# Ingegneria dei Dati:
# Data Integration

Mattia Micaloni, Carlo Proserpio, Alessandro Di Girolamo

Project repository:

https://github.com/Xhst/data-engineering-projects/tree/main/project-5

**Keywords**: Ingegneria dei Dati, Data Integration, Record Linkage, Entity Resolution, Schema matching, Blocking, Pairwise matching

## 1. Introduzione

Il progetto riguarda il problema di integrazione di dati di compagnie provenienti da sorgenti differenti ed eterogenee. La Data Integration è un processo che comprende tre diversi passaggi: Schema Alignment, Record Linkage e Data Fusion. In questo progetto ci siamo concentrati su Schema Alignment e Record Linkage, cercando di usare e confrontare metodi diversi, vecchi e nuovi.

## 2. Caratteristiche Sorgenti

In totale sono state integrate 16 sorgenti, molto eterogenee tra loro. Le diversità sono sia su come sono presentati i dati (formati diversi: csv, json, jsonl, xls); sia sul contenuto, ovvero i campi e i valori.
L'unico campo presente, seppur con nomi diversi, in tutte le sorgenti è quello del nome dell'azienda.

## 3. Schema Alignment

Per realizzare lo schema mediato abbiamo pensato di usare una nostra soluzione basata sull'uso di LLM e LM con una supervisione umana finale. Abbiamo fatto generare a *LLaMa 3.3 70b* delle descrizioni per ogni campo. Utilizzando il seguente prompt:

```
'''You will receive a name of a field, with some values associated,
of a dataset and you have to provide a description for that field.
The dataset is about Companies.
In your response you must only include a brief description of the field,
```

```
maximum 20 words.
Please provide a description for the field:
{field}

Example of values for the field:
{values}'''
```

fornendo il nome del campo e 5 esempi per quel campo. Una volta generate le descrizioni abbiamo fatto l'embedding di ogni descrizione utilizzando il modello *paraphrase-MPNet-base-v2* e abbiamo poi fatto un clustering utilizzando *HDBSCAN* (*Hierarchical Density-Based Spatial Clustering of Applications with Noise*). Con il clustering abbiamo raggruppato i campi con descrizioni semanticamente simili da cui poi siamo partiti per selezionare i campi dello schema mediato. Abbiamo revisionato manualmente i cluster e assegnato un nome a ciascuno. Abbiamo selezionato un totale di 24 campi per lo schema mediato con il nome del campo corrispondente al nome del cluster. In alcuni casi abbiamo separato un cluster in più campi.
Lo schema mediato comprende i seguenti campi: *company_name, company_type, link, number_of_employees, founded, founders, market_cap, annual_net_income_usd, total_assets_usd, total_equity_usd, total_liabilities_usd, annual_result_year_ending, industry, address,rank, ceo, telephone, facebook, twitter, instagram, pinterest, postal_code, investors, notes*. In totale sono presenti 65895 valori.
Il tempo richiesto per generare le descrizioni, fare gli embedding e il clustering su un totale di 132 campi (tra tutte le sorgenti) è stato di circa 6.3 minuti (include i tempi di attesa tra le richieste, che sono sommati a circa 2.2 minuti), mentre il tempo umano richiesto per dare i nomi ai blocchi e fare i mapping per lo schema mediato è stato di circa 10 minuti.

# 4. Blocking

Per il blocking abbiamo utilizzato tecniche basate su *Locality Sensitive Hashing* (*LSH*). LSH è una tecnica usata principalmente in ambito big data, con vettori di dimensione elevata. Per questo motivo, viene usata una prima funzione hash (MinHash) per ridurre la dimensionalità dei vettori, mantenendo però «invariata» la similarità, nel nostro caso la Jaccard. I vettori vengono poi confrontati con varie funzioni hash «classiche» e ogni elemento dei vettori viene inserito in un bucket in base a «quanto è simile ad altri» (con un threshold). La precisione e i tempi di calcolo variano in base al numero di funzioni hash e ad altri parametri legati al threshold.
In entrambi i metodi abbiamo fatto il blocking sul campo *company_name*, pre-processato i valori per standardizzarli il più possibile: rimosse le porzioni comprese tra parentesi; separato il testo scritto in PascalCase; rimossi accenti, spazi extra, punteggiatura e caratteri speciali ed infine rimosse parole "comuni" (esempio: company, inc, srl, corp..). Dopo la prima fase di pulizia abbiamo creato i token per LSH. Nel primo metodo i token corrispondono alle parole mentre nel secondo i token corrispondono ai bi-grammi. Inoltre,

