



Analisi multimodale di topic tramite il clustering di tweet e immagini

Dipartimento di Ingegneria Informatica, Automatica e Gestionale
Facoltà di Ingegneria dell'Informazione, Informatica e Statistica
Laurea in Ingegneria Informatica

Giovanni Tagliaferri

Matricola 1905922

Relatore

Prof.ssa Irene Amerini

Correlatore

Dr. Luca Maiano

Anno Accademico 2021/2022

Analisi multimodale di topic tramite il clustering di tweet e immagini
Tesi di Laurea Triennale. Sapienza Università di Roma

© 2022 Giovanni Tagliaferri. Tutti i diritti riservati

Questa tesi è stata composta con L^AT_EX e la classe Sapthesis.

Email dell'autore: tagliaferri.1905922@studenti.uniroma1.it

Dedicato a me stesso.

Sommario

Nell'ambito di questa tesi si propone una tecnica di analisi di tweet utilizzando metodi di clustering basati testi e immagini. Il lavoro presenta una panoramica sul mondo dell'intelligenza artificiale, il *machine learning* e il *deep learning*, per arrivare poi a trattare in modo più dettagliato il clustering. Vengono riportati diversi algoritmi tra cui *k-means*, *DBSCAN* e *HDBSCAN* che verranno utilizzati per effettuare il clustering di testi e immagini estratti dai tweet. Viene prima mostrato come costruire un dataset di tweet di interesse e come questi vengono elaborati e trasformati in *embeddings* (vettori numerici) tramite le tecniche del *Natural Language Processing*, in modo che il computer possa comprenderli e applicarvi gli algoritmi.

L'analisi viene condotta sul tema del conflitto bellico tra Russia e Ucraina dal 24 febbraio 2022 al 24 febbraio 2023 e i tweet vengono divisi in tre dataset analizzati nell'intervallo temporale in cui sono stati pubblicati. Gli algoritmi di clustering vengono applicati su ogni dataset e per ognuno vengono estratti dei topic che, confrontati con quelli degli altri, permettono di studiare a grandi linee l'evoluzione del conflitto.

Indice

1	Introduzione	1
2	Intelligenza Artificiale	3
2.1	Machine Learning	4
2.2	Machine Learning Supervisionato	5
2.2.1	Classificazione	6
2.2.2	Regressione	6
2.3	Machine Learning Non Supervisionato	7
2.3.1	Clustering	8
2.3.2	Association Rules	8
2.3.3	Dimensionality Reduction	8
2.4	Machine Learning Semi-Supervisionato	9
2.5	Apprendimento per rinforzo	10
2.6	Deep Learning	10
2.6.1	Reti Neurali	11
3	Clustering	13
3.1	Cos'è il Clustering?	13
3.2	Somiglianza e distanza	14
3.3	Indicatori di validità	17
3.3.1	Internal evaluation indicators	17
3.3.2	External evaluation indicators	21
3.4	Algoritmi di clustering	24
3.5	Centroid-based clustering	24
3.5.1	K-Means	25
3.6	Hierarchical Clustering	27
3.6.1	HC Agglomerativo	27
3.6.2	HC Divisivo	28
3.6.3	Calcolo della distanza	28
3.6.4	BIRCH	31
3.7	Density-Based Clustering	33
3.7.1	DBSCAN	35
3.8	Distribution-Based Clustering	37
3.8.1	Gaussian-Mixture model	38
3.9	Fuzzy Clustering	38
3.9.1	Fuzzy C-Means	39

3.10 HDBSCAN	40
4 Natural Language Processing	47
4.1 Cosa è il NLP?	47
4.2 Funzionamento del NLP	48
4.2.1 Bag of words (BoW)	48
4.2.2 TF-IDF	49
4.3 Word2Vec	50
4.3.1 Skip-Gram	51
4.3.2 CBOW	53
4.3.3 Embeddings per le immagini	54
4.4 Keyword extraction	55
4.4.1 Funzionamento	57
4.4.2 Utilizzi	57
4.4.3 BERT	58
5 Metodo proposto	61
5.1 Clustering dei testi	61
5.1.1 Embeddings e Curse of dimensionality	61
5.1.2 Feature Scaling	62
5.1.3 K-Means	63
5.1.4 DBSCAN	64
5.1.5 HDBSCAN	66
5.1.6 Keywords extraction e wordcloud	71
5.2 Clustering delle immagini	71
6 Analisi dei risultati	75
6.1 Analisi dei testi	75
6.1.1 Collezione testi Febbraio - Giugno 2022	76
6.1.2 Collezione testi Luglio - Novembre 2022	87
6.1.3 Collezione testi Dicembre 2022 - Febbraio 2023	91
6.2 Analisi delle immagini	95
6.2.1 Collezione immagini Febbraio - Novembre 2022	95
6.2.2 Collezione immagini Dicembre 2022 - Febbraio 2023	98
6.3 Topic estratti	104
7 Dettagli implementativi	107
7.1 Cosa è una API?	107
7.2 Esempi di API	109
7.3 Tipi di API e protocolli	110
7.4 API di Twitter	111
7.4.1 Accesso ai dati di Twitter	111
7.5 Creazione del dataset	112
7.5.1 Importazione dei tweet	112
7.6 NLP sui testi	115
8 Conclusione e sviluppi futuri	119

Capitolo 1

Introduzione

Prima di studiare un qualsiasi fenomeno o materia, un passo fondamentale da applicare a priori è quello di classificare i dati a disposizione in diverse categorie. L'uomo, sin da bambino, possiede questa innata capacità di categorizzare gli elementi del mondo che lo circonda sulla base di criteri dettati dall'osservazione e dall'esperienza: un bambino è in grado di suddividere i propri giocattoli per tipo o per colore, di identificare diverse specie animali in base alla grandezza, alle caratteristiche del corpo e al verso che fanno, o ancora capire (magari a proprie spese) con quali oggetti è meglio non giocare se non ci si vuole far male.

La classificazione dei dati nel mondo odierno gioca un ruolo importante negli ambiti più disparati: le aziende hanno bisogno di conoscere le caratteristiche dei propri clienti, in modo da poter indirizzare al meglio i loro prodotti e servizi al giusto target. Per esempio Netflix e Amazon tramite particolari algoritmi riescono a consigliare al cliente sempre film e prodotti secondo gusti e esigenze personali di ogni cliente; scienziati, governi e organizzazioni possono voler effettuare analisi statistiche su un particolare topic per capire come si evolve nel tempo, come potrebbe essere lo studio dell'andamento della recente campagna di vaccinazione contro il Covid-19; per effettuare l'analisi di un'immagine si passa attraverso la *segmentazione*, ossia distinguere all'interno di essa diversi soggetti di interesse, utile in ambito medico, nella realizzazione di un sistema di trasporto con guida automatica, nel riconoscimento facciale, in analisi satellitari, ecc.

Negli ultimi decenni dopo l'avvento dell'internet la produzione di dati informatici ha visto una crescita esponenziale grazie soprattutto alla nascita dei social network, che da un lato porta il grande vantaggio di ottenere informazioni pressoché illimitate, dall'altro rende impossibile per l'uomo analizzarli e classificarli manualmente. Basti pensare che su Twitter in media vengono pubblicati 6000 tweet al secondo e di conseguenza circa mezzo miliardo al giorno. Una caratteristica di questi dati è quella di essere *non etichettati*, ossia a priori non c'è nessun criterio che permette di

distinguerli l'uno dall'altro per suddividerli in gruppi. Quindi, oltre all'analisi in sé, anche la ricerca di un criterio di suddivisione diventa difficoltosa.

Questa problematica è soltanto una delle tante che trovano soluzione nel mondo dell'**Intelligenza artificiale** e del **Machine Learning** (capitolo 2). Le macchine, sebbene attualmente non abbiano la stessa intelligenza dell'uomo, in seguito a un processo di addestramento possono svolgere automaticamente dei compiti in maniera altamente efficace ed efficiente che per l'uomo risulterebbero impossibili da completare. In particolare il processo di raccolta e classificazione dei dati non etichettati prende il nome di *data mining* e la principale tecnica utilizzata per effettuarlo viene chiamata **clustering** ed è il punto focale di questa trattazione.

Nel capitolo 3 vengono trattati in dettaglio i principi su cui si basa il clustering e diverse tipologie di algoritmi, mostrando per ognuno i suoi pregi e difetti e su quali *dataset* (collezione di dati in forma tabellare) restituiscono i risultati migliori. In seguito ne verranno utilizzati solo alcuni per effettuare una analisi sugli eventi della guerra in Ucraina con lo scopo di estrarre i topic più rilevanti nel primo anno in cui si è svolta (capitoli 5 e 6). L'analisi si divide nelle seguenti quattro fasi:

1. la prima consiste nella raccolta di tweet riguardanti l'argomento e si concentra nel periodo che va dal giorno dell'invasione armata da parte della Russia il giorno 24 febbraio 2022 fino al 24 febbraio 2023 (capitolo 7);
2. la seconda prevede l'estrazione di testi e immagini dai tweet, i quali però non possono essere elaborati così come sono dal computer. Infatti le macchine non riescono comprendere il linguaggio umano direttamente e ad estrarre informazioni dalle immagini senza effettuare delle operazioni di pre-elaborazione, che consistono nell'analisi del linguaggio umano (*Natural Language Processing* trattato nel capitolo 4) e convertire i dati in formato numerico, questa volta comprensibile al computer. Per le immagini, inoltre, viene utilizzato anche un modello che ne genera una descrizione testuale per effettuare un ulteriore studio in combinazione con i testi dei tweet;
3. vengono applicati i diversi algoritmi di clustering, prima solo sui testi dei tweet, poi sulle immagini, estraendo così diversi topic dai tweet;
4. infine si mostrano i risultati ottenuti per ogni algoritmo e li si confrontano per decretare quale sia l'algoritmo più adatto per questo genere di analisi (capitolo 6).

Capitolo 2

Intelligenza Artificiale

L’Intelligenza Artificiale nasce nel tentativo di dare alle macchine un’intelligenza simile a quella dell’uomo. Attualmente è impossibile ricreare artificialmente una mente umana, ma è possibile emulare alcune attività. Esistono due definizioni di intelligenza:

- intelligenza vista come collezione di specifiche funzionalità;
- intelligenza come la capacità di apprendere abilità.

Assumendo vere entrambe le definizioni, l’intelligenza presente nelle macchine venne nominata per la prima volta come *Artificial Intelligence* (o *AI*) nella conferenza **Dartmouth Summer Research Project on Artificial Intelligence** nel 1956 e sin da quel momento le idee principali su come realizzarla si basavano sui seguenti approcci:

- imparare dagli esperti o *symbolic AI*: la conoscenza deriva dal sapere dell’uomo e viene rappresentata tramite regole e simboli che vengono poi manipolati e combinati in maniera logica per risolvere problemi e prendere decisioni;
- imparare dai dati: le macchine vengono addestrate e apprendono a partire dai dati di interesse ed è su questo che si baserà poi il **Machine Learning**.

Il primo approccio aveva delle limitazioni, tra le quali una fonte di conoscenza limitata del mondo reale, l’incapacità di gestire le incertezze, la difficoltà nel gestire grandi quantità di dati e una limitata capacità di apprendimento, pertanto nel corso del tempo è caduto in disuso ed oggi è noto con il nome di *Good Old-Fashioned AI*. Il secondo invece, grazie allo sviluppo della potenza di calcolo dei computer e all’aumento esponenziale della disponibilità di dati, è quello ad oggi più studiato e il machine learning è diventato quasi un omonimo di intelligenza artificiale.

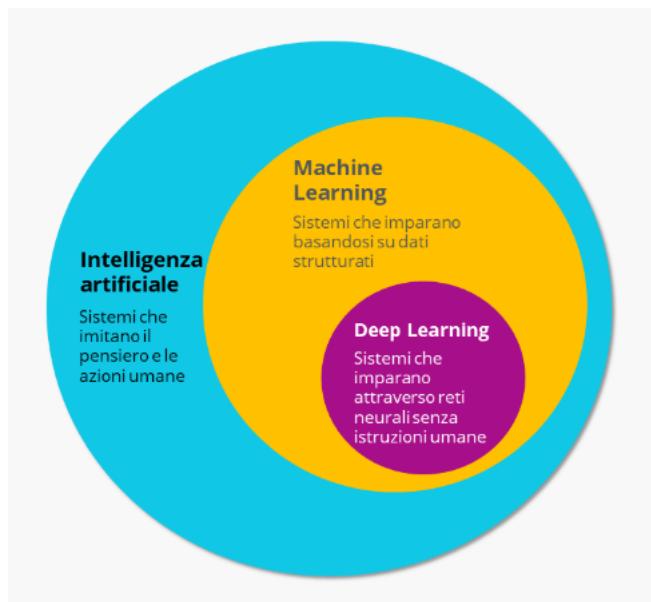


Figura 2.1. Schema intelligenza artificiale

Al concetto di intelligenza artificiale vengono spesso affiancati e, a volte erroneamente sovrapposti, quello di **Machine Learning** e **Deep Learning** (figura 2.1). Nel corso del capitolo faremo chiarezza su queste importanti nozioni.

2.1 Machine Learning

Il **Machine Learning** è un campo del mondo dell'intelligenza artificiale che si occupa di estrarre informazioni a partire da un insieme di dati provenienti dalla realtà. Negli ultimi anni c'è stata l'esplosione dei *Big Data*, ossia dati informatici di grandi dimensioni e difficili da analizzare manualmente, pertanto il machine learning offre uno strumento di analisi e ricerca di pattern all'interno di questa enorme mole di dati.

Lo sviluppo e l'implementazione di un modello di machine learning passa attraverso 5 livelli:

1. **Data Collection e Data Preparation:** il primo passo consiste nella raccolta dei dati di interesse in un *dataset* e prepararli secondo le esigenze del modello che lo andrà ad analizzare, portandolo nel giusto formato, estraendo le *features* (caratteristiche) più importanti e ridimensionarlo correttamente;
2. **Model Training o fitting:** è la fase di addestramento del modello secondo il dataset che gli viene passato, che consiste nell'apprendimento di determinati algoritmi e la ricerca di pattern e relazioni tra i dati;

3. **Model Evaluation:** dopo l'addestramento vengono valutati i risultati del modello per cercare di massimizzare la precisione e le prestazioni. Vengono effettuati diversi test su dataset diversi e si confrontano i risultati tra loro;
4. **Model Deployment:** si può utilizzare il modello per risolvere i problemi reali;
5. **Model Maintenance:** è importante monitorare costantemente le prestazioni del modello e prendere i dovuti accorgimenti affinché funzioni sempre correttamente o per migliorare ulteriormente le prestazioni.

Esistono diverse forme di machine learning a seconda dell'obiettivo da realizzare: **supervisionato, non supervisionato, semi-supervisionato, per rinforzo.**

2.2 Machine Learning Supervisionato

Nel **Machine Learning Supervisionato** il compito degli algoritmi, a partire da un insieme di dati in input e un insieme di dati in output, è quello di ricavare la funzione che lega ogni elemento in ingresso al corrispettivo in uscita. Dopo il processo di addestramento, l'algoritmo sarà in grado di predire l'uscita a partire da un ingresso che non ha mai visto prima d'ora applicando la funzione che ha trovato.

In modo più formale, date delle variabili input indicate con X e una variabile in uscita Y , l'algoritmo deve ricavare la funzione f tale che $Y = f(X)$.

Un esempio molto comune di machine learning supervisionato lo si ritrova nell'ambito delle previsioni meteorologiche. Per predire il meteo in una data località è necessario avere in input un insieme di parametri come i dati storici su temperatura, precipitazioni, vento, umidità, ..., in quel luogo, con il corrispettivo insieme di output rappresentato dagli eventi che si sono verificati (ha piovuto oppure è stato bel tempo, ha nevicato o no, ecc.). In seguito all'addestramento, il modello sarà in grado di stabilire più o meno accuratamente le previsioni del meteo a partire da parametri mai visti.

È importante capire che il modello può solo imitare esattamente ciò che gli viene mostrato, pertanto è fondamentale utilizzare esempi affidabili e imparziali. Inoltre, affinché l'apprendimento sia corretto, è necessaria un grande quantità di dati etichettati ed è questa la parte più difficile e dispendiosa dell'apprendimento supervisionato.

Essere in grado di adattarsi ai nuovi input e generare predizioni è la parte cruciale nella creazione di un modello di machine learning supervisionato. Durante l'addestramento si vuole massimizzare la **generalizzazione** cosicché il modello sia in grado di trovare, quanto più possibile, la reale funzione generale. Se il modello

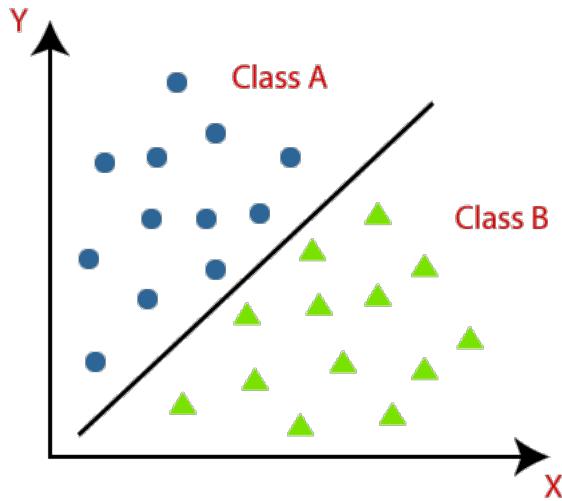


Figura 2.2. Classificazione

viene addestrato troppo su dati simili (*over-trained*) si causa il cosiddetto *over-fitting* e quindi non è in grado di adattarsi a nuovi input leggermente diversi da quelli che ha visto.

Il machine learning supervisionato prevede due categorie: **classificazione** e **regressione**.

2.2.1 Classificazione

La **classificazione** viene utilizzata per raggruppare tra loro elementi simili a partire da un insieme generale con lo scopo di classificarli. A un algoritmo di classificazione viene dato un insieme di dati (o *dataset*) con ognuno assegnato a una *label* (etichetta), ossia il nome della classe a cui appartiene. In seguito al processo di training, l'algoritmo deve essere in grado di classificare gli input, cioè assegnarli alla corretta classe tra quelle note. Un esempio può essere cercare di distinguere le mail in arrivo tra spam e non (problema di *classificazione binaria* perché sono previste solo due classi come in figura 2.2). Per addestrare il modello gli vengono passate inizialmente sia delle email spam che non in modo che possa ricavare la funzione per distinguere le due classi. Terminata questa fase sarà in grado di catalogare da sé le email in modo accurato.

2.2.2 Regressione

Se la classificazione viene vista come un processo di previsione di un valore discreto (quindi l'appartenenza a una classe), la **regressione** è un processo di previsione di un valore continuo, cioè che può valere un qualsiasi numero in un certo intervallo,

come potrebbe essere la predizione del prezzo di una casa in base alla sua grandezza, al numero di stanze, alla posizione, ecc.

La generica equazione di regressione ha la seguente forma:

$$y = w_1x_1 + w_2x_2 + \dots + w_nx_n + b \quad (2.1)$$

dove $x[i]$ è la proprietà del dato e i termini w_i sono i pesi e b il bias.

L'equazione 2.1 viene semplificata se si tratta di regressione *lineare*, ovvero avente una sola proprietà:

$$y = mx + b \quad (2.2)$$

ed è ovviamente una retta.

Sia la classificazione che la regressione possono essere impiegate in lavori complessi, come potrebbero essere la classificazione delle immagini, il rilevamento di oggetti, lo sviluppo di chat bot e anche la creazione di video molto realistici di persone che parlano.

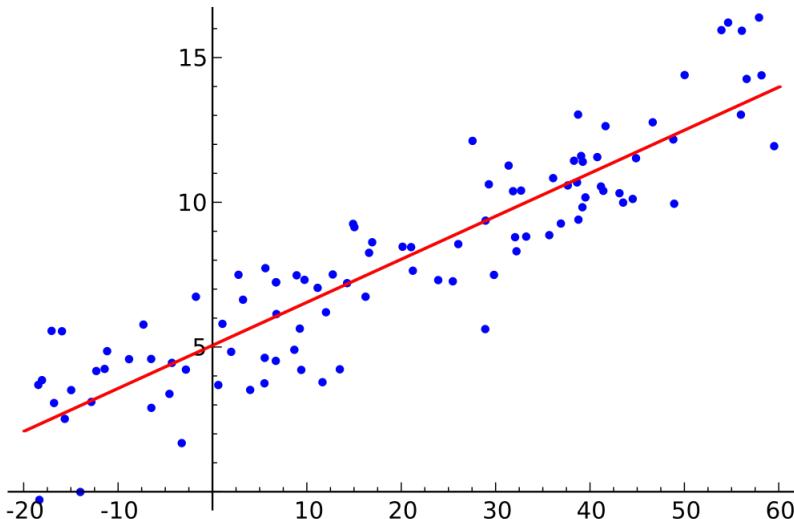


Figura 2.3. Regressione

2.3 Machine Learning Non Supervisionato

Nell'apprendimento **non supervisionato**, a differenza della tipologia precedente, vengono forniti solo i dati *non etichettati*, ovvero senza i rispettivi output: non ci sono classi o sezioni da ricercare a priori. Il compito degli algoritmi è quello di ricercare pattern nascosti tra i dati e creare da zero delle classi di elementi che rispettano determinate proprietà. Un esempio pratico tratto dalla realtà potrebbe

essere classificare un insieme di oggetti in base al colore: a priori non si sa come debbano essere divisi gli oggetti, semplicemente osservandoli si decide che il metodo più efficace per categorizzarli è in base al colore.

L'apprendimento non supervisionato può essere più complesso rispetto a quello supervisionato perché l'assenza di supervisione rende il problema meno chiaro e quindi la ricerca di un pattern diventa più vaga. Metaforicamente è quello che succede quando si cerca, per esempio, di imparare a suonare la batteria con un maestro piuttosto che da soli: con un maestro si impara più velocemente dato che fornirà una linea guida da seguire, da solo invece diventa più difficile anche solo capire da dove iniziare.

Tuttavia l'apprendimento non supervisionato entra in gioco quando si tratta di trovare migliori modi per risolvere un problema e di esplorare dei dati sconosciuti cercando di estrarre delle informazioni. Per ricercare dei pattern all'interno di dati non etichettati si utilizzano algoritmi che si basano su diversi approcci: **clustering** (quella più comune), **association rules** e **dimensionality reduction**.

2.3.1 Clustering

Il **Clustering** è una tecnica di machine learning non supervisionato che consiste nel dividere una popolazione di dati in un certo numero di gruppi o *cluster* con caratteristiche diverse tra loro, mentre gli elementi in uno stesso gruppo condividono delle proprietà. Dato che l'apprendimento è non supervisionato, non si stabilisce a priori quali gruppi avere, pertanto è l'algoritmo a scegliere un criterio, mentre il numero di cluster può essere specificato o no a seconda del modello. Il clustering è la tecnica utilizzata in questo progetto e viene approfondito nel capitolo 3.

2.3.2 Association Rules

Con questa tecnica si ricerca una regola che permette di stabilire delle relazioni tra le variabili all'interno di un dataset. Un approccio di questo tipo è molto sfruttato dalle aziende per capire la relazioni tra i vari prodotti e studiare le abitudini dei clienti per capire quali elementi vengono acquistati più spesso insieme. Per esempio su Amazon vi è la voce "spesso acquistato con", oppure Spotify crea le playlist di canzoni suggerite basandosi sugli ascolti delle persone.

2.3.3 Dimensionality Reduction

Sebbene durante il training il possesso di più dati generalmente porta a risultati più accurati, a volte può impattare le prestazioni dell'algoritmo e portare ad un *over-fitting*. Qui entra in gioco la tecnica del **dimensionality reduction** che porta,

appunto, a ridurre il numero delle caratteristiche di un dataset se sono troppo elevate e degli input per avere una quantità di dati più maneggevole, cercando di mantenere il più possibile l'integrità.

2.4 Machine Learning Semi-Supervisionato

L'apprendimento **semi-supervisionato** è un mix tra il supervisionato e il non supervisionato. L'addestramento di un modello parte da alcune label prestabilite come nell'apprendimento supervisionato, ma poi continua su dati non etichettati per cercare di raggrupparli secondo le etichette date. L'algoritmo più dettagliato è il seguente:

1. si assegnano manualmente delle etichette a un piccolo sottoinsieme del dataset di partenza, il quale viene utilizzato per addestrare un modello di base in modo supervisionato;
2. viene eseguito il processo di *pseudo-labeling*: si utilizza il modello addestrato per fare predizioni sul resto del dataset non ancora etichettato e genera le cosiddette *pseudo-labels*;
3. si stabilisce un livello di confidenza per ogni label tale che, se una determinata pseudo-label lo supera, allora viene inserita all'interno della label di partenza. Per esempio si ipotizzi di volere raggruppare foto di animali rappresentanti cani o gatti (le labels di partenza) e si imposti il livello di confidenza a 80%: se una foto dimostra di rappresentare un cane per almeno l'80%, allora viene inserita nell'insieme di input rappresentante foto di cani e il modello viene ri-addestrato con questo insieme aggiornato.

Nel caso in cui si voglia addestrare un modello di machine learning supervisionato, sicuramente l'accuratezza sarà migliore se ogni dato è etichettato. Tuttavia in molti casi si ha a disposizione un dataset talmente grande da non poter assegnare manualmente un'etichetta a ogni singolo elemento e qui entra in gioco l'apprendimento non supervisionato.

La sua applicazione migliora le prestazioni anche nel caso in cui si vuole addestrare un modello non supervisionato a partire da un dataset non etichettato in cui si sa già che le label sono limitate: infatti avere a disposizione anche un minimo numero di elementi etichettati costituisce un grande aiuto in termini di prestazioni e accuratezza.

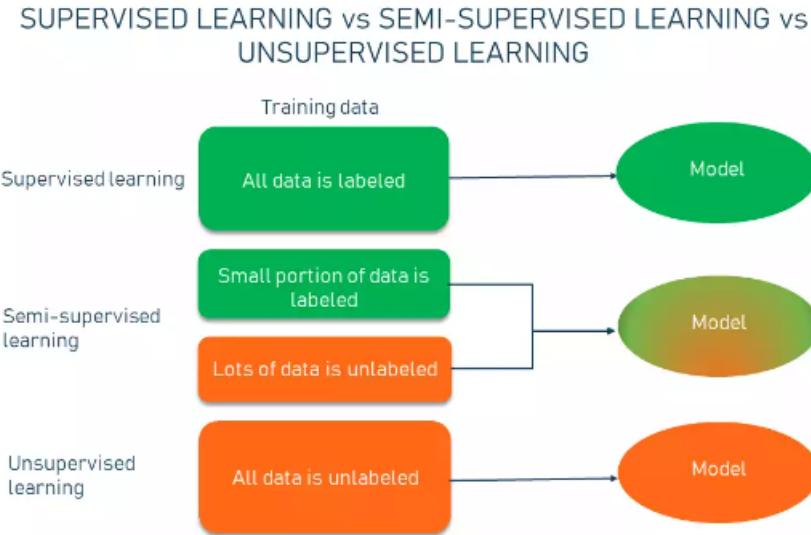


Figura 2.4. Confronto tra modelli di ML.

2.5 Apprendimento per rinforzo

È il metodo meno comune per via della sua complessità ma permette di ottenere comunque degli ottimi risultati. L'idea che c'è dietro si basa sul principio psicologico che feedback positivi o negativi possono influenzare il comportamento di un individuo. Come nell'addestramento di un cane, in caso di un comportamento positivo gli viene dato un premio per incentivarlo a comportarsi così, altrimenti viene punito per scoraggiarlo.

Lo stesso si applica nel caso degli esseri umani: se si riceve un premio a fronte di un comportamento positivo o del raggiungimento di un risultato, il cervello umano rilascia dopamina che rende l'individuo sereno e appagato e lo incentiva a fare meglio. A noi non serve l'approccio utilizzato nel machine learning supervisionato che prevede una supervisione costante, semplicemente ottenendo dei messaggi di rinforzo ogni tanto si riesce ad andare sulla buona strada.

Gli algoritmi di **apprendimento per rinforzo** vengono addestrati secondo un modello *trial and error*, cioè che impara attraverso gli errori. Questi sono più adatti, rispetto alle tipologie descritte nei paragrafi precedenti, ad operare su dataset **dinamici**, per esempio nei videogiochi, nella creazione di un sistema di guida autonoma o nell'ambito della robotica.

2.6 Deep Learning

Il **Deep Learning** è un sottoinsieme del Machine Learning dal quale si differenza per il tipo di dati con cui lavora e dal modo in cui lo fa. Il suo scopo è quello di

emulare l'attività del cervello umano di imparare dai propri errori e correggersi da sé, analizzando una grande quantità di dati. La differenza tra machine learning e deep learning sta nel fatto che il secondo rimuove tutte quelle operazioni di pre-lavorazione dei dati necessari per un modello ML (capitolo 2.1). Nel caso in cui, per esempio, a partire da un insieme di foto di animali li si voglia categorizzare per specie, per il modello di ML è necessario l'intervento dell'uomo che deve applicare delle operazioni di *data preparation* e *model fitting* prima che il modello possa effettuare la distinzione; il modello di DL sarà in grado da sé di identificare quali *features* sono più rilevanti per effettuare la distinzione e affinare costantemente la precisione della classificazione.

2.6.1 Reti Neurali

Un modello di DL è identificato da quella che sia chiama **Rete Neurale** o *Artificial Neural Network (ANN)*, un sistema costituito da vari livelli o *layer* ognuno contenente diversi nodi o **neuroni** (da il nome di rete neurale come metafora del sistema nervoso umano). Gli unici livelli visibili sono il livello di input (dove entrano i dati in ingresso) e quello di output (dove escono i dati prodotti), mentre quelli intermedi sono detti *hidden* (nascosti). I nodi sono connessi a quelli del livello successivo e ognuno ha un proprio **peso** e un **bias** o **valore di soglia**: i pesi sono i parametri con cui vengono moltiplicati i dati e se l'output di un nodo supera il valore di soglia, il neurone viene attivato e il dato viaggia verso il nodo successivo e diventa il suo input.

Ogni nodo ha il suo modello di regressione lineare nella seguente forma:

$$\sum w_i x_i + bias = w_1 x_1 + w_2 x_2 + \dots + w_n x_n + bias \quad (2.3)$$

e un esempio di output può essere:

$$\sum w_i x_i + bias \geq 0 \Rightarrow out = f(x) = 1 \quad (2.4)$$

che indica che il neurone è attivo, oppure

$$\sum w_i x_i + bias < 0 \Rightarrow out = f(x) = 0 \quad (2.5)$$

non attivo.

Quando un layer è determinato, i pesi vengono assegnati e il loro valore indica l'importanza di una data variabile. Gli input vengono moltiplicati per i pesi e sommati tra loro, poi passati al layer successivo se il neurone è attivo. Il processo di propagare i dati dal layer di input a quello di output è detto **forward propagation**.

Alcuni modelli vengono addestrati con il processo inverso detto **back propagation** che consiste nell'aggiornare i pesi e i bias calcolando l'errore con l'output

desiderato.

Le reti neurali si dividono in:

- **Convolutional Neural Network (CNN):** utilizzate spesso per effettuare operazioni sulle immagini, come la classificazione, l'estrazione di caratteristiche per stabilire se sono vere o no oppure per riconoscere degli oggetti presenti;
- **Recurrent Neural Network (RNN):** utilizzate spesso per fare predizioni riguardo eventi in ambito economico o nel riconoscimento del linguaggio umano.

Capitolo 3

Clustering

3.1 Cos'è il Clustering?

Come già accennato nel capitolo 1, lo studio di un evento prevede a priori la categorizzazione dei dati. Per esempio, ipotizziamo di volersi informare sulla disgraziata situazione attuale della guerra tra Russia e Ucraina iniziata nel febbraio 2022 tramite i post che gli utenti di tutto il mondo scrivono su Twitter, per capire quali sono i temi maggiormente trattati, quali eventi sono più rilevanti, cosa pensano le persone riguardo al conflitto o quali idee propongono per migliorare la situazione. Dopo aver importato i dati necessari (in questo i tweet relativi alla guerra), all'inizio diventa difficile estrapolare delle informazioni partendo da questo calderone, pertanto un'operazione che si fa è quella di ricercare una struttura che permetta di organizzare e analizzare meglio i tweet. La tecnica impiegata è quella del clustering.

Il **Clustering** è la tecnica più diffusa di machine learning non supervisionato. Il compito del clustering è quello di raggruppare dati simili in **cluster** (gruppi) a partire da un insieme più complesso e assegnare ad ognuno un'etichetta, al contrario del caso di apprendimento supervisionato in cui le etichette o **labels** sono note a priori (in tal caso di parla di classificazione).

Il clustering è uno strumento molto potente utilizzato in differenti settori:

- segmentazione del mercato;
- analisi dei social network;
- raggruppamento dei risultati di ricerca;
- diagnostica per immagini;
- segmentazione delle immagini;
- rilevamento di anomalie.

