Università degli Studi di Perugia
Dipartimento di Matematica e Informatica

Tesi di Laurea Triennale in Informatica

Approccio Deep Learning al Topic Modeling:

Analisi di annunci di lavoro tramite BERT

Relatrice

Prof. Valentina Poggioni

Laureando

Tommaso Bagiana

Anno Accademico 2024-2025

Lo dico solo perché a volte si incontra un uomo, non dirò un eroe... perché, che cos'è un eroe? Ma a volte si incontra un uomo, e sto parlando di Drugo, a volte si incontra un uomo che è l'uomo giusto al momento giusto nel posto giusto, là dove deve essere. E quello è Drugo, a Los Angeles. E anche se quell'uomo è un pigro, e Drugo lo era di sicuro, forse addirittura il più pigro di tutta la contea di Los Angeles, il che lo mette in competizione per il titolo mondiale dei pigri... Ma a volte si incontra un uomo... a volte si incontra un uomo... Ah! Ho perso il filo del discorso! Bah, al diavolo! È più che sufficiente come presentazione.

- Lo Sconosciuto, Il Grande Lebowsky

Indice

Capitolo 1

Introduzione

Panoramica introduttiva del progetto, motivazione della ricerca e obiettivi principali.

1.1 Cos'è il Topic Modeling

Descrivere in termini generali l'obiettivo del topic modeling, le tecniche classiche e i vantaggi rispetto ad approcci basati su keyword.

1.2 Analisi del paper *The Dynamics of Data Analytics Job Skills: a Longitudinal Analysis*

Almgerbi, De Mauro, Kahlawi, Poggioni (2025) presentano uno studio longitudinale sull'evoluzione delle competenze richieste negli **annunci di lavoro in ambito Data Analytics** tra il 2019 e il 2023. La loro ricerca mira a catturare le dinamiche temporali nella domanda di competenze applicando il topic modeling basato su **Latent Dirichlet Allocation (LDA)** a grandi corpora di annunci di lavoro online (Online Job Advertisements, OJA).

1.2.1 Obiettivi e motivazioni

Gli autori sottolineano come la rapida trasformazione digitale e la diffusione dei Big Data abbiano generato una crescente domanda di professionisti competenti in statistica, programmazione, tecnologie cloud e intelligenza artificiale. Tuttavia, la crescita dell'offerta formativa accademica e professionale non procede allo stesso ritmo delle esigenze dell'industria. Poiché ruoli come *Data Scientist*, *Business Analyst* e *Big Data Engineer* presentano competenze sovrapposte, lo studio adotta il termine più ampio **Data Analytics** per includere tutti questi ambiti professionali. Gli annunci di lavoro online vengono quindi trattati come una fonte affidabile e in tempo reale per monitorare l'andamento del mercato del lavoro.

1.2.2 Novità nella letteratura

Sebbene numerosi studi abbiano applicato tecniche di text mining o topic modeling agli annunci di lavoro, questo lavoro è il **primo a condurre un'analisi longitudinale su più anni**. La letteratura precedente si è concentrata su settori specifici (ad esempio marketing o IT) oppure ha impiegato strumenti proprietari e corpora statici. La metodologia proposta offre invece un quadro replicabile per **monitorare l'evoluzione delle competenze nel tempo**, fornendo indicazioni operative per le risorse umane e per le politiche formative.

1.2.3 Metodologia

Lo studio segue un processo articolato in quattro fasi:

1. **Raccolta dati:** gli annunci sono stati estratti da diversi siti web nel 2019 e nel 2023 utilizzando le stesse sei keyword (*big data, data science, business intelligence, data mining, machine learning, data analytics*), ottenendo un dataset bilanciato di 16 060 annunci (8 030 per anno).

2. **Pre-processing:** accurata pulizia del testo (rimozione di HTML e punteggiatura, filtraggio di stopword e n-gram), stemming, esclusione dei testi non in lingua inglese o troppo brevi e costruzione del dizionario/corpus per il topic modeling.

3. **Topic modeling:** esecuzione di più run LDA con $k = 5$–20 topic; il numero ottimale ($k = 12$) è stato selezionato combinando punteggi di coerenza e valutazione qualitativa degli esperti.

4. **Analisi:** interpretazione dei topic, confronto longitudinale (2019 vs. 2023) e studio dell'evoluzione del vocabolario.

1.2.4 Risultati

Sono stati individuati dodici topic distinti: *Financial Applications, Sales and Marketing Applications, Foundational Statistics, Cybersecurity Applications, Project Management, Business Intelligence, Databases, Scientific Research Applications, Cloud and Big Data Engineering, Machine Learning, Software Engineering, Senior Management.*
L'analisi comparativa ha messo in evidenza alcune tendenze chiave:

- **Crescente specializzazione settoriale:** l'aumento della domanda in finanza (+6%) e marketing (+10%) evidenzia il passaggio da ruoli generalisti ad analisti focalizzati su domini specifici.

- **Dalle competenze teoriche a quelle applicative:** il calo di *Foundational Statistics* (–13%) e la crescita di *Machine Learning* (+12%) indicano una transizione verso competenze pratiche in ambito AI e NLP.

- **Commoditizzazione dell'infrastruttura:** la diminuzione di richieste per *Database* (–39%) e *Cloud Engineering* (–7%) suggerisce l'impatto di automazione e servizi cloud sempre più user-friendly.

- **Maggiore bisogno di software e governance:** l'incremento di *Software Engineering* (+13%) e *Senior Management* (+8%) segnala l'importanza crescente di capacità di leadership, governo dei processi e integrazione software.

1.2.5 Evoluzione del vocabolario

L'analisi delle frequenze (termini con più di 500 occorrenze) mostra un chiaro cambio tecnologico:

- **Termini in crescita:** *Databricks* (+551%), *PyTorch* (+226%), *NLP* (+130%), *Modelling* (+129%), *GCP* (+114%), a testimonianza del ruolo crescente di framework cloud e machine learning.
- **Termini in diminuzione:** *Hadoop* (–50%), *Java* (–51%), *SSRS* (–61%), *CSS* (–51%), che rappresentano il declino di tecnologie legacy orientate all'infrastruttura.

