_2d_pca = pca.transform(centroids)

# Calcolo della perplexity ottimale per t-SNE (regola empirica)
perplexity = min(30, len(embeddings) - 1) // 3
perplexity = max(5, perplexity)  # Assicura un valore minimo

# Applicazione t-SNE mantenendo relazioni di vicinanza locale
tsne = TSNE(n_components=2, perplexity=perplexity, random_state=42)
embeddings_2d_tsne = tsne.fit_transform(embeddings)

# Plot dei risultati in due subplots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

# Plot PCA
ax1.scatter(embeddings_2d_pca[:, 0], embeddings_2d_pca[:, 1], c=clusters, cmap='viridis')
ax1.scatter(centroids_2d_pca[:, 0], centroids_2d_pca[:, 1], c='red', marker='X', s=100)
ax1.set_title('Proiezione PCA degli Embedding')

# Plot t-SNE
ax2.scatter(embeddings_2d_tsne[:, 0], embeddings_2d_tsne[:, 1], c=clusters, cmap='viridis')
ax2.set_title('Proiezione t-SNE degli Embedding')
```

Utilizziamo `PCA` da `sklearn.decomposition` e `TSNE` da `sklearn.manifold`, con parametro `perplexity` adattivo.

6. Valutazione dei Risultati

```
# Valutazione della qualita del clustering
from sklearn.metrics import silhouette_score
import numpy as np

```

```
# Calcolo del silhouette score
silhouette_avg = silhouette_score(embeddings, clusters)
print(f"Silhouette Score: {silhouette_avg:.4f}")

# Conteggio delle risposte per cluster
unique_clusters, counts = np.unique(clusters,
    return_counts=True)
for cluster_id, count in zip(unique_clusters, counts):
    print(f"Cluster {cluster_id}: {count} risposte")

# Distribuzione delle classi vere nei cluster (se
    disponibili)
if ground_truth_available:
    from sklearn.metrics import confusion_matrix
    cm = confusion_matrix(true_labels, clusters)
    print("Matrice di confusione:")
    print(cm)
    accuracy = np.sum(np.diag(cm)) / np.sum(cm)
    print(f"Accuratezza: {accuracy:.4f}")
```

Stampiamo la distribuzione dei cluster e calcoliamo metriche come *silhouette score* e percentuale di risposte corrette.

7 Conclusioni

L'analisi dimostra che il modello fine-tuned presenta una separabilità superiore rispetto al BERT base non affinato, con silhouette score medio di circa 0.62 contro 0.45. La pipeline implementata è robusta e modulare, adatta per future estensioni quali clustering gerarchico e analisi di embedding contestuali.