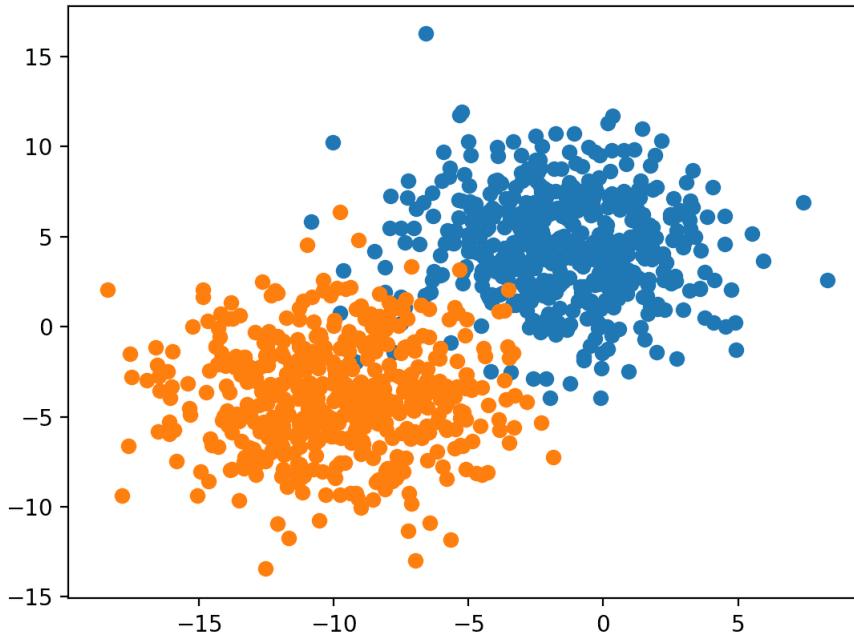


Figura 3.1. Clustering

Affinché si possano catalogare gli elementi simili tra loro è necessario stabilire una **misura di somiglianza** e della **distanza**, ovvero un criterio per determinare se due elementi sono abbastanza simili o distanti per appartenere o no allo stesso cluster. È facile comparare quando hanno poche caratteristiche, mentre quando questo numero sale fino ad averne centinaia la comparazione diventa difficile.

3.2 Somiglianza e distanza

La **somiglianza** indica il grado di quanto due elementi si assomigliano, cioè se hanno più o meno le stesse proprietà. Esistono diverse funzioni utilizzate per calcolare la distanza e sono riassunte in tabella 3.1.

La **distanza** è una misura che indica quanto due elementi dell'insieme si discostano l'uno dall'altro. Esistono diversi metodi per calcolarla dipendentemente dalla tipologia di dati e dallo scopo finale, quelli più comuni sono riassunti in tabella 3.2. Molti algoritmi di clustering popolari si basano proprio sul calcolo della distanza, tra cui *k-means*, *hierarchical clustering* e *density-based clustering* (vedi paragrafo 3.4).

Nome	Formula o metodo	Spiegazione
Jaccard	$J(A, B) = \frac{ A \cap B }{ A \cup B }$	Misura la somiglianza tra due insiemi A e B . Il termine $ X $ indica il numero di elementi nell'insieme. Si può inoltre calcolare l'omonima distanza nel seguente modo: $JDistance = 1 - JSimilarity$.
Hamming	Determina il minimo numero di sostituzioni necessarie per trasformare un dato in un altro.	Più il numero di sostituzioni è piccolo, più la somiglianza è maggiore. È spesso utilizzato sulle stringhe e su dati che possono essere ricondotti ad esse (per esempio la sequenza di DNA o dati binari).
Coseno	$\cos \alpha = \frac{\mathbf{x}_i^T \cdot \mathbf{x}_j}{\ \mathbf{x}_i\ \ \mathbf{x}_j\ }$	Gli elementi vengono caratterizzati come vettori e in base al valore del $\cos \in [-1, 1]$ si determina quanto sono vicini dove 1 indica la somiglianza assoluta.

Tabella 3.1. Funzioni per il calcolo della somiglianza.

Nome	Formula	Spiegazione
Minkowski	$\left(\sum_{l=1}^d x_{il} - x_{jl} ^n \right)^{\frac{1}{n}}$	A seconda del valore di n la distanza prende il nome di: 1. Manhattan se $n = 1$ 2. Euclidea se $n = 2$ 3. Chebyshev se $n \rightarrow \infty$
Euclidea standardizzata	$\left(\sum_{l=1}^d \left \frac{x_{il} - x_{jl}}{s_l} \right ^2 \right)^{\frac{1}{2}}$	Rappresenta una distanza euclidea pesata sulla deviazione standard s_l
Correlazione di Pearson	$1 - \frac{Cov(x_i, x_j)}{\sqrt{D(x_i)} \sqrt{D(x_j)}}$	Misura la distanza in base alla correlazione lineare, dove Cov indica la covarianza e D la varianza.
Mahalanobis	$\sqrt{(x_i - x_j)^T S^{-1} (x_i - x_j)}$	Misura la distanza in base alla covarianza, dove S è la matrice di covarianza del cluster.

Tabella 3.2. Funzioni per il calcolo della distanza.

3.3 Indicatori di validità

L'**indicatore di validità** o ***evaluation indicator*** è uno strumento che consente di controllare la validità dell'algoritmo e vedere se la clusterizzazione ha generato risultati affidabili.

Gli indicatori si possono dividere in due categorie:

- ***internal evaluation indicators*** o **indicatori interni**: l'indicatore attinge solo alle informazioni presenti nei dati per testare l'algoritmo. In generale può affermare quale algoritmo è il migliore da applicare sui dati analizzati e il numero ottimale di cluster, avendo però una precisione meno elevata rispetto agli ***external evaluation indicators***;
- ***external evaluation indicators*** o **indicatori esterni**: questi indicatori utilizzano informazioni esterne ai dati in sé (come il numero delle etichette dei cluster) e sono principalmente utilizzati per scegliere un algoritmo di clustering ottimale da applicare sul dataset posseduto.

Sebbene gli external indicators siano più affidabili, in molte situazioni in cui non si è in possesso di informazioni esterne, l'unica scelta per testare la validità ricade sugli *internal*.

3.3.1 Internal evaluation indicators

Dato che lo scopo del clustering è quello di raccogliere in un cluster gli elementi tanto più simili tra loro e di separare quelli più diversi, la validazione interna si basa sui seguenti criteri:

1. **compattezza**: misura quanto sono collegati gli oggetti nello stesso cluster. Per fare ciò viene sfruttata la varianza (più essa è bassa, migliore è la compattezza) oppure su tecniche basate sul calcolo della distanza massima o media tra coppie di elementi oppure dal centro del cluster;
2. **separazione**: misura quanto è distinto o ben separato un cluster dagli altri, per esempio calcolando la distanza tra i centri dei cluster oppure la distanza minima tra coppie di elementi tra cluster diversi.

Generalmente la maggior parte degli indici combina le due proprietà sopra descritte nel seguente modo:

$$Index = \frac{\alpha \times \text{separazione}}{\beta \times \text{compattezza}} \quad (3.1)$$

dove α e β sono i pesi.

Esistono circa una decina di metodi per la valutazione interna e sono riassunti in tabella 3.3, mentre nella sezione seguente verranno analizzati in dettagli i due più utilizzati.

Silhouette coefficient

Per misurare quanto la clusterizzazione sia precisa si calcola la distanza media tra i cluster, ossia per ogni punto in un cluster vede quanto esso sia simile rispetto al proprio cluster comparato con i punti degli altri.

Dato un punto i , l'**indice di silhouette** S_i si calcola applicando il seguente algoritmo:

1. per ogni punto i si calcola la distanza media a_i tra i e i punti del cluster a cui appartiene;
2. per tutti gli altri cluster C tali che $i \notin C$ si calcola la distanza media $d(i, C)$ tra i e i punti di C . La minima di esse viene definita come $b_i = \min_C d(i, C)$ e può essere vista come la distanza tra i e il cluster adiacente più vicino al suo;
3. infine si calcola S_i :

$$S_i = \frac{b_i - a_i}{\max(a_i, b_i)} \quad (3.2)$$

che può assumere valori $(-1, 1)$.

Per esempio se $a_i = 0.5$, $b_i = 1$ allora:

$$S_i = \frac{1 - 0.5}{\max(0.5, 1)} = \frac{0.5}{1} = 0.5$$

Su S_i si possono fare le seguenti considerazioni:

- punti con indici di silhouette molto vicini a 1 sono ben clusterizzati. Questa condizione è vera se $a_i \ll b_i$;
- un indice piccolo (S_i molto vicino a 0) indica che i può appartenere a due cluster;
- punti con un indice negativo (nel caso $a_i > b_i$) sono probabilmente posizionati nel cluster sbagliato.

Dunn index

Indice che viene calcolato tramite i seguenti passaggi:

1. per ogni cluster si calcola la distanza tra ogni suo oggetto e quelli presenti negli altri cluster;

2. la minima distanza così calcolata si chiama *min_separation* ed indica la distanza tra i due cluster;
3. per ogni cluster si calcola la distanza tra gli oggetti in esso e la massima si chiama *max_diameter* che indica la compattezza all'interno del cluster;
4. infine si calcola l'indice:

$$D = \frac{\text{min_separation}}{\text{max_diameter}} \quad (3.3)$$

Se il dataset contiene cluster compatti e ben separati, allora *min_separation* è grande e *max_diameter* è più piccolo, quindi l'indice deve essere massimizzato.

In tabella 3.3 sono riportati altri metodi oltre ai due precedentemente trattati per stimare la validità interna. Nella colonna dei valori ottimali, *elbow* indica che dal grafico della curva dell'indice bisogna prendere il valore nel punto in cui la funzione forma un "gomito", mentre *min* e *max* indica che il valore dell'indice deve essere rispettivamente minimizzato o massimizzato.

Notazione tabella 3.3: D : centro di c ; n : numero di oggetti in D ; c : centro di D ; P : numero di attributi di D ; NC : numero di clusters; C_i : i -esimo cluster; n_i : numero di oggetti in C_i ; c_i : centro di C_i ; $\sigma(C_i)$: vettore della varianza di C_i ; $d(x,y)$: distanza tra x e y ; $\|X_i\| = (X_i^T \cdot X_i)^{\frac{1}{2}}$.

Misura	Formula	Valore ottimale
Root-mean-square std dev	$\left\{ \frac{\sum_i \sum_{x \in C_i} x - c_i ^2}{[P \sum_i (n_i - 1)]} \right\}^{\frac{1}{2}}$	Elbow
R-squared	$\frac{\sum_{x \in D} x - c ^2 - \sum_i \sum_{x \in C_i} x - c_i ^2}{\sum_{x \in D} x - c ^2}$	Elbow
Modified Hubert Γ statistic	$\frac{2}{n(n-1)} \sum_{x \in D} \sum_{y \in D} d(x, y) [d_{(x \in C_i, y \in C_j)}(c_i, c_j)]$	Elbow
Calinski-Harabasz index	$\frac{\sum_i n_i d^2(c_i, c)}{\sum_i \sum_{x \in C_i} d^2(x, c_i)(n - NC)}$	Max
I index	$\left\{ \frac{1}{NC} \cdot \frac{\sum_{x \in D} d(x, c)}{\sum_i \sum_{x \in C_i} d(x, c_i)} \cdot \max_{i,j} d(c_i, c_j) \right\}^p$	Max
Davies-Bouldin indicator	$\frac{1}{n} \sum_{i=1}^n \max_{i \neq j} \left(\frac{\sigma_i + \sigma_j}{d(c_i, c_j)} \right)$	Min
Xie-Beni index	$\sum_i \sum_{x \in C_i} \frac{d^2(x, c_i)}{n \cdot \min_{j \neq i} d^2(c_i, c_j)}$	Min

Tabella 3.3. Metodi di validazione interna.

3.3.2 External evaluation indicators

Il processo di validazione esterna non prende in considerazione i dati da clusterizzare, bensì dati esterni ad essi come possono essere il numero di etichette e i risultati di alcuni *benchmark*. Questi ultimi possono essere pensati come una serie di oggetti pre-classificati (spesso derivanti direttamente dal giudizio umano) che stabiliscono il cosiddetto **gold standard**, ovvero il risultato migliore che si può ottenere. Per esempio, si ipotizzi che il risultato ottimale della ricerca della parola *Jaguar* conduca a esattamente tre cluster distinti: l'animale, il marchio automobilistico e l'omonima terza versione del sistema operativo MacOs del 2002 (MacOs Jaguar). Per la validazione si considera quanto l'algoritmo di clustering si avvicini a identificare queste tre classi note a priori.

Di sotto sono elencati i principali criteri su cui si basa la validazione esterna.

Purezza

Ad ogni cluster gli viene assegnata la classe di elementi più frequente contenuta in esso, poi si conta il numero corretto di elementi assegnati e si divide per il N (numero di elementi totali). Idealmente se il rapporto vale 1 l'accuratezza è massima:

$$\text{purezza}(\Omega, C) = \frac{1}{N} \sum_k \max_j |\omega_k \cap c_j| \quad (3.4)$$

dove $\Omega = \{w_1, w_2, \dots, w_K\}$ è l'insieme dei cluster e $C = \{c_1, c_2, \dots, c_J\}$ è l'insieme delle classi desiderate.

Rand indicator

Valuta quanto i cluster generati dall'algoritmo si avvicinano al gold standard:

$$RI = \frac{TP + TN}{TP + TN + FP + FN} \quad (3.5)$$

dove TP è il numero dei *true positive* (veri positivi), TN dei *false negative* (falsi negativi), FP dei *false positive* (falsi positivi), FN dei *false negative* (falsi negativi), tali che $TP + TN + FP + FN = \binom{N}{2}$ dove N è la dimensione del dataset. Questi numeri derivano dalla corretta assegnazione o meno di ogni elemento al cluster desiderato, ovvero:

- TP è il numero di elementi posizionati nello stesso cluster sia dall'algoritmo che nel gold standard;
- TN è il numero di elementi posizionati in cluster diversi sia dall'algoritmo che nel gold standard;

- FP è il numero di elementi raggruppati insieme dall'algoritmo ma non dal gold standard;
- FN è il numero di elementi non raggruppati insieme dall'algoritmo ma che dovrebbero esserlo secondo il gold standard.

Nel caso ideale in cui non ci sono falsi positivi e falsi negativi, l'indice vale 1 che indica l'accuratezza massima, e cioè che l'algoritmo coincide con il gold standard. Il problema principale del *rand index* è quello di assegnare lo stesso peso ai falsi positivi e falsi negativi che può portare a dei risultati poco realistici per alcuni algoritmi. Il prossimo criterio (*F-Measure*) corregge questa problematica.

F-Measure

Vengono introdotte due grandezze esterne, la ***precision P*** (precisione) e il ***recall R*** (recupero), definite come segue:

$$P = \frac{TP}{TP + FP} \quad (3.6)$$

$$R = \frac{TP}{TP + FN} \quad (3.7)$$

La precisione è il rapporto tra gli elementi correttamente raggruppati e il totale tra quelli del cluster, mentre il richiamo è il rapporto tra gli elementi correttamente raggruppati e il totale di quelli che sarebbero stati dovuti inseriti nel cluster (vedi figura 3.2).

La F-Measure si calcola nel seguente modo:

$$F_\beta = \frac{(\beta^2 + 1) \cdot P \cdot R}{\beta^2 \cdot P + R} \quad (3.8)$$

dove β è un parametro reale scelto in modo che l'importanza del recupero valga β volte quella della precisione.

Jaccard index

È un valore che varia tra 0 e 1 che indica quanto due dataset sono simili tra loro (identici se vale 1), pertanto lo si può applicare considerando un cluster A del gold standard e il corrispettivo B generato dall'algoritmo. L'indice si calcola con la seguente formula:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} = \frac{TP}{TP + FP + FN} \quad (3.9)$$

Da notare che TN non viene considerato.

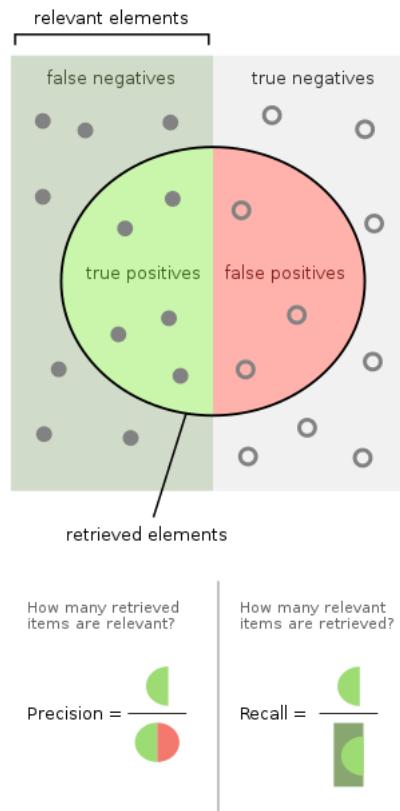


Figura 3.2. Precisione e richiamo

Dice index

Simile all'indice di Jaccard ma considera il doppio di TP :

$$J(A, B) = \frac{2|A \cap B|}{|A| + |B|} = \frac{2TP}{2TP + FP + FN} \quad (3.10)$$

Fowlkes-Mallow index

Misura la somiglianza tra i cluster generati dall'algoritmo e quelli del gold standard. Più il valore dell'indice è alto, più i cluster si somigliano e quindi l'accuratezza è maggiore:

$$FM = \sqrt{\frac{TP}{TP + FP} \cdot \frac{TP}{TP + FN}} \quad (3.11)$$

3.4 Algoritmi di clustering

Esistono diversi algoritmi di clustering, ognuno più adatto per essere applicato su un dataset diverso. La formazione dei cluster dipende da diversi parametri come la distanza più corta o la densità dei punti, quindi prima di scegliere un algoritmo è bene capire se esso scala in maniera efficiente sui dati che si hanno a disposizione. Ipotizziamo di avere un algoritmo che calcola la somiglianza tra ogni coppia di dati del dataset, quindi il costo di esecuzione è proporzionale al quadrato del numero n di elementi, ovvero $O(n^2)$. È facile capire che in un dataset di milioni di elementi questo algoritmo non è idoneo perché il costo è estremamente elevato.

Inoltre è più semplice creare misure di somiglianza se il numero delle caratteristiche o *features* degli elementi è basso, mentre se esso aumenta fino ad averne anche centinaia, bisogna adattarsi di conseguenza.

Prima di descrivere le varie tipologie di clustering, questi possono essere divisi in due categorie:

- **hard clustering:** ogni elemento del dataset appartiene a un solo cluster;
- **soft clustering:** invece di inserire un elemento in un cluster, gli viene assegnata la probabilità di appartenenza a ogni cluster generato, permettendo agli elementi di fare parte contemporaneamente di più gruppi.

3.5 Centroid-based clustering

Il **Centroid-based clustering** (clustering basato sui centroidi), o anche **Partition-based**, è la più semplice tipologia di algoritmi di clustering. I punti appartenenti a un cluster sono identificati in base alla vicinanza a un determinato valore centrale detto **centroide**. Il dataset viene diviso in un numero dato di cluster ognuno avente come centro un centroide, in seguito per ogni punto del dataset si calcola la distanza con i centroidi e verrà assegnato al cluster avente distanza minima. Per il calcolo della distanza si usa uno dei metodi descritti nel paragrafo 3.2.

Lo svantaggio principale di questo approccio è proprio nella scelta iniziale del numero di cluster, che quindi implica la conoscenza dell'insieme di dati non sempre possibile e comporta una bassa sensibilità, dato che una scelta sbagliata può compromettere i risultati. Tuttavia, essendo molto efficiente, facile e veloce, viene comunque utilizzato largamente in diversi ambiti, quali la segmentazione del mercato, dei clienti e delle immagini, nella ricerca di topic all'interno di un documento testuale, ecc.

L'algoritmo più popolare basato sui centroidi è sicuramente il **K-Means**.

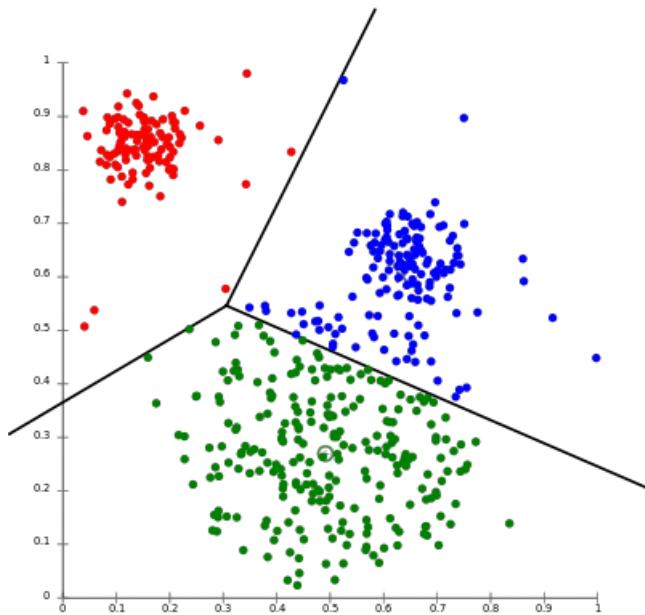


Figura 3.3. Esempio di K-Means con 3 cluster

3.5.1 K-Means

Il **K-Means** è un algoritmo di clustering basato sui centroidi che come criterio per la formazione dei cluster utilizza la distanza Euclidea (distanza di Minkowski per $n = 2$ in tabella 3.2).

Assumendo che n sia la dimensione del dataset $X = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$, k la dimensione dell'insieme dei cluster $S = \{S_1, \dots, S_k\}$ ognuno descritto dalla media $\boldsymbol{\mu}_j$ degli elementi contenuti in esso, l'algoritmo tende a minimizzare la seguente funzione obiettivo **within-cluster sum of squares (WCSS)**:

$$\text{WCSS} = \sum_{i=1}^k \sum_{x \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2 \quad (3.12)$$

I termini nella seconda sommatoria rappresentano l'**inerzia**, ossia la distanza tra il centroide $\boldsymbol{\mu}_i$ di un cluster S_i e i punti \mathbf{x} che ne fanno parte, ed è una misura della coesione interna ai cluster. L'algoritmo tenta di minimizzare questa quantità, affinché i punti siano vicini il più possibile con quelli dello stesso cluster e distanti con quelli di altri.

L'algoritmo k-means prevede i seguenti passaggi:

0. l'utente sceglie il numero k di cluster da ricercare;

1. si inizializzano casualmente i centroidi;
2. si assegna ciascun dato al centroide più vicino in base alla distanza;
3. ricalcola le coordinate dei centroidi come media μ dei punti assegnati a quel cluster;
4. si ripetono i passi 2 e 3 finché i centroidi restano gli stessi o cambiano restando al di sotto di un certo valore di soglia, oppure si è raggiunto il numero massimo di iterazioni.

Se n è il numero di vettori d -dimensionali del dataset, k è il numero dei cluster e i è il numero di iterazioni affinché l'algoritmo converga, il costo temporale dell'algoritmo è $O(nkdi)$.

Il migliore valore di k si può stimare utilizzando il metodo della **Silhouette** e la **Elbow technique**.

Il **metodo della Silhouette** per ogni elemento in un cluster utilizza la distanza media con altri elementi nello stesso cluster e la minima distanza con gli elementi degli altri insiemi (spiegato in dettaglio nel paragrafo 3.3.1).

La **Elbow technique** fornisce uno strumento grafico per la scelta di un valore migliore di k . Per applicarlo si esegue l'algoritmo k-means per un range di valori di k e per ognuno si calcola l'inerzia. In seguito si grafica in funzione di k e come numero di cluster finale si sceglie il valore di k in corrispondenza del cosiddetto "gomito" della funzione (da qui *elbow technique*).

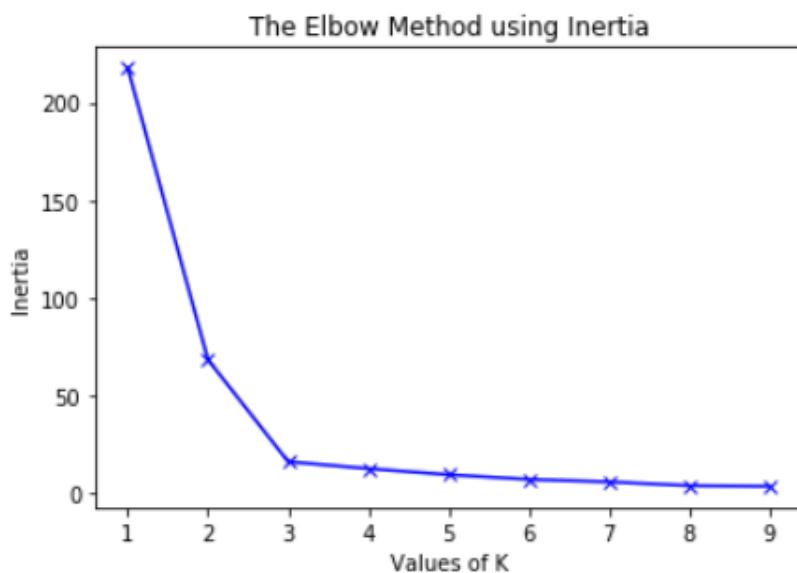


Figura 3.4. Elbow technique

L'idea su cui si fonda questo principio sta nel fatto che aumentando il numero di cluster migliora il *fit*, cioè il clustering è più preciso in quanto l'inerzia diminuisce, fino a quando non si genera un *over-fitting* in cui elementi in cluster già formati vengono suddivisi in insiemi più piccoli inutilmente. Pertanto sopra un certo valore di k non ha più senso continuare a dividere gli elementi e questo viene indicato dal gomito della funzione. Nell'esempio in figura 3.4 il valore ottimale di k sarebbe 3.

Questo metodo fornisce un risultato euristico, cioè non rigoroso e a volte piuttosto ambiguo e impreciso rispetto al metodo della silhouette, dato che non sempre la funzione da considerare non ha un *elbow* evidente, tuttavia è molto semplice e intuitivo e può fornire una stima rapida del numero di cluster da scegliere.

3.6 Hierarchical Clustering

Nel **Hierarchical Clustering** (clustering gerarchico), anche noto come **Connectivity-based clustering**, i cluster sono rappresentati come una struttura gerarchica a forma di albero e ogni elemento è connesso al suo vicino dipendentemente dalla loro distanza di prossimità. I cluster si visualizzano graficamente come un **dendrogramma**, un particolare albero che consente di studiare la somiglianza tra gli elementi in seguito al processo di clustering. Esistono due tipi di algoritmi di clustering gerarchico: agglomerativo e divisivo.

3.6.1 HC Agglomerativo

È un approccio di tipo *bottom-up* in cui inizialmente ogni elemento viene considerato come un cluster a sé. A ogni iterazione dell'algoritmo i cluster simili vengono uniti tra loro finché infine non si trova un unico cluster. L'algoritmo si articola nei seguenti passaggi:

0. si sceglie il numero k di cluster da ottenere;
1. si calcola la matrice di prossimità contenente le distanze tra ogni coppia di punti del dataset, ossia la matrice in cui ogni cella contiene la distanza reciproca tra due elementi;
2. si ripetono i seguenti passaggi finché non rimane un solo cluster:
 - (a) trova due cluster più vicini in base alla distanza;
 - (b) unisce i due cluster in uno unico;
 - (c) aggiorna la tabella di prossimità sostituendo le righe e le colonne degli elementi del cluster con un'unica riga e un'unica colonna rappresentanti il cluster e si aggiornano i valori delle distanze (vedi paragrafo 3.6.3).

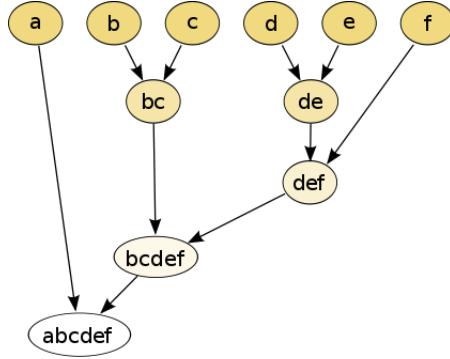


Figura 3.5. Esempio di HC agglomerativo: tagliando il dendrogramma dopo la seconda riga si ottengono i cluster $\{a\}$, $\{b, c\}$, $\{d, e\}$, $\{f\}$.

3. il cluster finale rappresenta la radice del dendrogramma.

Il dendrogramma tiene traccia dei passaggi tramite i quali si è arrivati a raccogliere tutti i dati. Per identificare un certo numero di cluster k , si taglia il dendrogramma a una certa altezza con una linea orizzontale in modo che essa non tocchi nessun **punto di merge** (nodi dell'albero): da ogni punto di intersezione tra la linea e il grafico si ha un cluster (figura 3.5).

3.6.2 HC Divisivo

È esattamente l'opposto dell'approccio agglomerativo: è di tipo *top-down* dove tutti i punti inizialmente vengono considerati parte di un unico cluster che poi, tramite il processo di clustering, viene diviso nei due cluster meno simili tra loro. Si applica questo processo iterativamente a ogni cluster ottenuto e vengono generati via via sotto-cluster sempre più specifici. Si termina finché o non ci può essere un'ulteriore divisione dei dati oppure vi è stata una terminazione logica (non ha più senso continuare a raggruppare gli elementi).

L'approccio divisivo riesce a generare cluster più accurati rispetto all'agglomerativo, tuttavia è più complesso da realizzare.

3.6.3 Calcolo della distanza

Nel clustering gerarchico la distanza tra gli elementi può essere calcolata seguendo vari approcci:

- **MIN o single-linkage:** a ogni passaggio i cluster simili sono quelli che hanno la coppia di elementi con distanza minima tra loro. In modo più formale, definita con $D(X, Y)$ la distanza tra i cluster X e Y , la *single-linkage function*

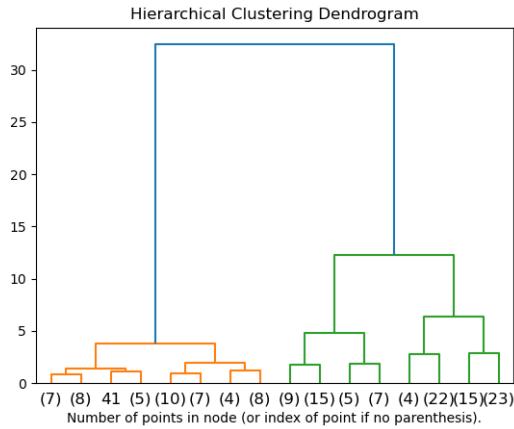


Figura 3.6. HC divisivo

si definisce dalla seguente espressione:

$$D(X, Y) = \min_{x \in X, y \in Y} d(x, y) \quad (3.13)$$

tale che $d(x, y)$ è la distanza tra i due punti $x \in X$ e $y \in Y$. Il vantaggio di questo approccio è che può separare insiemi di dati di forme non ellittiche se lo spazio tra i cluster è abbastanza grande, tuttavia non riesce a separare bene i cluster se tra di essi c'è del rumore;

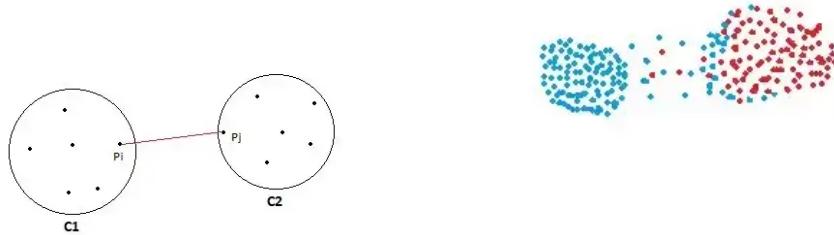


Figura 3.7. Single linkage

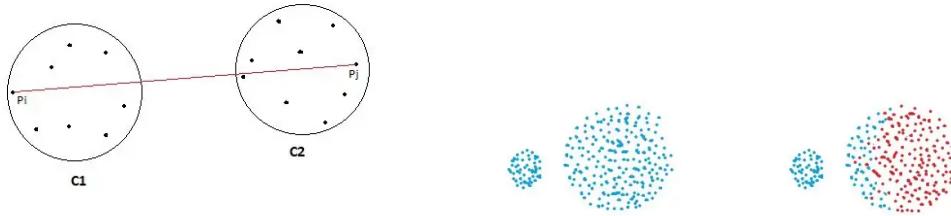
Figura 3.8. Problema single linkage

- **MAX o complete-linkage:** è esattamente l'opposto di MIN, i cluster sono simili se la distanza tra una coppia di loro elementi è massima, cioè presi due cluster X, Y e definita la distanza tra loro come $D(X, Y)$, la *complete-linkage function* è data da:

$$D(X, Y) = \max_{x \in X, y \in Y} d(x, y) \quad (3.14)$$

tale che $d(x, y)$ è la distanza tra i due punti $x \in X$ e $y \in Y$. Con questo

approccio si riescono a separare bene i cluster con molto rumore tra loro, tuttavia tende a rompere grandi cluster e propende verso cluster sferici.



