



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
Drugó, a volte si incontra un uomo che è l'uomo giusto al momento giusto
nel posto giusto, là dove deve essere. E quello è Drugó, a Los Angeles. E
anche se quell'uomo è un pigro, e Drugó 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 Lebowski

Indice

1 Introduzione	5
1.1 Cos'è il Topic Modeling	5
1.2 Stato dell'arte	5
1.2.1 Obiettivi e motivazioni	6
1.2.2 Novità nella letteratura	6
1.2.3 Metodologia	6
1.2.4 Risultati	7
1.2.5 Conclusioni	8
1.2.6 Rilevanza per questo progetto	8
2 BERTopic	10
2.1 Funzionalità	11
2.1.1 Embedding	11
2.1.2 Dimensionality Reduction	13
2.1.3 Clustering	16
2.1.4 CountVectorizer	23
2.1.5 c-TF-IDF	24
2.1.6 Maximal Marginal Relevance (MMR)	25
3 Implementazione di BERTopic	27
3.1 Embedding	27
3.1.1 Mean-max pooling	29

3.2	Dimensionality Reduction	30
3.3	Clustering	33
3.3.1	CountVectorizer	34
4	Data Cleaning	37
4.1	Perché il Data Cleaning è necessario	37
4.2	Divisione in paragrafi	40
4.2.1	Spacy	41
4.2.2	Valutazione del modello	48
4.2.3	Fallback	48
4.3	Classificazione paragrafi	50
4.3.1	Etichettatura	50
4.4	Altre strategie di data cleaning usate	56
4.5	Possibili miglioramenti	59
5	Risultati finali	60
6	Conclusioni	61
	Ringraziamenti	63

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 Stato dell'arte

Almgerbi, De Mauro, Kahlawi, Poggioni 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 $k = 5\text{--}20$ 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.

Inoltre 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.

1.2.5 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.

1.2.6 Rilevanza per questo progetto

Questo studio fornisce le basi concettuali e metodologiche per il nostro lavoro: In primis 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 e enfatizza la combinazione tra metriche

quantitative di coerenza e interpretabilità umana, principio mantenuto nel nostro approccio tramite UMAP, HDBSCAN e probabilità di topic.

Capitolo 2

BERTopic

L'avvento dei moderni *LLM* ha portato ad un'evoluzione del **NLP** (Natural Language Processing) con la conseguente nascita di nuovi framework e tecniche per le analisi più disparate. Il *topic modeling* non è esente da questo fenomeno, in particolare i modelli basati su **transformers** (cioè modelli che convertono testo in embeddings dipendenti dal contesto) si sono rilevati particolarmente efficaci nell'*NLP*. **Modelli preaddestrati** sono molto utilizzati perché permettono di creare rappresentazioni **accurate** e **rappresentative** di testi, potendo fare affidamento su dataset di training molto grandi e molto diversificati. In questo contesto BERTopic è un *framework* al passo con il corrente stato dell'arte, che offre alcuni vantaggi rispetto a tecniche antecedenti, fra cui permette di catturare il **significato semantico** dei documenti (riconosce ad esempio i sinonimi). Per dataset eterogenei e rumorosi come gli annunci di lavoro, questa capacità di individuare strutture semantiche latenti rende BERTopic una scelta particolarmente vantaggiosa rispetto ai modelli tradizionali. Inoltre in BERTopic specificare il numero di classi è **opzionale**. Inoltre BERTopic è composto da più *submodules* con funzioni diverse, il che permette un'alta personalizzazione.

2.1 Funzionalità

All'interno di Bert Topic è possibile distinguere alcune funzionalità particolarmente utili per questo progetto:

1. Embedding
2. Dimensionality Reduction
3. Clustering
4. CountVectorizer
5. c-TF-IDF

BERTopic permette l'utilizzo di algoritmi diversi per ogni funzionalità, per personalizzare il proprio *topic model*, ad esempio il clustering può essere effettuato da *HDBSCAN* o da *k-Means*. Illustreremo ora tutti i moduli da noi usati, vedremo nello specifico la loro funzione e come la loro alterazione modifica il risultato finale.

2.1.1 Embedding

Il primo modulo è l'embedder, il cui compito è quello di trasformare i documenti testuali in rappresentazioni numeriche poi utilizzabili dagli algoritmi di machine learning.

Il testo viene innanzitutto **tokenizzato**: un token può corrispondere a una parola intera, a una sua parte oppure a una sequenza di caratteri frequente, in base alla strategia di tokenizzazione adottata.

Ciascun token è successivamente convertito in un vettore numerico, l'*embedding*. Questi vettori abitano uno spazio ad alta dimensionalità in cui la vicinanza geometrica riflette la somiglianza semantica dei token: parole con significato analogo (ad esempio *developer* e *programmer*) finiscono in posizioni contigue.

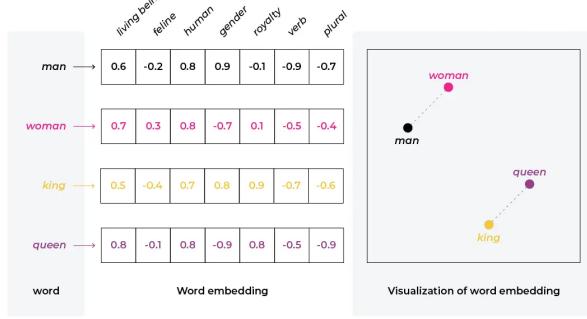


Figura 2.1: Esempio illustrativo di relazioni semantiche nello spazio degli embedding: differenze vettoriali come $E(\text{man}) - E(\text{woman})$ codificano attributi latenti (qui il genere).

Nella Figura 2.1 si osserva come la direzione del vettore $E(\text{man}) - E(\text{woman})$ (dove $E(\cdot)$ è la funzione di embedding) catturi una proprietà semantica interpretabile.

La dimensionalità di questo spazio dipende dal modello: *all-mpnet-base-v2* produce vettori da 768 componenti, mentre modelli di grandi dimensioni come GPT-3 arrivano a 12 288 dimensioni. Negli embedder moderni (ad esempio quelli basati su Transformer), ad esempio *Sentence-BERT* (SBERT), MPNet o GPT, ogni token viene inoltre rappresentato *in funzione del contesto* in cui compare. Di conseguenza la parola “bank” avrà embedding diversi se usata in “river bank” oppure in “investment bank”.

A partire dagli embedding dei token è possibile costruire *sentence embeddings* che sintetizzano la struttura sintattico-semantica di un paragrafo. SBERT, in particolare, applica un meccanismo di pooling (tipicamente mean o max pooling)¹ alle rappresentazioni prodotte dal Transformer, ottenendo un unico vettore che compendia l’informazione distribuita sull’intera frase.

L’uso di *sentence embeddings* consente di calcolare **distanze semantiche tra documenti** con misure come cosine similarity o euclidean distance,

¹https://sbert.net/docs/package_reference/sentence_transformer/models.html

rendendoli ideali per compiti quali *clustering* e *topic discovery*.