1.2.6 Conclusioni

Il paper mostra come il mercato del lavoro in ambito Data Analytics stia evolvendo verso ruoli **specializzati, guidati dall'AI e orientati al software**. Le competenze di leadership, governance e integrazione dei sistemi acquistano peso accanto alle competenze tecniche. Gli autori propongono inoltre una **metodologia longitudinale di topic modeling replicabile** applicabile anche ad altri domini professionali. Tra i limiti figurano il bias intrinseco degli annunci online e la rapida obsolescenza dei trend tecnologici. Per il futuro si suggerisce l'integrazione di ulteriori fonti (sondaggi, curricula formativi) per anticipare meglio le competenze emergenti.

1.2.7 Rilevanza per questo progetto

Questo studio fornisce le basi concettuali e metodologiche per il nostro lavoro:

- stabilisce il topic modeling longitudinale basato su LDA come benchmark per il confronto con i metodi moderni basati su embedding come **BERTopic**;
- conferma l'importanza di analizzare le **dinamiche temporali** nei corpora di annunci di lavoro;
- enfatizza la combinazione tra metriche quantitative di coerenza e interpretabilità umana, principio mantenuto nel nostro approccio tramite UMAP, HDBSCAN e probabilità di topic.

1.3 Approccio con BERT

Introdurre il modello BERT, motivare la sua scelta per il topic modeling e spiegare a livello concettuale la pipeline prevista.

Capitolo 2

BERTopic

Descrivere l'architettura di BERTopic e le motivazioni per cui è stata scelta per l'analisi degli annunci.

2.1 Pipeline

Presentare la pipeline end-to-end, evidenziando gli input, le trasformazioni intermedie e l'output finale del modello.

2.1.1 Embedding

Indicare come vengono generati gli embedding con BERT (o varianti), eventuali fine-tuning e impostazioni rilevanti.

2.1.2 Dimensionality Reduction

Illustrare l'algoritmo di riduzione dimensionale adottato (es. UMAP), i parametri principali e l'impatto sulla qualità dei topic.

2.1.3 Clustering

Spiegare il metodo di clustering (es. HDBSCAN), i criteri di scelta dei parametri e come viene determinato il numero di topic.

2.1.4 Tokenizer

Dettagliare il tokenizer utilizzato, le strategie di pre-processing e qualsiasi personalizzazione per il dominio degli annunci di lavoro.

2.1.5 Cosa si può migliorare

Discutere criticità note, possibili ottimizzazioni della pipeline e idee per estensioni future di BERTopic nel progetto.

Capitolo 3

Data Cleaning

3.1 Perché il Data Cleaning è necessario

Come abbiamo visto nel capitolo precedente, gli *embeddings* dei documenti ne rappresentano la **semantica**. Di conseguenza due documenti di simile significato saranno convertiti in vettori vicini.

I documenti che sono **vicini** nello spazio semantico e che si trovano in una **zona densa** di punti vengono quindi inseriti nello stesso cluster. Questo pone due importanti restrizioni nel dataset:

1. I documenti devono essere **semanticamente coerenti**, poiché ogni frase inutile influisce sulla posizione del documento nello spazio semantico; di conseguenza il rumore compromette la **coerenza dei cluster**.

2. Le frasi che riguardano argomenti non importanti per lo studio (e.g. stipendi, paragrafi legali, descrizioni aziendali, ecc.), oltre a influire sulla posizione dell'*embedding* nello spazio, creano cluster non utili ai fini dell'analisi, poiché comparendo in quasi tutti i documenti generano zone dense.

Il secondo punto è particolarmente delicato, perché cluster fittizi che raggruppano documenti in base a fattori irrilevanti non solo creano "topic spaz-

zatura", cioè non informativi, ma riducono anche la sensibilità ai dettagli distintivi di un documento (e.g. mansioni, abilità richieste), sottraendo ai cluster effettivi documenti importanti.

Per visualizzare l'effetto di un data cleaning accurato confrontiamo il comportamento del modello su un dataset non preprocessato (Figura 3.1).

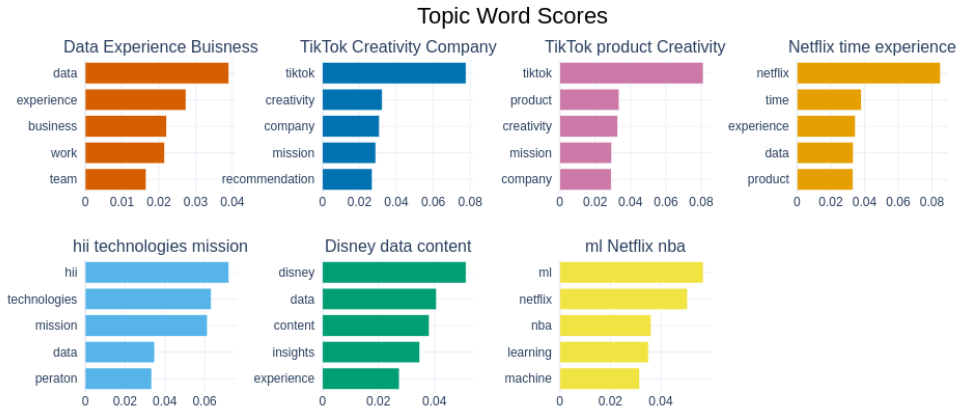

Figura 3.1: Barplot ottenuto da *topic modeling* senza *preprocess*.

Come visto nel capitolo precedente, a ogni parola è associato uno *score* che ne rappresenta l'importanza all'interno del topic. Questo barplot ci permette di fare alcune considerazioni importanti sulla natura del dataset e sulla direzione che deve assumere la pulizia dei dati. Innanzitutto notiamo che i nomi delle aziende, come TikTok e Netflix, hanno un peso molto grande; ciò è coerente con la natura del dataset, composto da offerte di lavoro basate negli USA, quindi è plausibile che Big Tech e altre multinazionali compaiano nella maggior parte degli annunci. Altre parole poco informative che compaiono in più topic sono legate al gergo aziendale (e.g. mission, team) e derivano dal *blurb* aziendale spesso presente negli annunci. Già con queste considerazioni preliminari otteniamo un buon punto di partenza per stabilire **cosa** eliminare dal dataset.