- **group average:** per stabilire la somiglianza tra due cluster X e Y calcola la distanza $D(X, Y)$ media tra ogni coppia di punti dei due, ossia:

$$D(X, Y) = \frac{1}{|X||Y|} \sum_{x \in X} \sum_{y \in Y} d(x, y) \quad (3.15)$$

Con questo approccio si separano bene i cluster se c'è del rumore in mezzo ma è orientato verso cluster circolari;

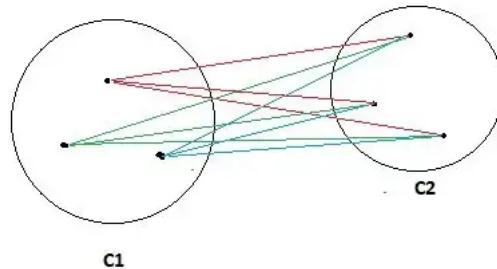


Figura 3.11. Group average

- **metodo di Ward:** simile al metodo *group average* ma nella funzione obiettivo si tiene conto delle distanze al quadrato:

$$D(X, Y) = \frac{1}{|X||Y|} \sum_{x \in X} \sum_{y \in Y} d(x, y)^2 \quad (3.16)$$

Valgono le stesse considerazioni del precedente metodo;

- **centroidi:** due cluster sono simili se è minima la distanza tra i due centroidi.

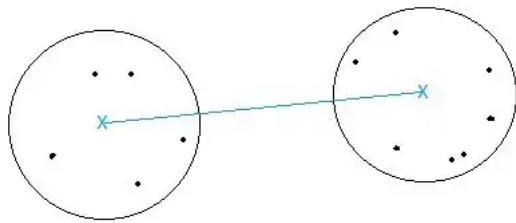


Figura 3.12. Centroidi

Dal punto di vista spaziale, il clustering gerarchico è molto dispendioso perché a ogni passaggio necessita di salvare la matrice di prossimità e quindi il costo è $O(n^2)$. Anche il costo temporale è elevato dato che, se n è il numero di elementi, bisogna compiere almeno n iterazioni in ognuna delle quali si ricalcola e si salva la matrice di prossimità, quindi il costo è $O(n^3)$. Date queste considerazioni il clustering gerarchico è poco adatto per essere applicato a grandi quantità di dati. Tuttavia nel paragrafo seguente viene riportato un algoritmo di clustering gerarchico adatto a clusterizzare anche dataset molto vasti.

3.6.4 BIRCH

Il **BIRCH** (*Balanced Iterative Reducing and Clustering using Hierarchies*) è un algoritmo utilizzato per clusterizzare in modo gerarchico grandi dataset. Il suo vantaggio più grande è quello di poter raggruppare in modo incrementale e dinamico (non necessita tutto il dataset in input) i dati con metrica multidimensionale in ingresso (cioè i dati i cui attributi possono essere rappresentati da esplicite coordinate in uno spazio euclideo) in modo da poter garantire il risultato migliore a partire dai vincoli di memoria e temporali dati. BIRCH richiede di scansionare il dataset attualmente a disposizione una sola volta e solitamente viene utilizzato insieme ad altri algoritmi di clustering come k-means.

L'algoritmo riceve in input un set di N elementi con e il numero desiderato di cluster k e si sviluppa nei seguenti passaggi:

1. **costruzione dell'albero CF:** il *Clustering Feature tree* (figura 3.13) è un albero bilanciato realizzato da un insieme *CF nodes*, ossia nodi costituiti da una serie elementi contenenti le informazioni sulle loro caratteristiche (*clustering features* appunto) utilizzate nelle operazioni di clustering e (per i nodi non foglia) un puntatore ai propri figli. I nodi foglia contengono i cluster e sono

connessi a quelli adiacenti tramite liste doppiamente puntate. Questa struttura permette di lavorare con parti del dataset complessivo senza avere la necessità di passarlo come input all'inizio.

Ogni *Clustering Features* (*CF*) è una tripla ordinata del tipo $CF = (N, \vec{LS}, \vec{SS})$ dove:

- N è il numero di elementi nella *CF*;
- $\vec{LS} = \sum_{i=1}^N \vec{X}_i$ è la somma lineare dei dati nel cluster;
- $\vec{SS} = \sum_{i=1}^N (\vec{X}_i)^2$ è la somma quadratica dei dati.

Il numero massimo di elementi che possono essere presenti in un sotto-cluster nel nodo foglia viene chiamato *threshold T*, il numero massimo di *CF* nei nodi foglia viene indicato con L e il numero massimo di figli che ogni nodo interno può avere si chiama *Branching-Factor B*. La costruzione del *CF* avviene in attraverso 4 passaggi:

- (a) per ogni nuovo record in arrivo BIRCH compara la sua somma lineare o quadratica con quella di ogni *CF* della radice e lo passa al nodo radice che ci si avvicina di più;
 - (b) il record "scende" l'albero della radice scelta, le sue *CF* vengono comparate con i nodi che incontra e viene passato al nodo non foglia che si avvicina di più;
 - (c) il record viene comparato con i nodi foglia del nodo scelto al punto (b) e viene passato alla foglia con caratteristiche più simili;
 - (d) vengono eseguiti uno dei due punti seguenti:
 - i. il raggio della foglia, includendo il nuovo record, non eccede T e quindi viene assegnato al sub-cluster della foglia. La foglia e le *CF* del nodo padre vengono aggiornati con le informazioni del nuovo nodo;
 - ii. il raggio della foglia sfiorerebbe T , allora viene formata una nuova foglia costituita solo dal nuovo record.
2. (passo opzionale) l'algoritmo scansiona i nodi foglia per costruire degli alberi *CF* più piccoli e rimuove gli *outliers* (dati che non rientrano in nessun cluster);
 3. dopo che l'albero è stato infine costruito, si applica un algoritmo di clustering a scelta sui *CF* dei sotto-cluster dei nodi foglia, ad esempio k-means.

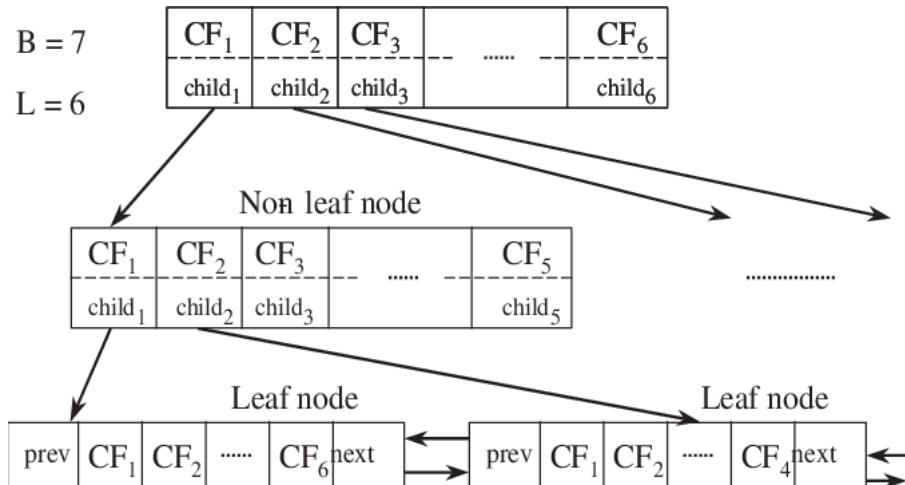


Figura 3.13. Esempio di albero cf

Il vantaggio principale di BIRCH è che ogni decisione per creare i sotto-cluster viene effettuata senza scansionare ogni elemento del dataset e ogni cluster già formato. Inoltre osservando lo spazio euclideo, se vede che i dati non sono uniformemente distribuiti e quindi non sono tutti di eguale importanza, scarta direttamente quelli meno importanti. Tuttavia il fatto di essere applicabile solo su uno spazio metrico è la sua principale limitazione, dato che non può essere utilizzato i presenza di variabili categoriali.

3.7 Density-Based Clustering

A differenza dei precedenti metodi di clustering che dipendono dalla distanza e dalla prossimità tra gli elementi, il **Density-based clustering** si focalizza sulla densità di distribuzione dei dati. I dati sono raggruppati in regioni di alta concentrazione legate da regioni a bassa concentrazione considerate come rumore. Grazie a un criterio di questo tipo i cluster possono avere forme e dimensioni arbitrarie in modo da non ignorare nessun dato importante, mentre il rumore e gli *outliers* (punti anomali che non hanno caratteristiche in comune con nessun gruppo) sono ben identificati e vengono scartati, a differenza di algoritmi come k-means che non hanno proprio questo concetto.

Il density-based clustering si basa sui concetti di **ϵ -neighborhood** e **densità**. Dato un punto p nel dataset e un $\epsilon > 0$, allora l' **ϵ -neighborhood** di p è l'insieme dei punti distanti al massimo ϵ da p (in forma compatta $nbhd(p, \epsilon)$). Ponendoci in uno spazio bidimensionale, si può pensare di tracciare un cerchio centrato in p di raggio ϵ , allora l' ϵ -neighborhood è costituito dai punti che si trovano all'interno del

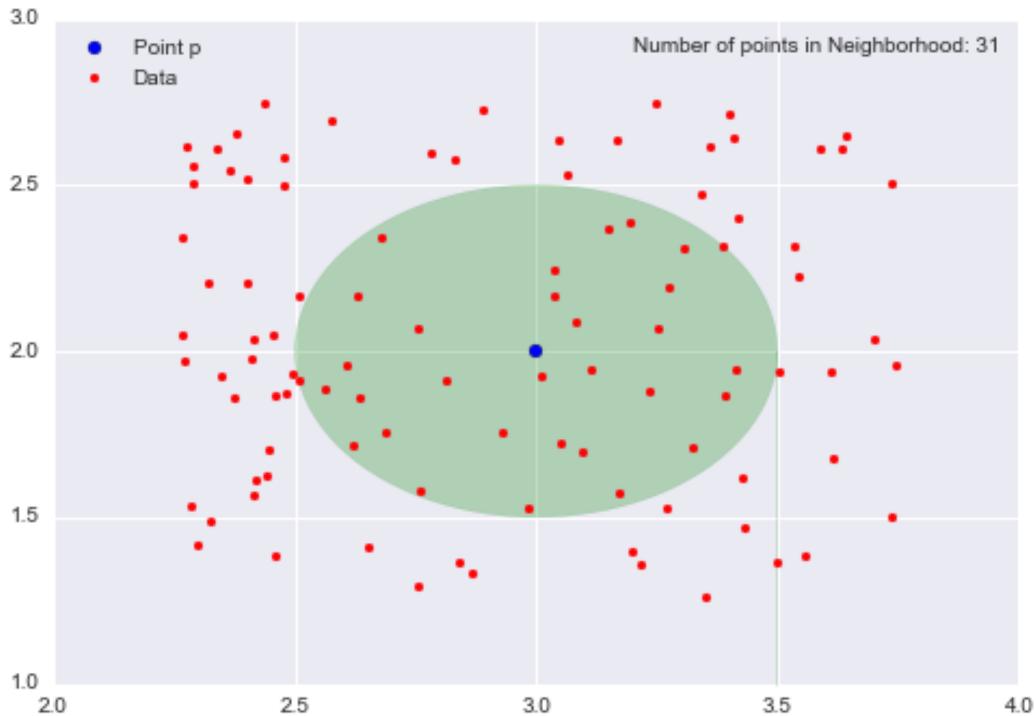


Figura 3.14. ϵ -neighborhood con $\epsilon = 0.5$

cerchio. Nel caso tridimensionale vale lo stesso concetto dove al posto del cerchio viene tracciata una sfera.

In figura 3.14 si ha un dataset di 100 elementi nell’intervallo $[1, 3] \times [2, 4]$, il punto p ha coordinate $(3, 2)$ e $\epsilon = 0.5$: ϵ -neighborhood comprende i 31 punti presenti nel cerchio verde. Se invece si pone $\epsilon = 0.15$ si nota in figura 3.15 che l’ ϵ -neighborhood si riduce di molto e comprende solo 3 punti. Questo mostra quanto la scelta di ϵ possa influenzare il clustering.

Per definire il concetto di **densità** si può fare un parallelo con la fisica, in cui è definita come il rapporto tra la massa e il volume. Per calcolare la densità di un ϵ -neighborhood nel caso bidimensionale, la massa può essere vista come il numero di punti presenti nell’insieme e il volume (anzi, in questo caso la superficie) vale $\pi\epsilon^2$. Per il caso in figura 3.14 la densità locale in $p = (3, 2)$ vale:

$$\text{densità} = \frac{\text{massa}}{\text{superficie}} = \frac{31}{\pi 0.5^2} = \frac{31}{\frac{\pi}{4}} = 39.5 \quad (3.17)$$

Questo dato da solo non ha alcun valore, tuttavia se si calcola la densità locale in ogni punto del dataset si può dire che due punti appartengono allo stesso cluster se sono vicini (cioè contenuti nello stesso neighborhood) e hanno densità locale simile. Se si riduce il valore di ϵ , si possono costruire neighborhoods più piccoli

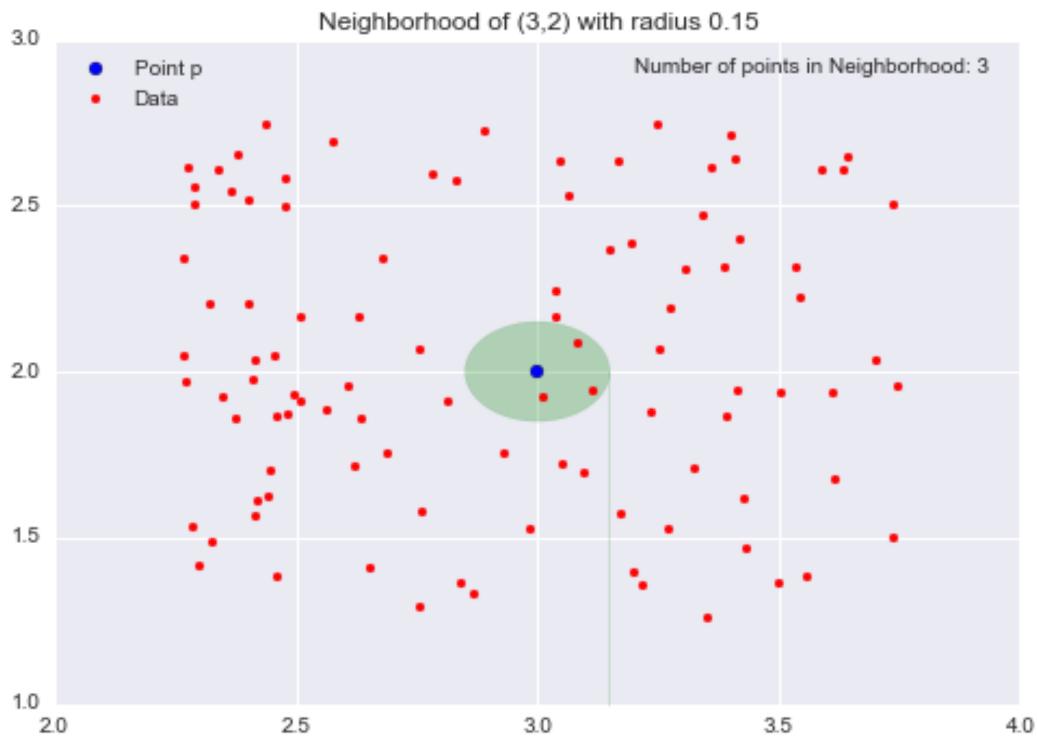


Figura 3.15. ϵ -neighborhood con $\epsilon = 0.15$

che contengono meno punti. L’idea quindi è di ricercare neighborhoods con densità elevate che contengono la maggior parte dei dati, anche se la loro grandezza è relativamente piccola.

Il più famoso algoritmo di density-based clustering è il DBSCAN che utilizza e modifica leggermente i concetti spiegati poc’anzi.

3.7.1 DBSCAN

A differenza degli altri algoritmi di clustering descritti nei capitoli precedenti, **DBSCAN** (*Density-Based Spatial Clustering of Applications with Noise*) non necessita come input il numero di cluster che andrà a generare. Infatti determinerà i cluster di forme arbitrarie direttamente dall’insieme dei dati a partire da alcuni parametri fondamentali:

- **ϵ :** il raggio per definire l’ ϵ -neighborhood;
- **minPts:** il numero minimo affinché un neighborhood si possa definire un cluster.

Grazie a questi parametri, DBSCAN suddivide i punti in tre categorie:

1. **core points:** un punto p si dice *core-point* se $nhd(p, \epsilon)$ contiene almeno $minPts$ punti. Riprendendo i concetti di densità, massa e volume del paragrafo

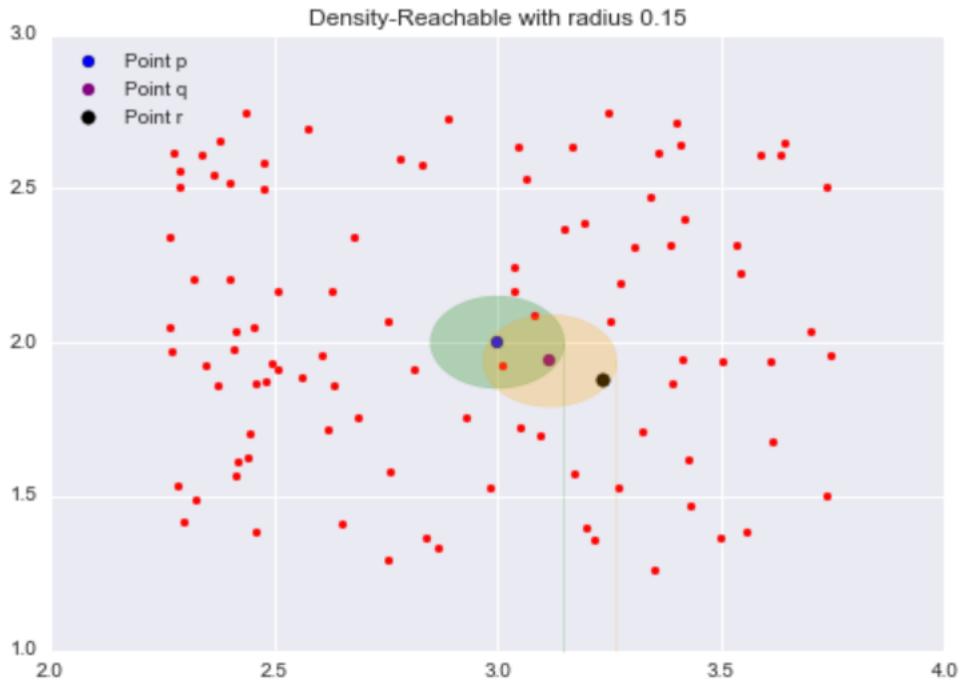


Figura 3.16. Raggiungibilità dei punti con DBSCAN

3.7, sapendo che la massa di $nbhd(p, \epsilon)$ è variabile e il volume è costante, modificando $minPts$ cambia la densità minima del neighborhood affinché p possa essere un core point: aumentandolo, la densità minima aumenta e quindi servono più punti vicini a un punto per ottenere un cluster, e viceversa;

2. **border points:** un punto q si dice *border-point* se $nbhd(q, \epsilon)$ contiene meno di $minPts$ punti, ma q è raggiungibile da un core point;
3. **outliers:** punti che non sono né core point né border point.

Esistono due diversi concetti di raggiungibilità: *directly reachable* e *density reachable*.

Un punto q si dice ***directly reachable*** da p se q si trova in $nbhd(p, \epsilon)$, mentre un punto r si dice ***density reachable*** da p se è *directly reachable* da un altro punto q che appartiene a $nbhd(p, \epsilon)$ (figura 3.16).

Secondo queste definizioni, gli outliers sono quei punti con non sono né directly reachable né density reachable da un core point, quindi non sono assegnati a nessun cluster e vengono scartati.

Di seguito vengono elencati i passaggi dell'algoritmo DBSCAN:

1. si inizia da un punto p a caso che non è stato ancora visitato e si individua $nbhd(p, \epsilon)$ a partire da un ϵ dato;

2. se $nbhd(p, \epsilon)$ contiene almeno $minPts$ punti, allora p viene marchiato come "visitato" e diventa il core point del cluster appena formato, altrimenti p viene marchiato come "rumore" per questa iterazione (può cambiare successivamente) e si continua in cerca di un core point. Quando viene trovato si passa al passaggio successivo dell'algoritmo;
3. se un punto q successivo appartiene al cluster formato, allora anche $nbhd(q, \epsilon)$ farà automaticamente parte di esso. Questo passo si ripete proseguendo per tutti i punti che sono directly e density reachable;
4. dopo che il ciclo termina, si passa in rassegna dei punti non ancora visitati e ripetono i primi 3 passi per individuare eventuali altri cluster e rumore;
5. l'algoritmo termina quando non ci sono più punti non visitati e restituisce cluster e outliers.

DBSCAN visita ogni un punto almeno una volta e, se per visitare il suo vicinato si utilizzata una struttura tale da impiegare $O(\log n)$, allora il costo complessivo è $O(n \log n)$. Questo però se ϵ viene scelto con criterio, cioè se in media a ogni iterazione vengono restituiti $O(\log n)$ punti. Se così non fosse il costo può arrivare anche a $O(n^2)$.

3.8 Distribution-Based Clustering

Nel **Distribution-Base clustering** il raggruppamento dei dati viene effettuato considerando la **distribuzione di probabilità** dei valori delle loro caratteristiche: i cluster includono gli elementi che hanno la probabilità statistica più alta di appartenervi. Inoltre ognuno ha un punto centrale, quindi più un dato è vicino, maggiore è la sua probabilità di entrare nel cluster.

Questo metodo di clustering ha diversi vantaggi:

- **flessibilità:** supporta numerose distribuzioni di probabilità diverse quali la Gaussiana, la Binomiale, ecc.;
- **soft-clustering:** alcuni algoritmi come il GMM (Gaussian-Mixture model, paragrafo 3.8.1 che si basa sulla distribuzione Gaussiana) prevede un assegnazione *soft* delle etichette, cioè a ogni punto non viene assegnata una sola etichetta, bensì tutte le probabilità di appartenenza relative a ogni cluster;
- **scalabilità:** è adatto per essere applicato su grandi dataset.

Tuttavia sono presenti alcuni svantaggi come:

- **complessità:** alcuni algoritmi a volte hanno costi computazionali troppo elevanti;
- algoritmi come il GMM nella fase di avvio tendono a decidere arbitrariamente i parametri iniziali che può portare a risultati fuorvianti;
- altri richiedono dati già classificati e in alcuni casi non è possibile fornirli, pertanto fanno assunzioni sulla natura dei dati (vedi paragrafo 3.8.1).

Di seguito viene analizzato il GMM, un algoritmo che adotta questo approccio.

3.8.1 Gaussian-Mixture model

Il **GMM** si basa sull'assunzione che i dati sono generati da un miscuglio di un numero finito di diverse distribuzioni gaussiane con parametri non noti, ognuna delle quali rappresenta un cluster. Inizialmente tramite un altro algoritmo di clustering (come k-means) o in maniera del tutto casuale, vengono creati dei cluster con i rispettivi centri. In seguito il modello cercherà di affinare la distribuzione dei dati negli cluster creati e di stimare i parametri delle distribuzioni (media, varianza, covarianza) con la **Expectation Maximization technique**. Dopo l'inizializzazione dei parametri, questa tecnica prevede due fasi che si alternano tra loro:

1. **Expectation step (E-step):** l'algoritmo calcola la probabilità *a posteriori* di ogni punto di appartenere a ogni cluster in base alla stima attuale dei parametri, poi le utilizza per stimare il logaritmo della verosimiglianza completa dei dati sempre in relazione ai parametri;
2. **Maximization-step (M-step):** l'algoritmo massimizza il valore atteso del logaritmo della verosimiglianza completa dei dati rispetto ai parametri per ottenere nuove stime di questi ultimi. In seguito si ritorna alla E-step utilizzando i nuovi parametri.

Le due fasi vengono ripetute finché il logaritmo dei dati non converge.

3.9 Fuzzy Clustering

Il **Fuzzy clustering** (dove *fuzzy* vuol dire sfocato, confuso) è un esempio di *soft-clustering* che permette agli elementi del dataset di appartenere a più cluster contemporaneamente. In particolare ad ogni elemento viene assegnato un grado di appartenenza a ogni cluster che varia da 0 a 1 e indica la probabilità che quel dato appartenga al cluster. Il valore viene calcolato a partire dalla distanza tra il dato e il centro del cluster: più i due sono vicini, maggiore è il grado di appartenenza.

L'algoritmo più popolare che sfrutta questo approccio è il **Fuzzy C-means**.

3.9.1 Fuzzy C-Means

Il **Fuzzy C-Means** (FCM) è un algoritmo di fuzzy clustering che assomiglia per alcuni aspetti a k-means. I passi dell'algoritmo sono i seguenti:

0. l'utente a priori specifica il numero dei cluster;
1. a ogni dato si assegnano dei coefficienti casuali di appartenenza a ogni cluster;
2. si ripetono i seguenti punti finché l'algoritmo non converge, cioè quando il cambio dei coefficienti tra due iterazioni non supera la soglia prestabilita ϵ :
 - (a) si calcolano i centroidi per ogni cluster (vedi formula 3.18);
 - (b) per ogni dato si calcolano i coefficienti di appartenenza a ogni cluster.

I centroidi di ogni cluster si calcolano effettuando il media pesata dei punti con il loro grado di appartenenza relativo a ogni cluster. In modo più formale, a ogni punto x viene assegnato il coefficiente $w_k(x)$ di appartenenza al k -esimo cluster, pertanto il centroide c_k del cluster è dato da:

$$c_k = \frac{\sum_x w_k(x)^m x}{\sum_x w_k(x)^m} \quad (3.18)$$

dove m è l'iper-parametro che controlla quanto il cluster debba essere "sfocato".

Come per k-means (par. 3.5.1) si può definire una funzione obiettivo da minimizzare. Assumendo che n sia la dimensione del dataset $X = \{x_1, \dots, x_n\}$, c il numero di cluster, $C = \{c_1, \dots, c_c\}$ la lista dei centroidi e $W = w_{i,j} \in [0, 1]$ con $i = 1, \dots, n$, $j = 1, \dots, k$ la **matrice di partizione** in cui l'elemento $w_{i,j}$ indica il grado di appartenenza dell'elemento x_i al cluster centrato in c_j , si può scrivere la seguente funzione obiettivo \mathcal{F} da minimizzare:

$$\mathcal{F} = \sum_{j=1}^c \sum_{i=1}^n w_{i,j}^m \|x_i - c_j\|^2 \quad (3.19)$$

dove

$$w_{i,j} = \frac{1}{\sum_{k=1}^c \left(\frac{\|x_i - c_k\|}{\|x_i - c_j\|} \right)^{\frac{2}{m-1}}} \quad (3.20)$$

La differenza con k-means sta nel fatto che in quest'ultimo si può pensare di utilizzare dei coefficienti di appartenenza che valgono solo esattamente 0 o 1, mentre nel fuzzy c-means variano nell'intervallo continuo $[0, 1]$. Nel limite per $m \rightarrow 1$, i coefficienti di appartenenza $w_{i,j}$ convergono a 0 o 1 e il fuzzy c-means va a coincidere con k-means. Nel caso si assenza di sperimentazione e scarsa conoscenza del dataset,

m solitamente viene impostato a 2. Anche se l'algoritmo minimizza la varianza all'interno dello stesso cluster, non si può sempre dire che il minimo della funzione sia globale, questo perché la sensibilità varia notevolmente in base alla scelta iniziale arbitraria dei pesi $w_{i,j}$.

Il Fuzzy clustering trova impiego negli ambiti della bioinformatica, dell'analisi delle immagini e nel marketing.

3.10 HDBSCAN

Esistono degli algoritmi che si basano su un approccio ibrido tra quelli descritti sinora che garantiscono prestazioni migliori e cluster più corretti. Nel progetto è stato utilizzato **HDBSCAN** (*Hierarchical DBSCAN*) che estende DBSCAN convertendolo in un algoritmo di clustering gerarchico in modo da renderlo più efficace su dati reali che sono molto rumorosi e con cluster all'apparenza poco definiti. Il procedimento di clustering può essere diviso in cinque passaggi.

1. **Trasformazione dello spazio:** lo spazio dei dati può essere visto metaforicamente come un *mare*, pertanto l'idea iniziale è quella di cercare delle *isole* in mezzo al rumore in cui i dati hanno densità maggiore. L'algoritmo, inoltre, si basa sulla tecnica *single-linkage* (par. 3.6.3) che è molto sensibile al rumore, quindi un singolo punto nel posto sbagliato può causare un ponte tra le isole unendole tra loro erroneamente. Per migliorare la robustezza quello che si fa prima del clustering vero e proprio è cercare di identificare il mare come un insieme di punti con densità minima, i quali sono distanti il più possibile sia tra loro che dai punti delle isole.

Viene definita la **mutual reachability distance** tra una coppia di punti a e b nel seguente modo:

$$d_{mreach-k} = \max\{core_k(a), core_k(b), d(a, b)\} \quad (3.21)$$

dove k è un parametro scelto che indica il numero dei k punti più vicini a un dato punto x , $core_k(x)$ è il minimo raggio del cerchio centrato in x in grado di trattenere k punti vicini, $d(a, b)$ è la distanza lineare tra a e b , $d_{mreach-k}$ è la massima tra le tre distanze.

Per avere un esempio visivo, guardando la figura 3.17 in cui si considerano tre punti, blu, verde e rosso, e $k = 5$, si vede che la *mutual reachability distance* tra il punto blu e il verde è pari a $core_k(p_verde)$, mentre tra il verde e il rosso si sceglie proprio la distanza $d(p_verde, p_rosso)$ tra i due.

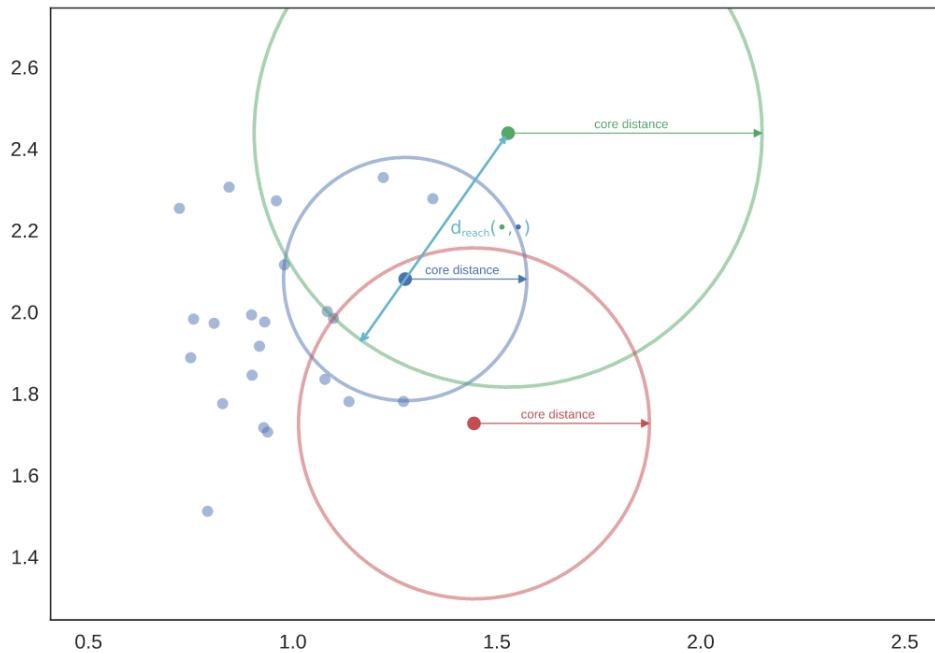


Figura 3.17. Esempio di *mutual reachability distance*.

2. **Costruzione del *minimum spanning tree*:** dopo aver calcolato le distanze mutue tra ogni punto si possono identificare le *isole*. Il dataset viene rappresentato da un grafo pesato in cui i vertici sono i dati e i rami sono le *mutual reachability distance* tra i due punti. Partendo da un certo valore di soglia abbastanza alto che viene gradualmente abbassato, a ogni passaggio si scollegano gli archi che lo superano. In questo modo si otterranno degli insiemi di punti connessi tra loro e staccati dagli altri, fino ad avere un dendrogramma come nel clustering gerarchico.

Nella pratica questo procedimento è molto dispensioso perché ci sono n^2 archi da analizzare, pertanto quello che si fa è cercare di trovare il minimo insieme di rami tali che scartandone uno si causerà una disconnessione tra i componenti e, in più, che non ci siano dei rami con peso minore che colleghino quei dati. La struttura scelta è il *minimum spanning tree* (albero con minimo ricoprente), cioè l'albero con il minore numero di archi collegante tutti i dati. Viene utilizzato l'algoritmo di Prim che costruisce l'albero un nodo alla volta, aggiungendo a ogni passaggio l'arco con minore peso che connette i nodi già uniti ad uno non ancora collegato.