2.1.2 Dimensionality Reduction

I vettori generati dall'embedder hanno dimensione alta, a prescindere dal modello che si sceglie e ciò potrebbe rendere difficile il *clustering*. Per questo è importante introdurre nella pipeline un algoritmo di riduzione della dimensione. L'idea di questi algoritmi è quella di ridurre la dimensione di un insieme di vettori preservandone la **struttura topologica locale**. Di default BERTopic usa **UMAP**² per questo compito poiché *preserva sia la struttura locale, che quella globale degli spazi ad alte dimensioni*. UMAP lavora costruendo un *grafo fuzzy locale*: ogni punto viene connesso ai suoi `n_neighbors` più vicini e la connessione è pesata in base alla distanza. Questo grafo rappresenta la topologia locale dello spazio originale. UMAP cerca poi una rappresentazione a bassa dimensione che minimizzi la differenza tra il grafo originale e il grafo proiettato.

L'obiettivo è mantenere la densità e la vicinanza dei punti simili, consentendo allo stesso tempo di contrarre regioni sparse. Questo bilanciamento consente di preservare la struttura locale pur comprimendo lo spazio globale. Vediamo più nel dettaglio questi punti.

Fuzzy simplicial complex

Per costruire il grafo iniziale ad alta dimensionalità, **UMAP** crea una struttura chiamata *fuzzy simplicial complex*. In pratica, si tratta di una rappresentazione di un grafo pesato, in cui i pesi degli archi rappresentano la **probabilità che due punti siano connessi**.

Per determinare la connessione tra due punti, UMAP³ estende un **raggio locale** a partire da ciascun punto e collega i punti quando i rispettivi raggi si

²https://maartengr.github.io/BERTopic/getting_started/dim_reduction/dim_reduction.html

³<https://pair-code.github.io/understanding-umap/>

sovrappongono. La scelta di questo raggio è cruciale: Se è troppo piccolo, il risultato sarà un insieme di **piccoli cluster isolati**; se è troppo grande, tutti i punti finiranno per essere **collegati tra loro**, perdendo così la struttura locale.

UMAP risolve questo problema scegliendo un raggio locale **adattivo**, basato sulla distanza rispetto al k -esimo vicino più prossimo di ciascun punto. Successivamente, il grafo viene reso “sfumato” (*fuzzy*) diminuendo la probabilità di connessione man mano che la distanza aumenta.

Infine, imponendo che ogni punto sia connesso almeno al proprio vicino più prossimo, UMAP garantisce un equilibrio tra la **preservazione della struttura locale** e la **coerenza della struttura globale**.

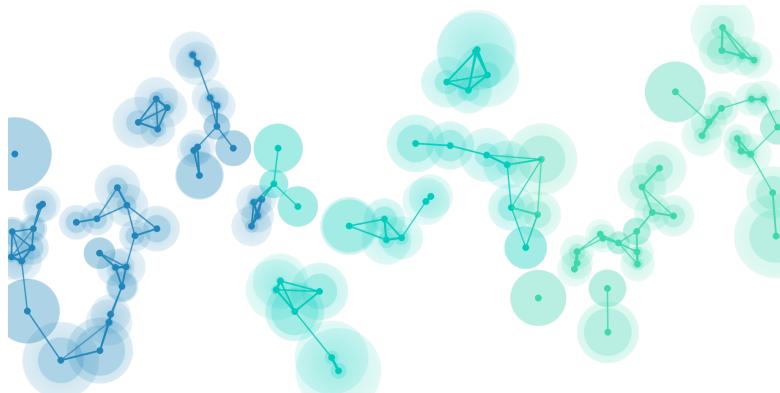


Figura 2.2: Proiezione UMAP nel piano bidimensionale. Dopo l’intersezione con il primo *neighbour*, il cerchio diventa *sfumato*, riducendo il peso della connessione.⁴

Parametri

I due parametri più importanti di UMAP sono `n_neighbors` e `min_dist`, poiché regolano l’equilibrio tra struttura *locale* e *globale* nella proiezione finale.⁵

⁴<https://pair-code.github.io/understanding-umap/>

⁵<https://pair-code.github.io/understanding-umap/>

n_neighbors. Determina il numero di vicini utilizzato per costruire il grafo fuzzy iniziale. Valori bassi enfatizzano le relazioni locali, isolando microstrutture anche molto sottili; valori alti favoriscono invece la coerenza complessiva, preservando maggiormente la geometria globale dell'insieme di punti. Per visualizzare l'effetto di `n_neighbors` consideriamo la proiezione bidimensionale di un modello 3D campionato con 50 000 punti. Le immagini mostrano come la struttura globale emerga gradualmente all'aumentare del numero di vicini, mentre i dettagli fini sono preservati soltanto per valori più bassi.

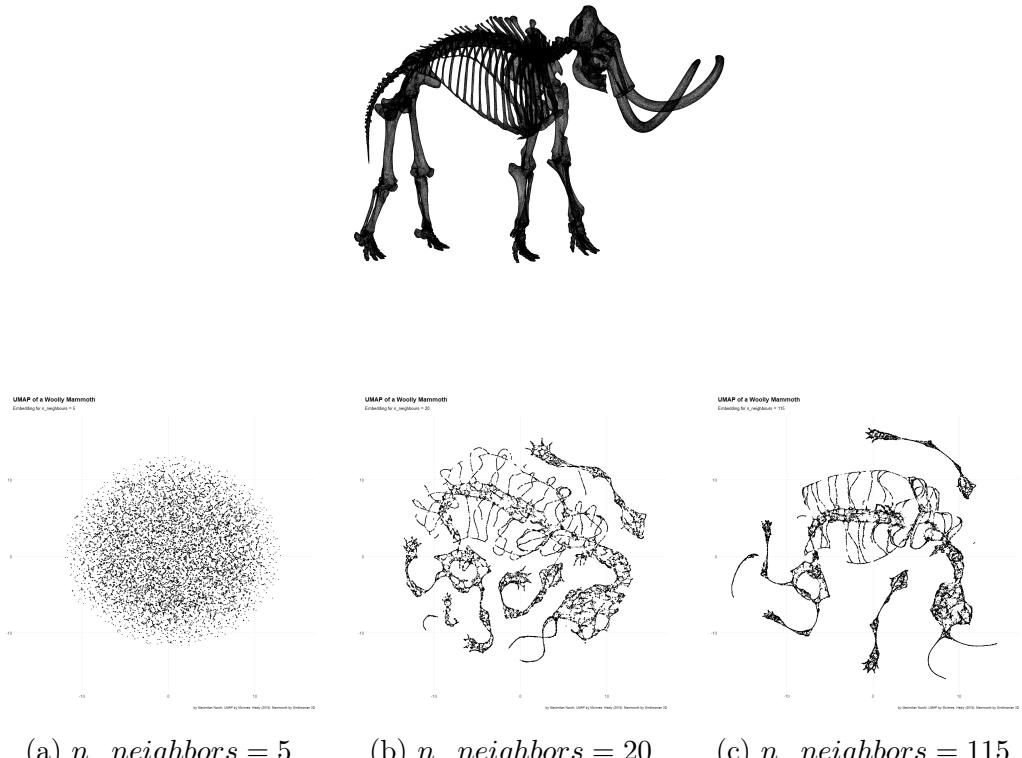


Figura 2.3: Proiezione UMAP di un modello 3D con 50 000 punti in un piano 2D. Fonti: Smithsonian 3D Digitization Program e Max Noichl.

Si osservi come per valori ridotti di `n_neighbors` la proiezione UMAP mantenga dettagli locali (curve delle zanne o delle zampe), ma perda la forma complessiva del mammut; all'aumentare del parametro la sagoma globale diventa riconoscibile, a scapito delle piccole variazioni geometriche.