Un buon punto di partenza, ma non sufficiente. Per capire bene cosa eliminare dobbiamo osservare i dati grezzi e comprendere meglio la natura del corpus. Riportiamo di seguito un esempio che riteniamo rappresentativo, frutto dell'analisi dei dati grezzi, che ci ha permesso di decidere quali porzioni di testo andassero rimosse e quali invece preservate.

Ruolo
We are seeking an experienced and proactive Business Intelligence Engineer or lead to join our dynamic team. As a BI Engineer, you will be responsible for [...]

Abilità
Ability to work in a fast-paced, high-energy environment and bring sense of urgency & attention to detail skills to the table. [...]

Responsabilità
Responsibilities:

ETL processes — design, develop, and maintain ETL processes using Informatica IICS (Integration Cloud Services) and IDMC (Intelligent Data Management Cloud), ensuring efficient data extraction, transformation, and loading from various source systems. [...]

Benefit
Expected salary ranges between 100,000 and 150,000 USD annually. [...]

Blurb aziendale
About Regal Rexnord. Regal Rexnord is a publicly held global industrial manufacturer with 30,000 associates around the world who help create a better tomorrow [...]

Call to action
For more information, including a copy of our Sustainability Report, visit RegalRexnord.com.

Disclaimer su inclusività
Equal Employment Opportunity Statement.

Regal Rexnord is an Equal Opportunity and Affirmative Action Employer. All qualified applicants will receive consideration for employment without regard to race, color, religion [...]

Come inviare curriculum
Notification to agencies:

please note that Regal Rexnord Corporation and its affiliates and subsidiaries ("Regal Rexnord") do not accept unsolicited resumes or calls from third-party recruiters or employment agencies. [...]

Figura 3.2: Esempio annotato di annuncio di lavoro e sezioni considerate nella fase di data cleaning.

Questa struttura si riscontra nella maggior parte degli annunci analizzati. I paragrafi che riteniamo cruciali per gli obiettivi dello studio sono *Ruolo*, *Abilità* e *Responsabilità*, perché descrivono la **natura del lavoro**. Gli al-

tri blocchi, ovvero *Benefit*, *Blurb aziendale*, *Call to action*, *Disclaimer su inclusività* e *Come inviare curriculum*, costituiscono il rumore che intendiamo rimuovere. Abbiamo quindi identificato **cosa** eliminare; nella successiva sezione ci occuperemo del **come**.

3.2 Divisione in paragrafi

La suddivisione di un testo in paragrafi semanticamente coerenti è un problema aperto nella disciplina dell'*nlp*, ed è stato uno dei problemi più complessi affrontati in questo studio. Un primo approccio che abbiamo tentato è stato suddividere le descrizioni degli annunci in base a segni di punteggiatura, caratteri speciali (e.g. `\n`) e numero di parole. Il problema di questo approccio è che introduce dei metaparametri, come ad esempio numero di divisioni massime, lunghezza minima, ecc., che poco hanno a che fare con il significato del testo. Come risultato abbiamo ottenuto una divisione abbastanza omogenea nella lunghezza, ma grossolana nella coerenza semantica. Inoltre gli annunci di lavoro hanno una struttura ortografica poco coerente: qualche annuncio suddivide le informazioni con un elenco, altri separano tramite newline, altri ancora non separano affatto i paragrafi. Si è reso quindi necessario una ricerca di un' altro tipo di struttura comune, o se non altro fortemente ricorrente, nella segmentazione degli annunci.

Studiando nuovamente il dataset abbiamo notato che molti paragrafi iniziano con un'intestazione: nell'esempio sopra possiamo notare: "Responsabilities:" e "Equal Employment Opportunity Statement". Moltissimi annunci usano questa divisione, probabilmente per motivi di leggibilità data la lunghezza, ma non tutti; dunque abbiamo scelto la divisione basata su intestazioni come strategia principale e la vecchia strategia basata sulla punteggiatura come *fallback* nel caso di paragrafi troppo lunghi, questo perché se un paragrafo è lungo è probabile che contenga frasi con significati distanti, inoltre il modello che vedremo nella sezione successiva predilige blocchi di testo con poche frasi.

Riconoscere le intestazioni

Come riconoscere un'intestazione è più complicato di come si potrebbe pensare, un primo tentativo che abbiamo svolto utilizzava delle *espressioni regolari*, ad esempio:

```
JOB_CUES_PAT = re.compile(
    r"(?i)\b("
    r"job\s+description|about\s+the\s+role|responsabilit|activit|"
    r"requirements?|qualifications?|what\s+you'll\s+(do|be\s+doing)|"
    r"skills|nice\s+to\s+have|profilo"
    r")\b"
)
```

Però le regex si sono rivelate troppo rigide allo scopo, ad esempio un intestazione tipo "At Google you will:" non verrebbe catturata.

Un altro tentativo è stato quello di selezionare le righe che fossero composte da massimo n parole o che terminassero con un carattere *":"*, chiaramente anche questo tentativo è risultato fallimentare poiché alcune intestazioni si rivelano ben più lunghe di quanto si potrebbe immaginare, mentre altre frasi brevi potrebbero essere confuse per intestazioni. Dunque ci serve un metodo che sia sufficentemente flessibile, per questo abbiamo scelto di addestrare una rete neurale.

La libreria scelta per l'addestramento è *Spacy*.

3.2.1 Spacy

Spacy è una libreria per *nlp* basata su *pipelines* composte da moduli modificabili e allenabili. Questa componente altamente personalizzabile della libreria la rende ideale per il nostro studio, infatti in momenti diversi utilizzeremo componenti differenti a seconda del bisogno.