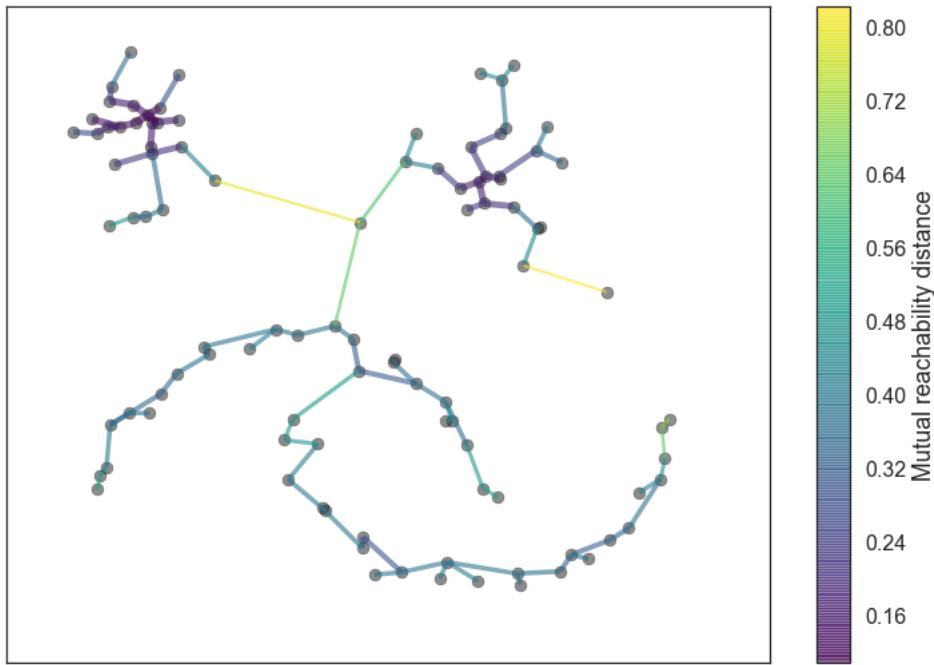


Figura 3.18. Minimum spanning tree.

3. **Gerarchia tra i cluster:** l'albero viene convertito in un dendrogramma con le componenti gerarchicamente connesse. Il modo più semplice è farlo in ordine inverso: gli archi vengono ordinati in ordine crescente per distanza e, iterando, si crea un nuovo cluster unendo due archi.

Nel clustering gerarchico classico, i cluster vengono estratti tracciando una linea orizzontale a una certa altezza nel dendrogramma, tuttavia questa è una scelta che stabilisce una densità fissa, cosa che si vuole evitare per trattare meglio casi di dataset con densità variabile. Pertanto ci vorrebbe un modo per tagliare il grafico a altezze diverse.

4. **Condensazione dell'albero:** la grande struttura gerarchica viene semplificata facendo in modo che ogni nodo dell'albero abbia meno dati connessi. Guardando la figura 3.19 l'idea chiave è immaginare che a ogni ramificazione, invece di dividere il cluster principale in due più piccoli, ci sia un solo cluster persistente che sta "perdendo" punti. Per fare ciò si utilizza il parametro **minimum cluster size** che stabilisce il numero minimo di elementi che deve avere un insieme affinché possa definire alla fine un cluster.

In seguito si scorre il dendrogramma precedentemente ottenuto e a ogni punto di separazione si confronta il numero di elementi che ha ogni gruppo con *minimum cluster size*: se uno dei due contiene meno punti di *minimum cluster size*, si dichiara quel gruppo e tutti i suoi successori come "punti separati dal

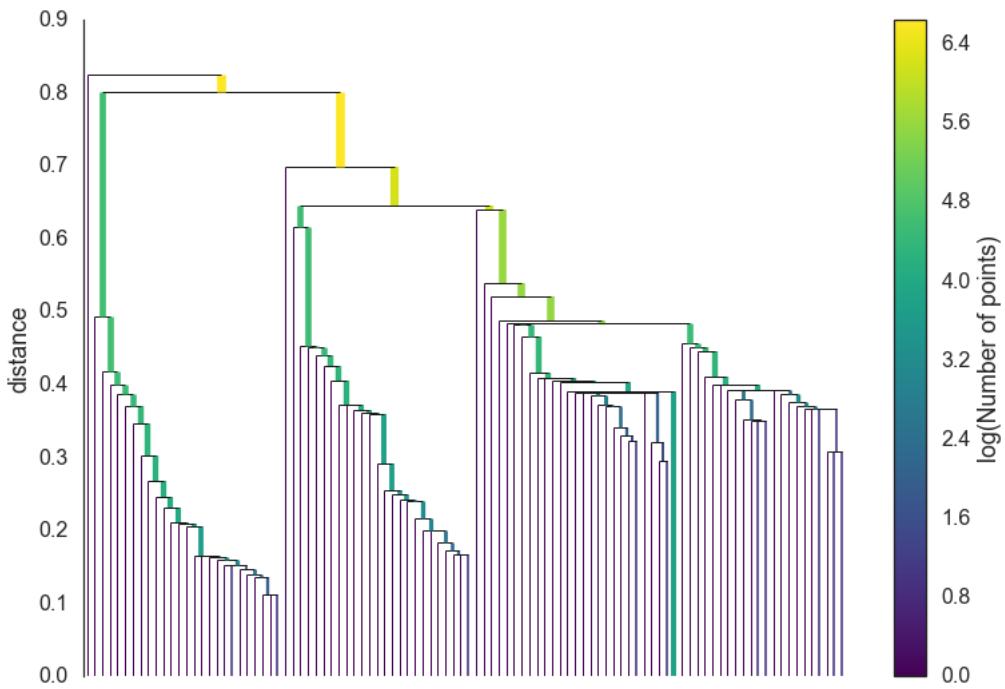


Figura 3.19. HDBSCAN Dendrogramma.

"cluster" principale (cioè l'altro), il quale conterrà l'informazione sui punti scartati e a che distanza è avvenuta la separazione; se invece entrambi hanno abbastanza punti, allora vengono visti come due cluster e si prosegue l'analisi ai loro sottoinsiemi.

Terminato questo processo si otterrà un albero più piccolo con meno nodi, ognuno contenente l'informazione su come la dimensione del cluster decresce al variare della distanza. In figura 3.20 si ha un esempio con *min_cluster_size* = 5.

5. **Estrazione dei cluster:** per estrarre i cluster si potrebbe scegliere un ϵ arbitrario e tagliare il dendrogramma a questa altezza (ottendo praticamente i risultati di DBSCAN), tuttavia questo modo non considera la densità variabile del dataset e si potrebbero avere ripercussioni sulla precisione e la compattezza dei cluster, quindi si vuole cercare un metodo per tagliare il grafico a un'altezza variabile.

Il nuovo principio che si adotta è il seguente: i cluster vengono selezionati se sono **persistenti**, cioè se "vivono" per molto tempo. Graficamente questo si può vedere in figura 3.20 se colorano una lunga regione di spazio e abbastanza larga da contenere più dati possibili del dataset. Selezionando un cluster, automaticamente non si può selezionare nessuno dei suoi discendenti. Esiste

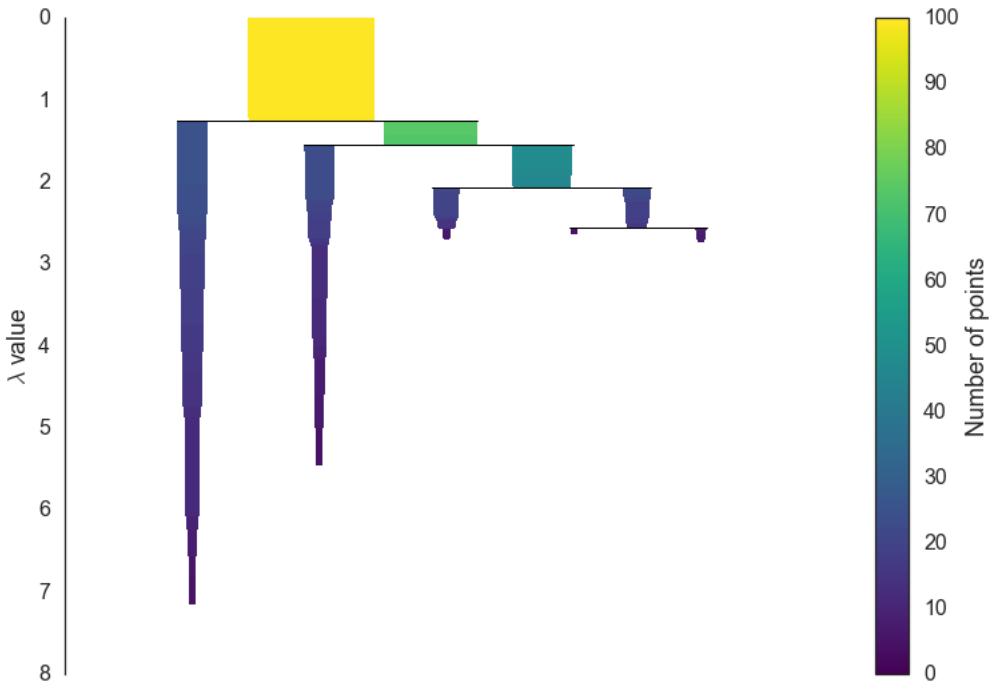


Figura 3.20. Condensed tree.

un metodo più formale per stabilire la persistenza dei cluster che si basa sul concetto di **lifetime**.

Come già spiegato per DBSCAN (par. 3.7.1), dato ϵ e preso un punto a del dataset, allora a è un punto di rumore se $\epsilon < \text{core}(a)$, altrimenti a è un *core-point* se $\epsilon \geq \text{core}(a)$, quindi genera il suo cluster. Da ciò se evince che i cluster iniziano a esistere a partire da un certo valore di ϵ e inoltre cessano di esistere a partire da altri valori perché possono unirsi insieme in seguito a un *merge*.

Per un cluster C siano $\epsilon_{\min}(C)$ e $\epsilon_{\max}(C)$ rispettivamente i valori per cui C inizia a esistere e scompare a fronte di un *merge* con un altro che abbia almeno *min_cluster_size* oggetti. La quantità $\epsilon_{\max}(C) - \epsilon_{\min}(C)$ viene chiamata **lifetime** di C che indica per quanto tempo un cluster esiste: indicativamente se un cluster dura per più tempo, allora sarà più stabile e preciso.

Un cluster C cresce nel tempo perché gli vengono aggiunti gradualmente degli elementi, pertanto è necessario parlare anche della *lifetime* degli oggetti in relazione al cluster. L'appartenenza di un punto x a C è determinata da

$$\epsilon_{\min}(x, C) = \max(\text{core}(x), \epsilon_{\min}(C))$$

cioè il massimo tra il valore $\text{core}(x)$ tale per cui x definisce esso stesso un

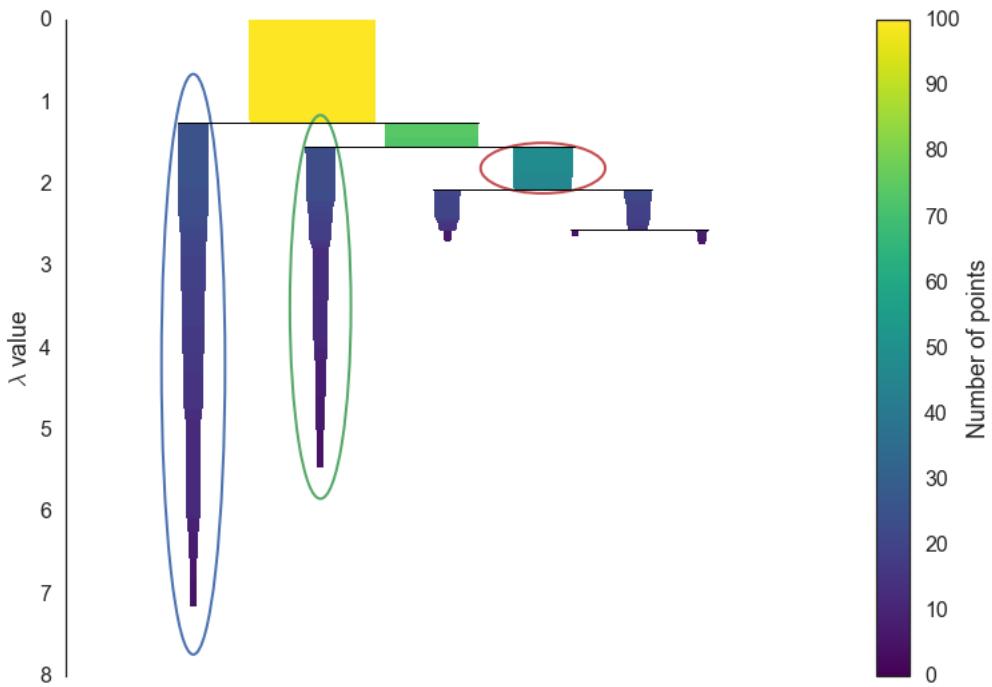


Figura 3.21. Estrazione dei cluster.

cluster e $\epsilon_{min}(C)$ dopo il quale C viene definito.

Adesso si definisce la grandezza **densità** λ come $\lambda = \frac{1}{\epsilon}$, quindi per ogni cluster si ha $\lambda_{min} = \frac{1}{\epsilon_{max}}$ e $\lambda_{max} = \frac{1}{\epsilon_{min}}$. A partire dalla densità per ogni cluster C si calcola sua **stabilità** $S(C)$:

$$S(C) = \sum_{x \in C} \left(\lambda_{max}(x, C) - \lambda_{min}(C) \right) = \sum_{x \in C} \left(\frac{1}{\epsilon_{min}(x, C)} - \frac{1}{\epsilon_{max}(C)} \right) \quad (3.22)$$

La stabilità permette infine di scegliere i cluster migliori facendo le seguenti considerazioni:

- dal dendrogramma si scelgono i cluster che massimizzano la stabilità e si escludono i loro figli;
- se la somma della stabilità dei cluster figli supera quella del padre, si considerano i figli come cluster invece del padre.

In figura 3.21 si ha un esempio grafico di ciò che è stato descritto in questo ultimo punto.

Capitolo 4

Natural Language Processing

Una volta che si è in possesso dei tweet desiderati, è necessario applicare una serie di operazioni sul testo e sulle immagini a disposizione prima di poter applicare gli algoritmi di clustering di cui siamo a conoscenza. Infatti il computer lavora con i numeri e non riesce direttamente a riconoscere il linguaggio umano. Inoltre all'interno di tweet sono presenti numerose informazioni, simboli e parole inutili ai fini dell'analisi, dunque è ben eliminarli per filtrare solo il contenuto importante. E ancora, su Twitter postano utenti provenienti da tutto il mondo che parlano lingue diverse; nel progetto si è scelto, per comodità e per avere a disposizione più materiale, di considerare solamente i tweet scritti in inglese, pertanto è necessario implementare un metodo che permetta di filtrare i tweet per lingua. Tutti questi problemi (e molti altri ancora) sono risolvibili grazie al **NLP**.

4.1 Cosa è il NLP?

Il **Natural Language Processing (NLP)** è un campo dell'intelligenza artificiale che permette di rendere comprensibile il linguaggio umano alle macchine. Il NLP combina le conoscenze in ambito linguistico, informatico e della AI per cercare regole e strutture di linguaggio e creare sistemi intelligenti che riescano a capire, analizzare e estrapolare delle informazioni dal testo.

Più precisamente, il NLP è utilizzato per comprendere il linguaggio umano analizzando diversi aspetti come sintassi, semantica, pragmatica e morfologia, poi la *computer science* trasforma queste regole di linguaggio in modelli di machine learning per risolvere problemi specifici. Un esempio comune lo si trova nell'applicazione di posta elettronica Gmail: le email vengono automaticamente in sezioni differenti (principale, promozioni, social e spam) grazie a una funzionalità del NLP chiamata *keyword extraction* (par. 4.4). Ciò consente, tramite la lettura del testo e l'estrazione

Very **intuitive platform**, I'll **definitely recommend** it.
 The **chat support** is **excellent**, really **fast** in their replies
 and very **helpful**.

Usability

Positive

Customer Support

Figura 4.1. Estrazione di topic da un testo con il NLP.

di alcune parole chiave, di capire il contesto delle mail e assegnarle alla giusta categoria, migliorando così l'esperienza dell'utente.

L'utilizzo del NLP porta altri numerosi benefici, tra cui:

- realizzare analisi su larga scala, permettendo alle macchine di capire e analizzare una quantità di testi non strutturati immensa, tra cui post sui social media, risposte e commenti, recensioni, notizie di cronaca, articoli sportivi, ecc.;
- consentire alle macchine di imparare a classificare e instradare le informazioni in modo rapido, preciso ed efficiente, senza (o quasi) l'aiuto dell'uomo;
- adattare le tecniche di NLP per bisogni personali o per i propri business, impostando dei criteri come più si preferisce.

4.2 Funzionamento del NLP

Come già accennato, le macchine non comprendono direttamente i testi scritti a parole, pertanto è necessario a priori convertirli in formato numerico. Questa operazione è chiamata **text vectorization (vettorizzazione)** e consiste nel trasformare parole e frasi in vettori numerici. Esistono diversi approcci per effettuare questa operazione, alcuni realizzati prima dell'avvento di deep learning, altri invece successivi più potenti.

4.2.1 Bag of words (BoW)

È un approccio basato sul conteggio, molto semplice ma allo stesso tempo fornisce una rappresentazione del testo che il computer può elaborare velocemente. Si ha a disposizione un dizionario di parole (appunto una *bag of words*) e si definisce un vettore della stessa dimensione del dizionario: dato un testo, le componenti del vettore sono le relative **frequenze** delle parole del dizionario che compongono il testo. Per esempio dato il dizionario {Sandro, è, un, bravo, avvocato, studente,

volenteroso, sfaticato, Andrea} e il testo *Sandro è un bravo studente e Andrea è un avvocato*, il vettore sarà $(1, 2, 2, 1, 1, 1, 0, 0, 1)$.

Per migliorare questa rappresentazione quello che si fa è rimuovere le congiunzioni, i numeri, i caratteri speciali e le **stop words**, ossia parole inutili ai fini dell'analisi del testo (per esempio le proposizioni, articoli e anche alcuni verbi come *want*, *can*, *be* e relative coniugazioni).

Gli svantaggi di questo approccio sono facilmente deducibili: il vettore non cattura il significato del testo e per testi molto grandi c'è bisogno di un dizionario almeno altrettanto vasto.

4.2.2 TF-IDF

Il **Term frequency-inverse document frequency (TF-IDF)** è una tecnica di BoW **pesato** che, dato un documento o una collezione di documenti, cerca di quantificare l'importanza di una parola in base alla frequenza con cui compare in un documento in relazione alla frequenza complessiva nella collezione. Le parole che hanno una alta frequenza in un documento molto probabilmente avranno un peso alto, tuttavia se sono presenti frequentemente all'interno dell'intera collezione, allora il loro peso scenderà.

Dato un termine i , lo $score_{i,j}$ viene calcolato per ogni documento d_j della collezione e dipende da due fattori:

- $tf_{i,j}$ è il rapporto tra il numero di volte $n_{i,j}$ in cui i compare nel documento j diviso per la lunghezza del documento $|d_j|$:

$$tf_{i,j} = \frac{n_{i,j}}{|d_j|} \quad (4.1)$$

- idf_i è il logaritmo in base 10 del rapporto tra il numero di documenti totali $|D|$ e il numero di documenti d che contengono i :

$$idf_i = \log_{10} \frac{|D|}{|\{d : i \in d\}|} \quad (4.2)$$

$$score_{i,j} = tf * idf \quad (4.3)$$

Sebbene questo metodo migliori il semplice **BoW**, ha comunque dei difetti, per esempio il fatto che in piccoli documenti anche una sola menzione di una parola è importante e necessita quindi di un peso più alto. Questi problemi vengono risolti dal metodo **Best-Ranking 25 (BM25)** in cui si tiene conto anche del rapporto tra la lunghezza del documento attuale e dell'intera collezione, oltre all'utilizzo di

alcuni parametri che impostano quanto sia importante la frequenza del termine nella collezione e quanto sia importante la lunghezza del documento.

4.3 Word2Vec

Dopo la diffusione del deep learning, si è cercato un modo migliore per rappresentare il testo in formato numerico. Nei metodi precedenti, i testi e le parole venivano visti solo come una sequenza di lettere senza un significato vero e proprio, mentre adesso si cerca di dare loro un significato.

Il primo metodo in questo ambito fu il **Word2Vec** che permette di rappresentare la semantica e le proprietà sintattiche delle parole tramite i cosiddetti **word embeddings**, dei vettori che rappresentano le parole in un nuovo dominio numerico. Il modello riceve in input del contenuto testuale e crea un vocabolario di parole distinte, le quali vengono poi mappate in uno spazio di n -dimensionale ognuna tramite il suo unico *embedding*, con n che può valere anche diverse centinaia. È ovviamente impossibile visualizzare graficamente o mentalmente uno spazio di centinaia di dimensioni, per questo per avere un'approssimazione grafica è lecito ridurre il tutto a due dimensioni.

Gli scopi di usare gli embeddings sono i seguenti:

- visto che i dati vengono rappresentati come vettori, per calcolare la somiglianza basta misurare la distanza metrica tra i due utilizzando una tecnica a scelta;
- vengono forniti come input a modelli di machine learning per operazioni di apprendimento supervisionato;
- vengono utilizzati per trovare relazioni tra i dati per operazioni di clustering.

Gli embeddings possono essere utilizzati per rappresentare come vettori numerici qualsiasi tipi di dati per effettuare le operazioni descritte in precedenza. In questo capitolo ci soffermeremo su come realizzare gli embeddings per i testi, mentre nel paragrafo successivo 4.3.3 si vedrà come generarli per le immagini.

Una rappresentazione storica degli embeddings è la cosiddetta **one-hot encoding**, ossia a partire di una frase di V parole si costruiscono dei vettori di dimensione $V \times 1$ con tutti i bit a 0 tranne uno. Per esempio dalla frase *Sandro è uno studente universitario*, si ottengono i seguenti embeddings di dimensione 5×1 :

$$Sandro = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \text{è} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \text{uno} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \text{studente} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \text{universitario} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Con questa rappresentazione a ogni parola si riesce ad assegnare un vettore univoco, tuttavia si capisce bene che in casi di vocabolari molto lunghi, la dimensione dei vettori diventa enorme. Inoltre aggiungendo delle parole al dizionario, la rappresentazione di tutte le parole cambia. Il problema più grande resta il fatto che la rappresentazione non incapsula il significato delle parole.

Il modello Word2Vec è stato creato proprio per risolvere questa problematica e ha due implementazioni principali: **Skip-Gram** e **CBOW**. Entrambi i modelli si basano sul **contesto**, cioè l'embedding di una parola viene ricavato partendo dalle parole vicine: se un gruppo di parole si trova sempre vicino alla stessa parola, allora avranno degli embeddings simili. Allo stesso modo parole riguardanti elementi simili (come per esempio le nazioni) saranno mappate più vicine.

Per etichettare quanto le parole siano simili tra loro, si stabilisce una **finestra di contesto** che determina quali parole vicine vanno considerate. Per esempio, data una finestra di contesto di dimensione $C = 2$, per ogni parola si prendono due parole prima e due dopo:

Frase	Copie
Sandro è uno studente universitario	(Sandro , è), (Sandro , uno)
Sandro è uno studente universitario	(è, Sandro), (è, uno), (è, studente)
Sandro è uno studente universitario	(uno, Sandro), (uno, è), (uno, studente), (uno, universitario)
Sandro è uno studente universitario	(studente, è), (studente, uno), (studente, universitario)
Sandro è uno studente universitario	(universitario, uno), (universitario, studente)

Tabella 4.1. Esempio con dimensione della finestra di contesto pari a 2.

La parola in grassetto a ogni passaggio è quella per cui si cercano le coppie. Non importa quanto le parole siano lontane all'interno della finestra: non si differenziano le parole se distanti 1 o 2, basta che si trovano nella finestra.

4.3.1 Skip-Gram

Con l'algoritmo **Skip-Gram** si cerca di ottenere il contesto delle parole a partire dalla **main word** (parola principale). Nel nostro esempio, scegliendo come *main word* la parola *studente*, si procede in questo modo:

1. si codificano le parole del testo usando la rappresentazione *one-hot encoding*;
2. si selezionano le finestre di contesto contenenti la parola principale, quindi in questo caso (**studente**, è), (**studente**, uno), (**studente**, universitario);

3. si utilizza una rete neurale (figura 4.2) che come input prende il vettore $V \times 1$ dell'encoding della parola principale (con V dimensione del vocabolario) e lo passa all'*hidden layer* (livello nascosto) tramite la matrice $W_{N \times V}$ dove N è la dimensione scelta per gli embedding, ottenendo effettivamente l'embedding $E_{N \times 1}$ della parola;
4. si moltiplica l'embedding ottenuto per la matrice dei pesi $W'_{V \times N}$ che alla fine del processo di addestramento conterrà gli embedding delle parole in uscita. Dal prodotto di $E_{N \times 1}$ per ogni riga di $W'_{V \times N}$ si ottengono C vettori y_i V -dimensionali, uno per ogni parola nelle finestre di contesto;
5. si passano i vettori ottenuti al cosiddetto *softmax layer* che per ognuno computa un vettore V -dimensionale con una distribuzione di probabilità: le componenti del vettore hanno valori compresi tra 0 e 1 e la loro somma da 1. Ogni vettore va interpretato come un'approssimazione dell'encoding della relativa parola: dal confronto dei due si aggiornano i pesi delle matrici W e W' (figura 4.3).

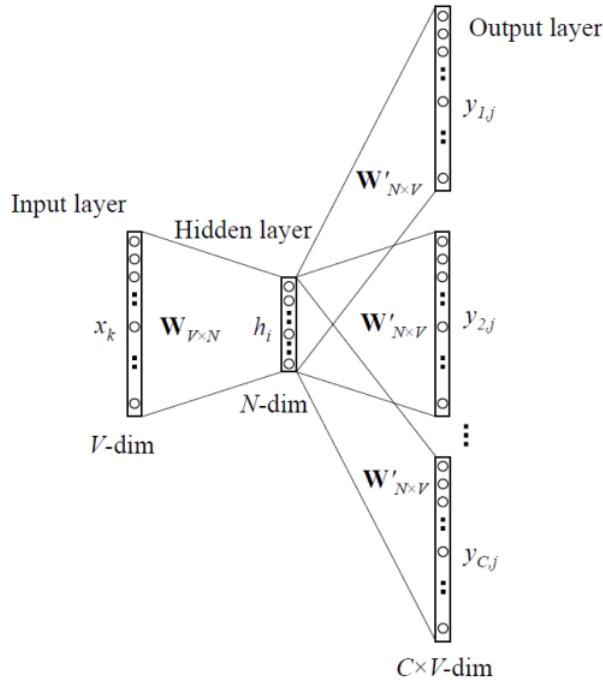


Figura 4.2. Schema Skip-Gram

Finita la fase di addestramento su tutto il vocabolario, si avrà una matrice di pesi $W_{V \times N}$ che connette gli input al livello nascosto, la quale applicata a una parola genererà il suo embedding. Se il processo è stato effettuato correttamente, la

$$\begin{bmatrix} 0.1 \\ 0.2 \\ 0.96 \\ 0 \\ 0.1 \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

Figura 4.3. Confronto tra il vettore probabilistico e il *one-hot encoding*. Il fatto che la componente più grande del primo vettore (0.96) abbia lo stesso indice dell'1 nel secondo, è indice di una buona approssimazione.

rappresentazione incapsula anche la semantica e le parole simili saranno più vicine nello spazio vettoriale, cioè i loro embeddings si assomiglieranno.

4.3.2 CBOW

L'algoritmo **Continuous Bag of Words (CBoW)** è simile a Skip-Gram ma applica l'operazione opposta: partendo dal contesto cerca di predire la parola principale. Supponendo di volere trovare l'embedding della parola studente *studente*, l'algoritmo è il seguente:

1. si codificano le parole del testo usando la rappresentazione *one-hot encoding*;
2. si selezionano le finestre contenenti la parola principale, quindi (**studente**, è), (**studente**, uno), (**studente**, universitario)
3. la rete neurale prende in input gli encoding di dimensione $V \times 1$ delle parole presenti e calcola la media degli embedding calcolati per ogni encoding tramite la matrice $W_{N \times V}$, ottenendo un vettore di dimensione $N \times 1$;
4. tramite l'applicazione della matrice $W'_{V \times N}$ e del *softmax layer* si ottiene una distribuzione di probabilità che approssima l'encoding della parola principale. In base all'errore tra i due si aggiornano i pesi.

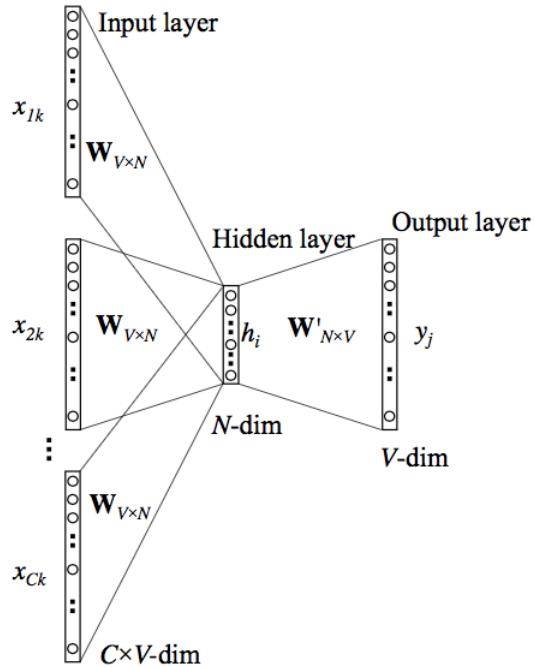


Figura 4.4. Schema CBOW

Una volta finito l’addestramento si avrà la matrice $W_{N \times V}$ in grado di trasformare le parole date negli embedding e ancora una volta terrà conto della semantica.

Sebbene i due metodi si basino su un’idea simile, non si comportano allo stesso modo su un determinato dataset: è stato provato che Skip-Gram lavora meglio su dataset piccoli e può rappresentare meglio parole meno frequenti; CBoW è più veloce nell’addestramento di Skip-Gram e rappresenta meglio le parole più frequenti.

4.3.3 Embeddings per le immagini

Nei paragrafi precedenti sono stati descritti i motivi per cui è utile analizzare un contenuto testuale e metodi per farlo. Nel mondo attuale può essere utile applicare delle operazioni simili sulle immagini: per esempio sul noto servizio di Pinterest se si cerca "tagli di capelli ricci" automaticamente appariranno foto riguardanti l’argomento, e cliccando su una in particolare ne verranno proposte altre sempre più simili. Inoltre se si dirige un’azienda di vendita di prodotti, può essere utile suggerire ai clienti prodotti simili ad alcuni che hanno già acquistato.

Come per i testi, anche le immagini subiscono un processo di vettorizzazione affinché sia possibile confrontarle, quindi ad ognuna viene assegnato un embedding. Un’immagine in formato digitale viene scomposta in **pixel**, la più piccola unità indivisibile che la costituisce. Ipotizziamo di avere un’immagine di dimensione 3×3 che viene rappresentata con in figura 4.5.

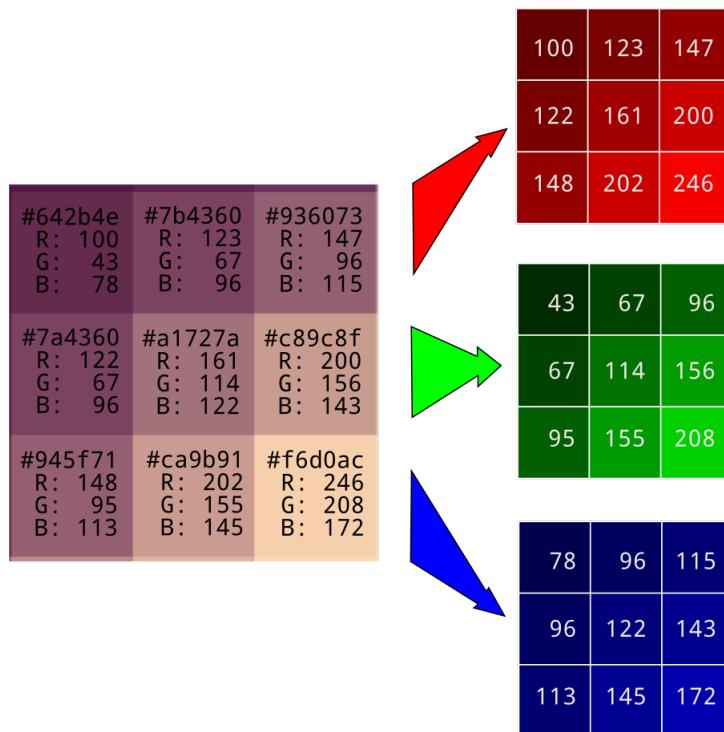


Figura 4.5. Immagine in digitale.

Ogni immagine può essere scomposta in tre colori primitivi: rosso, verde e blu. Ogni cella dell’immagine contiene i valori dei pixel, che possono variare nell’intervallo $[0, 255]$. Per ogni colore si ha una matrice 3×3 , quindi per ognuno avremo 9 pixel per un totale di $3 * 9 = 27$ pixel.

Nel mondo reale le immagini non sono così piccole, basti pensare che il formato comune 1080p ha una dimensione di 1920×1080 comporta un totale di $1920 * 1080 * 3 = 6220800$ pixel da comparare con altre immagini della stessa dimensione. È facile intuire che il costo computazionale diventa notevole, pertanto è necessario cercare di rappresentare le immagini in un dominio più piccolo: qui entrano in gioco gli embeddings.