`min_dist` controlla la densità minima consentita nella proiezione a bassa dimensione. Un valore vicino a zero permette ai punti di collassare in regioni molto dense, mettendo in risalto cluster ben separati; aumentando il parametro, UMAP distribuisce i punti con maggiore uniformità, sacrificando la compattezza dei gruppi a favore di una rappresentazione più omogenea. La Figura 2.4 mostra come diverse configurazioni di `min_dist` modifichino la distribuzione dei punti: valori estremamente bassi generano isole molto concentrate, mentre impostazioni più elevate distendono l'intero spazio proiettato.



Figura 2.4: Effetto di `min_dist` sulla proiezione UMAP del modello 3D da 50 000 punti. Fonti: Smithsonian 3D Digitization Program e Max Noichl.

2.1.3 Clustering

Dopo aver ridotto la dimensione dell'input, dobbiamo creare dei *cluster* di embedding simili da cui estrarremo i topic.

Il modello che consiglia BERTopic⁶ è *HDBSCAN*, un algoritmo di clustering basato su densità, in modo da identificare cluster a densità variabile senza dover fissare un singolo parametro di soglia (a differenza di *DBSCAN*).

Il *clustering* viene ottenuto passando per 5 fasi⁷:

1. Trasformazione dello spazio basata sulla densità
2. Minimum spanning tree del grafo pesato delle distanze
3. Costruzione della gerarchia di aggregazione dei punti
4. Condensazione della gerarchia in base a `min_cluster_size`
5. Estrazione dei cluster dall'albero condensato

Vediamo i passaggi più nel dettaglio con l'ausilio di un esempio grafico.
In Figura 2.5 è mostrato l'insieme di punti su cui applicheremo HDBSCAN.

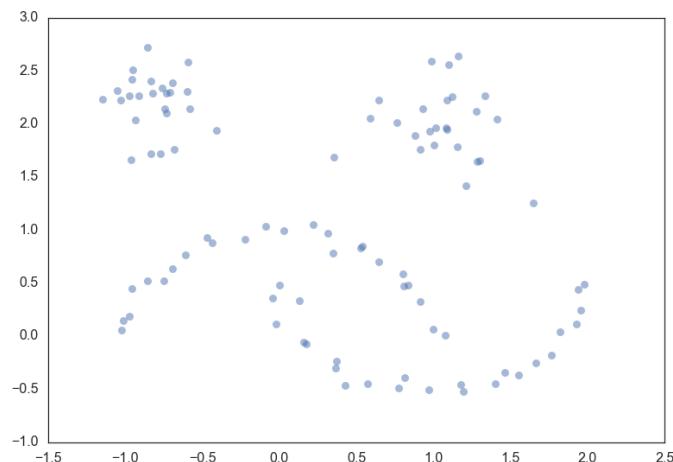


Figura 2.5: Dataset iniziale *ad hoc* generato per testare HDBSCAN

⁶https://maartengr.github.io/BERTopic/getting_started/clustering/clustering.html

⁷https://hdbSCAN.readthedocs.io/en/latest/how_hdbSCAN_works.html

1. Trasformazione dello spazio in funzione della densità

Il primo passo dell'algoritmo HDBSCAN consiste nel *trasformare lo spazio dei dati* in modo che rifletta non solo la distanza geometrica tra i punti, ma anche la **densità locale** delle regioni in cui essi si trovano. Questo passaggio consente di adattare la nozione di distanza alle caratteristiche strutturali del dataset, migliorando la robustezza rispetto a variazioni di scala e rumore.

Per ogni punto x_i viene calcolata la **core distance**, definita come la distanza dal suo k -esimo vicino più prossimo, dove $k = \text{min_samples}$. Essa rappresenta una stima della densità locale: valori piccoli indicano aree dense, mentre valori grandi indicano zone più sparse o isolate.

Successivamente, la distanza tra due punti a e b viene ridefinita come **mutual reachability distance**, che tiene conto delle rispettive densità locali:

$$d_{\text{mreach}}(a, b) = \max(\text{core_dist}(a), \text{core_dist}(b), d(a, b))$$

dove $d(a, b)$ è la distanza originale tra a e b . Usando questa distanza, i punti densi rimangono alla loro distanza originale, mentre quelli rarefatti vengono allontanati, ciò rende l'algoritmo **robusto al rumore**.

2. Costruzione del *Minimum Spanning Tree* (MST)

Dopo la trasformazione dello spazio in funzione della densità locale, HDBSCAN costruisce un grafo completamente connesso in cui ogni punto rappresenta un nodo e ogni arco tra due punti è pesato in base alla *mutual reachability distance* calcolata nel passo precedente. Questo grafo descrive la struttura di connettività del dataset, tenendo conto sia della distanza geometrica sia della densità locale delle regioni considerate.

A partire da tale grafo, viene poi calcolato il **Minimum Spanning Tree** (MST), ossia l'albero di connessioni che collega tutti i punti minimizzando la somma complessiva dei pesi degli archi. Il MST conserva quindi solo le

connessioni essenziali per mantenere la connettività globale, eliminando i collegamenti ridondanti e le connessioni tra regioni scarsamente dense.

Questo passaggio ha due obiettivi principali:

- **Semplificare la struttura dei dati**, riducendo la complessità del grafo e mantenendo una rappresentazione compatta della topologia di densità;
- **Preparare la costruzione della gerarchia dei cluster**, poiché nei passi successivi gli archi del MST verranno rimossi in ordine crescente di distanza per individuare i gruppi di punti che restano connessi, ossia i futuri cluster.

il Minimum Spanning Tree dunque connette i punti più vicini secondo la loro densità locale e costituisce la base per la costruzione della gerarchia dei cluster nel passo successivo.

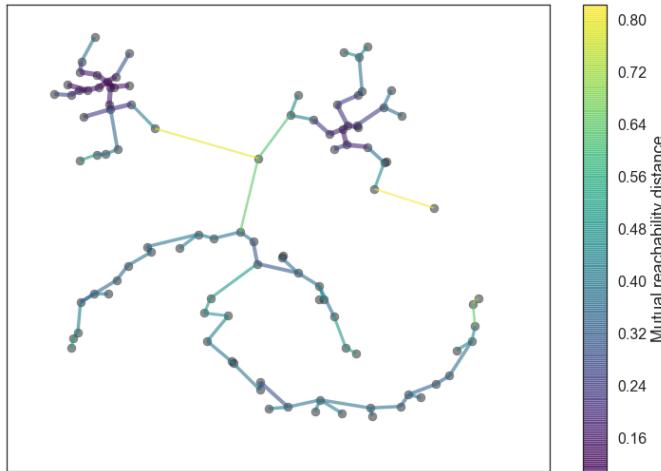


Figura 2.6: Minimum Spanning Tree costruito sulle distanze di mutual reachability: gli archi a peso minore collegano regioni dense e fungono da base per la gerarchia.

3. Costruzione della gerarchia dei cluster

A partire dal *Minimum Spanning Tree* costruito nel passo precedente, HDBSCAN procede alla **costruzione di una gerarchia di cluster** basata sulle componenti connesse del grafo. Questo processo consiste nel rimuovere progressivamente gli archi del MST in ordine crescente di peso, ovvero dalla connessione più “debole” (distanza maggiore) a quella più “forte” (distanza minore).