per entrambe le tecniche abbiamo due varianti. Infatti possiamo aggiungere al vettore generato l'acronimo del nome. Da notare come questo possa aumentare la recall ma diminuire la precision. Il tempo richiesto per fare il blocking sulle parole con LSH è stato di circa 2.5 minuti, mentre per i bi-grammi con LSH è stato di circa 3.4 minuti.

# 5. Pairwise Matching

Lo step finale del pairwise matching è stato svolto con tre diversi metodi. Il primo utilizzando la libreria recordlinkage, in particolare con il metodo di *Jaro-Winkler* che misura la *edit distance* tra due stringhe. Gli altri due metodi invece utilizzano tecniche di *deep learning* e sono *DITTO* e *DeepMatcher*. Sia DITTO che DeepMatcher hanno richiesto una fase di addestramento e fine tuning dei parametri, i dati di addestramento sono per 1/4 creati a mano a partire dal dataset e per 3/4 sintetici generati utilizzando ChatGPT4o e poi controllati manualmente.

## 5.1 Ditto

Ditto è un sistema che offre blocking e pairwise matching basato sull'uso di modelli di linguaggio pre-addestrati della famiglia *BERT* (per il nostro progetto abbiamo usato *distilBERT*). Abbiamo modificato il codice affinché si allineasse con i nostri requisiti e formati di file.
L'addestramento consiste in un fine-tuning del LM scelto a partire dal training set. Per quanto rigurda la predizione invece, Ditto esegue un fine-tuning del threshold usando il validation set e poi esegue le predizioni.

# 6. Valutazione

La valutazione è stata eseguita sulla base di un ground truth costruito a mano contenente 150 coppie positive uniche (due elementi di due coppie diverse non sono mai in match). Sono state inserite nel GT anche coppie *difficili* con acronimi e casi particolari.

Per la valutazione del sistema si sono quindi eseguite le varie tecniche di blocking sul GT e successivamente sono state create tutte le coppie possibili a partire dai blocchi ottenuti.
Queste ultime sono state l'input dei vari metodi di pairwise matching, su cui abbiamo calcolato *Precision*, *Recall*, *F-measure* e tempi di calcolo (CPU per Jaro-Winkler e GPU per gli altri).
Nella tabella 1 sono mostrati i risultati della valutazione usando varie combinazioni di metodi di blocking e pairwise matching.

| | | Precision | Recall | F-Measure | Tempo |
|---|---|---|---|---|---|
| **LSH Words** | **Jaro-Winkler** | 0.96 | 0.94 | 0.95 | 27s (CPU) |
| | **DeepMatcher** | 0.99 | 0.84 | 0.91 | 3.2m (GPU) |
| | **DITTO** | 0.97 | 0.98 | 0.97 | 3.4m (CPU) |
| **LSH Bi-Grams** | **Jaro-Winkler** | 0.96 | 0.94 | 0.95 | 45s (CPU) |
| | **DeepMatcher** | 0.99 | 0.83 | 0.9 | 17.3m (GPU) |
| | **DITTO** | 0.97 | 0.98 | 0.97 | 15.6m (CPU) |

Tabella 1: Precision, Recall, F-Measure e tempo di calcolo per le diverse combinazioni tra i metodi di Blocking e di Pairwise Matching.

Si può notare come Jaro-Winkler offra prestazioni elevate senza una grande perdita di qualità, mentre DeepMatcher reputa troppo spesso le coppie come negative, probabilmente perché non è stato effettuato un tuning sul threshold. Ditto offre i risultati migliori, mentre per quanto riguarda i tempi si può notare come i due modelli abbiano risultati simili con la differenza che Ditto è stato fatto girare su una Gamma *GeForce RTX 4060* (8GB) mentre DeepMatcher su *NVIDIA GeForce GTX 1660 SUPER* (6GB).