Nel progetto è stato utilizzato un modello preaddestrato **clip-ViT-B-32** della libreria *SentenceTransformers* di Python che mappa le immagini in embedding di lunghezza 512 (vedi capitolo 5).

4.4 Keyword extraction

Il **Keyword Extraction** è un metodo automatico per estrarre l’informazione più rilevante a partire da un testo, quindi parole o frasi che ne riassumono al meglio l’argomento trattato. Un’operazione del genere può essere applicata su testi di

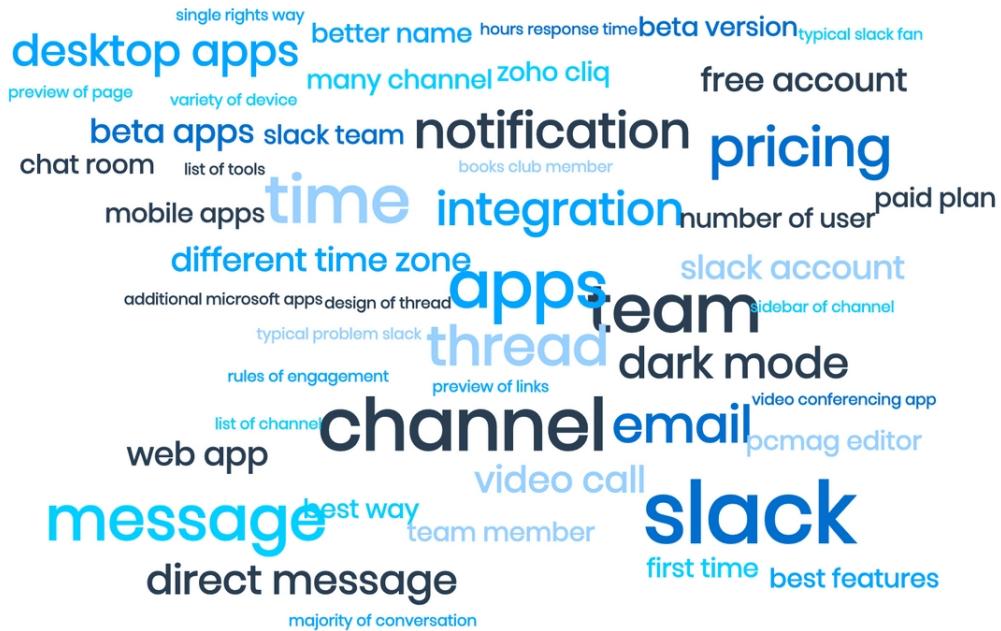


Figura 4.6. Wordcloud di Slack.

qualsiasi tipo (recensioni, articoli di giornali, libri, tweet, ecc.) con uno scopo diverso: per esempio, un venditore potrebbe voler analizzare le centinaia di recensioni ricevute, pertanto con un modello di *keyword extraction* può velocemente identificare le parole maggiormente sono state utilizzate dagli utenti per descrivere i propri prodotti.

I modelli di keyword extraction permettono di estrarre sia parole singole (*key-words*) che gruppi di parole (*key-phrases*), le quali possono essere utilizzate per identificare in modo rapido le informazioni presenti in un cluster. In figura 4.6 è riportata una *word clouds* di keywords relative a delle recensioni online di Stack.

Dato che nel mondo di oggi si produce costante una quantità di dati non strutturati enorme (cioè non organizzati in una struttura specifica e quindi difficili da analizzare), usare un modello di *keywords extraction* porta i seguenti vantaggi:

- **scalabilità:** automatizzare il processo di estrazione di parole chiave permette di analizzare qualsiasi quantità di dati;
- **criterio consistente:** l'estrazione delle keywords viene effettuata sulla base di un criterio e dei parametri scelti dall'utente;
- **analisi *real-time*:** si può effettuare l'operazione di keywords extraction su tweet, post, recensioni, sondaggi, ecc., in tempo reale, quindi si ottiene un andamento di come evolve un particolare topic nel tempo.

4.4.1 Funzionamento

Esistono diversi approcci per estrarre le keywords:

- **approccio statistico:** è il più semplice tra quelli descritti, estrae le keywords in base alla frequenza con cui le parole sono presenti in un testo. Si può usare un approccio basato sul conteggio del tipo *Bag of Words* (par. 4.2.1) oppure uno più elaborato come *TF-IDF* (par. 4.2.2);
- **approccio linguistico:** basato sulle informazioni linguistiche dei testi e delle parole che contengono. Per esempio si considerano la morfologia e la sintassi per assegnare un punteggio più alto alle parole in base al loro tipo, in particolare i nomi sono più adatti per essere keywords perché solitamente contengono più informazioni rispetto ad altre parole;
- **approccio basato sui grafi:** il testo viene rappresentato come grafo i cui nodi sono le parole collegate da rami (tutti unidirezionali oppure no) che possono essere etichettati con la relazione che intercorre tra le parole. Le keywords sono estrarre in base all'importanza che viene assegnata ai nodi, misurata in base al **grado** (*degree D*) del nodo v calcolato nel seguente modo:

$$D_v = \frac{(D_v^{in} + D_v^{out})}{N - 1} \quad (4.4)$$

dove D_v^{in} è il grado in ingresso (numero degli archi entranti in v), D_v^{out} è il grado in uscita, N è il numero totale di vertici. I nodi con grado massimo vengono estratti come keywords;

- **approccio machine learning:** esistono diversi tipi algoritmi di machine learning per estrarre le keywords da un testo. Nel progetto è stato utilizzato l'algoritmo **BERT** sviluppato da Google (4.4.3).

4.4.2 Utilizzi

Il metodo del keywords extraction trova impiego in qualsiasi ambito:

- **analisi dei social e delle recensioni:** le persone sui social network come Twitter condividono i propri pensieri sugli argomenti più disparati, dalla politica del proprio paese all'ultimo aggiornamento del loro videogioco preferito. Per le aziende o gli individui direttamente coinvolti in questi temi è bene capire cosa pensano le persone esterne dei loro prodotti, delle loro idee politiche, dei servizi che offrono, ecc.;

- **business intelligence:** nel caso si voglia lanciare un nuovo business, può essere interessante capire in anticipo le richieste degli eventuali clienti, oppure analizzare la futura concorrenza per evitare di fare gli stessi errori o prendere spunto dai suoi punti di forza;
- **ottimizzazione dei motori di ricerca:** i motori di ricerca quali Google utilizzano un modello di keywords extraction per migliorare la ricerca di contenuti sul web.

4.4.3 BERT

BERT(*Bidirectional Encoder Representations from Transformers*) è un framework open source di machine learning realizzato da Google per operazioni di NLP. Si basa sui **Transformers**, un modello di deep learning di più livelli in cui ogni elemento in output è connesso a ogni elemento in input in base a dei pesi calcolati dinamicamente dalle loro connessioni, e a differenza di altre reti neurali standard l'input viene processato tutto insieme.

Il termine *bidirezionale* indica che, a differenza di altri algoritmi di NLP che leggono il testo in input in modo sequenziale (da destra verso sinistra o viceversa), BERT legge l'intera sequenza di parole tutta in una volta grazie a *transformers*. In questo modo riesce ad apprendere il contesto di una parola in base alle diverse parole che la circondano nel testo.

Questo approccio differenzia BERT da altri metodi di NLP sviluppati in passato, come lo stesso Word2Vec (4.3), in particolare dal punto di vista degli embeddings generati. In Word2Vec, in cui l'input viene analizzato in maniera sequenziale, per stabilire il significato di una parola e generare il suo embeddings vengono considerate solo le parole lette fino a quel momento; in BERT invece, siccome il testo viene analizzato tutto insieme, ogni parola può influenzare il contesto complessivo e il significato di ognuna può essere molteplice. Questo metodo simula in maniera migliore il comportamento di ragionamento della mente umana.

Per estrarre le keywords con BERT si utilizza il metodo **KeyBERT**, il quale genera gli embeddings del testo considerato e applica la tecnica del *cosine similarity* per trovare le parole che gli somigliano di più che teoricamente sono quelle che meglio riassumono il contenuto del testo.

```

1 def keywords_gen(ids):
2     testi = []
3     for id in ids:
4         for tw in tweets:
5             if int(tweets[tw]['id']) == int(id):
6                 testi.append(tweets[tw]['text'])
7

```

```
8     testi = " ".join(testi)
9     keywords = kw_model.extract_keywords(testi, keyphrase_ngram_range
10        =(3,5), top_n=15)
11     for k in keywords:
12         if key_match(k[0]):
13             return k[0]
14     if len(keywords) == 0:
15         return "Outliers"
16     return keywords[0][0]
```

Codice 4.1. Funzione che genera le keywords con BERT ricevendo gli id dei tweet desiderati.

Capitolo 5

Metodo proposto

In questo capitolo vengono descritte le procedure per applicare le nozioni descritte nella trattazione sui dati di nostro interesse. In particolare si vedrà innanzitutto come vengono analizzati i testi con le tecniche del NLP, poi come convertire testi e immagini in embeddings per applicare infine gli algoritmi di clustering scelti, ovvero *k-means*, *DBSCAN* e *HDBSCAN*.

Per l'esecuzione del codice è stato utilizzato **Google Colaboratory** (abbreviato **Colab**), un programma browser che dà la possibilità di eseguire codice Python in ambienti chiamati *colab notebooks*. Il vantaggio di Colab sta nel fatto che mette a disposizione dello sviluppatore la potenza di calcolo delle GPU di Google direttamente da casa propria, permettendo così l'esecuzione di tutte queste operazioni di machine learning che un PC "normale" non sarebbe in grado o richiederebbe un tempo di elaborazione lunghissimo.

Per implementare gli algoritmi di clustering si utilizza **Scikit-Learn** (*sklearn*), una libreria open-source che offre gli strumenti per costruire modelli di machine learning in Python.

5.1 Clustering dei testi

In questa sezione vengono mostrati i passi per effettuare il clustering dei soli testi dei tweet, dalla preparazione dei dati all'implementazione degli algoritmi di clustering scelti.

5.1.1 Embeddings e Curse of dimensionality

Per calcolare gli embeddings dei tweet è stato utilizzato il modello pre-addestrato *clip-ViT-B-32-multilingual-v1* sviluppato da OpenAI in grado di mappare sia testi che immagini in uno spazio vettoriale di 512 dimensioni. Quando si ha a che fare

con dati con così tante dimensioni, si può incorrere nel problema chiamato **curse of dimensionality**: dato che l'algoritmo basa la ricerca dei cluster sulla misura della distanza, in uno spazio con tante dimensioni gli elementi appariranno più o meno equidistanti compromettendo l'analisi, e inoltre i tempi di esecuzione aumentano vertiginosamente anche su dataset non troppo grandi.

La risoluzione di questo problema passa attraverso l'applicazione di un modello di riduzione delle features, in cui i dati vengono convertiti in un dominio di dimensioni ridotte. Ovviamente il modello non si limita a troncare le caratteristiche in eccesso, bensì cerca una corrispondenza tra quelle di partenza e le nuove create nel nuovo dominio.

Il metodo descritto viene implementato in Python tramite la libreria *umap* e gli embeddings vengono convertiti in uno spazio 100-dimensionale.

5.1.2 Feature Scaling

Dopo aver ottenuto gli embeddings dei testi si esegue il cosiddetto **feature scaling**, un'operazione fondamentale nella creazioni di modelli di machine learning (e quindi anche nel clustering). Le caratteristiche possono avere valori più o meno grandi e dato che l'algoritmo può solo vedere numeri, questo può portare a errori di valutazione. Per esempio, si suppone per un prodotto alimentare in vendita in un negozio si abbiano due caratteristiche, peso e prezzo: se il peso è molto maggiore del prezzo, per l'algoritmo la caratteristica più importante sarà la prima, quando in realtà contano tutte e due perché definiscono due proprietà diverse del prodotto. Affinché l'algoritmo valuti entrambe allo stesso modo è necessario, appunto, scalarle, cioè renderle comparabili e impedire che differenze di valori molto grandi tra le caratteristiche compromettano il risultato.

I due modi più comuni per scalare sono:

- **normalizzazione:** si scalano tutti i punti negli intervalli $[0, 1]$ o $[-1, 1]$;
- **standardizzazione:** si trasforma il dataset in modo da avere media nulla e varianza pari a 1 per ogni variabile (utilizzata nel progetto). Per ogni variabile x il suo valore standardizzato x_{std} vale:

$$x_{std} = \frac{x - \mu}{\sigma} \quad (5.1)$$

dove μ e σ sono rispettivamente la media e la varianza della variabile.

In Python l'implementazione è semplificata dall'utilizzo di **StandarScaler** della libreria *sklearn*.

Un vantaggio dello scaling è il miglioramento della velocità con cui l'algoritmo converge.

In seguito gli embeddings dei tweet vengono salvati nel dataframe `df_text` ognuno con il proprio id e finalmente possono essere sottoposti al clustering.

5.1.3 K-Means

Come spiegato nel paragrafo 3.5.1, prima di applicare k-means bisogna prima stimare il numero di cluster migliore per l'algoritmo utilizzando il metodo della silhouette e/o l'*elbow method*. La funzione `kmeans_elbow` nella sezione di codice 5.1 prende in input gli embeddings e restituisce il grafico dell'inerzia in funzione di k e gli indici di silhouette per ogni valore di k in modo da poter scegliere quello migliore.

```

1 def kmeans_elbow(emb):
2     inerzia = []
3     output = []
4     K = range(4,25)
5     for k in K:
6         km = KMeans(n_clusters=k)
7         y = km.fit_predict(emb)
8         try:
9             sil = silhouette_score(X=emb, labels=y)
10            print(f"per k={k} => silhouette_score={sil}")
11
12        except:
13            print(f"per k={k} => silhouette_score=NA")
14        output.append((k, sil))
15        inerzia.append(km.inertia_)
16
17    plt.figure(figsize=(16,8))
18    plt.plot(K, inerzia, 'bx-')
19    plt.xlabel('Numero di cluster')
20    plt.ylabel('Inerzia')
21    plt.title(f'Elbow tecnique per K-Means\n{titolo}')
22    plt.savefig(f'{foto}_elbow.png', bbox_inches='tight')
23    plt.show()

```

Codice 5.1. Elbow method per k-means

Dopo aver dedotto dal grafico il valore più appropriato del numero di cluster e averlo salvato in `n_cluster`, si effettua il clustering con k-means, passando alla funzione 5.2 gli embeddings, `n_cluster`, il dataframe con i tweet `df_kmeans` e il parametro `i = 0` che corrisponde all'indice della nuova colonna da aggiungere nel dataset in cui si salvano le label generate dal clustering per ogni tweet. L'output è il grafico (bidimensionale) dei cluster ognuno descritto dalle rispettive keywords generate dalla funzione `keywords_gen` che le crea ricevendo in input la lista degli id dei tweet che fanno parte di un cluster selezionato.

```

1 def kmeans_cluster(df, n_clusters, emb, i):
2     km = KMeans(n_clusters=n_clusters)
3     y = km.fit_predict(emb)
4     sil = silhouette_score(X=emb, labels=y)
5     print(f"silhouette score: {sil}")
6
7     df[i] = y    # creo la colonna delle labels ottenute nella i-esima
8             # clusterizzazione sempre nel
9             # dataframe originale df
10 unilab = np.unique(y)
11 keywords = []
12 for l in unilab:
13     ids = df['id'].loc[df[i] == l].tolist()    # estraggo la lista
14             # degli id appartenenti a un cluster
15     k = keywords_gen(ids)                      # genero le keywords
16     keywords.append(k)
17
18     figure(figsize=(12, 12))
19     ax = plt.axes()
20     ax.set_facecolor("white")
21     scatter = plt.scatter(emb[:, 0], emb[:, 1], c=y, cmap="tab20_r")
22     plt.legend(handles=scatter.legend_elements(num=unilab)[0], labels=
23                 keywords, title="Clusters", fontsize="small", loc="upper left",
24                 bbox_to_anchor=(1, 1))
25     plt.title(f"K-Means {titolo}")
26     plt.savefig(f'{foto}_kmeans.png', bbox_inches='tight')
27
28
29 n_clusters=16
df_kmeans=df_text.copy()      # df_text contiene gli id dei tweet con i
                             # relativi embeddings
30
31 kmeans_cluster(df_kmeans, n_clusters, embeddings, 0)

```

Codice 5.2. Clustering con k-means

5.1.4 DBSCAN

Per DBSCAN è necessario impostare correttamente i valori di *eps* e *min_samples*, ossia i parametri ϵ e *minPts* descritti nel paragrafo 3.7.1. Per il secondo si sceglie un valore arbitrario che non supera la dimensione delle caratteristiche dei dati e può essere messo a punto effettuando diversi test.

Per *eps* invece, essendo il parametro più importante, la scelta è un po' più complicata e deve essere fatta con criterio. Avendo già scelto *min_samples*, per qualsiasi punto del dataset si calcola la distanza con i rispettivi *min_samples* punti

più vicini: la distanza con il più lontano tra questi corrisponde al valore minimo di eps affinché il punto scelto inizialmente definisca il suo cluster.

In Python il calcolo descritto viene semplificato dal metodo **NearestNeighbors(n_neighbors=k)** di *sklearn* in cui k rappresenta il numero di punti con cui si vuole calcolare la distanza. È bene notare che tra questi viene anche considerato il punto stesso, la cui distanza è pari a 0.

```

1 def dbscan_eps(emb, k):
2
3     neigh = NearestNeighbors(n_neighbors=k, metric="manhattan")
4     nbrs = neigh.fit(emb)
5     distances, indices = nbrs.kneighbors(emb)
6
7     figure(figsize=(10, 8), dpi=100)
8     distances = np.sort(distances, axis=0)
9     distances = distances[:, 1]
10    plt.title(f"Stima di eps\n{n{titolo}}")
11    plt.xlabel("Punti")
12    plt.ylabel("Eps")
13    plt.grid(True)
14    plt.plot(distances)
15    plt.savefig(f'{foto}_eps.png', bbox_inches='tight', dpi=150)

```

Codice 5.3. Stima di eps .

Le distanze vengono rappresentate in un grafico in ordine crescente e il valore di eps ottimale è in corrispondenza del gomito della funzione. Infatti, intuitivamente, prendendo un valore troppo piccolo molti punti risulteranno come punti isolati o si ottengono tanti piccoli cluster, mentre un valore troppo grande tende a far mischiare tra loro diversi cluster.

Il clustering con DBSCAN viene poi effettuato con la seguente funzione:

```

1 def dbscan_cluster(df, eps, min, emb, i):
2     db = DBSCAN(eps=eps, min_samples=min, metric="manhattan").fit(emb)
3     y = db.labels_
4     sil = silhouette_score(X=emb, labels=y)
5     num_outliers = list(y).count(-1)
6     unilab = np.unique(y)
7     print(f"eps = {eps}, min samples = {min}, silhouette score: {sil},
8           numero outliers: {num_outliers} su {len(emb)}, numero di cluster: {
9             len(unilab)})")
10    df[i] = y      # creo la colonna delle labels ottenute nella i-esima
11                                # clusterizzazione sempre nel
12                                # dataframe originale df
13
14    keywords = []

```

```

13     for l in unilab:
14         if l == -1:
15             keywords.append("Outliers")
16             continue
17         ids = df['id'].loc[df[i] == l].tolist()    # estraggo la lista
18         degli id appartenenti a un cluster
19         k = keywords_gen(ids)
20         keywords.append(k)    # ottengo solo la parola/frase
21
22         figure(figsize=(12, 12), dpi=100)
23         plt.title(f"DBSCAN {titolo}")
24         ax = plt.axes()
25         ax.set_facecolor("white")
26         scatter = plt.scatter(emb[:,0], emb[:,1], c=y, cmap="tab20_r")
27         plt.legend(handles=scatter.legend_elements(num=unilab)[0], labels=
28         keywords, title="Clusters", fontsize="small", loc="upper left",
29         bbox_to_anchor=(1, 1))
30         plt.savefig(f'{foto}_dbscan.png', bbox_inches='tight', dpi=150)
31         plt.show()
32
33     return

```

Codice 5.4. Clustering con DBSCAN

5.1.5 HDBSCAN

HDBSCAN prevede la scelta di due parametri:

- **min_cluster_size** che indica il numero minimo di elementi che deve avere un cluster per essere definito tale. Intuitivamente aumentando *min_cluster_size* diminuisce il numero dei cluster trovati perché alcuni vengono uniti tra loro e cluster che precedentemente esistevano (separati) potrebbero essere considerati come rumore;
- **min_samples** indica il numero minimo di elementi vicini a un *core point* (praticamente il *minPts* per DBSCAN). La sua scelta determina quanto i cluster siano conservativi: un valore grande tenderà a rendere i cluster più grandi ma selettivi, quindi più punti saranno rumore.

Inoltre l'accuratezza e le prestazioni del clustering migliorano notevolmente rispetto a DBSCAN grazie al procedimento descritto nel paragrafo 3.10, dato che il valore di ϵ viene scelto dinamicamente.

HDBSCAN possiede il metodo *relative_validity_* che restituisce un valore compreso tra 0 e 1 che permette di stimare la validità del clustering sui dati in possesso a partire dai parametri scelti: più il valore è alto, più il clustering è affidabile.

Tramite la funzione *hdbSCAN_validity* viene calcolato l'indice di validità con varie combinazioni di parametri, poi restituisce la coppia *min_samples*, *min_cluster_size* che ha generato il valore migliore.

```

1 def hdbSCAN_validity(emb, min_samples, min_size):
2     print(f"Numero embeddings: {len(emb)}")
3     valArr = []
4     samples = min_samples
5     size = min_size
6
7     for i in range(10):
8         size = min_size
9         for j in range(10):
10            if size > samples:
11                hd = hdbSCAN(min_samples = samples,
12                             min_cluster_size = size, gen_min_span_tree=True).fit(emb)
13                val = hd.relative_validity_
14                valArr[val] = {
15                    "samples": samples,
16                    "size": size
17                }
18                size += 5
19                samples += 5
20
21    # restituisco i valori di min_samples e min_cluster_size che danno
22    # il punteggio migliore
23    sortVal = sorted(valArr.items(), reverse=True)
24
25    score = sortVal[0][0]
26    return sortVal[0][1]['samples'], sortVal[0][1]['size']

```

Codice 5.5. Funzione per determinare i parametri migliori.

I parametri così ricavati vengono ricavati per effettuare il clustering con HDBSCAN. Per ottenere risultati migliori, la funzione *hdbSCAN_cluster* effettua ricorsivamente il clustering degli eventuali outliers che ottiene, ricalcolando i parametri con *hdbSCAN_validity* a ogni passaggio, finché non riesce più ad ottenere cluster. I cluster vengono graficati in un *scatter plot* con le rispettive labels, ovvero le keywords estratte dai testi dei tweet che ne fanno parte. La funzione, inoltre, restituisce le keywords concatenate tra loro utilizzate poi per realizzare una *word cloud*.

```

1 # funzione che effettua il clustering e nelle label mostra le keywords
2 # per ogni cluster
3 def hdbSCAN_cluster(df, emb, i, start_samples, start_size):
4     if i == 0: # primo passaggio: clusterizzo tutto
5         samples, size = hdbSCAN_validity(emb, start_samples, start_size)

```

```

6         hd = hdbscan.HDBSCAN(min_samples = samples, min_cluster_size =
7             size)
8         y = hd.fit_predict(emb)
9         labels = hd.labels_
10        df[i] = labels      # creo la colonna delle labels ottenute nella
11            # esima clusterizzazione sempre nel
12            # dataframe originale df
13        unilab = list(np.unique(labels))
14        num_outliers = list(y).count(-1)
15        print(f"min_samples = {start_samples}, min_cluster_size = {
16            start_size}, numero outliers: {num_outliers} su {len(emb)}, numero
17            di cluster: {len(unilab)}")
18
19        # cerco le keywords associate a ogni cluster
20        # per ogni cluster, cerco i tweet che ne fanno parte, concateno
21        # i testi e vi applico il modello che genera le keywords
22
23        keywords = []
24        for l in unilab:
25            # print("label ", l)
26            if l == -1:
27                keywords.append("Outliers")
28            else:
29                ids = df['id'].loc[df[i] == l].tolist()    # estraggo la
30                lista degli id appartenenti a un cluster
31                k = keywords_gen(ids)
32                keywords.append(k)
33
34        figure(figsize=(12, 12))
35        plt.title(f"HDBSCAN {titolo}\nClusterizzazione n. {i}")
36        ax = plt.axes()
37        ax.set_facecolor("white")
38        scatter = plt.scatter(emb[:, 0], emb[:, 1], c = y, cmap =
39            "tab20_r")
40        plt.legend(handles=scatter.legend_elements(num=unilab)[0],
41            labels=keywords, title="Clusters", fontsize="small", loc="upper
42            left", bbox_to_anchor=(1, 1))
43        plt.savefig(f"{foto}_hdbscan{i}.png", bbox_inches='tight', dpi
44            =150)
45        plt.show()
46        print()
47
48        try:
49            keywords.remove("Outliers")
50        except:
51            pass
52        words = " ".join(keywords)

```

```
43
44     # termine se ottengo solo -1 OPPURE non ottengo nessun -1
45     if set(unilab) == {-1} or (-1 not in unilab):
46         return words
47     else:
48         df_outliers = df.loc[df[i] == -1]      # estraggo gli
49         embeddings degli outliers appena ottenuti
50         embOutliers = df_embeddings(df_outliers)
51         i += 1      # incremento l'indice delle labels
52
53         words += (hdbscan_cluster(df, embOutliers, i, start_samples,
54             start_size))    # richiamo la funzione sugli embeddings degli
55             outliers
56
57         return words
58
59     else:    # clusterizzo solo gli outliers finche' ottengo solo
60         outliers
61         samples, size = hdbscan_validity(emb, start_samples, start_size
62     )
63
64     hd = hdbscan.HDBSCAN(min_samples = samples, min_cluster_size =
65     size)
66     df_outliers = df.loc[df[i-1] == -1]      # estraggo gli
67     embeddings degli outliers del passaggio precedente
68     y = hd.fit_predict(emb)
69     labels = hd.labels_
70     unilab = list(np.unique(labels))
71     num_outliers = list(y).count(-1)
72     print(f"min_samples = {samples}, min_cluster_size = {size},
73         numero outliers: {num_outliers} su {len(emb)}, numero di cluster: {len(unilab)}")
74
75     list_labels = [] # lista delle label nelle clusterizzazioni
76     per i > 0: per labels diverse da -1 salvo '', per labels uguali -1
77         # salvo il valore delle labels dopo la nuova
78         clusterizzazione
79     j = 0
80     for row in df[i-1]:
81         if row != -1: list_labels.append('')
82         else:
83             list_labels.append(labels[j])
84             j+=1
85     df[i] = list_labels    # nuova colonna di labels nel dataframe
86     di partenza
87
88     # cerco le keywords associate a ogni cluster
```

```

78     # per ogni cluster, cerco i tweet che ne fanno parte, concateno
79     i testi e vi applico il modello che genera le keywords
80     keywords = []
81     for l in unilab:
82         if l == -1:
83             keywords.append("Outliers")
84         else:
85             ids = df['id'].loc[df[i] == l].tolist()
86             k = keywords_gen(ids)
87             keywords.append(k)    # ottengo solo la parola/frase
88
88     figure(figsize=(12, 12))
89     plt.title(f"Clusterizzazione n. {i}")
90     ax = plt.axes()
91     ax.set_facecolor("white")
92     scatter = plt.scatter(emb[:,0], emb[:,1], c=y, cmap =
93                           "tab20_r")
94     plt.legend(handles=scatter.legend_elements(num=unilab)[0],
95                labels=keywords, title="Clusters", fontsize="small", loc="upper
96                left", bbox_to_anchor=(1, 1))
97     plt.savefig(f"{foto}_hdbSCAN{i}.png", bbox_inches='tight', dpi
98 =150)
99     plt.show()
100    print()
101
102    try:
103        keywords.remove("Outliers")
104    except:
105        pass
106    words = " ".join(keywords)
107    # termino se ottengono solo -1 OPPURE non ottengo nessun -1
108    if set(unilab) == {-1} or (-1 not in unilab):
109        return words
110    else:    # proseguo nella clusterizzazione
111        df_outliers = df.loc[df[i] == -1]    # estraggo gli
112        embeddings degli outliers
113        embOutliers = df_embeddings(df_outliers)
114        i += 1
115
116        words += (hdbSCAN_cluster(df, embOutliers, i, start_samples
117                                , start_size))    # richiamo la funzione sugli embeddings degli
118        outliers
119
120    return words

```

Codice 5.6. Clustering con HDBSCAN

5.1.6 Keywords extraction e wordcloud

Una volta aver ottenuto i cluster, le loro descrizioni vengono create estraendo delle *keywords* dalla concatenazione di tutti i testi dei tweet facenti parte dello stesso insieme, utilizzando la funzione 4.1 *keywords_gen* descritta nel paragrafo 4.4.3.

Dalla concatenazione di tutti i testi per ogni intervallo temporale vengono generate delle wordclouds contenenti le parole più rilevanti per ciascuno, utilizzate per vedere come evolvono a grandi linee i topic principali nel corso del tempo.

```

1 def wordcloud_gen(words):
2     stopwords = set(STOPWORDS)
3     image = Image.open(r"/content/ucraina.png").convert("RGB")
4     mask = np.array(image)
5     colors = ImageColorGenerator(mask)
6     wordcloud = WordCloud(width = 800, height = 800, background_color =
7         'white', stopwords = stopwords, min_font_size = 10, mask=mask, mode
8         ="RGBA", collocations=False, max_words=400, min_word_length=4).
9     generate(words)
10
11 # plot the WordCloud image
12 plt.figure(figsize = (10, 10), facecolor = None)
13 plt.imshow(wordcloud.recolor(color_func=colors), interpolation="bilinear")
14 plt.axis("off")
15 plt.tight_layout(pad = 0)
16
17 print(colors)
18 plt.show()

```

Codice 5.7. Generazione wordcloud.

5.2 Clustering delle immagini

Per calcolare gli embeddings delle immagini viene utilizzato il modello pre-addestrato *cli-ViT-B-32*, mentre per generare le *caption* (una descrizione di esse) il modello *nlpconnect/vit-gpt2-image-captioning* (codice 5.8).

```

1 start = 0
2 end = 256
3 size = 256
4 lung = len(list_images)
5 # list_out = []
6 preds = []
7 print("lung ",lung)
8
9 for i in range(lung):

```

```

10    print("i ", i)
11    print("start ", start)
12    print("end ", end)
13    if end < lung:
14        for img in list_images[start:end]:
15            pixel_values = feature_extractor(images=img, return_tensors="pt").pixel_values
16                pixel_values = pixel_values.to(device)
17                output = mod_caption.generate(pixel_values)
18                pred = tokenizer.batch_decode(output, skip_special_tokens=True)
19                pred_string = " ".join(map(str, pred))
20                preds.append(pred_string)
21                start = end
22                end += size
23        else:
24            for img in list_images[start:lung]:
25                pixel_values = feature_extractor(images=img, return_tensors="pt").pixel_values
26                    pixel_values = pixel_values.to(device)
27                    output = mod_caption.generate(pixel_values)
28                    pred = tokenizer.batch_decode(output, skip_special_tokens=True)
29                    pred_string = " ".join(map(str, pred))
30                    preds.append(pred_string)
31        break

```

Codice 5.8. Generazione delle caption.

Il clustering delle immagini viene effettuato solo con HDBSCAN e le label dei cluster derivano dalla concatenazione dei testi e delle caption delle immagini dei tweet che ne fanno parte. Le immagini vengono mostrate poi suddivise per cluster (codice 5.9).

```

1 def get_clusters_img(df):
2 unilab = unique_labels(df, 0)
3
4 for l, k in zip(unilab, keywords):
5     df_aux = df["img"].loc[df[0] == l]
6     # df_aux = pd.DataFrame(df["img"].loc[df[0] == l])
7     n = len(df_aux)
8     n_row=int(math.sqrt(n))
9     n_col=int(math.sqrt(n))
10    f, axs = plt.subplots(n_row, n_col, figsize=(16, 16))
11    axs = axs.flatten()
12    f.suptitle(f"\{k\}", fontsize=16)
13    df_aux.reset_index(inplace=True, drop=True)
14    for img, ax in zip(df_aux, axs):

```

```
15     ax.imshow(img)
16     ax.axis('off')
17     ax.plot()
18
19     plt.savefig(f'{foto}_cluster{1}.png', bbox_inches='tight', facecolor
20 = "white", dpi=150)
21 plt.show()
21 get_clusters_img(df_img_hdbscan)
```

Codice 5.9. Immagini divise per cluster.

Capitolo 6

Analisi dei risultati

In questo capitolo, dai risultati ottenuti applicando gli algoritmi di clustering *k-means*, *DBSCAN* e *HDBSCAN* prima sui testi dei tweet e poi sulle immagini, si cercherà di stabilire quale sia il miglior algoritmo per effettuare questo genere di analisi, spiegando le differenze tra ognuno, per poi riassumere tutti i topic estratti in una tabella per ottenere un andamento degli eventi accaduti in questo anno di conflitto.