Man mano che gli archi vengono rimossi, il grafo si frammenta in un numero crescente di componenti connesse. Ogni componente risultante viene interpretata come un potenziale cluster a un determinato livello di densità. In questo modo, HDBSCAN genera una *gerarchia di densità* in cui i cluster “nascono” e “muoiono” al variare della soglia di distanza, descrivendo come la struttura dei dati cambia al modificarsi del livello di densità considerato.

Il risultato di questa fase è un albero gerarchico che rappresenta la relazione di inclusione tra cluster: le radici corrispondono ai cluster più generali (densità minore), mentre i rami e le foglie rappresentano i cluster più specifici e densi. Questa rappresentazione gerarchica è fondamentale per le fasi successive, poiché consente di identificare in modo naturale i livelli di densità più stabili e significativi all’interno del dataset.

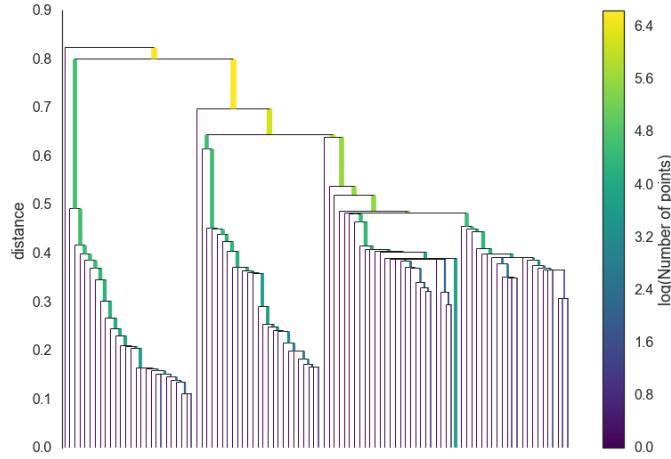


Figura 2.7: Esempio di gerarchia dei cluster derivata dalla rimozione progressiva degli archi del MST: i nodi mostrano come i gruppi nascano e si separino a densità crescenti.

4. Condensazione della gerarchia dei cluster

Una volta costruita la gerarchia completa delle componenti connesse, HDBSCAN applica un processo di **condensazione** basato sul parametro `min_cluster_size`. L’obiettivo è semplificare la struttura gerarchica, eliminando i rami instabili e i cluster di dimensioni troppo ridotte per essere considerati significativi.

Durante questa fase, la gerarchia dei cluster viene attraversata e vengono mantenuti soltanto quei gruppi che rispettano la dimensione minima richiesta. I cluster che non raggiungono la soglia stabilità vengono rimossi oppure fusi con componenti più grandi a densità simile, producendo una rappresentazione compatta e più interpretabile della struttura dei dati.

Il risultato di questa operazione è un *condensed tree*, ossia un albero di condensazione in cui ogni nodo rappresenta un cluster potenzialmente stabile e ogni arco riflette una relazione di inclusione o fusione tra cluster. Questo passaggio riduce notevolmente la complessità della gerarchia originale, preservando solo le regioni dello spazio che presentano un’evidente coesione interna e scartando i gruppi effimeri o rumorosi.

La condensazione costituisce quindi la base per la successiva selezione dei cluster più stabili, che rappresentano le strutture di densità realmente persistenti all'interno del dataset.

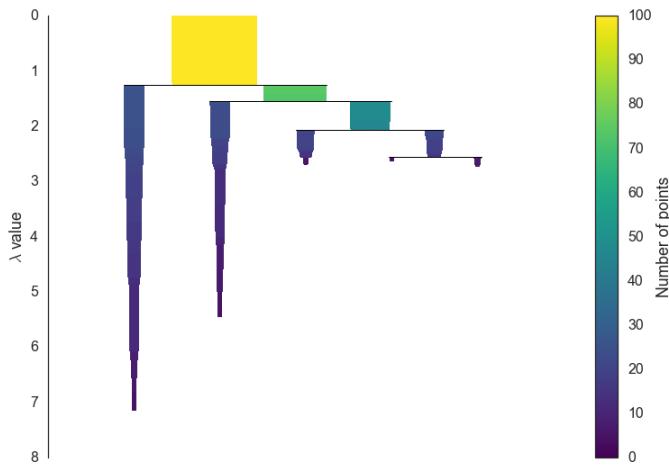


Figura 2.8: Albero condensato generato applicando `min_cluster_size`: vengono eliminati i rami instabili e restano solo i cluster con sufficiente supporto.

5. Estrazione dei cluster stabili dalla struttura condensata

Dopo aver ottenuto l'albero di condensazione, HDBSCAN seleziona i **cluster più stabili** in base alla loro persistenza lungo i diversi livelli di densità. Ogni cluster nella struttura condensata è caratterizzato da un intervallo di esistenza: esso “nasce” quando si separa da un cluster padre e “muore” quando, aumentando la soglia di densità, si frammenta o scompare.

Per ciascun cluster, HDBSCAN calcola una misura di **stabilità** che quantifica la sua persistenza all'interno della gerarchia. Durante questa fase, vengono mantenuti soltanto i cluster con una stabilità elevata, mentre i punti appartenenti a regioni meno dense o a cluster transitori vengono etichettati come *rumore* (label -1). Questo approccio consente di identificare automaticamente le strutture di densità più consistenti, senza richiedere la specifica del numero di cluster a priori.

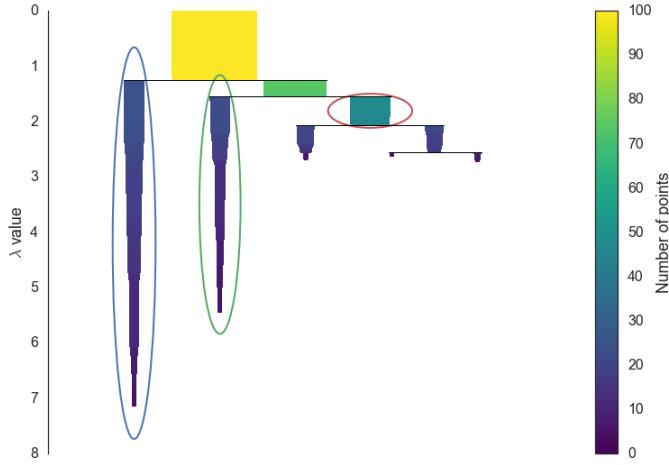


Figura 2.9: Cluster finali selezionati in base alla stabilità: i colori identificano le componenti persistenti, mentre i punti grigi rappresentano il rumore (label -1).

2.1.4 CountVectorizer

Nel topic modeling, dopo la creazione dei topic, si cerca una **rappresentazione** che descriva i topic in termini lessicali.

Il **CountVectorizer** rappresenta il primo passo per collegare la rappresentazione semantica dei documenti, ottenuta tramite gli *embeddings*, a una rappresentazione lessicale interpretabile. Il suo compito è quello di trasformare un insieme di testi in una matrice numerica in cui ogni riga rappresenta un documento e ogni colonna un termine del vocabolario; ciascun elemento M_{ij} indica quante volte il termine j compare nel documento i .

In questo modo, il **CountVectorizer** costituisce il ponte tra la rappresentazione numerica continua degli *embeddings* e la rappresentazione simbolica necessaria per calcolare le statistiche di c-TF-IDF, che servono a identificare le parole più rappresentative di ciascun topic.