Le *pipelines* elaborano un testo e restituiscono un *Doc*, un oggetto che permette di accedere alle informazioni del testo ottenute dalla pipeline. Un Doc è una sequenza di Python composta da *token*. In spacy i token sono parole o segni di punteggiatura. Gli *span* invece sono sottosequenze del documento, composte da token contigui.

```
nlp = spacy.blank("en")        # Creazione pipeline default
doc = nlp("Hello world!")

for token in doc:
    print(token.text)
```

Output:

```
Hello
world
!
```

```
span = doc[1:3]
print(span.text)
```

Output:

```
world!
```

I moduli della pipeline lavorano salvando le informazioni sui token, sugli span o sul Doc. Uno dei moduli custom allenabili è lo SpanCategorizer (spancat in breve) che permette di riconoscere ed etichettare span.[1] Possiamo quindi interpretare le intestazioni come *span* e allenare un modello per riconoscerle. Lo spancat è composto da due parti:

1. Funzione *suggester*, che indica quali span sono candidati alla categorizzazione.

2. Rete neurale responsabile della classificazione, suddivisa in più sottoreti

[1] `https://spacy.io/api/spancategorizer`

Per configurare il modello (e l'allenamento), spaCy richiede un file `config.cfg`, vediamo adesso il suggester e tutte le componenti della rete neurale.

Suggester

Il *Suggester* utilizzato è quello di default, `spacy.ngram_suggester.v1`, che marca come candidati tutti gli span possibili entro una certa lunghezza.

```
[components.spancat.suggester]
@misc = "spacy.ngram_suggester.v1"
sizes = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
```

Figura 3.3: Configurazione del *suggester*: usa il componente `spacy.ngram_suggester.v1` e valuta tutti gli span fino a 20 token per coprire la varietà degli header.

Ho impostato 20 come lunghezza massima a causa della varietà degli header: il modello risulta molto flessibile, ma anche più pesante, perché all'aumentare degli n-grammi possibili aumentano i controlli da effettuare. Vediamo adesso come è composta la rete neurale, partendo dalla sua prima componente: **Tok2Vec**.

Tok2Vec

È suddiviso in due sottoreti embedder ed encoder. L'*embedder* converte i token in vettori ignorando il contesto.

```
[components.tok2vec.model.embed]
@architectures = "spacy.MultiHashEmbed.v2"
width = 96
attrs = ["NORM","PREFIX","SUFFIX","SHAPE"]
rows = [5000,1000,2500,2500]
include_static_vectors = false
```

Figura 3.4: Configurazione dell'*embedder*: vettori da 96 dimensioni con attributi morfologici distinti (`NORM`, `PREFIX`, `SUFFIX`, `SHAPE`) e hash table calibrate su 5000/1000/2500/2500 voci; gli embedding esterni non vengono usati.

Gli attributi indicano quali caratteristiche del token contribuiscono all'embedding e forniscono alla rete informazioni morfologiche e ortografiche utili per riconoscere pattern linguistici: `NORM` è la parola normalizzata (senza maiuscole, accenti o varianti equivalenti, e.g. `'s` diventa `is`); `PREFIX` e `SUFFIX` sono autoesplicativi; `SHAPE` descrive la forma ortografica (`Data`: `Xxxx`, `NASA`: `XXXX`, `2025`: `dddd`). Il componente `MultiHashEmbed.v2` crea embedding separati per ciascun attributo,[2] così ogni token è rappresentato dal significato, dalla forma ortografica e da porzioni specifiche della parola. In `rows` impostiamo la dimensione massima della tabella hash per ogni attributo; trattandosi di matrici che associano a ciascun valore un vettore, abbiamo dimensionato le tabelle in base alla varietà attesa degli attributi, prevedendo ad esempio molte più forme normalizzate rispetto ai prefissi e rimanendo entro il bound consigliato (1000–10000).
L'*encoder* modifica i vettori in funzione del *contesto*.

[2] `https://spacy.io/api/architectures#MultiHashEmbed`

```
[components.tok2vec.model.encode]
@architectures = "spacy.MaxoutWindowEncoder.v2"
width = 96
depth = 4
window_size = 1
maxout_pieces = 3
```

Figura 3.5: Encoder `MaxoutWindowEncoder`: stesso spazio vettoriale a 96 dimensioni, quattro strati con finestra contestuale di un token per lato e tre proiezioni Maxout per modellare non linearità.

I vettori vengono processati da una rete *feed-forward*: ogni strato calcola tre proiezioni lineari e applica *Maxout*, scegliendo quella con valore massimo per ogni finestra. L'uso di *Maxout*, al posto di una semplice ReLU, permette di modellare dipendenze contestuali più complesse.

Reducer

Questo componente sintetizza uno span in un unico vettore di dimensione `hidden_size`.

```
[components.spancat.model.reducer]
@layers = "spacy.mean_max_reducer.v1"
hidden_size = 128
```

Figura 3.6: Reducer `mean_max`: combina media e massimo dei token in un vettore di dimensione 128 prima dello strato di scoring.

Il componente `MeanMaxReducer` calcola la media dei token, il massimo dei token e concatena i due vettori in un'unica rappresentazione, sulla quale applica un *hidden layer*; la documentazione indica la presenza del layer ma non ne esplicita la struttura.[3]

[3] `https://spacy.io/api/architectures#mean_max_reducer`

Scorer

Il layer finale mappa il vettore dello span sulla probabilità di classe, utilizzando una funzione di attivazione **sigmoide**.

```
[components.spancat.model.scorer]
@layers = "spacy.LinearLogistic.v1"
nO = 1
nI = null
```

Figura 3.7: Scorer logistico lineare: un'unica uscita sigmoide (`nO = 1`) alimentata dal vettore di dimensione 128 prodotto dal reducer.