6.1 Analisi dei testi

I tweet importati sono relativi al primo anno del conflitto russo-ucraino e sono divisi in tre dataset a seconda dell'intervallo temporale in cui sono stati pubblicati: il primo va dal 24 febbraio 2022 (primo giorno di guerra) al 30 giugno 2022 e conta 1525 tweet, il secondo va dal 1 luglio al 30 novembre e conta 1390 elementi, il terzo va dal 1 dicembre al 24 febbraio 2023 ed è composto da 10318 tweet.

Nei paragrafi che seguono, per ogni intervallo temporale vengono mostrati in ordine: lo *scatter plot* rappresentante l'insieme dei punti di partenza, i rispettivi grafici ottenuti applicando gli algoritmi di clustering descritti, ognuno con annessa una legenda contenente le keywords del relativo cluster.

Nel paragrafo 6.1.1 vengono fatte delle considerazioni (applicabili anche nei casi successivi) sulla scelta del migliore algoritmo per effettuare questo genere di analisi, vedendo come dai cluster e delle wordcloud si possano ricavare i principali topic in questo anno di guerra.

I criteri su cui si basa la scelta del migliore algoritmo sono:

- **correttezza dei cluster:** quanto correttamente vengono raggruppati tra loro elementi simili e quindi quanto i cluster sono affidabili;

- **scelta dei parametri:** la scelta dei parametri gioca un ruolo fondamentale perché valori poco accurati possono condurre a risultati inutili o sbagliati. Si desidera ovviamente che l'algoritmo sia robusto a cambiamenti di parametri e che la scelta sia la più semplice e intuitiva possibile;
- **cluster stabili:** eseguendo più volte l'algoritmo su un dataset con gli stessi parametri, si desidera che il risultato sia sempre lo stesso, cosa che non è sempre scontata;
- **prestazioni:** aumentando le dimensioni del dataset, è bene che le prestazioni non ne risentano particolarmente.

6.1.1 Collezione testi Febbraio - Giugno 2022

Prima di applicare gli algoritmi è necessario stabilire i parametri da utilizzare. Kmeans prevede la scelta del solo *n_cluster* (numero di cluster da ricercare), ma per quanto possa risultare banale, in realtà non lo è affatto. Infatti la scelta manuale di *n_cluster* prevede la conoscenza a priori del dataset, che nel caso trattato non è possibile averla, e un valore sbagliato può portare alla creazione di cluster fuorvianti.

DBSCAN necessita di *min_samples* e *eps* e, tra i tre algoritmi utilizzati, è probabilmente il più difficile da mettere a punto. Infatti, sebbene per *min_samples* la scelta può essere abbastanza semplice (un valore piccolo implica cluster più numerosi e selettivi e viceversa), stabilire un *eps* corretto è assai più complesso e per nulla intuitivo, il quale dipende sia dalla scelta di *min_samples* che dalla metrica per misurare la distanza tra i punti.

La stima di *n_cluster* per kmeans e *eps* di DBSCAN viene effettuata utilizzando rispettivamente le funzioni descritte nel capitolo 5 nelle sezioni 5.1 e 5.3. Esse producono le curve mostrate nelle figure 6.2 e 6.3 da cui, estraendo il valore in prossimità del gomito, si ricavano i parametri desiderati.

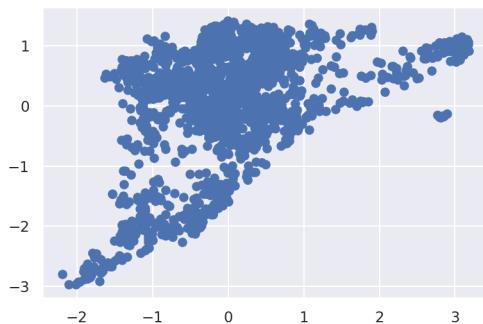


Figura 6.1. Scatter plot tweet febbraio-giugno.

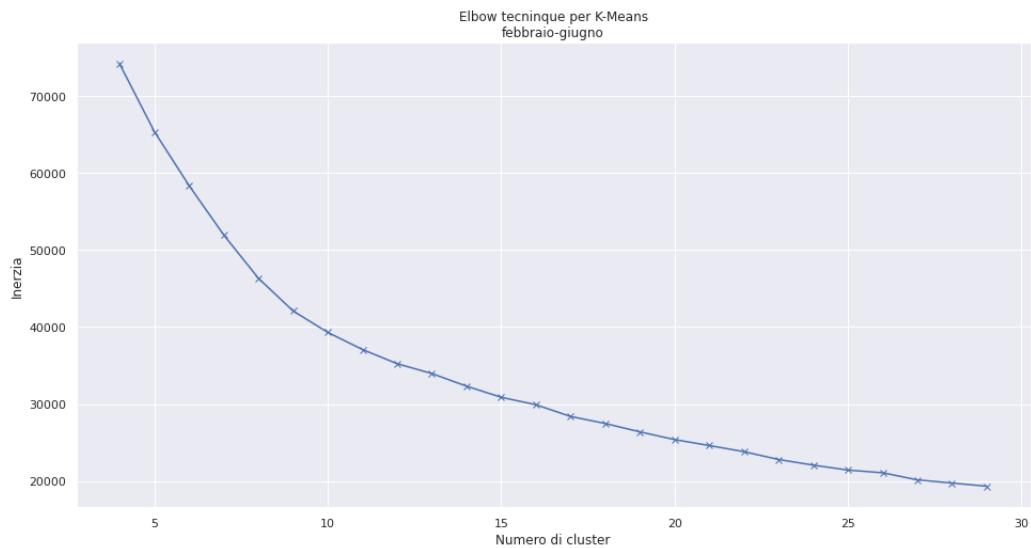


Figura 6.2. Elbow technique per k-means.

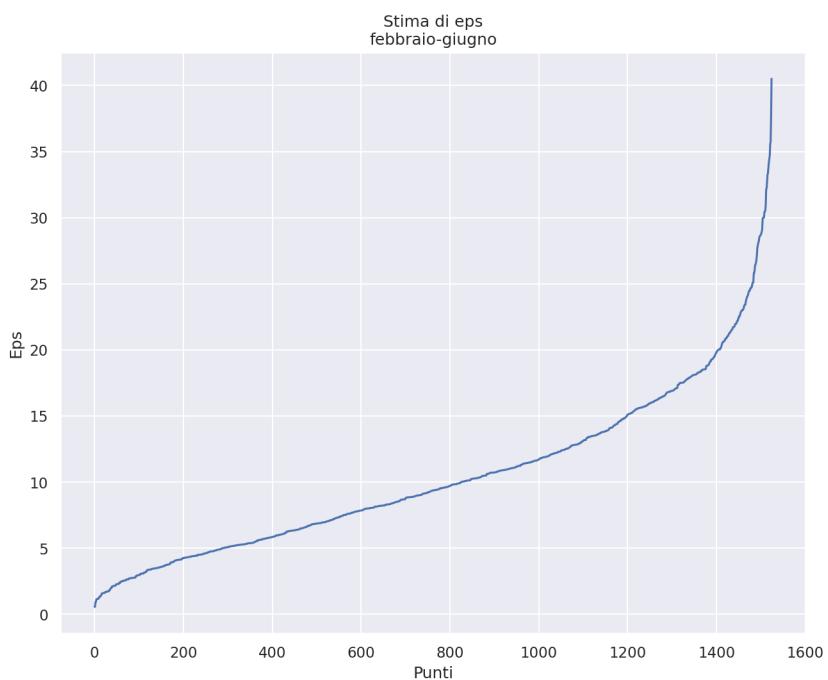


Figura 6.3. Scelta di eps per DBSCAN.

Questo procedimento è comunque molto approssimativo e richiede un'analisi manuale da parte del programmatore, che deve osservare i grafici, estrarre i valori all'apparenza migliori e non sempre evidenti, per poi eseguire diversi test per dimostrarne la validità.

Per HDBSCAN i parametri sono più intuitivi da scegliere e la funzione *hdbScan_validity* (sezione 5.5) permette di automatizzare il procedimento: essa non produce un risultato grafico da cui stimare i grafici, bensì restituisce direttamente i valori che garantiscono un clustering più accurato grazie a un'analisi numerica.

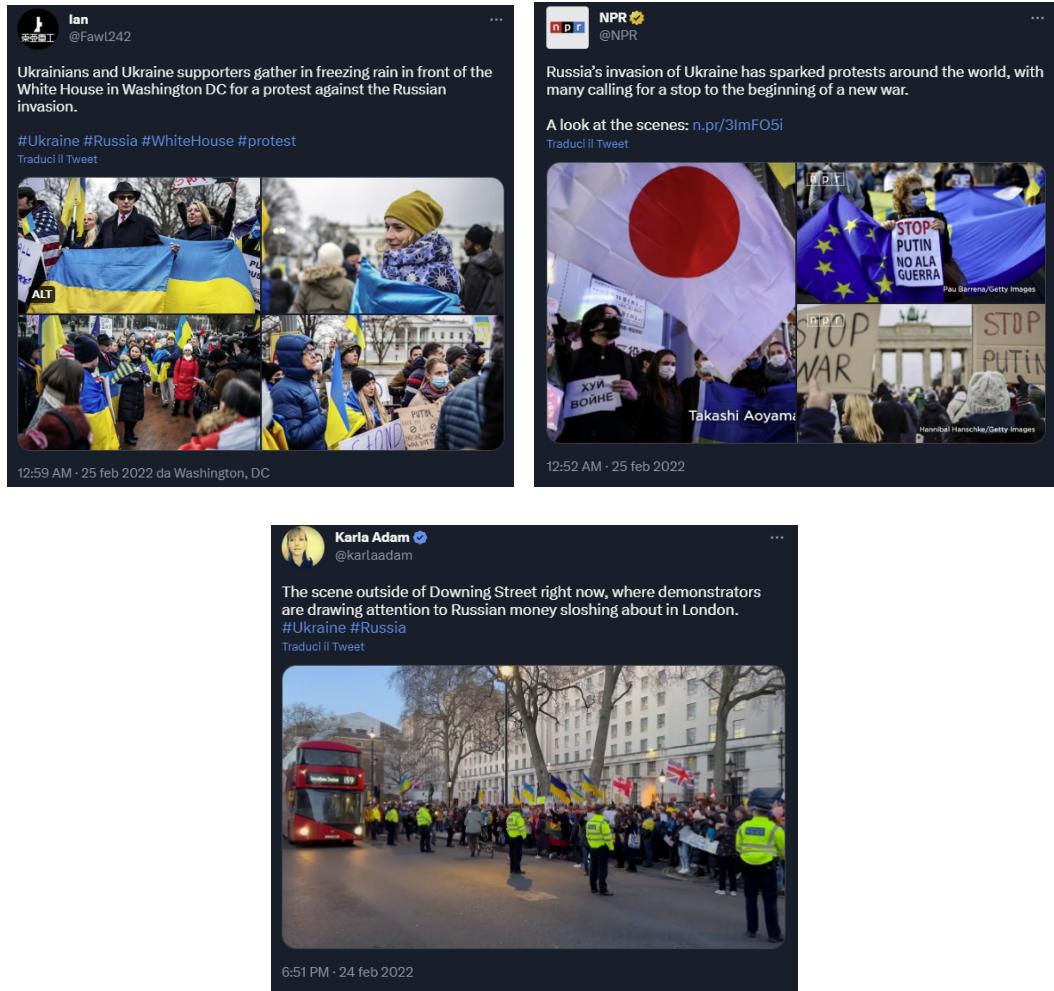


Figura 6.4. Tweet raggruppati in modo simile da k-means e HDBSCAN.

In termini di prestazioni, se si tratta di dataset piccoli come in questo caso di circa 1500 elementi, i tre algoritmi si comportano allo stesso modo impiegando circa 1 secondo per generare i cluster. Quando si opera su dataset grandi (dai 10000 elementi in su), k-means è il più veloce complessivamente impiegando 2 secondi (anche a parità di cluster trovati), contro gli oltre 12 di DBSCAN e di HDBSCAN, tuttavia

perde in accuratezza. Infatti, come già descritto nel paragrafo 3.5.1, k-means non ha il concetto di *outliers* e quindi ogni elemento viene inserito nel cluster con centroide più vicino ad esso, anche se logicamente il testo del tweet non dovrebbe farne parte.

A titolo di esempio si considerino due cluster con lo stesso nome da k-means e HDBSCAN (figure 6.7 e 6.12) come *protest invasion ukraine russia* e da cui si estraggano dei tweet per confrontarli. Tramite un'analisi manuale si vede che il cluster di kmeans contiene sia tutti i tweet presenti nel corrispettivo di HDBSCAN (e quindi scelti correttamente), sia altri che invece non rispecchiano l'identità del cluster.

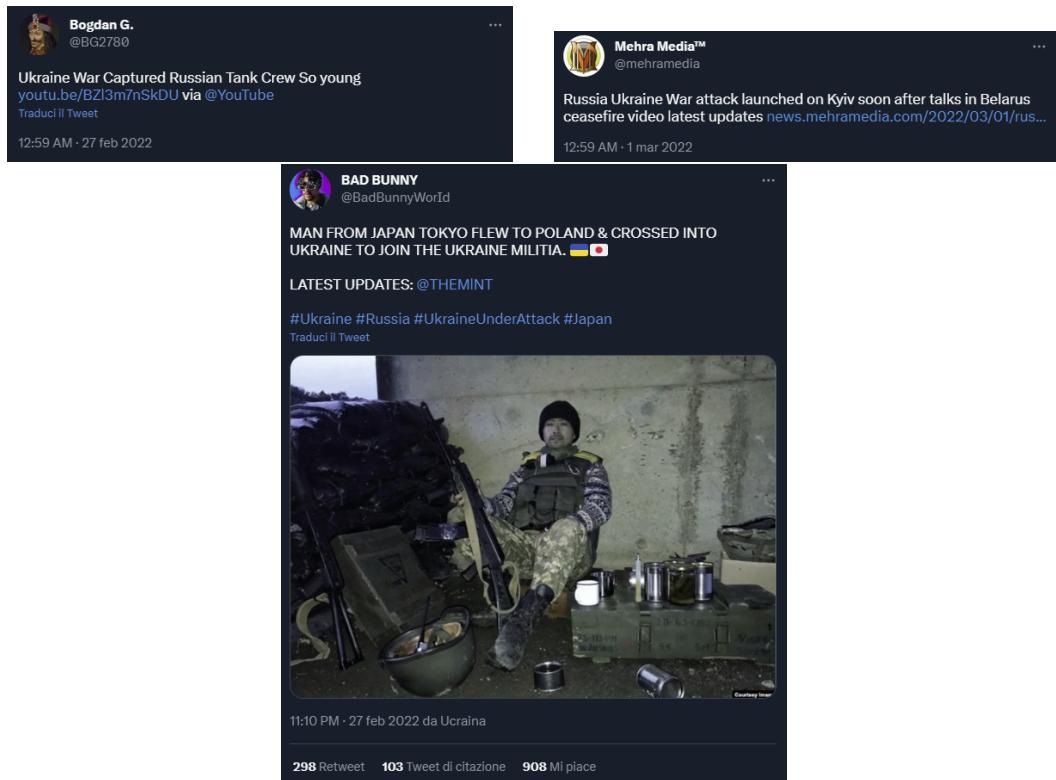


Figura 6.5. Tweet raggruppati erroneamente da k-means.

Come si può vedere in figura 6.4, i tweet in comune contengono parole presenti nel nome del cluster come *protests*, *ukraine*, *russia*, o comunque riguardanti il tema della protesta come *demonstrators* (manifestanti).

Guardando la figura 6.5, invece, ci si accorge come il topic del tweet non rispecchi quello del cluster (uno parla della cattura di alcuni soldati russi, uno di un attacco missilistico, uno dell'arruolamento di un soldato giapponese nell'esercito ucraino). Da ciò si evince che l'accuratezza di k-means ha dei limiti: se da un lato riesce a raccogliere con buona approssimazione tra loro tweet riguardanti uno stesso topic, dall'altro il contenuto del cluster viene "sporcato" con tweet che invece non vi hanno



Figura 6.6. Tweet identificati come *outliers* da HDBSCAN ma non da k-means.

a che fare. Nel caso trattato fortunatamente il significato del cluster non si è perso, ma può capitare che più tweet "sbagliati" raggruppati insieme ad altri diano un cluster privo di significato.

Un'altra limitazione di k-means riguarda il fatto di non avere il concetto di *outliers*, posseduto invece da DBSCAN e HDBSCAN, e questo obbliga l'algoritmo a inserire in cluster tweet che dovrebbero essere scartati perché privi di un contenuto utile ai fini dell'analisi. È il caso di tweet già molto brevi oppure costituiti da molte *stopwords*, link, menzioni, hashtag, simboli o numeri che vengono filtrati a priori (vedi paragrafo 7.6) lasciando un testo estremamente corto da essere irrilevante, oppure semplicemente il loro contenuto non sia adatta a nessun cluster.



Figura 6.7. K-means con $n_cluster = 15$ e $silhouette_score = 0.3218800127506256$.

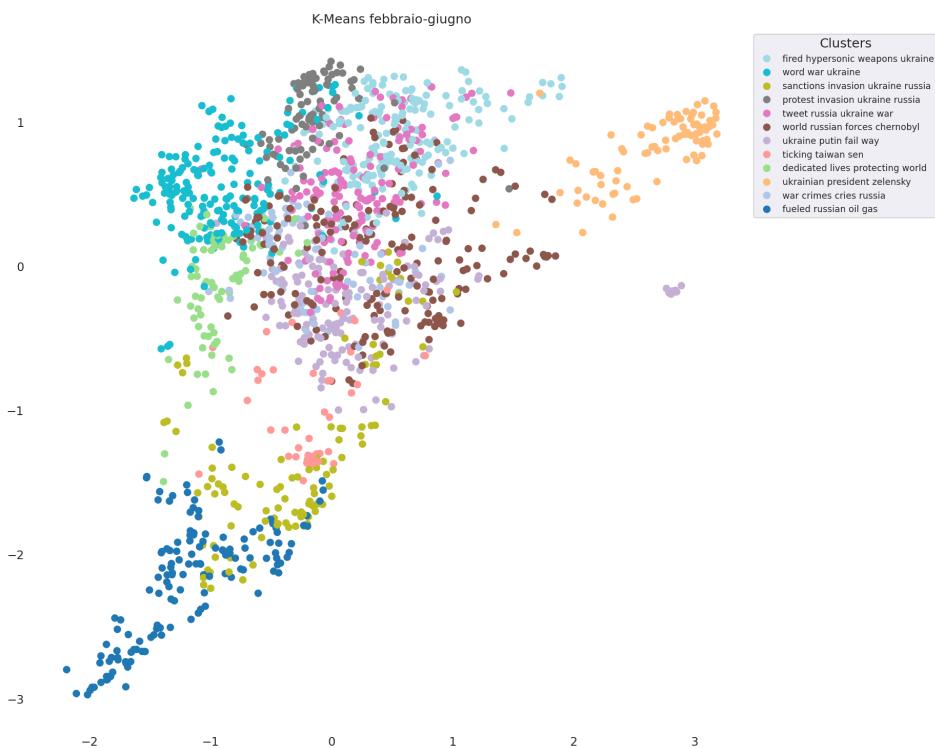


Figura 6.8. K-means con $n_cluster = 15$ e $silhouette_score = 0.32395192980766296$.

I tweet in figura 6.6 vengono classificati come *outliers* HDBSCAN dato che, eliminando stopwords e hashtag, i loro testi non hanno informazioni interessanti (rispettivamente i testi saranno ridotti a "russia ropes", "demographics end" e "russia video war"), mentre kmeans li inserisce rispettivamente nei cluster *war ukraine putin global*, *word war ukraine* e *tweet russia putin global*.

Un altro punto a favore di HDBSCAN sta nel fatto che, sebbene ad una prima operazione di clustering non si riescano a raggruppare tutti i tweet su un certo topic, tentativi successivi solo sui tweet marchiati come *outliers* possono aiutare a estrarre ulteriori cluster (si veda la figura 6.13).

Per quanto riguarda la stabilità utilizzando gli stessi parametri per effettuare diversi test, sia DBSCAN che HDBSCAN la rispettano, mentre non è lo stesso per kmeans. Infatti, eseguendo più test utilizzando lo stesso $n_cluster$, i cluster finali possono essere diversi, di conseguenza ne risente la veridicità dell'analisi. Nelle figure 6.7 e 6.8 vengono mostrate due esecuzioni di kmeans sullo stesso dataset con $n_cluster = 12$ e, confrontando i cluster generati, si nota come differiscano sia alcuni cluster che l'indice della silhouette.

Inoltre HDBSCAN è anche stabile a variazioni (contenute) di parametri, al contrario di DBSCAN: una volta aver selezionato l'intervallo di *eps* e *min_samples* ipoteticamente più corretti, si applica l'algoritmo più volte sul dataset incrementando

di volta in volta i parametri provando varie combinazioni. Infine, comparando i risultati e il metodo della silhouette, si sceglie la coppia di parametri migliore. Anche una variazione di poche unità può cambiare di molto il clustering, per esempio abbassando il numero di cluster trovati o marchiando molti più punti come outliers (vedi figure).



Figura 6.9. DBSCAN con $\text{eps} = 29, \text{min_samples} = 20$ con 10 cluster generati, 328 outliers e $\text{silhouette_score} = 0.2013704478740692$.

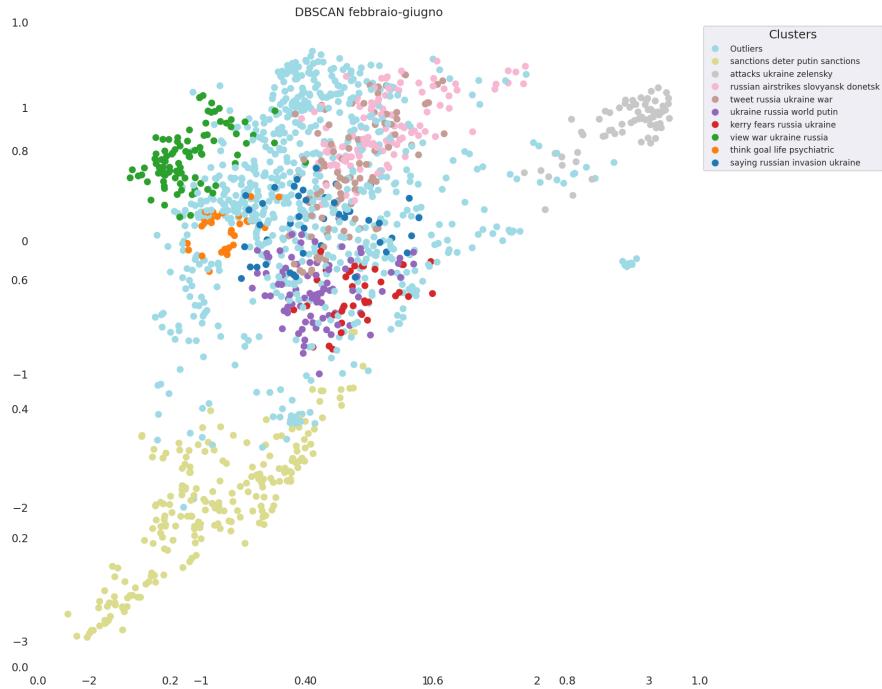


Figura 6.10. DBSCAN con $\text{eps} = 29, \text{min_samples} = 30$ con 10 cluster generati, 730 outliers e $\text{silhouette_score} = 0.1032649427652359$.



Figura 6.11. DBSCAN con $\text{eps} = 20, \text{min_samples} = 20$ con 6 cluster generati, 1249 outliers e $\text{silhouette_score} = -0.08166419714689255$.

Grazie al modo riportato in precedenza con cui vengono scelti i parametri di HDBSCAN, è possibile applicare l'algoritmo ricorsivamente sugli outliers generati ad ogni passata dell'algoritmo finché non vengono più generati cluster oppure quando non ci sono più outliers. Il clustering degli outliers non dà sempre risultati precisi, dato che comunque si opera su dati precedentemente scartati, ma risulta utile per vedere se è possibile ricavare ulteriori informazioni.

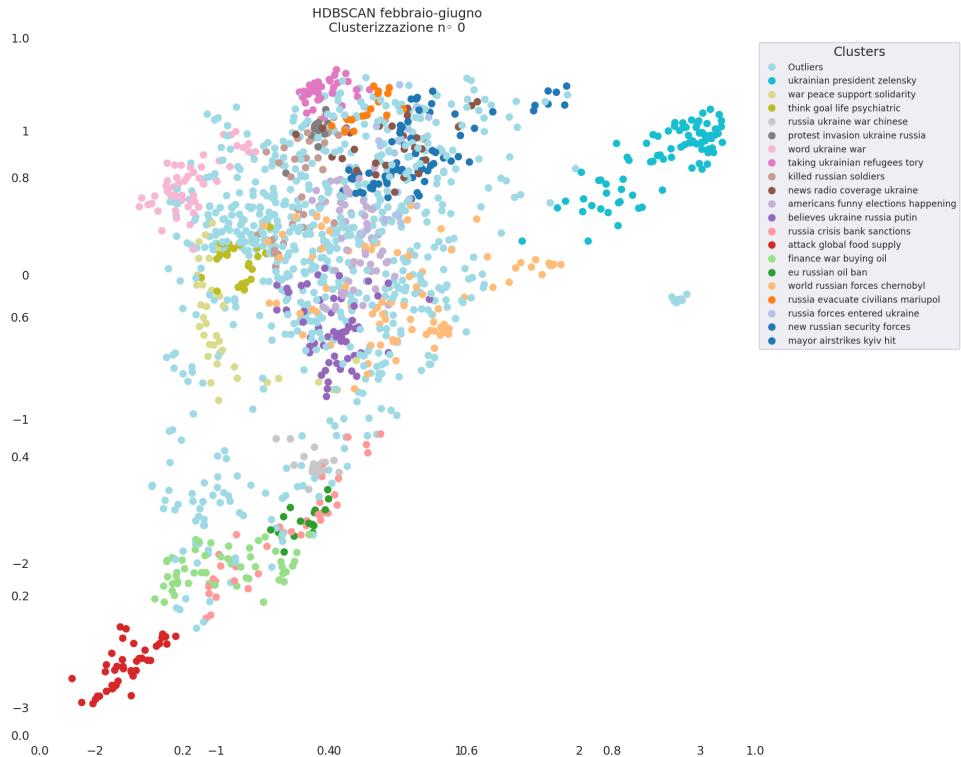


Figura 6.12. HDBSCAN con $\text{min_samples} = 5$, $\text{min_cluster_size} = 18$ e 20 cluster generati, 703 outliers.

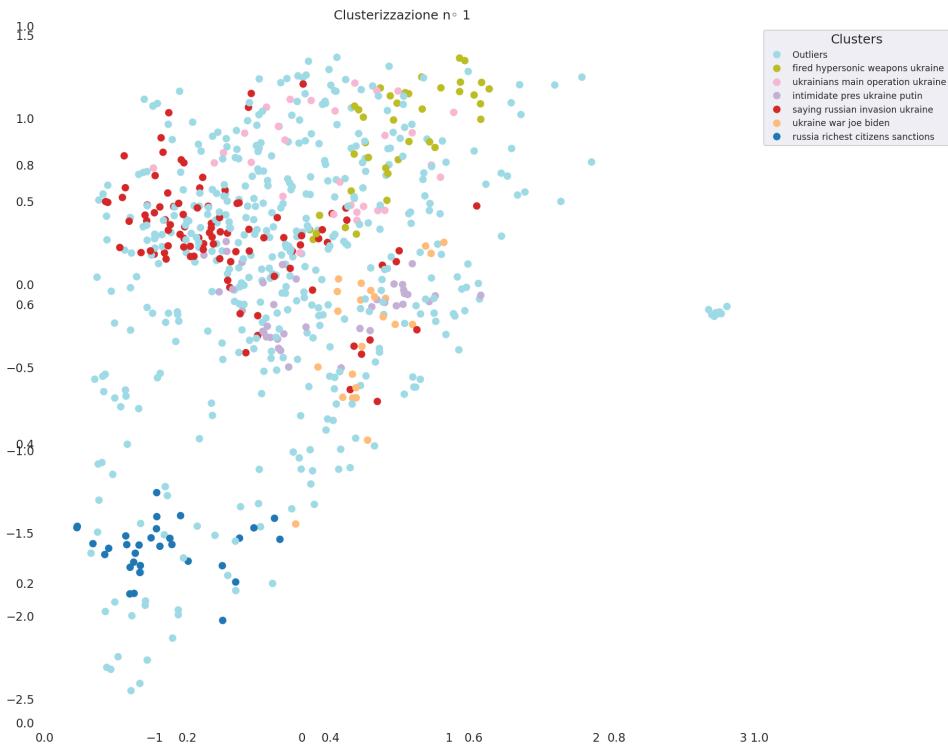


Figura 6.13. HDBSCAN febbraio-giugno.

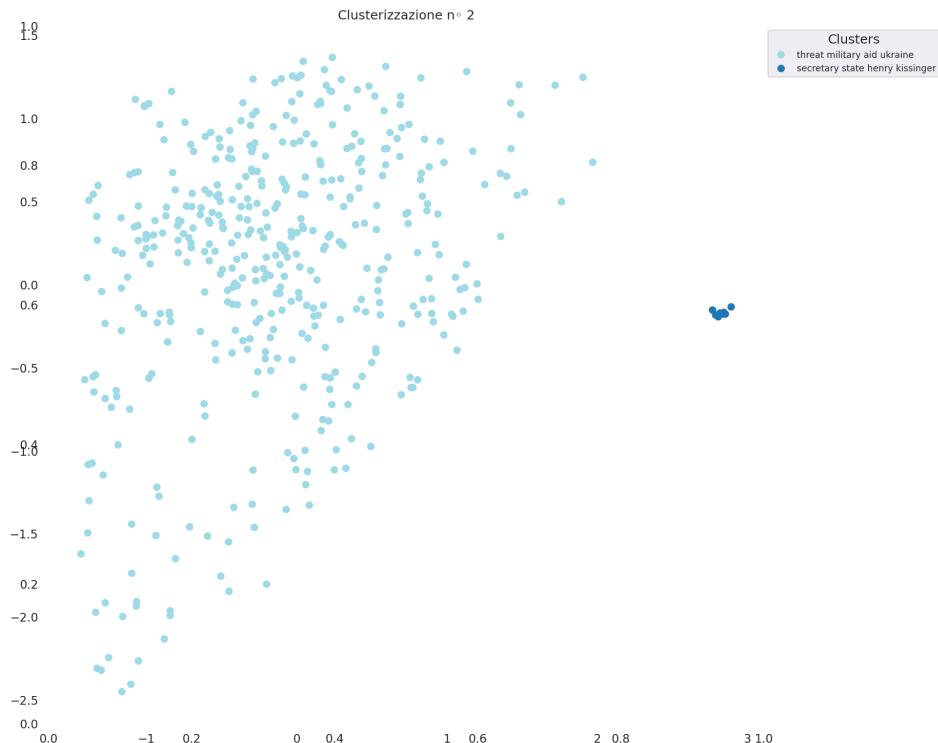


Figura 6.14. HDBSCAN febbraio-giugno.

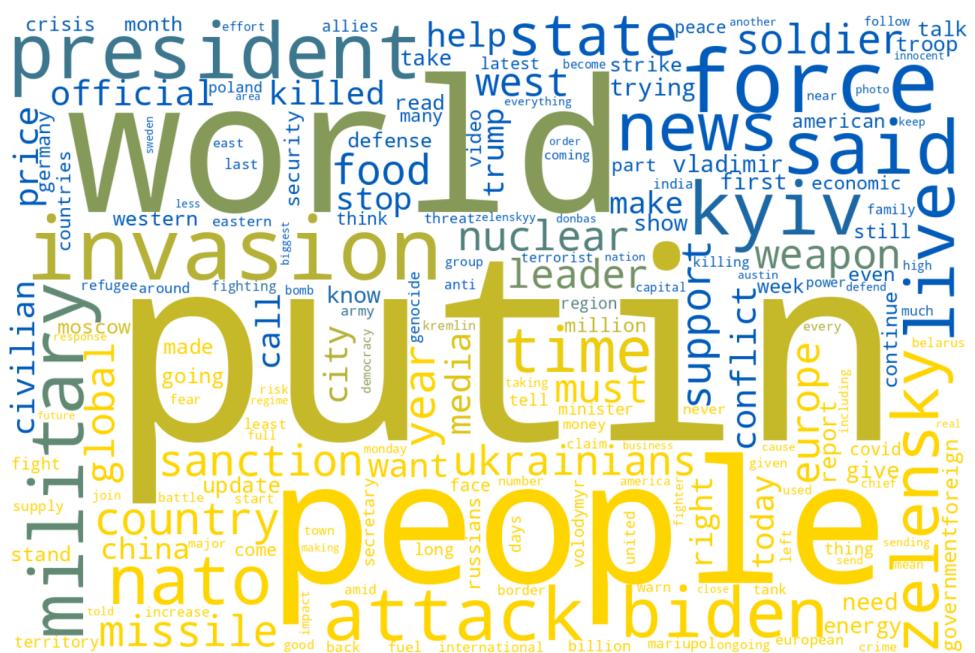


Figura 6.15. Wordcloud dei tweet da febbraio a giugno.