2.1.5 c-TF-IDF

Come descritto nella documentazione ufficiale⁸, il c-TF-IDF può essere visto come l'adattamento della formula TF-IDF a un contesto multi-classe: tutti i documenti di una stessa classe vengono concatenati in un unico documento, da cui si calcola la frequenza dei termini x . Questa frequenza viene normalizzata con norma L1 e costituisce la **term frequency**.

Per un termine x appartenente alla classe c , il peso c-TF-IDF è definito come:

$$W_{x,c} = \|\text{tf}_{x,c}\| \times \log\left(1 + \frac{A}{f_x}\right)$$

dove:

- $W_{x,c}$ è il peso del termine x nella classe c ;
- $\text{tf}_{x,c}$ è la frequenza del termine x nella classe c ;
- f_x è la frequenza totale del termine x considerando tutte le classi;
- A rappresenta il numero medio di parole per classe.

Il termine $\|\text{tf}_{x,c}\|$ indica la normalizzazione della frequenza all'interno del topic, mentre il fattore $\log\left(1 + \frac{A}{f_x}\right)$ riduce il peso dei termini comuni tra le classi, privilegiando invece quelli che si presentano frequentemente in una sola classe. Questo consente di ottenere una rappresentazione più discriminante e stabile dei topic rispetto alla versione standard del TF-IDF, in quanto la rarità viene calcolata rispetto ai topic anziché rispetto ai singoli documenti.

⁸<https://maartengr.github.io/BERTopic/api/ctfidf.html>

Come viene usato il peso

Per ciascun topic c , BERTopic ordina i termini in base al peso c-TF-IDF in ordine decrescente e produce una lista di coppie *termine-peso*. Tale lista è accessibile tramite il metodo `topic_model.get_topic(topic_id)`.⁹

Il framework utilizza inoltre le prime parole più pesanti per generare un nome sintetico del topic, consultabile con `topic_model.get_topic_info()`.

2.1.6 Maximal Marginal Relevance (MMR)

c-TF-IDF è una tecnica basata esclusivamente sulla rilevanza statistica e quindi tende spesso a selezionare termini ridondanti o molto simili tra loro. Ciò accade perché le parole semanticamente affini come “scientist”, “scientists”, “science” condividono contesti lessicali simili e presentano valori di peso elevati, riducendo la capacità discriminativa dell’etichetta del topic.

Per ovviare a questo problema, BERTopic può impiegare un meccanismo chiamato **Maximal Marginal Relevance (MMR)**. L’obiettivo di MMR è bilanciare la *rilevanza* di un termine rispetto al topic con la sua *novità* rispetto ai termini già selezionati, ottenendo così una rappresentazione più informativa e diversificata.

MMR seleziona iterativamente i termini D_i che massimizzano la seguente funzione:

$$\text{MMR}(D_i) = \arg \max_{D_i \in R \setminus S} \left[\lambda \cdot \text{Rel}(D_i, Q) - (1 - \lambda) \cdot \max_{D_j \in S} \text{Sim}(D_i, D_j) \right]$$

dove:

- R è l’insieme dei candidati termini,
- S è l’insieme dei termini già selezionati,

⁹https://maartengr.github.io/BERTopic/api/bertopic.html#bertopic._bertopic.BERTopic.get_topic

- $\text{Rel}(D_i, Q)$ misura la rilevanza del termine D_i rispetto al topic Q (spesso basata sulla similarità coseno con il vettore medio del topic),
- $\text{Sim}(D_i, D_j)$ rappresenta la similarità semantica tra due termini,
- $\lambda \in [0, 1]$ è un parametro di bilanciamento tra rilevanza e diversità.

Un valore di λ elevato privilegia la rilevanza (selezionando termini molto centrali ma potenzialmente ridondanti), mentre un valore più basso favorisce la diversità semantica, producendo etichette più varie e descrittive. In pratica, valori compresi tra 0.5 e 0.7 risultano un buon compromesso nella maggior parte dei casi, specialmente in domini con linguaggio ripetitivo come gli annunci di lavoro.

In BERTopic, l'algoritmo MMR viene applicato alla rappresentazione finale dei topic per scegliere le parole chiave che ne definiscono il nome. Ciò consente di ottenere etichette più leggibili e significative, riducendo la presenza di lemmi sinonimici o morfologicamente simili e migliorando la capacità interpretativa del modello. Vediamo adesso come abbiamo deciso di impostare il framework per fare *topic modeling* su annunci di lavoro.

Capitolo 3

Implementazione di BERTopic

3.1 Embedding

I modelli principali consigliati nella documentazione¹ sono i **sentence transformers** (SBERT). Ci sono molti modelli SBERT pre-addestrati tra cui scegliere, la documentazione ufficiale nei suoi esempi usa spesso all-MiniLM-L6-v2 un modello estremamente leggero e veloce, però non sufficientemente preciso nel catturare sfumature semantiche complesse o relazioni a lungo raggio. paraphrase-MiniLM-L12-v2 e paraphrase-mpnet-base-v2, sono più accurati, ma più specializzati nel catturare relazioni di parafrasi, fra testi molto simili, sono inoltre inadatti a testi particolarmente lunghi come gli annunci di lavoro. **all-mpnet-base-v2**, invece, ha una alta **precisione semantica**, e si comporta bene con documenti di lunghezza **medio-lunga**. Ha però un limite di lunghezza di **512 token**. Ecco alcune informazioni sulla lunghezza del nostro dataset (dopo la pulizia):

¹https://maartengr.github.io/BERTopic/getting_started/embeddings/embeddings.html

	n chars	n words	n tokens mpnet
count	5357.00	5357.00	5357.00
mean	3315.48	457.92	594.54
std	1591.28	219.01	283.37
min	91.00	14.00	23.00
50%	3087.00	427.00	554.00
75%	4060.00	562.00	729.00
90%	5213.40	717.00	934.00
95%	6188.20	846.20	1103.40
99%	8797.52	1203.44	1562.72
max	19470.00	2739.00	3753.00
Testi che superano i 512 token: 3075 (57.40%)			
Totale annunci: 5357			

Figura 3.1: Statistiche globali del dataset (totale annunci: 5357).

Quando SBERT riceve un input troppo lungo, lo **tronca** questo significa che applicando l'embedder così come è verrebbe tagliata una porzione molto grande del dataset.

Le prime opzioni che abbiamo considerato sono:

1. Affidarsi ad un modello con un limite di dimensione input più alto (ad esempio **all-roberta-long-v1**)
2. Frammentare le descrizioni e fare il topic modeling in segmenti di quest'ultime.

Il vantaggio della prima opzione è che i modelli *long-context*, mantengono attenzione tra termini distanti, permettendo di cogliere relazioni semantiche a lungo raggio, cosa impossibile se si divide il testo in *chunk*. Non è però una caratteristica che abbiamo ritenuto significativa, data la natura del nostro *dataset*, infatti i nostri paragrafi hanno natura sciolta, uno potrebbe parlare di competenze e un altro di mansioni, la coerenza globale è più importante in testi di natura **discorsiva**. Inoltre se la lunghezza del testo è grande è probabile che contenga più temi, un unico embedding rischia di mescolare **sezioni semanticamente diverse**, ottenendo topic più grossolani.