Il layer usato qui è uno strato lineare seguito da una **sigmoide**. Poiché SBERT (usato per la classificazione dei paragrafi) predilige documenti composti da poche frasi, e per limitare il numero di paragrafi spuri (cioè che comprendono informazioni di classi diverse), abbiamo scelto 0.2 come probabilità minima per classificare uno span come header, in modo da avere una segmentazione più fine. Il contro di questo approccio è che se il documento è troppo corto SBERT non ha il contesto necessario per fare una classificazione robusta, parleremo del problema nella sezione relativa al *fallback*.

Allenamento

Come detto in precedenza il file *config.cfg* definisce anche l'allenamento della rete. Il *corpus* è composto da 149 descrizioni selezionate inizialmente a caso dal dataset completo, poi a causa della comparsa di numerosi annunci dalla stessa azienda abbiamo deciso di sostituire alcuni annunci della suddetta con altri che mai comparivano nei 149 estratti, questo per evitare overfitting e migliorare la capacità del modello di generalizzare. In queste descrizioni abbiamo individuato 1083 intestazioni.
Abbiamo quindi suddiviso il corpus in *train* usato per l'allenamento e *dev* usato per la valutazione. la grandezza del *dev set* è del 10% del corpus, siamo consci del fatto che sia una porzione di dati troppo esigua per avere delle

valutazioni statistiche robuste, cionondimeno abbiamo deciso volutamente di creare una disparità in favore del training set poiché il modello è composto da più strati, tutti da allenare (embedder, encoder, reducer e scorer), dunque abbiamo evitato di sottrarre informazioni utili all'apprendimento. Vediamo adesso nello specifico come è impostato il training:

```
[training]
dev_corpus = "corpora.dev"
train_corpus = "corpora.train"
seed = ${system.seed}
gpu_allocator = ${system.gpu_allocator}
dropout = 0.1
accumulate_gradient = 1
patience = 1600.
L'F-score rimane buono in entrambi i casi, questo vuol dire che il modello
    generalizza bene, anche se perde qualche intestazione marginale.
max_epochs = 0
max_steps = 20000
eval_frequency = 200
```

Figura 3.8: Parametri di training: dropout fissato a 0.1, aggiornamento dopo ogni batch (`accumulate_gradient = 1`), early stopping con `patience` 1600 valutazioni e `max_steps` 20000; valutazione sul dev ogni 200 step.

Il corpo viene suddiviso in batch di grandezza variabile, i batch non sono composti da documenti, ma da token, per migliorare le performance quando l'allenamento avviene su GPU. A ogni passo di training, la dimensione del batch viene moltiplicata per un fattore 1.001, partendo da 100 token fino a un massimo di 1000; in questo modo batch piccoli vengono usati all'inizio del training, per avere gradienti più rumorosi ma anche più esplorativi, man mano che il training prosegue si prediligono batch sempre più grandi per avere gradienti più stabili e precisi. *Nota: valutare se inserire info riguardo l'ottimizzatore che è Adam.*

3.2.2 Valutazione del modello

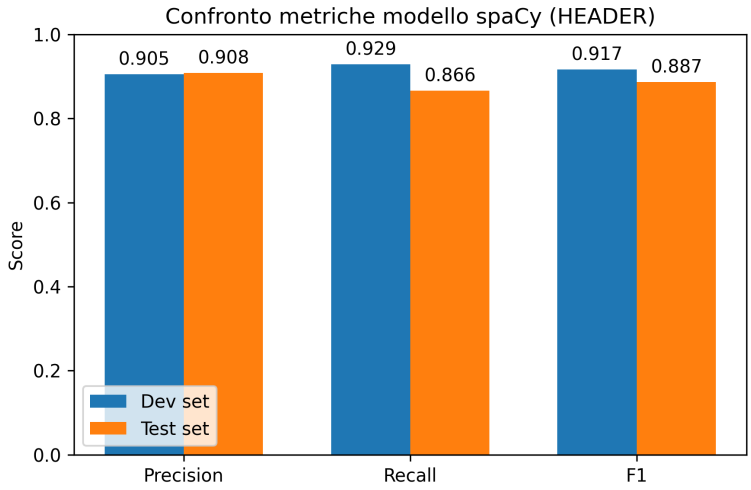

Figura 3.9: Confronto delle metriche *precision*, *recall* e *F-score* tra la valutazione fatta nel dev set e quella nel test set.

Il *test set* è composto da altre 20 descrizioni per un totale di 150 intestazioni. Il modello mostra una *precisione* elevata e una *sensibilità* complessivamente soddisfacente. La *recall*, come evidenziato in Figura 3.9, diminuisce in modo sensibile, ma rimane entro un intervallo accettabile per l'obiettivo di segmentazione. L'*F-score* resta stabile e su valori buoni, a indicare che la rete generalizza bene: qualche intestazione marginale viene persa, ma il bilanciamento tra falsi positivi e falsi negativi rimane favorevole.

3.2.3 Fallback

Come affermato in precedenza, abbiamo implementato una strategia di *fallback* per suddividere i paragrafi troppo lunghi. Il primo approccio testato è stato quello di utilizzare la *cosine similarity* fra frasi:

1. si divide il paragrafo in frasi;

2. si calcola l'embedding di ogni frase con SBERT;

3. a partire dalla seconda frase si misura la *cosine similarity* con quella precedente e, quando scende sotto soglia, si spezza il paragrafo.

In teoria il metodo avrebbe dovuto raggruppare frasi semanticamente affini e separare cambi di argomento. In pratica, molti paragrafi omogenei presentavano similarità basse e, in generale, la *similarità tra frasi* non risultava abbastanza correlata al cambio di argomento. Siamo quindi tornati a una suddivisione più ortografica basata su soglie di lunghezza (in termini di *token* spaCy), numero minimo di frasi e doppi `\n\n`.

```
nlp_sent = spacy.blank("en")
nlp_sent.add_pipe("sentencizer")

def count_sentences(part: str) -> int:
    doc = nlp_sent(part)
    return len(list(doc.sents))

def _split_long_paragraph(span, max_tokens=120, min_sents=2):
    text = span.text
    if len(span) < max_tokens:
        return [text]

    parts = [p.strip() for p in text.split("\n\n") if p.strip()]
    buffer = ""
    paragraphs = []

    for i, part in enumerate(parts):
        n_sents = count_sentences(part)
        if n_sents <= min_sents and i < len(parts) - 1:
            buffer += "\n" + part
        else:
            paragraphs.append((buffer + "\n" + part).strip())
            buffer = ""

    if buffer:
        paragraphs.append(buffer.strip())

    return paragraphs
```

Figura 3.10: Funzione di fallback: sfrutta i token e il modulo `sentencizer` di spaCy (all'interno di `count_sentences`) per riconoscere le frasi in un blocco di testo.