6.1.2 Collezione testi Luglio - Novembre 2022

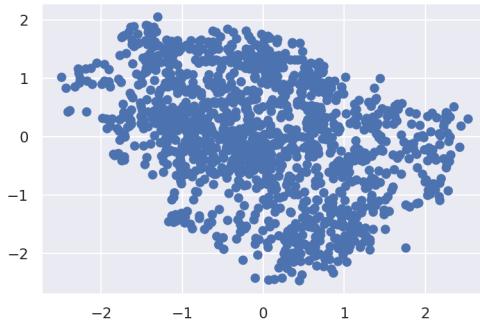


Figura 6.16. Scatter plot tweet luglio-novembre.

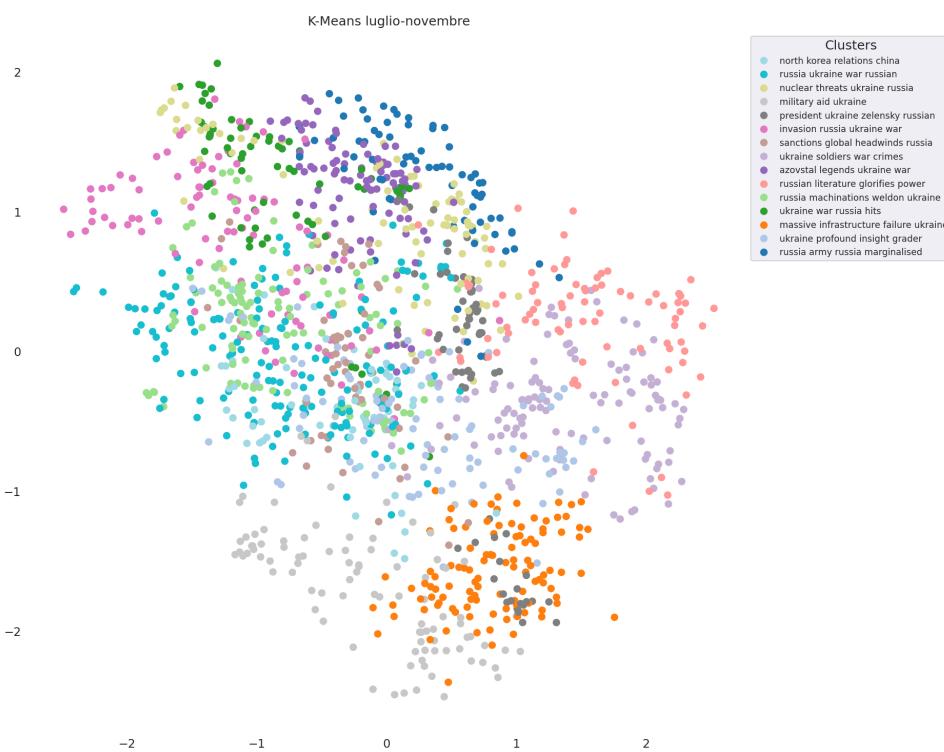


Figura 6.17. K-means con $n_cluster = 15$ e $silhouette_score = 0.3039248287677765$.

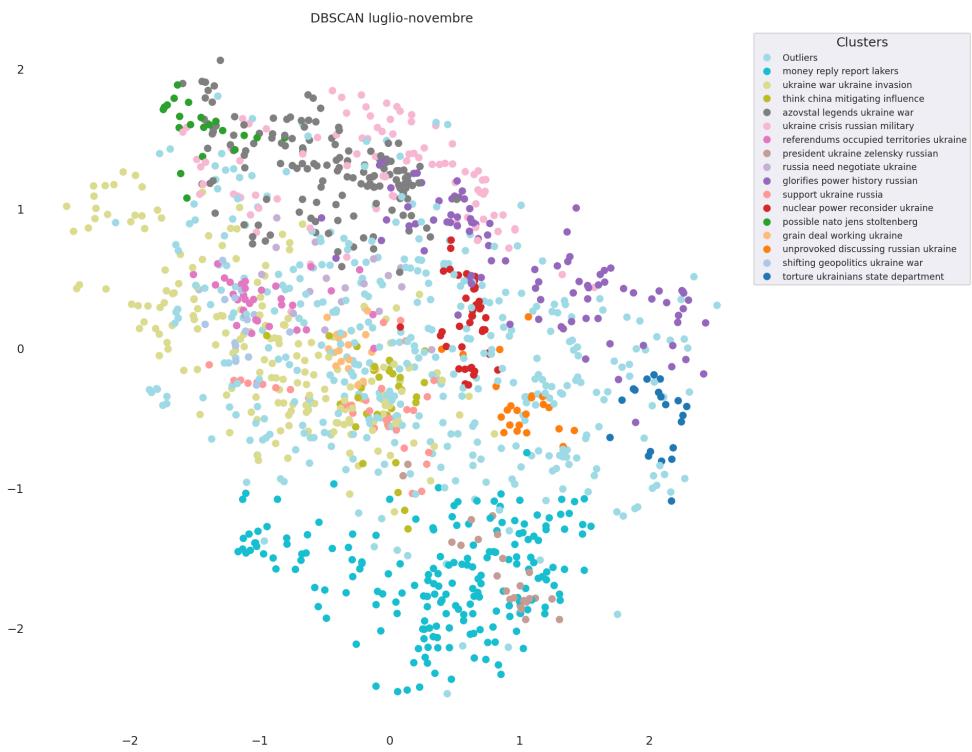


Figura 6.18. DBSCAN con $\text{eps} = 30$, $\text{min_samples} = 20$ e 14 cluster generati, 510 outliers e $\text{silhouette_score} = 0.14053146541118622$.

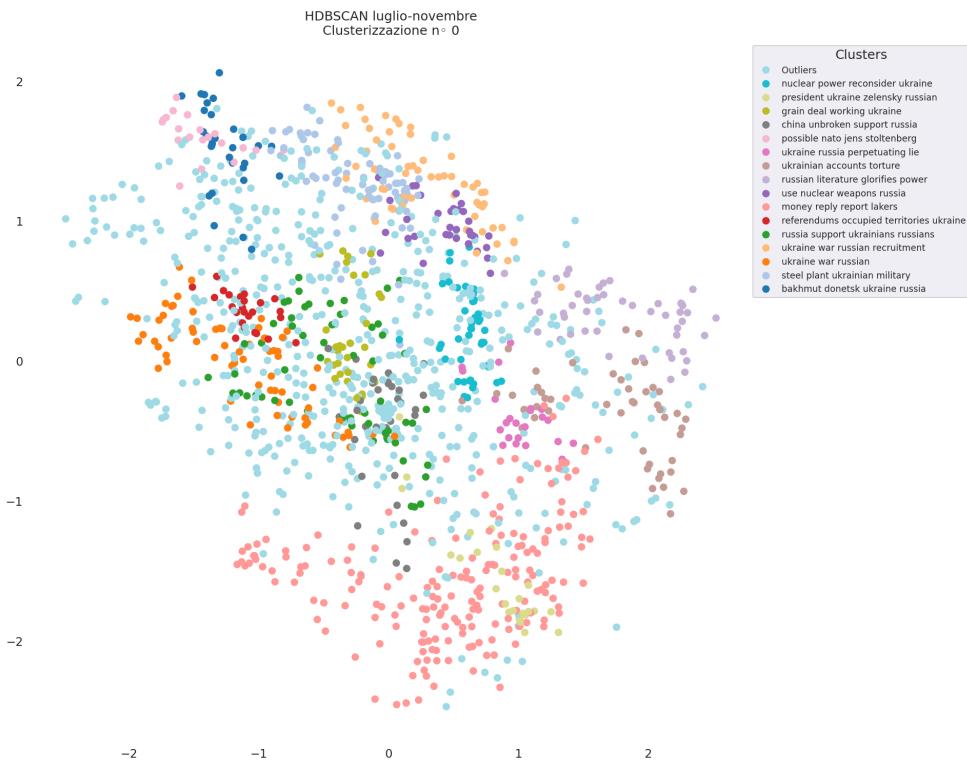


Figura 6.19. HDBSCAN con $\text{min_samples} = 5$, $\text{min_cluster_size} = 10$ e 16 cluster generati, 479 outliers.

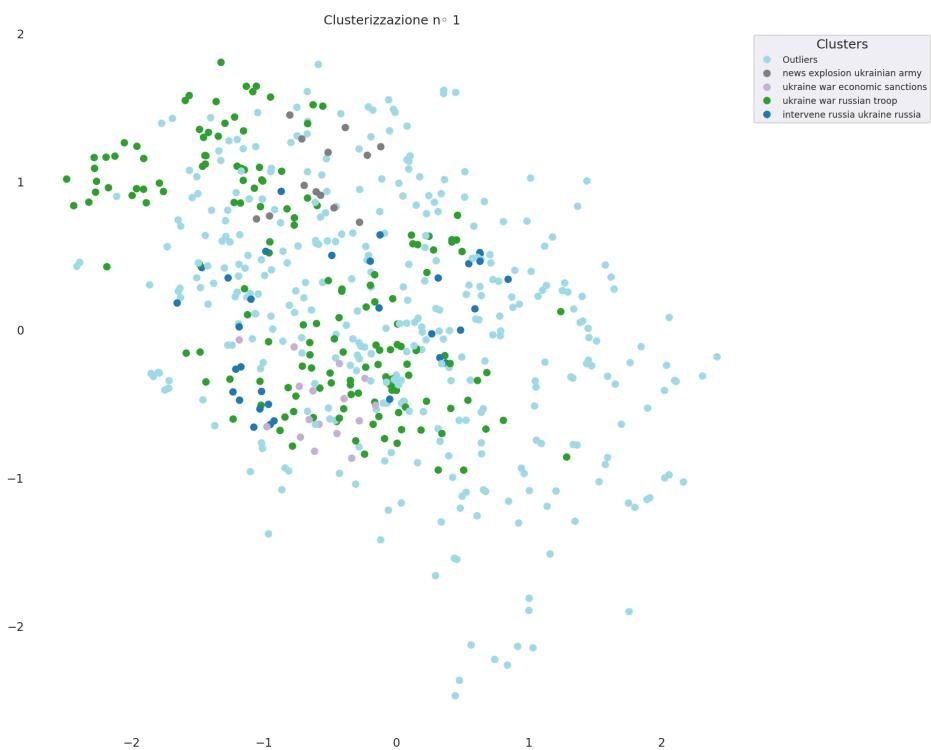


Figura 6.20. HDBSCAN luglio-novembre.



Figura 6.21. Wordcloud dei tweet da luglio a novembre.

6.1.3 Collezione testi Dicembre 2022 - Febbraio 2023

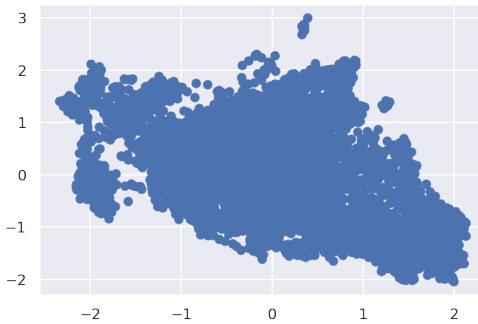


Figura 6.22. Scatter plot tweet dicembre.

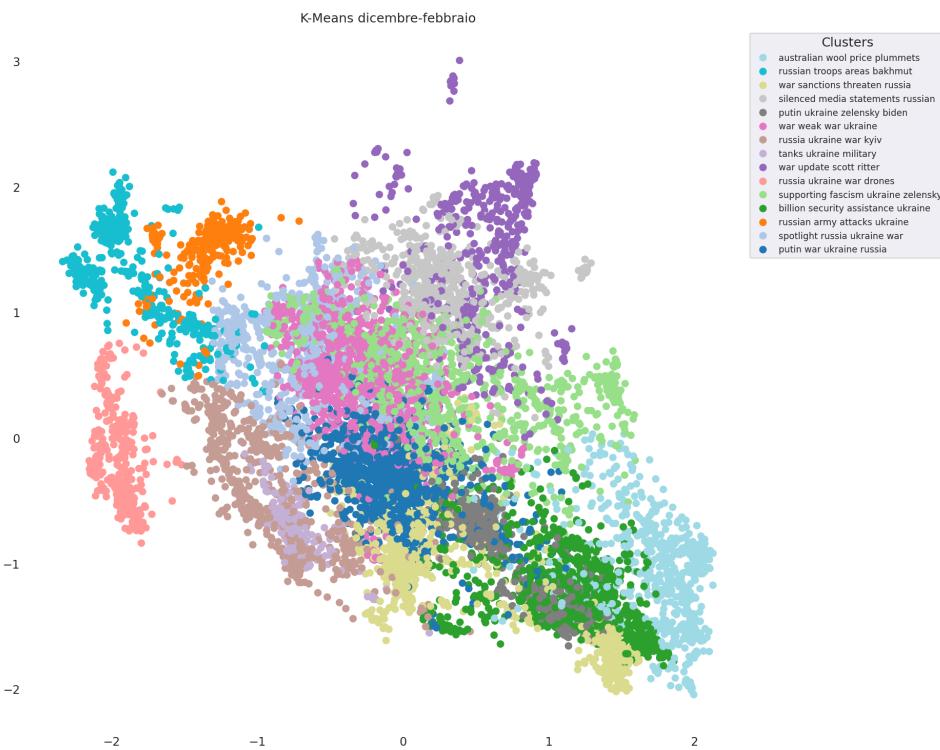


Figura 6.23. K-means con $n_cluster = 15$ e $silhouette_score = 0.3104861080646515$.

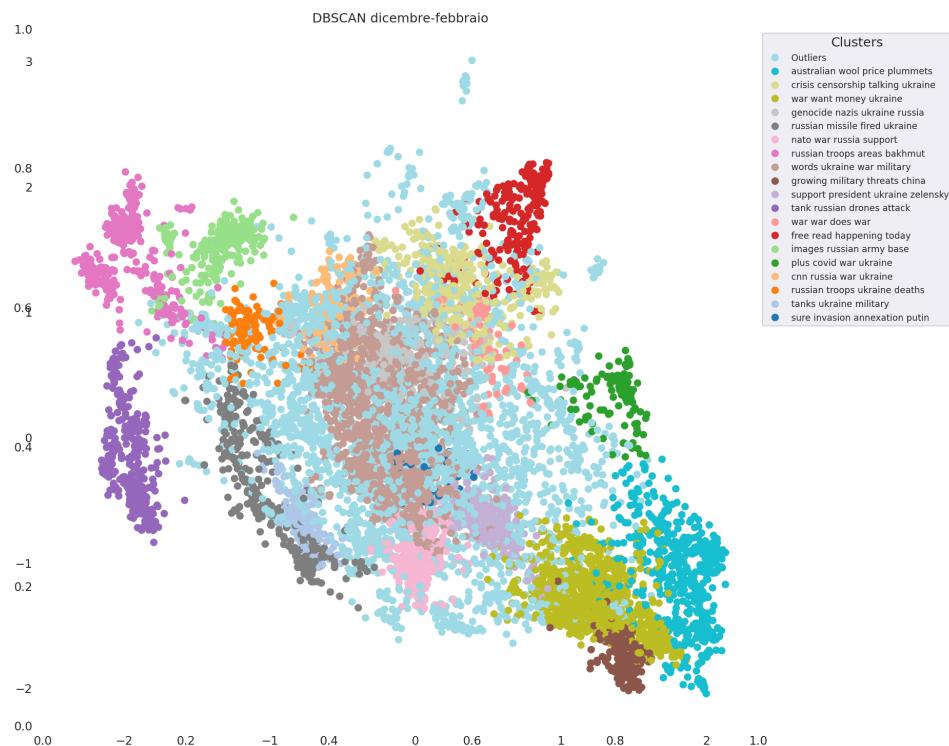


Figura 6.24. DBSCAN con $\text{eps} = 20$, $\text{min_samples} = 60$ e 19 cluster generati, 3187 outliers e $\text{silhouette_score} = 0.13501596450805664$.

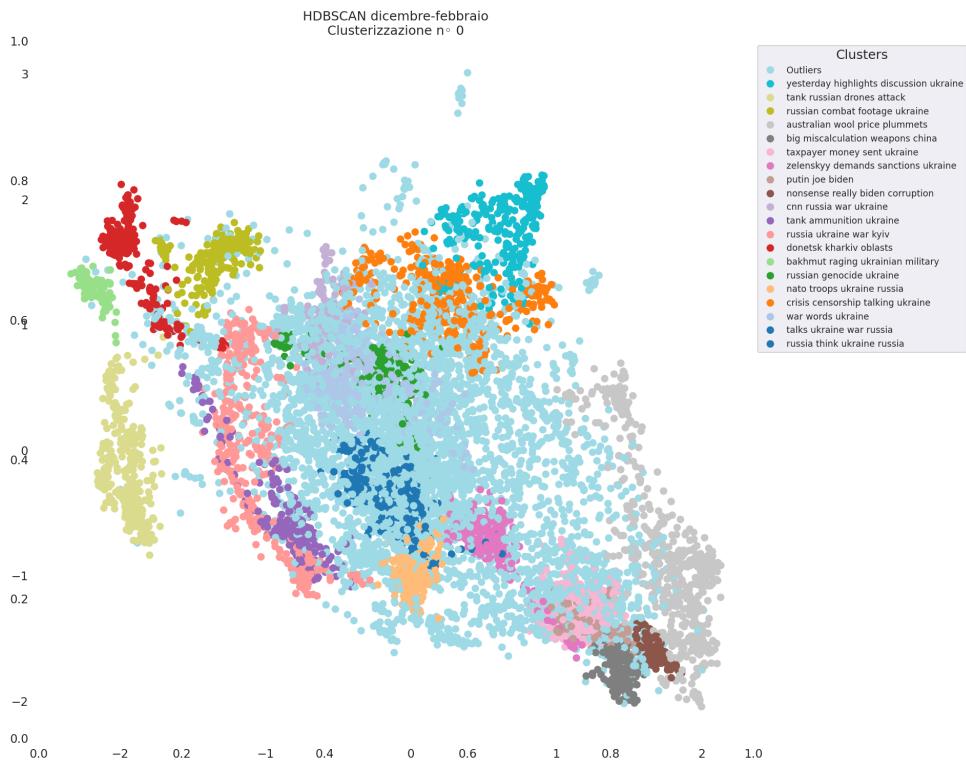


Figura 6.25. HDBSCAN con $\text{min_samples} = 19$, $\text{min_cluster_size} = 26$ e 20 cluster generati, 4836 outliers.

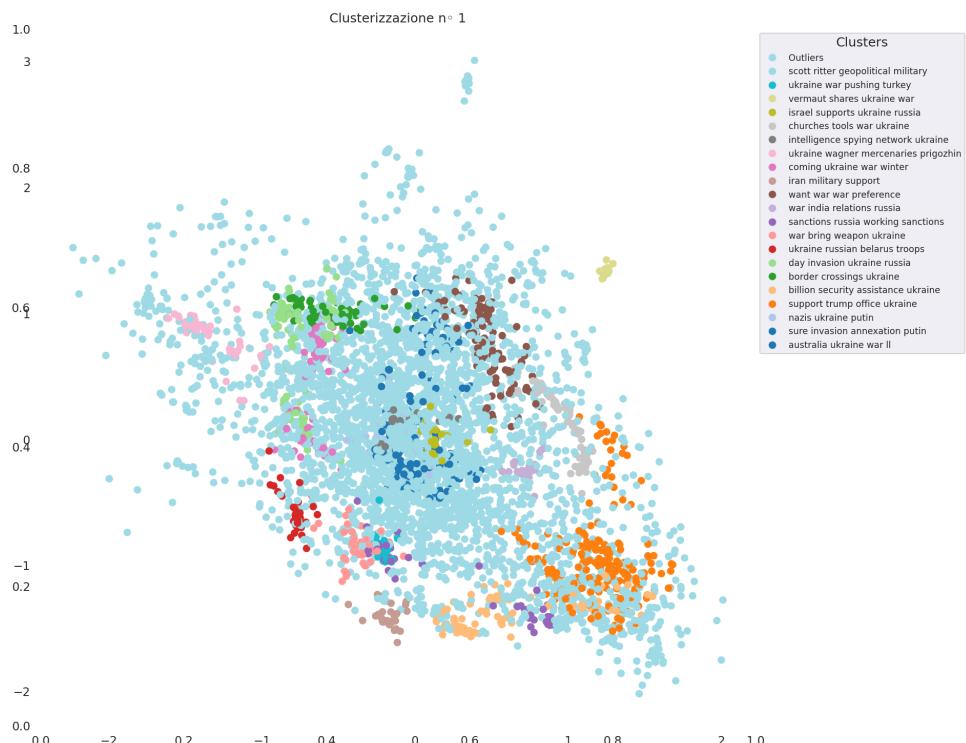


Figura 6.26. HDBSCAN dicembre-febbraio.

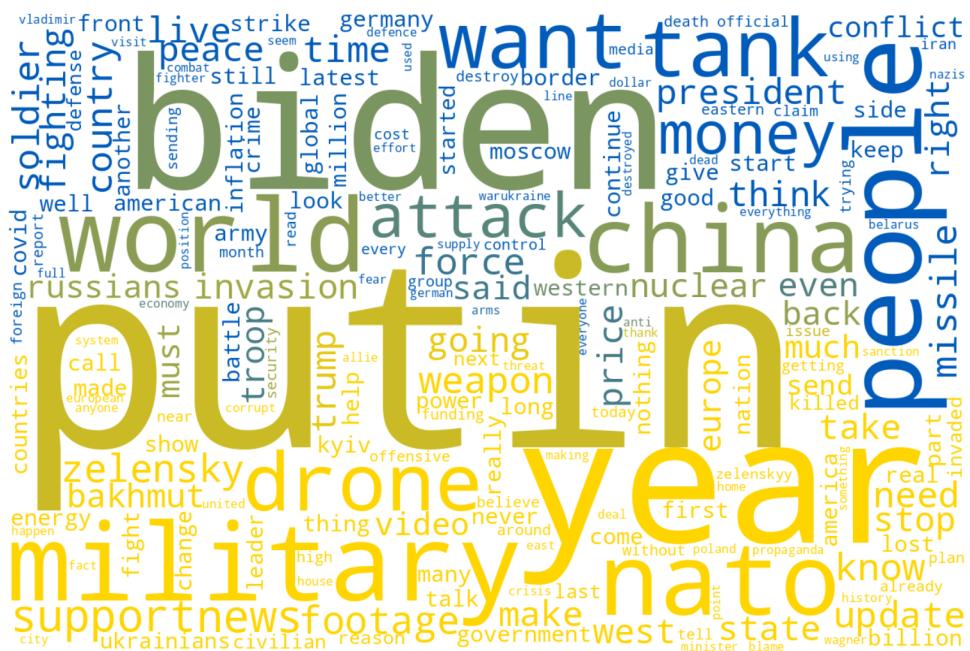


Figura 6.27. Wordcloud dei tweet da dicembre a febbraio 2023.

6.2 Analisi delle immagini

Per lo studio delle immagini, i tweet sono stati divisi in due dataset: uno nell'intervallo temporale da febbraio a novembre composto da 73 tweet, l'altro da dicembre a febbraio 2023 di dimensione 1062.

Date le dimensioni ridotte degli insiemi (al contrario dei dataset dei testi) viene applicato una sola volta HDBSCAN dato che, come si è riscontrato nel paragrafo 6.1, è migliore rispetto a k-means e DBSCAN.

6.2.1 Collezione immagini Febbraio - Novembre 2022

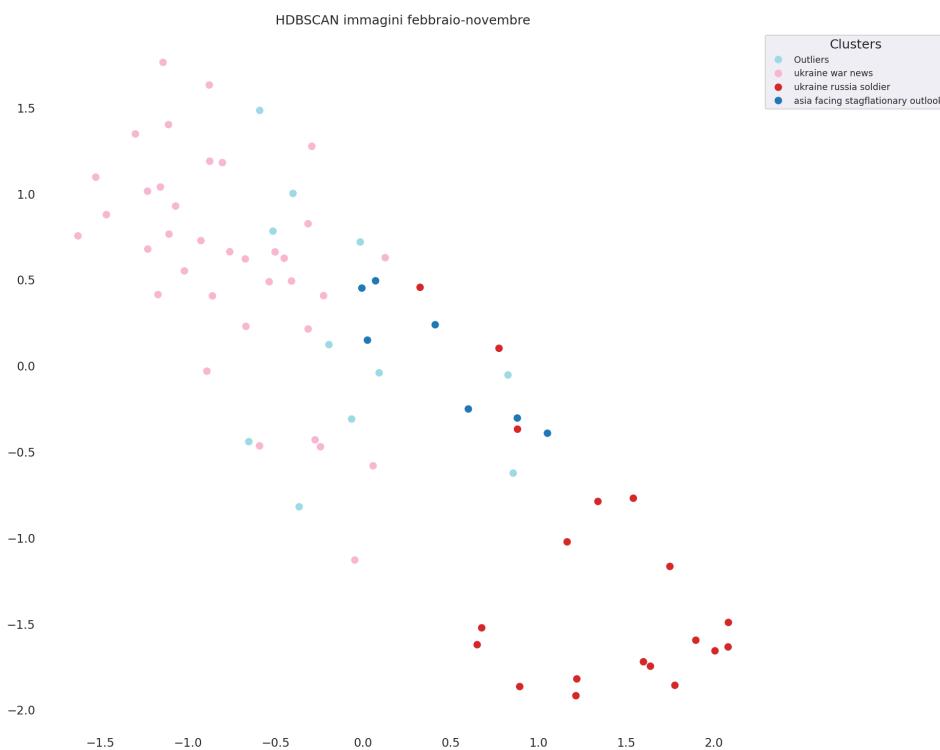


Figura 6.28. HDBSCAN con $\text{min_samples} = 3, \text{min_cluster_size} = 5$ e 4 cluster generati, 11 outliers.

Sulle immagini viene applicato HDBSCAN e si genera uno scatter plot (figura 6.28) in cui le descrizioni dei cluster sono generate estraendo le keywords dai rispettivi testi dei tweet che ne fanno parte concatenati con le *caption* delle immagini. In seguito le immagini vengono mostrate divise per cluster nelle figure seguenti con le rispettive descrizioni del cluster.

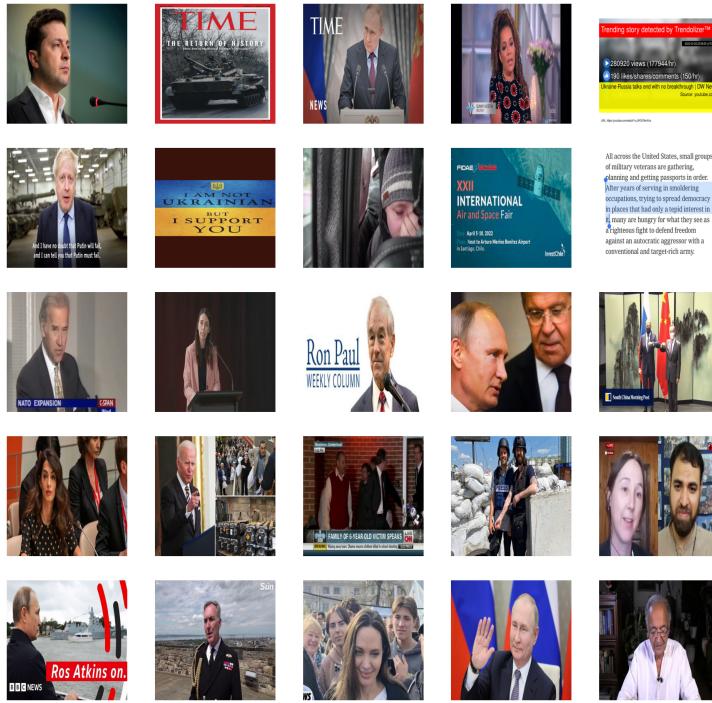


Figura 6.29. Ukraine war news.

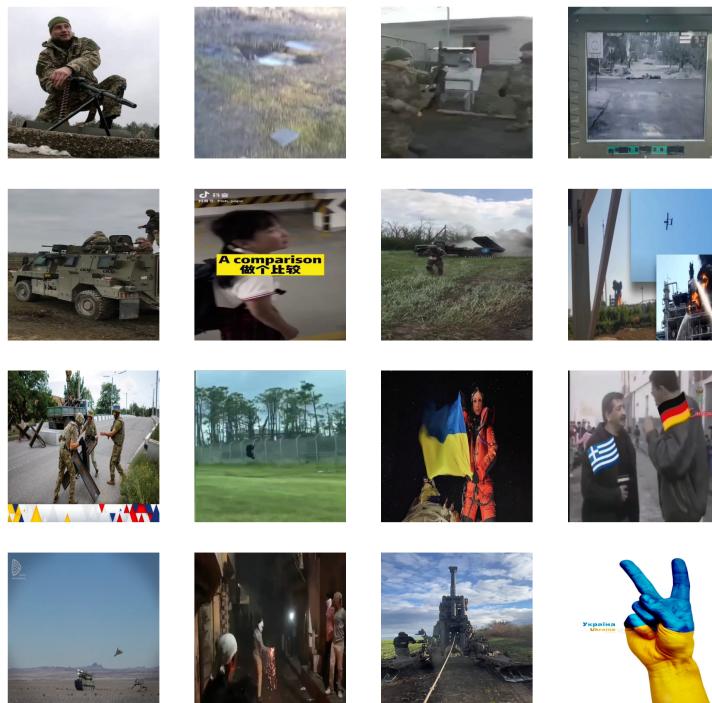


Figura 6.30. Ukraine russia soldier.

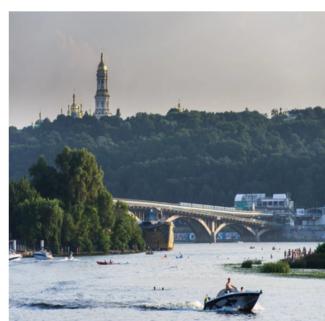


Figura 6.31. Asia facing stagflationary outlook.

6.2.2 Collezione immagini Dicembre 2022 - Febbraio 2023

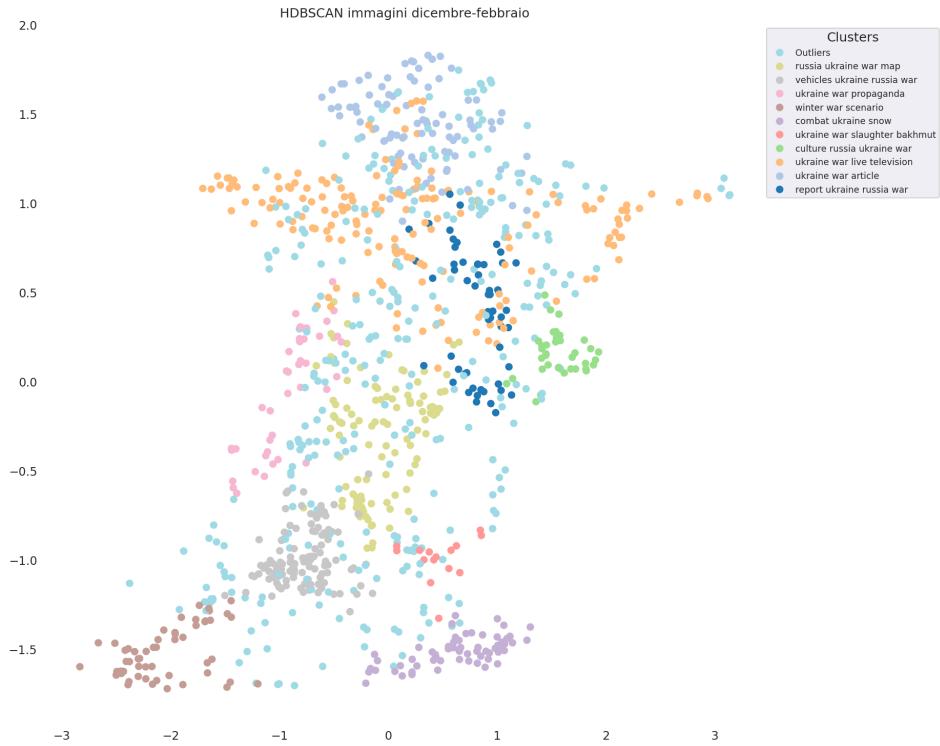


Figura 6.32. HDBSCAN con $\text{min_samples} = 11$, $\text{min_cluster_size} = 14$ e 6 cluster generati, 318 outliers.



Figura 6.33. Russia ukraine war map.

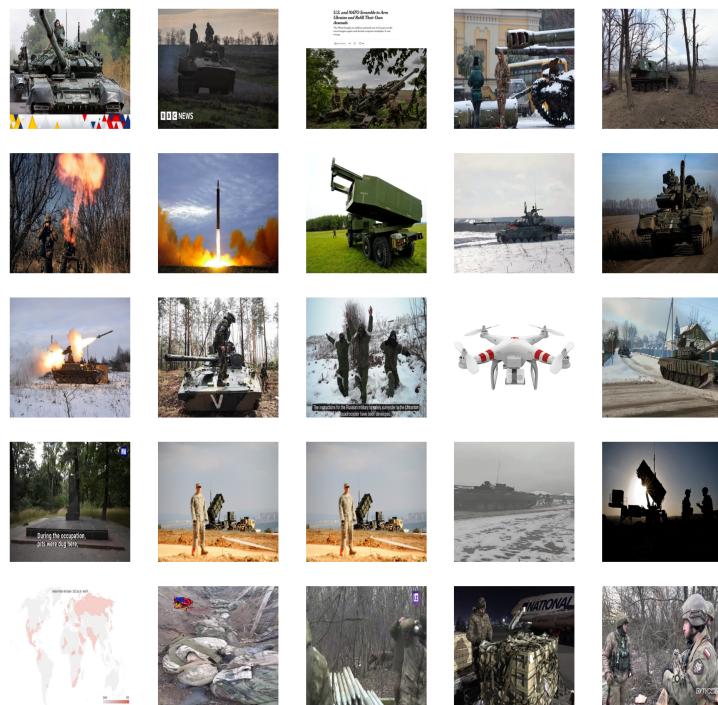


Figura 6.34. Vehicles ukraine russia war.