La seconda opzione invece, oltre che generare topic più **puliti** e **interpretabili**, consente di aumentare molto il numero di documenti in input e questo è un vantaggio per dataset esegui come il nostro. Però anche questo caso presenta delle criticità: non avremmo i topic relativi agli annunci come uniche entità, ma saranno relativi ai segmenti e andrebbero aggregati a posteriori per ottenere una **distribuzione di topic per annuncio**. In più in questo modo, gli annunci più lunghi avranno **più peso**, semplicemente perché hanno più paragrafi. La situazione si complica ulteriormente se i documenti contengono paragrafi simili, questo creerebbe dei **cluster** artefatti e quindi topic che non rispecchiano la natura del dataset. Un altro problema è che con questo metodo il modello è completamente ignaro della struttura globale dell'annuncio. Questo approccio è più attuabile per risolvere un problema di **classificazione**, non per il *topic modeling*.

La soluzione che abbiamo scelto è dunque quella di *mean-max-pooling*.

3.1.1 Mean-max pooling

Con questa tecnica si ottiene un embedding per ogni annuncio partendo dagli embedding dei paragrafi, seguendo i passaggi seguenti:

1. calcolare gli embedding di ciascun paragrafo;
2. normalizzare ogni embedding;
3. calcolare media aritmetica e massimo elemento per elemento;
4. concatenare i due vettori risultanti;
5. normalizzare nuovamente il vettore concatenato.

In questo modo si ottiene un embedding unico di dimensione doppia rispetto l'output dell'embedder che può catturare sia il **significato generale**

dell'annuncio, sia le **feature salienti** dei singoli paragrafi: queste due componenti possono produrre vettori più **discriminativi** e quindi più facilmente separabili nello spazio semantico.

Per attuare questo approccio serve inoltre una strategia di segmentazione degli annunci che soddisfi al contempo **coerenza semantica** e criteri di lunghezza: il segmento non può essere troppo corto, altrimenti SBERT non ha il contesto per estrarre topic significativi, e non può ovviamente superare la lunghezza massima consentita. Approfondiamo questo aspetto nel capitolo **Pre-processing**.

3.2 Dimensionality Reduction

Il parametro più importante di UMAP è *n neighbors*. I valori con cui abbiamo sperimentato sono: 5, 15, 25, 35, 50 e 100 per calibrare il livello di granularità dei topic: valori molto bassi frammentano in gruppi minimi, mentre impostazioni alte tendono a fondere i cluster semanticamente più affini; una scelta intermedia (ad esempio `n_neighbors = 50`) offre un equilibrio adeguato tra dettaglio locale e coerenza globale.

Per giungere a questa scelta sono stati provati diversi set di iperparametri; riportiamo di seguito quelli usati per la configurazione finale, così da facilitarne la riproducibilità:

```
umap_model = UMAP(  
    n_neighbors=n,  
    n_components=10,  
    min_dist=0.0,  
    metric="cosine",  
    random_state=1,  
)  
hdbscan_model = HDBSCAN(  
    min_cluster_size=50,
```

```

min_samples=15,
metric="euclidean",
cluster_selection_method="eom",
prediction_data=True,
cluster_selection_epsilon=0.1,
)

```

<i>n_neighbors</i>	<i>n_topics</i>	mean cluster size	std cluster size	topics < 100	topics > 500	total docs assigned
5	20	180.50	180.42	9	2	3610
15	15	208.87	267.01	8	2	3133
25	12	259.42	237.23	4	2	3113
35	12	235.33	176.29	4	1	2824
50	11	286.36	248.59	3	1	3150
100	2	2396.50	2090.91	0	2	4793

Tabella 3.1: Sintesi degli esperimenti al variare di *n_neighbors*: numero di topic trovati, dimensione media e deviazione standard dei cluster, oltre al conteggio dei topic piccoli (< 100 documenti), di quelli molto grandi (> 500 documenti) e al totale di documenti assegnati.

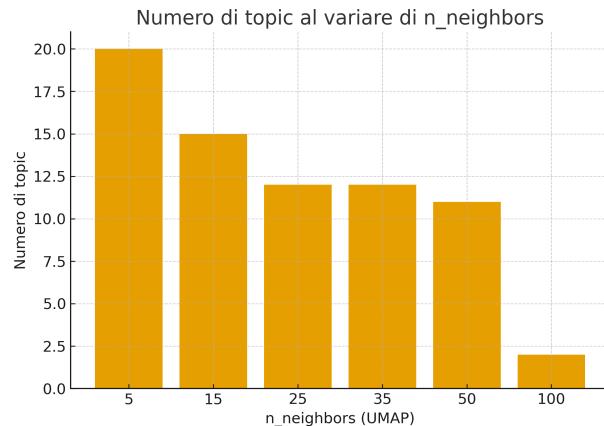


Figura 3.2: Numero di topic generati da HDBSCAN al variare di *n_neighbors*: la scelta intermedia ($n_{neighbors} = 50$) massimizza la granularità senza collassare in pochi cluster.

I plot in Figura 3.3 mostrano la proiezione bidimensionale generata da `BERTopic.visualize_documents()` per tre configurazioni distinte. Con `n_neighbors = 15` UMAP costruisce un grafo molto **frammentato**: le connessioni tra i documenti sono deboli, molti punti rimangono isolati e HDBSCAN li etichetta come *outlier*, generando quindi un topic -1 (il topic dove vengono inseriti i documenti di cui si è incerti) estremamente popoloso (2224 documenti). All'estremo opposto (`n_neighbors = 100`) il grafo è troppo **compatto**: le relazioni globali dominano e i topic tendono a fondersi. Con `n_neighbors = 50`, invece, i cluster si addensano in regioni coerenti, evidenziando macro-categorie professionali distinte (marketing, cybersecurity, biomedical engineering, *etc.*). L'incrocio tra indicatori numerici e analisi visiva conferma quindi `n_neighbors = 50` come valore finale.

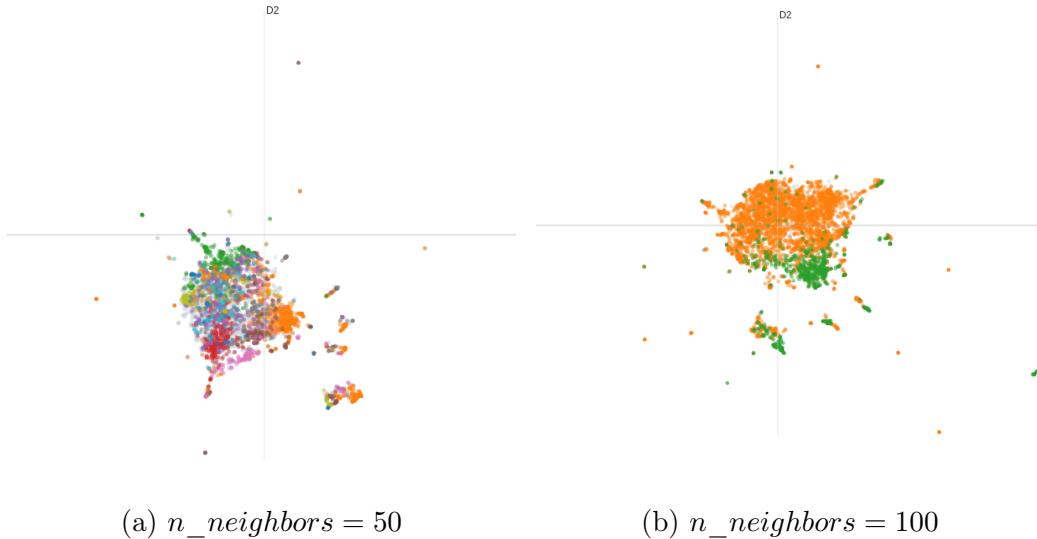


Figura 3.3: Proiezioni generate da `BERTopic.visualize_documents()`: i punti (embedding ridotti a 2 dimensioni) sono colorati in base al topic. Confronto tra `n_neighbors` pari a 50 e 100.