L'euristica divide il testo sugli `\n\n`, accumula i blocchi troppo corti in un buffer e restituisce sotto-paragrafi con almeno due frasi e meno di 120 token, così SBERT riceve segmenti informativi senza perdere il contesto. L'uso di una doppia strategia garantisce una buona granularità nella suddivisione del testo, infatti annunci in cui non sono presenti né intestazioni, né doppi new line sono pochi e hanno quindi un peso basso in BERTopic.

3.3 Classificazione paragrafi

Una volta completata la divisione in paragrafi, analizziamo come abbiamo affrontato la classificazione. Dobbiamo gestire testi di lunghezza variabile ma comunque limitati a poche frasi. Abbiamo deciso di utilizzare un modello della libreria *Sentence Transformers* per creare gli embedding dei paragrafi, a cui abbiamo poi applicato un layer di *Logistic Regression* che classifica gli embedding. Anche in questo caso abbiamo adottato *all-mpnet-base-v2*, poiché fornisce embedding semantici di alta qualità, come visto nel capitolo precedente.

3.3.1 Etichettatura

Il primo step per l'allenamento del modello è creare il *training set* e il *test set*. Un test preliminare è stato effettuato etichettando 178 paragrafi con 10 etichette diverse; di seguito riportiamo le etichette con il relativo numero di paragrafi:

JOB	91
BLURB	25
BENEFIT	23
DEI	15
LEGAL	10
LOCATION	5
SCHEDULE	3
CONTRACT	3
CALL TO ACTION	2
HOW TO APPLY	1

In particolare, **JOB** contiene i paragrafi che abbiamo precedentemente identificato con "Ruolo", "Abilità" e "Responsabilità"; **DEI** include i paragrafi dedicati alle politiche di inclusione ed equità; **LEGAL** raggruppa i disclaimer legali; **CONTRACT** raccoglie le specifiche burocratiche del contratto.

Come è facile notare solo la classe **JOB** risulta ben rappresentata; le uniche altre due classi con un numero sufficiente di esempi sono **BLURB** e **BENEFIT**. Per questo motivo abbiamo deciso di fondere le classi sottorappresentate, così da evitare un crollo delle performance sulle classi più piccole. La scelta della fusione è ricaduta su due opzioni alternative:

1. **Binario semplice**
2. **Multi-classe intermedia**

Il binario semplice risolve il problema della sotto-rappresentazione, ma rischia di generalizzare peggio poiché le classi **JOB** e **NON-JOB** risultano molto eterogenee. Il multi-classe, invece, aiuta SBERT a strutturare meglio lo spazio semantico; di contro richiede più dati per garantire una rappresentazione adeguata di tutte le classi. Abbiamo quindi optato per il multi-classe, incrementando significativamente la dimensione del *training set* per ottenere un addestramento più stabile.
Le classi finali adottate sono tre:

- **JOB**: come definita precedentemente.
- **BLURB-LEGAL**: unione di blurb, DEI, legal, call to action e how to apply.
- **OFFER-DETAIL**: unione di benefit, location, schedule, contract.

Useremo dunque più classi in fase di addestramento, ma il modello verrà impiegato come classificatore binario scartando le classi diverse da **JOB**. Il *dataset* finale è composto da 1315 paragrafi così etichettati:

JOB	700
BLURB-LEGAL	340
OFFER-DETAIL	275

Lo sbilanciamento a favore di **JOB** potrebbe apparire problematico: quando le classi sono sbilanciate un classificatore tende a privilegiare la classe maggioritaria, con il rischio di introdurre un *bias* verso **JOB**. Tuttavia il nostro obiettivo è pulire il dataset *evitando di tagliare informazioni potenzialmente rilevanti* per lo studio; di conseguenza, nel caso in cui il modello fosse incerto, è preferibile preservare il paragrafo per non eliminare contenuti importanti. In quest'ottica avere un'unica classe **JOB** (anziché distinguere Ruolo, Abilità e Responsabilità) può rivelarsi un vantaggio. Questa politica di "favoritismo" verso la classe **JOB** verrà riutilizzata sia nel training dell'embedder sia nelle politiche di cancellazione dei paragrafi post-classificazione, come illustrato nelle sezioni successive.

Allenamento

```
X_train, X_test, y_train, y_test = train_test_split(
    df["text"],
    df["label"],
    test_size=0.2,
    stratify=df["label"],
    random_state=10,
)
```

Figura 3.11: Il 20% del dataset è usato come test set per la valutazione. L'opzione `stratify=df["label"]` preserva la distribuzione delle classi tra train e test, indispensabile in presenza di classi sbilanciate.