Abbiamo fissato `min_dist` a 0.0 poiché l'algoritmo di *clustering* (HDBSCAN) restituisce risultati migliori con cluster densi e ben separati: eventuali regioni

vuote verrebbero classificate come outlier e assegnate al topic “spazzatura” (-1). Mantenere gli embedding compatti riduce la formazione di tali zone.

3.3 Clustering

Abbiamo però introdotto un parametro `min_samples`: al suo crescere aumenta la `core_distance` media e con essa i punti considerati rarefatti. Quindi più è alto `min_samples`, più embedding verranno considerati come *rumore*. Come per `n_neighbors`, mostriamo qui dei confronti con diversi valori di `min_samples`:

<code>min_samples</code>	<code>n_topics</code>	noise %	mean size	median size	max size	top topic %	outliers
5	12	42.28	257.67	190.0	882	28.53	2265
15	61	45.88	224.98	131.0	1581	11.52	11635
30	10	39.69	323.10	209.0	886	27.42	2126
60	2	14.80	2282.00	2282.0	3699	81.05	793

Tabella 3.2: Metriche chiave al variare di `min_samples`: numero di topic (`n_topics`), percentuale di rumore (noise), statistiche di dimensione dei cluster, quota del topic principale e conteggio degli outlier.

Per valori molto bassi (`min_samples = 5`) l’algoritmo tende a frammentare eccessivamente il corpus, generando numerosi micro-cluster spesso composti da pochi paragrafi. Sebbene questa impostazione aumenti il livello di dettaglio, i topic risultano meno stabili e semanticamente meno coerenti.

Aumentando il parametro a valori intermedi (`min_samples = 15`) si ottiene un equilibrio tra granularità e stabilità. I cluster formati risultano sufficientemente omogenei e interpretabili, mantenendo al contempo un numero di topic adeguato per una lettura tematica fine del corpus. Questo valore rappresenta il compromesso ottimale tra dettaglio informativo e robustezza del clustering.

Con valori più elevati (`min_samples = 30`) l’algoritmo diventa più selettivo: la quantità di rumore aumenta e alcuni cluster tendono a fondersi,

riducendo la capacità di distinguere tematiche affini ma distinte. Spingendosi ulteriormente verso valori molto alti (`min_samples` = 60) si osserva un drastico calo del numero di cluster e un incremento marcato dei punti etichettati come rumore, con il rischio di ottenere pochi macro-temi troppo generici.

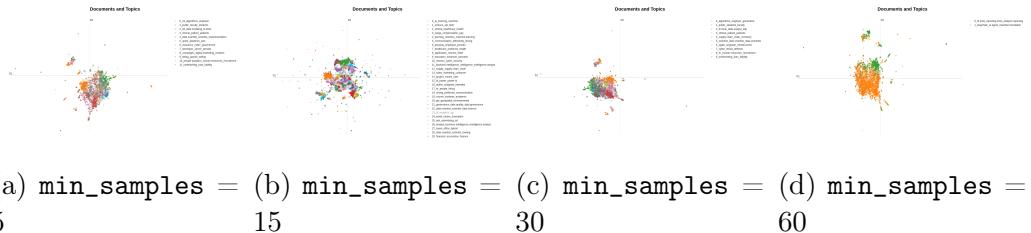


Figura 3.4: Effetto di `min_samples` sulla distribuzione dei topic: visualizzazioni prodotte da `BERTopic.visualize_documents()` in configurazioni crescenti del parametro.

Nel complesso, la configurazione con `min_samples` = 15 si è dimostrata la più bilanciata: garantisce la formazione di cluster stabili, coerenti e sufficientemente dettagliati, risultando quindi la scelta più appropriata per il corpus di annunci di lavoro considerato.

3.3.1 CountVectorizer

La configurazione del vettorizzatore influenza fortemente la qualità dei topic risultanti. Tra i principali parametri utilizzabili si distinguono:

Parametro	Descrizione
<code>stop_words</code>	Elenco di parole da escludere (articoli, preposizioni, ecc.). Può assumere valori come <code>'english'</code> o una lista personalizzata di termini.
<code>ngram_range</code>	Tuple che definisce la lunghezza minima e massima delle n-gram; ad esempio <code>(1, 2)</code> include sia unigrammi che bigrammi, consentendo di catturare espressioni come <i>data analysis</i> .
<code>min_df</code>	Numero minimo di documenti (o frazione) in cui un termine deve apparire per essere incluso nel vocabolario; filtra i termini troppo rari.
<code>max_df</code>	Numero massimo di documenti (o frazione) in cui un termine può apparire prima di essere escluso; filtra i termini troppo frequenti o generici.

Tabella 3.3: Principali parametri del `CountVectorizer`.

Alle normali *stop words* inglesi abbiamo aggiunto alcune specifiche che rispecchiano il gergo del nostro dataset:

```
custom_stopwords = [
    # anni e numeri
    "2024", "2025", "2026", "19xx", "20xx",

    # frasi burocratiche / pattern ricorrenti
    "regular", "job", "regular job", [...]
    # termini amministrativi o HR generici
    "eligibility", "degree", "foreign", "equivalent", [...]
    # altri pattern burocratici o poco informativi
    "agency", "equipment", "reports", "dashboards", "hoc",
    "solutions",
    # forme sinonimiche o doppioni rumorosi
```

```
"equivalency years", "years", "regular job code", "employment type"
"type regular job", "employment type", "regular job",
# placeholder testuali
"institutional", "equivalency", "perform essential",
]
```

Capitolo 4

Data Cleaning

4.1 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 comparando 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 4.1).

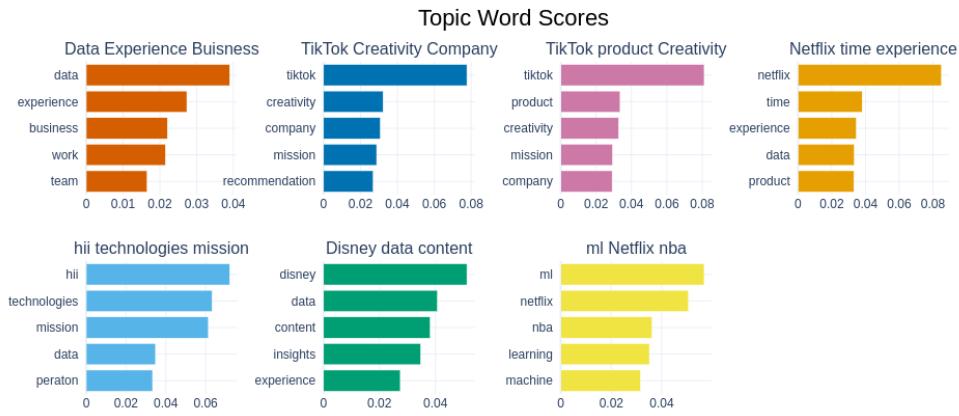


Figura 4.1: Barplot ottenuto da *topic modeling* senza *preprocess*.

Come visto nel capitolo precedente, a ogni parola è associato uno *score* che ne rappresenta l’importanza all’interno del topic. Il barplot in Figura 4.1, rappresentante il risultato finale in termini di topic e keywords calcolati sul dataset senza preprocessing, 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 di alcune 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, dove aziende Big Tech come quelle citate 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 4.2: 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 altri 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**.

4.2 Divisione in paragrafi

La suddivisione di un testo in paragrafi semanticamente coerenti è un problema aperto nella disciplina del *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("br/>    r"job\s+description|about\s+the\s+role|responsabilit|activit|"br/>    r"requirements?|qualifications?|what\s+you'll\s+(do|be\s+doing)|"br/>    r"skills|nice\s+to\s+have|profilo"br/>    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 sufficientemente flessibile, per questo abbiamo scelto di addestrare una rete neurale.

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

4.2.1 Spacy

Spacy è una libreria per *NLP* basata su *pipelines* composte da moduli modificabili e allenabili. Questa componente altamente personalizzabile della libreria

ria 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.

¹<https://spacy.io/api/spancategorizer>

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

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 4.3: 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.

Abbiamo 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-gram* 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 4.4: 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 4.5: 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 4.6: 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

³https://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"
n0 = 1
nI = null
```

Figura 4.7: Scorer logistico lineare: un'unica uscita sigmoide ($n0 = 1$) 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 conto 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 4.8: 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.*

4.2.2 Valutazione del modello

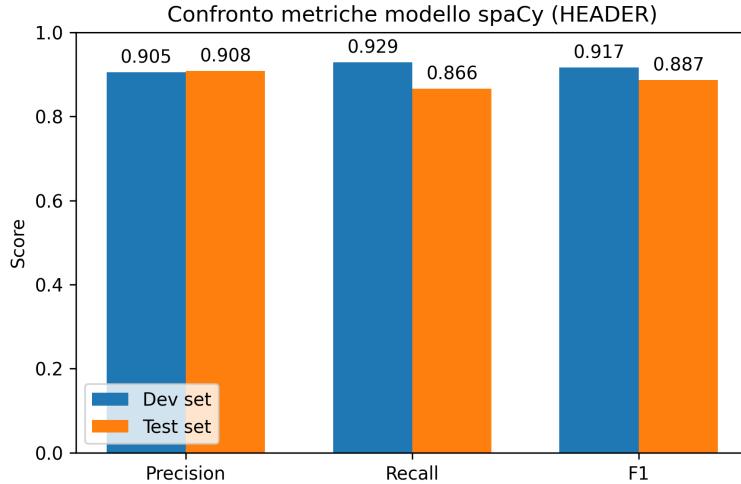


Figura 4.9: Confronto delle metriche *precision*, *recall* e *F-score* tra la valutazione fatta nel dev set e quella nel test set.

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 4.9, 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.

4.2.3 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;

```

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 4.10: 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.

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.

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.

4.3 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.

4.3.1 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 4.11: 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 4.12: 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 4.13.

Balanced (1.0)							
	Prec.	Rec.	F1		DET.	BLURB	JOB
DETAIL	0.87	0.93	0.90	DET.	253	13	7
BLURB	0.85	0.90	0.88	BLURB	20	306	13
JOB	0.97	0.92	0.94	JOB	18	41	637
Slightly unbalanced (1.5)							
	Prec.	Rec.	F1		DET.	BLURB	JOB
DETAIL	0.91	0.85	0.88	DET.	231	22	20
BLURB	0.89	0.81	0.85	BLURB	16	275	48
JOB	0.91	0.97	0.94	JOB	8	13	675
Unbalanced (2.0)							
	Prec.	Rec.	F1		DET.	BLURB	JOB
DETAIL	0.91	0.84	0.88	DET.	230	20	23
BLURB	0.89	0.80	0.85	BLURB	15	272	52
JOB	0.90	0.97	0.94	JOB	7	12	677

Figura 4.13: 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 4.14: 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.

4.4 Altre strategie di data cleaning usate

Dopo la pulizia a livello di paragrafo analizziamo l'effetto su BERTopic. La Figura 4.15 mostra il dendrogramma ottenuto con il dataset parzialmente

pulito.

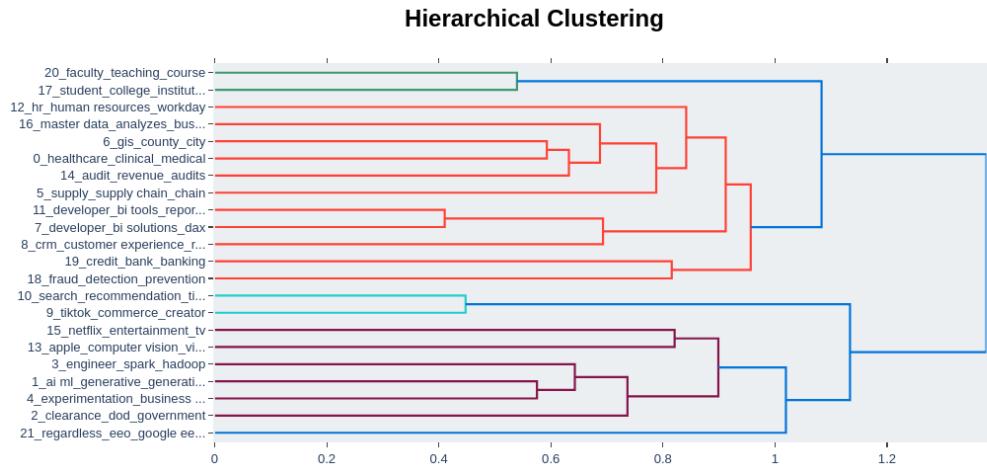


Figura 4.15: Dendrogramma di BERTopic su dataset parzialmente pulito.

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 4.16: 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 4.17 tramite BERTopic.

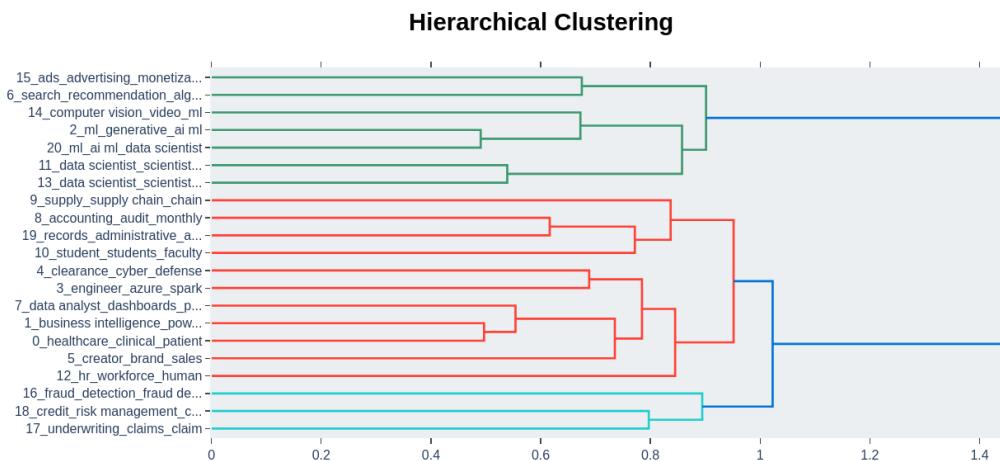


Figura 4.17: Dendrogramma di BERTopic su dataset completamente pulito.

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.

4.5 Possibili miglioramenti

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

Capitolo 5

Risultati finali

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

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

Capitolo 6

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.