Dopo la suddivisione del dataset e la creazione e normalizzazione degli embedding resta la classificazione effettiva, le opzioni che abbiamo preso in considerazione sono:

1. **Rete neurale**
2. **Fine-tuning del transformer**
3. **Logistic Regression**

Una rete neurale può apprendere relazioni non lineari tra gli embedding per catturare interazioni più complesse. Questo approccio può ottenere buone prestazioni in presenza di dataset di dimensioni maggiori, comporta inoltre una maggiore complessità di addestramento e un rischio più elevato di overfitting rispetto a modelli lineari.
Con *fine tuning del transformer* intendiamo il riaddestramento parziale (cioè solo degli ultimi layer) del modello Transformer stesso, aggiungendo in coda al modello una piccola testa di classificazione. Questa strategia offre un grande vantaggio in più rispetto alla precedente: l'adattamento al dominio. Infatti SBERT non sarebbe più un generatore di embedding statici, ma coglierebbe sfumature linguistiche specifiche degli annunci di lavoro, riducendo l'ambiguità tra frasi che nei dati generali erano simili, ma nel dominio degli annunci hanno significati distinti; questo potrebbe aumentare sensibilmente la capacità del modello di discriminare più accuratamente paragrafi simili. Chiaramente un approccio del genere richiederebbe un dataset di dimensioni ben maggiori, con almeno alcune migliaia di esempi. Abbiamo comunque ritenuto importante menzionarlo per eventuali sviluppi futuri.
L'opzione scelta è quindi la **Logistic Regression**, che è adatta alla dimensione del nostro dataset, con basso rischio di overfitting rispetto alle scelte precedenti; inoltre gli embedding di *all-mpnet-base-v2* sono già molto informativi, anche senza *fine tuning*, quindi LR deve solo apprendere frontiere di decisione lineari, non estrarre feature dal testo grezzo.

```
clf = LogisticRegression(
    max_iter=2000,
    class_weight={"JOB": 1.5, "BLURB": 1.0, "DETAIL": 1.0},
    random_state=10,
)
```

Figura 3.12: Configurazione del classificatore LR: massimo 2000 iterazioni, pesi maggiorati per la classe `JOB` (1.5) e bilanciamento minore per `BLURB` e `DETAIL`.

La *class weight* rappresenta il peso che ha una classe nella funzione di *loss*. Aumentando il peso di una classe il modello riceve gradienti più forti durante la *backpropagation* e impara quindi a classificarla meglio. Abbiamo prediletto pesi maggiori per **JOB** per le ragioni espresse nella sezione precedente. I valori testati per la *class weight* di **JOB** sono 1.0, 1.5 e 2.0; i risultati principali sono riassunti in Figura 3.13.

Balanced (1.0)

	Prec.	**Rec.**	**F1**
DETAIL	0.87	0.93	0.90
BLURB	0.85	0.90	0.88
JOB	0.97	0.92	0.94

	DET.	**BLURB**	**JOB**
DET.	253	13	7
BLURB	20	306	13
JOB	18	41	637

Slightly unbalanced (1.5)

	Prec.	**Rec.**	**F1**
DETAIL	0.91	0.85	0.88
BLURB	0.89	0.81	0.85
JOB	0.91	0.97	0.94

	DET.	**BLURB**	**JOB**
DET.	231	22	20
BLURB	16	275	48
JOB	8	13	675

Unbalanced (2.0)

	Prec.	**Rec.**	**F1**
DETAIL	0.91	0.84	0.88
BLURB	0.89	0.80	0.85
JOB	0.90	0.97	0.94

	DET.	**BLURB**	**JOB**
DET.	230	20	23
BLURB	15	272	52
JOB	7	12	677

Figura 3.13: Metriche di valutazione e matrici di confusione al variare del peso assegnato alla classe **JOB** nella *Logistic Regression*.

Il modello bilanciato è quello con un *f-score* più alto, però il nostro obiettivo rimane massimizzare la *recall* di JOB, in cui il modello leggermente sbilanciato è più forte al netto di un leggero peggioramento nelle altre classi (atteso) e nella precisione. Il modello più sbilanciato non offre un miglioramento della *recall*, ma solo un peggioramento della performance generale del modello. Dunque la scelta è ricaduta sul modello **Slightly unbalanced**.

Strategia di rimozione

Adesso abbiamo la probabilità di appartenenza ad una classe per ciascun paragrafo, dobbiamo scegliere come eliminare i paragrafi non informativi in

base a queste informazioni: Per mitigare i casi borderline abbiamo definito un filtro euristico che riassegna la classe finale sulla base delle probabilità restituite dal classificatore.

```
dominant = df[["prob_job", "prob_blurb_legal", "prob_offer_detail"]] \
    .idxmax(axis=1).str.replace("prob_", "", regex=False)
top_probs = df[["prob_job", "prob_blurb_legal", "prob_offer_detail"]].max(axis=1)

cond_blurb = df["prob_blurb_legal"] > blurb_threshold
cond_job = df["prob_job"] > job_threshold
cond_diff = (top_probs - df["prob_job"] < diff_threshold)
cond_low = top_probs < low_conf_threshold

prediction = np.select(
    [
        cond_blurb & ~cond_job,              # blurb forte e non job
        cond_job | cond_diff | cond_low      # se job, simile a job, o incerto
    ],
    ["blurb_legal", "job"],
    default=dominant                         # altrimenti usa la classe dominante
)
```

Figura 3.14: Logica di post-processing per l'assegnazione finale della classe del paragrafo.

Cosa fa questo codice:

1. un paragrafo viene etichettato come **blurb legal** solo quando la probabilità relativa supera la soglia e quella di **JOB** rimane sotto il limite stabilito; questo perché i paragrafi che più inquinano sono quelli delle descrizioni aziendali.

2. i paragrafi con alta confidenza su **JOB**, con probabilità simile a **JOB** o complessivamente ambigui (*low conf*) vengono classificati come **JOB**;

3. in tutti gli altri casi si ricorre alla classe dominante (*default*) proposta dal modello.

3.4 Altre strategie di data cleaning usate

Dopo la pulizia a livello di paragrafo analizziamo l'effetto su BERTopic. La Figura 3.15 mostra il dendrogramma ottenuto con il dataset parzialmente pulito.

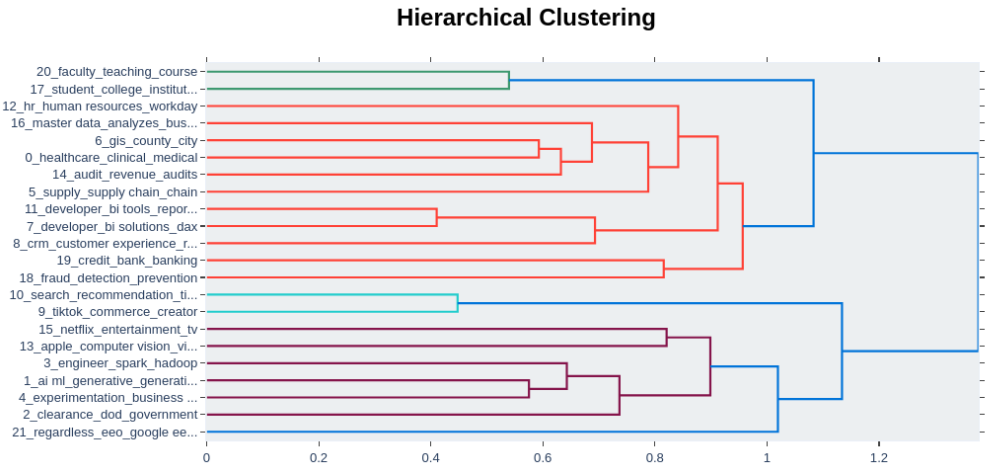

Figura 3.15: Dendrogramma di BERTopic su dataset parzialmente pulito.

Il miglioramento rispetto al dataset grezzo è evidente, ma persistono ancora numerosi **nomi aziendali** e **luoghi**. Ciò dipende sia dalla strategia conservativa adottata in classificazione (preferire **JOB** anziché rischiare falsi negativi), sia da paragrafi misti: frasi come "At Apple we are looking for a Data Analyst" contengono informazioni cruciali sul ruolo, ma introducono anche riferimenti non utili.

Diventa quindi necessario affiancare alla pulizia per paragrafi una pulizia **interna alle singole frasi**, così da eliminare riferimenti ad aziende e località e permettere a BERTopic di pesare maggiormente mansioni e competenze.

Pulizia tramite NER

Per raggiungere questo obiettivo integriamo nella pipeline spaCy un componente *NER* (Name Entity Recognizer) già addestrato che identifica *entità* testuali (aziende, persone, stati, ecc.). I token corrispondenti a entità non informative vengono rimossi dalle frasi dei paragrafi **JOB**.

```
def remove_entities(texts, removable_entities={"ORG", "PERSON", "GPE"}):
    cleaned = []
    for doc in ner_model.pipe(texts):
        tokens = []
        for token in doc:
            if token.ent_type_ in removable_entities \
                    and token.text.lower() not in TECH_WHITELIST:
                continue
            tokens.append(token.text_with_ws)
        cleaned.append("".join(tokens).strip())
    return cleaned
```

Figura 3.16: Euristica di rimozione delle entità: `ORG` identifica le organizzazioni (incluse le aziende), `GPE` le entità geopolitiche (stati, città).

In parallelo manteniamo una **whitelist** di termini tecnici (ad esempio *Oracle*, *GitHub*) per evitare di rimuovere entità che rappresentano competenze o strumenti rilevanti.

Risultato finale

Applicando anche questa pulizia intra-frasi otteniamo il dendrogramma della Figura 3.17 tramite BERTopic.

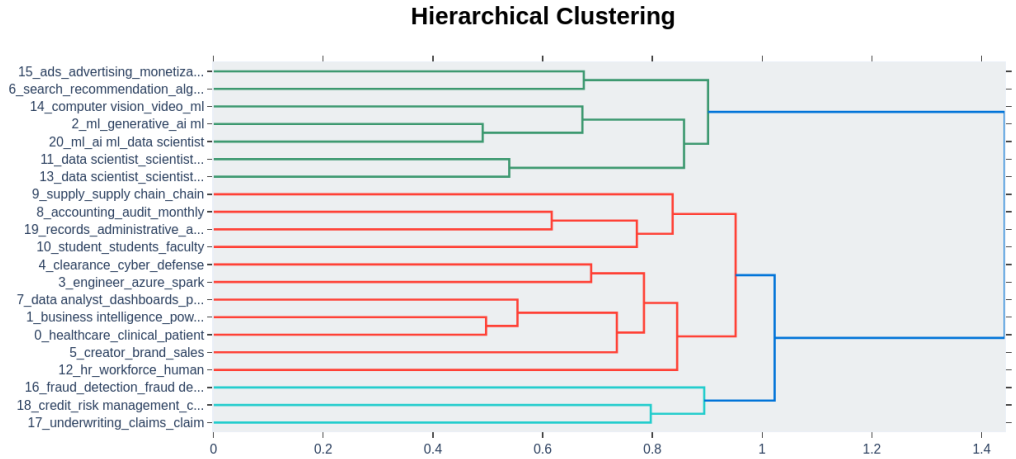

Figura 3.17: Dendrogramma di BERTopic su dataset completamente pulito.

I riferimenti a **brand** e **luoghi statunitensi** scompaiono quasi del tutto e l'attenzione si concentra su mansioni, competenze e requisiti realmente utili per l'analisi del mercato del lavoro.

3.5 Possibili miglioramenti

Analizziamo quì i punti critici che potrebbero essere migliorati e speculiamo su quali altre strategie potrebbero essere attuate in uno sviluppo successivo dello studio:

Capitolo 4

Risultati finali

Descrivere i topic ottenuti, le metriche di valutazione e le principali evidenze emerse dall'analisi degli annunci.

Inserire tabelle, grafici e commenti qualitativi che illustrino chiaramente l'efficacia dell'approccio BERTopic applicato al dataset.

Capitolo 5

Conclusioni

Sintetizzare i risultati principali e delineare i possibili sviluppi futuri.

Bibliografia

[1] Mariam Almgerbi, Andrea De Mauro, Hanan Kahlawi, and Valentina Poggioni. The dynamics of data analytics job skills: a longitudinal analysis. *Journal of Emerging Data Science*, 12(3):45–68, 2025.

[2] Maarten Grootendorst. Bertopic: Neural topic modeling with a class-based tf-idf procedure. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*, pages 1–9, 2022.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. Association for Computational Linguistics, Florence, 2019.

Ringraziamenti

Testo dei ringraziamenti da completare.