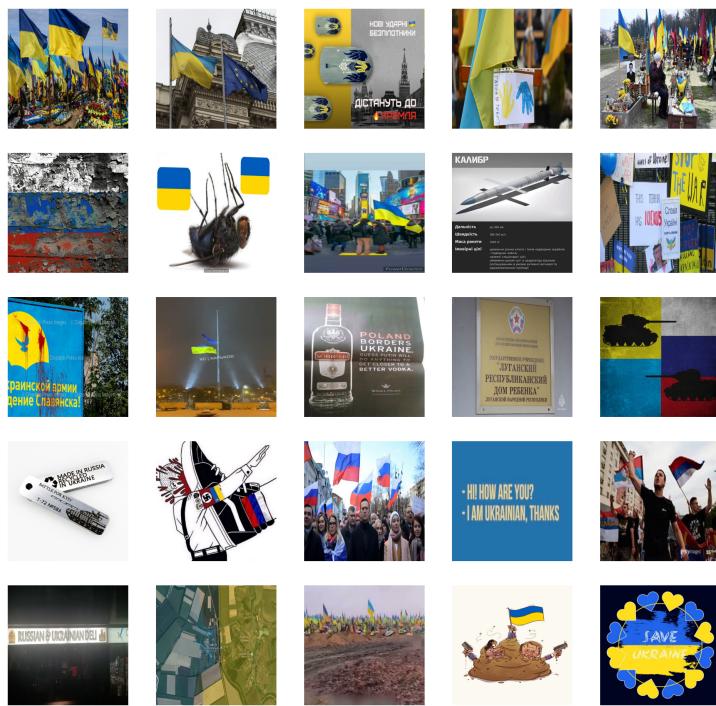


Figura 6.35. Ukraine war propaganda.

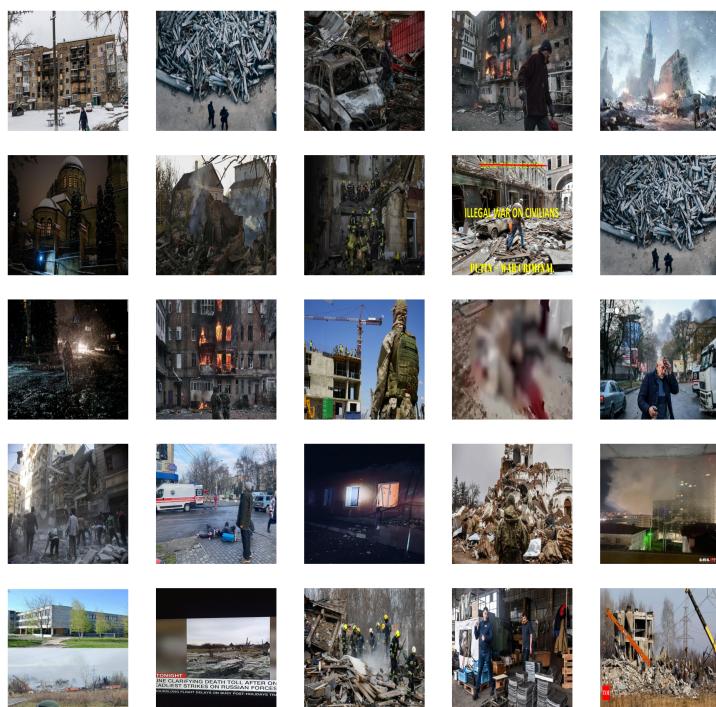


Figura 6.36. Winter war scenario.

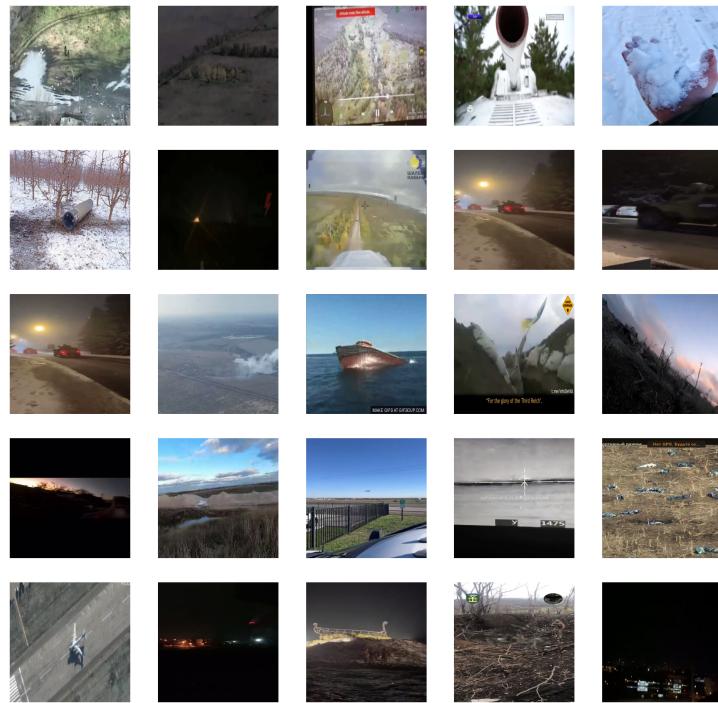


Figura 6.37. Combat ukraine snow.

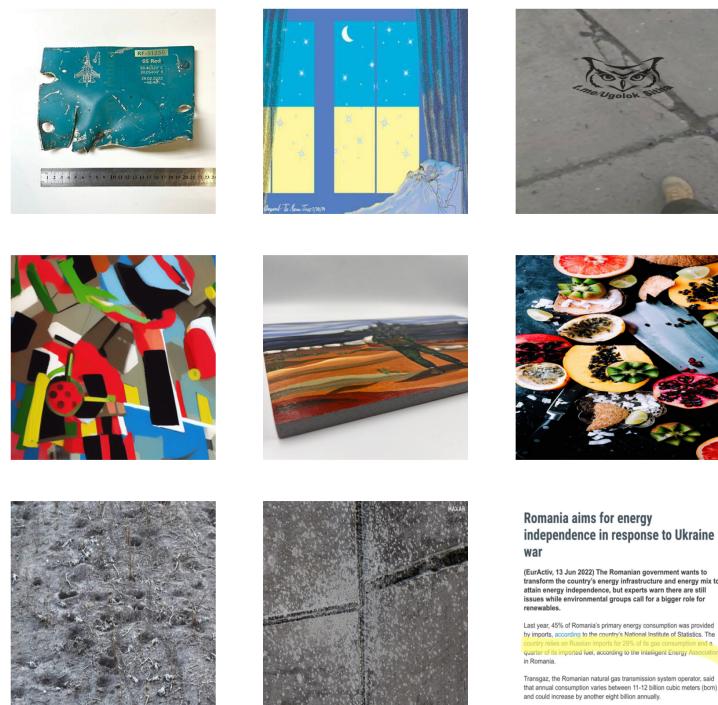


Figura 6.38. Ukraine war slaughter bakhmut.

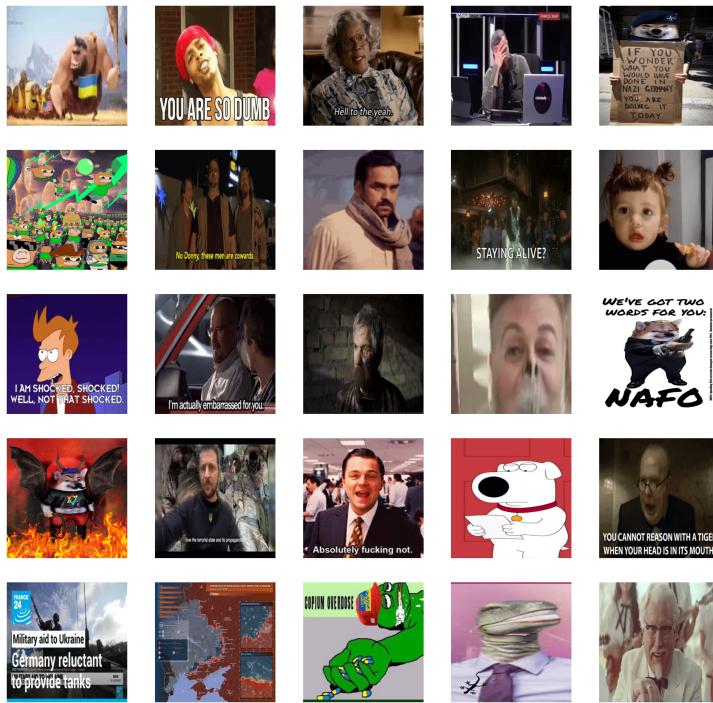


Figura 6.39. Culture russia ukraine war.

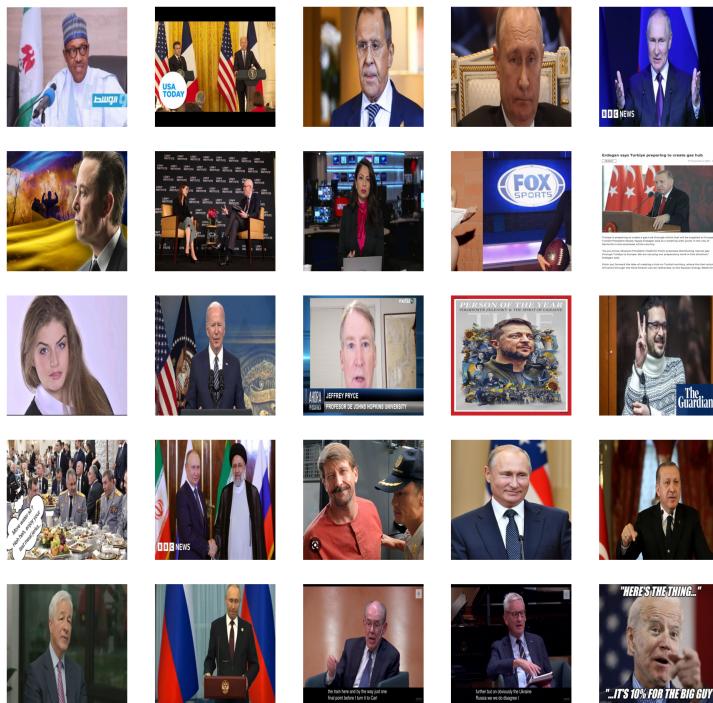


Figura 6.40. Ukraine war live television.

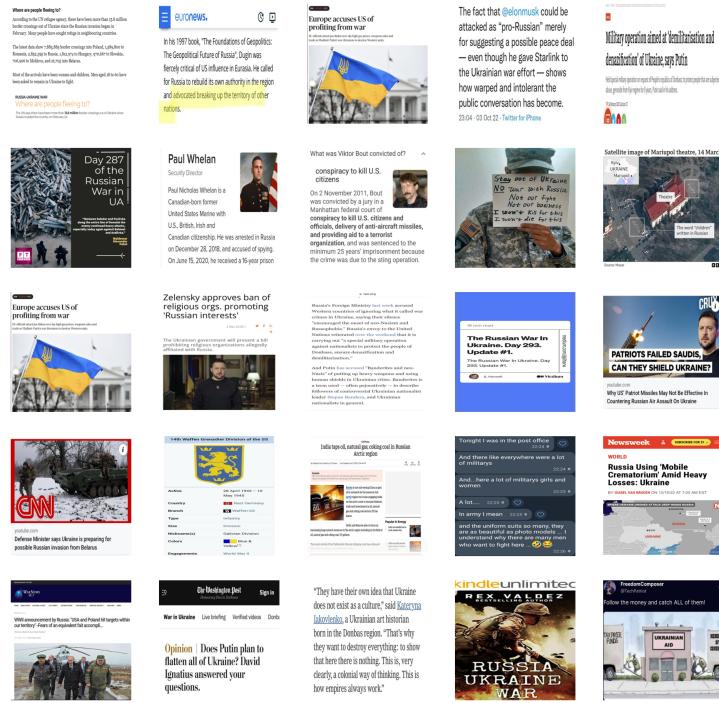


Figura 6.41. Ukraine war article.

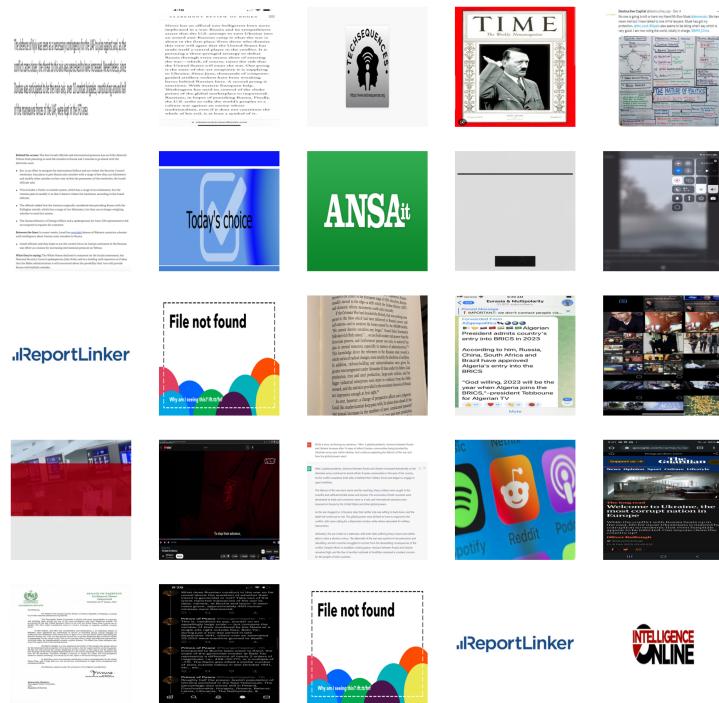


Figura 6.42. Report ukraine russia war.

6.3 Topic estratti

Dall'applicazione di HDBSCAN sui testi è possibile ricavare i topic riportati nelle tabelle 6.1, 6.2 e 6.3, che danno una panoramica sul corso della guerra in questo anno.

Topic febbraio-giugno	
ukrainian president zelensky war peace support solidarity russia ukraine war chinese protest invasion ukraine russia world ukraine war taking ukrainian refugees news radio coverage ukraine	american funny elections happening russia crisis bank sanctions attack global food supply finance war buying oil eu russian oil ban russian forces chernobyl russia evacuate civilians riupol new russian security forces mayor airstrikes kyiv hit

Tabella 6.1. Topic estratti dai testi di febbraio-giugno.

Topic luglio-novembre	
nuclear power reconsider ukraine president ukraine zelensky grain deal working ukraine china unbroken support russia possible nato jens stoltenberg ukraine russia perpetuating lie ukrainian accounts torture	referendums occupied territories ukraine use nuclear weapons russia war russian recruitment steel plant ukrainian military bakhmut donetsk ukraine russia ukraine war economic sanctions ukraine war russian troop

Tabella 6.2. Topic estratti dai testi di luglio-novembre.

Topic dicembre-febbraio 2023		
highlights discussion ukraine tank russian drones attack russian combat footage ukraine zelenksy demands sanctions putin joe biden nonsense biden corruption russia ukraine war kyiv donetsk kharkiv oblasts	russian genocide ukraine nato troops ukraine russia crisis censorship talking ukraine ukraine war pushing turkey israel supports ukraine coming ukraine war winter iran military support sanctions russia working	russian belarus troops day invasion ukraine russia border crossing ukraine nazis ukraine putin support trump office ukraine intelligence spying network ukraine india relations russia

Tabella 6.3. Topic estratti dai testi di dicembre-febbraio 2023.

Dalle immagini si ottengono invece i topic nelle tabelle 6.4, 6.5.

Topic immagini febbraio-novembre
ukraine war news ukraine russia soldier asia facing stagflationary outlook

Tabella 6.4. Topic estratti dalle immagini di febbraio-novembre.

Topic immagini dicembre-febbraio 2023
russia ukraine war map vehicles ukraine russia war ukraine war propaganda winter war scenario combat ukraine snow ukraine war slaughter bakhmut culture russia ukraine war ukraine war live television ukraine war article report ukraine russia war

Tabella 6.5. Topic estratti dalle immagini di dicembre-febbraio 2023.

Capitolo 7

Dettagli implementativi

Dopo aver descritto le principali tecniche per effettuare il clustering su varie tipologie di dati a disposizione, vediamo come costruire un dataset abbastanza grande su cui applicare le nostre conoscenze. Nel caso specifico che prevede la categorizzare delle informazioni relative agli eventi sfortunati della guerra in Ucraina presenti su Twitter, è stato utilizzato un particolare programma messo a disposizione dallo stesso social network per ottenere i tweet necessari, ossia una API di Twitter.

7.1 Cosa è una API?

In generale una **Application Programming Interface** (o **API**) è uno strumento che permette a due software di comunicare tra loro secondo una convenzione stabilita. Una API definisce come uno sviluppatore debba richiedere dei servizi da un sistema operativo o da altre applicazione, ed espone i dati entro diversi contesti e attraverso canali multipli. Le aziende (come Twitter in questo caso) possono sfruttare le API per condividere dati e funzionalità delle loro applicazioni con sviluppatori di terze parti, partner di business e dipartimenti interni alla loro compagnia.

I vantaggi di utilizzare una API sono molteplici:

- **collaborazione migliore:** le imprese utilizzano applicazioni cloud molte delle quali sono disconnesse. Le API permettono l'integrazione tra le piattaforme e le applicazioni in modo che possano comunicare tra loro, così che le compagnie possano automatizzare il flusso del lavoro e migliorare le prestazioni;
- **monetizzazione:** molte aziende inizialmente forniscono le loro API gratuitamente, in modo da costruire velocemente un vasto bacino di utenti e instaurare relazioni con potenziali partner di business. In seguito, se per esempio l'API si specializza nel trattamento di dati personali o operazioni, la si può monetizzare

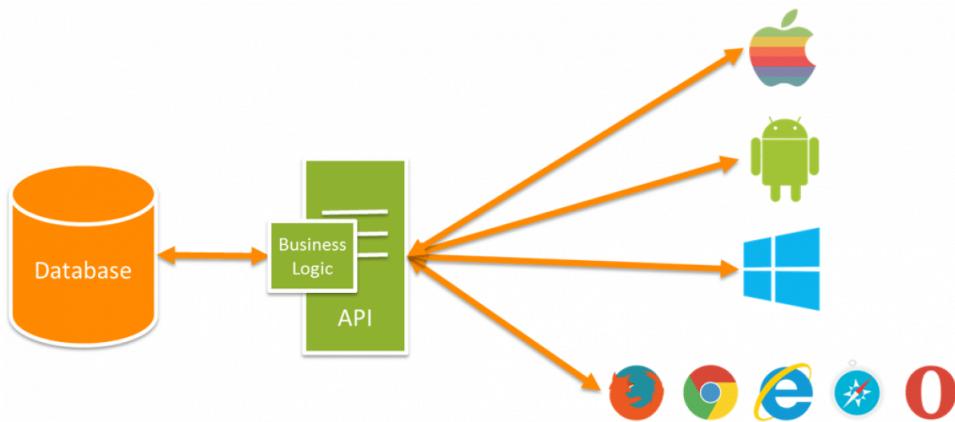


Figura 7.1. Api

vendendone l'accesso. È il caso di alcune compagnie che usano PayPal e sono disposte a pagare per avere un servizio di fiducia per gestire il proprio denaro;

- **flessibilità e scalabilità:** le API offrono un servizio a diversi programmi contemporaneamente, pertanto migliorando delle funzionalità interne chiunque le stia utilizzando ne gioverà;
- **sicurezza:** come accennato poco fa, alcune API offrono dei servizi di sicurezza ai loro clienti, implementando dei sistemi di controlli degli accessi, crittografia, ecc.

Agli sviluppatori che utilizzano una API non interessa sapere come questa sia implementata, ma si limitano a rispettare le regole per la comunicazione messe a disposizione da essa. La API risiede tra l'applicazione del client e il server, agendo come livello intermediario per trasferire i dati tra i sistemi. Le fasi per utilizzare una API sono le seguenti:

1. l'applicazione del client inizializza una **API call** o **richiesta** per ottenere dei dati. La richiesta viene processata da un'applicazione lato server attraverso l'**API's Uniform Resource Identifier (URI)** e include le informazioni dettagliate riguardo cosa si desidera. La API riceve le richieste e invia le risposte tramite l'**API endpoint**, uno spazio digitale univocamente identificato da un **Uniform Resource Locator (URL)**;
2. dove aver ricevuto una richiesta valida, la API ne effettua un'altra su un programma esterno o un web server per ottenere le informazioni;
3. il server manda le informazioni richieste alla API;
4. la API trasferisce i dati ottenuti all'applicazione del client di partenza.

Le API offrono sicurezza perché la loro posizione intermedia estrae le funzionalità dei due sistemi (di fatto scollega l'applicazione che sfrutta i dati da ciò che li fornisce). Inoltre nella API call sono spesso incluse le credenziali di accesso che riducono il rischio di attacchi al server, e un servizio chiamato *API gateway* può gestire gli accessi per limitare i rischi per la API.

Si consideri, per esempio, il caso di una API che offre un servizio per pagamenti online. Il cliente può inserire i dati della carta di credito sul sito che gli interessa, poi l'API, invece di utilizzare i dati in sé, crea un *token* identificativo della richiesta e lo usa per contattare il server esterno.

7.2 Esempi di API

Come già accennato, le API consentono alle aziende di diffondere le proprie funzionalità mantenendo comunque sicurezza e controllo, per questo sono diventate uno strumento di valore nel mondo attuale. Di seguito sono riportati alcuni esempi:

- **sistemi di login:** siti come Facebook, Twitter o Google sfruttano delle API per consentire il login rapido agli utenti, salvando i loro dati personali;
- **lavorazione di pagamenti:** alcuni siti accettano pagamenti utilizzando PayPal, permettendo agli utenti di non condividere i propri dati personali online;
- **comparazione dei costi dei viaggi:** alcuni siti come Booking mostrano e confrontano i costi di diversi viaggi per ogni data e destinazione. Ciò è reso possibile tramite delle API che forniscono alle applicazioni in tempo reale le novità riguardo alberghi e voli;
- **Google Maps:** lo stesso Google Maps è una API che fornisce agli utenti diverse informazioni sui posti in cui si trovano, come luoghi di interesse, informazioni sul traffico, percorsi più brevi per raggiungere una destinazione, ecc.;
- **Twitter:** ogni tweet contiene tanti attributi, come id, autore, testo, immagini, data e orario di scrittura, geo-localizzazione, ecc. Twitter mette a disposizione le proprie API agli sviluppatori per consentire loro di importare ciò che desiderano. Inoltre alcune permettono anche di scrivere tweet direttamente sul proprio profilo o interagire con altri in modo automatico, e questo consente, per esempio, lo sviluppo di chat-bot.

7.3 Tipi di API e protocolli

La maggior parte delle API attuali condividono i loro dati e funzionalità su internet e per questo sono chiamate **web API**. Esse si dividono in quattro categorie:

- **Open APIs:** sono di tipo *open source* e vi si può accedere tramite il protocollo HTTP. Sono anche denominate come **pubbliche** perché sono accessibili a chiunque e a volte possono richiedere una registrazione o la conoscenza della *API key*;
- **Internal APIs:** al contrario delle open, queste sono accessibili solo all'interno dell'organizzazione che le fornisce e pertanto sono anche dette **private**. Si usano spesso per consentire lo scambio di informazioni tra diversi sistemi all'interno della stessa azienda, migliorando di fatto la produttività e l'efficacia delle comunicazioni;
- **Partner APIs:** sono simili alle open API, ma più indicate per particolari compagnie e sviluppatori. Solitamente è consentito l'accesso agli sviluppatori in modalità autonoma, tuttavia è necessario un processo per ottenere le credenziali;
- **Composite:** combinano alcuni servizi o dati di diverse API che consentono agli sviluppatori di accedere a molteplici endpoint con una singola chiamata.

L'utilizzo delle API è gestito da alcuni protocolli che stabiliscono una serie di regole su come sono strutturate le richieste e le risposte e quali sono i tipi di dati accettati. I protocolli più comuni sono:

- **SOAP** (Simple Object Access Protocol): il formato dei messaggi è costruito con XML, i quali possono essere trasferiti attraverso i protocolli di trasporto HTTP e SMTP. Con SOAP è più semplice trasferire informazioni tra applicazioni o software in esecuzione su ambienti diversi o scritti in lingue diverse. Inoltre garantisce una sicurezza elevata;
- **REST** (Representational State Transfer): non c'è un formato standard specifico perché REST si basa su diverse architetture: i messaggi possono essere scritti in JSON, HTML, PHP o testo normale. Una *RESTful API* è chiamata tale se rispetta i vincoli imposti e solitamente è più semplice da implementare rispetto all'utilizzo di un protocollo SOAP per via delle dimensioni più ridotte dei singoli messaggi;
- **XML-RPC** (XML-*Remote Procedure Call*): si basa su un singolo specifico formato di XML, mentre SOAP usa un formato XML proprietario, per questo

XML-RPC è più semplice da capire. Inoltre, essendo più vecchio di SOAP, ha un modello di sicurezza più antiquato. Come si deduce dal nome utilizza le *remote procedure call* (chiamate a procedura remota), cioè chiama delle funzioni accessibili tramite internet;

- **JSON-RPC:** simile a XML-RPC ma utilizza JSON come formato dei dati.

7.4 API di Twitter

Come accennato nel paragrafo 7.2, Twitter mette a disposizione delle API che consentono agli sviluppatori di accedere a parte del servizio che offre l'azienda stessa. Ci sono due diversi tipi di **Twitter API**:

- la prima categoria permette l'accesso ai dati pubblici che gli utenti decidono di condividere sul social, quindi tweet, risposte, commenti, ecc.;
- la seconda categoria consente agli utenti di gestire le informazioni private, cioè non condivise apertamente con il resto del mondo (per esempio i messaggi diretti).

7.4.1 Accesso ai dati di Twitter

Per usufruire delle funzionalità delle Twitter API bisogna innanzitutto avere un account e registrare un'**applicazione**, che di default può accedere solo alle informazioni pubbliche. In seguito, se l'utente lo necessita, può fare richiesta di ulteriori autorizzazioni per accedere ad altri endpoint che si dividono in cinque gruppi.

Account e utenti

Comprende la gestione del proprio profilo e delle impostazioni dell'account, silenziare o bloccare utenti, gestire utenti e follower, richiedere informazioni sulle attività di un determinato account, ecc. Possono essere utili per offrire dei servizi alla comunità, per esempio in Virginia la protezione civile informa i cittadini di eventuali emergenze e allerte tramite questi endpoint.

Tweet e risposte

Gli sviluppatori possono importare tweet e risposte pubbliche di altri utenti effettuando ricerche (*query*) con parole chiave, oppure importare i tweet presenti attualmente nella propria home o quelli creati da un particolare utente. Le informazioni ricavate possono essere usate per studiare un particolare fenomeno (come la guerra in Ucraina

nel caso di questo progetto) e comprendere e combattere la disinformazione su un particolare argomento (utile soprattutto i questi anni di pandemia del Covid).

Messaggi diretti

L'accesso ai propri messaggi diretti (non quelli di altri utenti) consente agli sviluppatori di creare esperienze personalizzate, come lo sviluppo di chatbot o l'automatizzazione di un servizio di assistenza di una azienda.

Annunci

Alcune API consentono di aiutare le aziende a creare e gestire automaticamente campagne pubblicitarie su Twitter su misura delle varie categorie di pubblico, grazie all'identificazione di argomenti e interessi rilevanti.

Strumenti e SDK per i publisher

A disposizione di publisher e sviluppatori ci sono software che incorporano cronologie nelle pagine web, pulsanti di condivisione e altri contenuti di Twitter, per consentire ai brand di aggiungere conversazioni pubbliche in diretta alla propria esperienza web e di facilitare la condivisione di informazioni e articoli sui propri siti da parte dei clienti.

7.5 Creazione del dataset

In questa sezione viene mostrato il processo di creazione del dataset utilizzando le API descritte in precedenza e ripulire i testi da *stopwords*, numeri e caratteri irrilevanti. Il primo passo da eseguire è la creazione di una applicazione nel portale *developer* di Twitter e richiedere il profilo di tipo *elevated* per sbloccare l'utilizzo di alcune API e impostare il cap mensile di tweet importati a due milioni.

7.5.1 Importazione dei tweet

Le API utilizzate, per effettuare richieste nelle veci dell'account di Twitter, si basano sul metodo **OAuth 1.0a**, il quale, per effettuare l'autenticazione dell'utente e dell'applicazione, necessita delle seguenti credenziali presenti nel portale *developer*:

- **Api Key e Api Secret:** sono le credenziali fondamentali che corrispondono praticamente all'username e alla password di autenticazione della applicazione. Vengono utilizzati dalla API per capire quale app sta effettuando le richieste;

- **Access Token e Secret Token:** rappresentano l’user per il quale si stanno effettuando le richieste (se lo si genera all’interno dell’app allora rappresenta l’user che la possiede).

In Python le API e il metodo per l’autenticazione vengono forniti dalla libreria di Python **Tweepy** rispettivamente tramite le classi *tweepy.API* e *tweepy.OAuthHandler*. Sono state utilizzate due API, le quali entrambe prendono in input una query e restituiscono i tweet che la rispettano ma hanno alcune differenze:

- la API **search** per importare i tweet recenti (cioè postati da massimo 10 giorni) senza limitazioni riguardo al numero di tweet importati mensilmente e con la possibilità di scrivere query più accurate, inserendo filtri per escludere i *retweet* (così da avere, per quanto possibile, tutti tweet unici) e tweet di altre lingue diverse dall’inglese;
- la API premium **search_full_archive** per importare i tweet dei mesi passati ma con un cap mensile, un limite massimo di richieste giornaliere e mensili e l’impossibilità di utilizzare filtri nella query. Per risolvere questo problema si veda il paragrafo 7.6.

```

1 import tweepy
2
3 # credenziali
4 ACCESS_TOKEN = "1456205100-pHRLPhyPvE1VlC6Cty9qxyPWf9GFjJRAuEOofx5"
5 ACCESS_TOKEN_SECRET = "WAG2uGIYa6bGSO0aTN0mc88tUDIjSVkvXnJTHF0sfQ97B"
6 API_KEY = "BkcawOx9kHTLcprjB1vk6Lazd"
7 API_KEY_SECRET = "27FpVac8aHRpKZAtMeEDvt0l7dqFrxoWd6QVtUPavTYLPgDDJE"
8
9 # autenticazione
10 auth = tweepy.OAuthHandler(API_KEY, API_KEY_SECRET)
11 auth.set_access_token(ACCESS_TOKEN, ACCESS_TOKEN_SECRET)
12
13 api = tweepy.API(auth) # classe per chiamare le API

```

Di seguito il funzionamento della API *search* (per la *search_full_archive* il procedimento è simile).

La API *search* è stata utilizzata per raccogliere i tweet giorno per giorno da dicembre 2022 fino a febbraio 2023. Una volta a settimana la si applicava sui giorni passati per ottenere 300 tweet per ciascuno. Tramite i parametri *date_start* e *date_end* si seleziona il giorno attuale in cui la API deve operare, mentre *tweet_mode="extended"* permette di ottenere informazioni più dettagliate sui tweet.

Nella lista *tweets* vengono salvati i tweet temporaneamente per ogni richiesta, mentre poi vengono aggiunti nel dataframe *df_raw* ("raw" perché il contenuto dei tweet non è stato ancora elaborato). Dei tweet viene salvato l'id, il testo completo, l'username dell'utente che l'ha scritto, la posizione, la data di creazione, il numero di retweet, l'url e l'eventuale immagine che possiede.

```

1 # data una query, importo un certo numero di tweets e creo un dataset
2
3 query = '"ukraine war" OR "ukraine russia" OR "ukraine zelensky" OR "
4     ukraina putin" OR "ukraine putin" -filter:retweets'
5
6 # scarico 300 tweets al giorno
7 limit = 300
8 date_start = datetime.date(2022,12,1)
9 date_end = date_start + datetime.timedelta(days=1)
10
11 list_tweets = []
12 for i in range(7):    # importo i tweets per una settimana
13     print(f"Importo i tweets del {date_start}")
14     tweets=tweepy.Cursor(api.search, q=query, tweet_mode='extended',
15                           lang='en', since=date_start, until=date_end, wait_on_rate_limit=
16                           True).items(limit)
17
18     print(tweets)
19     for tweet in tweets:
20         try:
21             media = tweet.entities[ "media "][0][ "media_url "]
22         except:
23             media = ""
24         url = f"https://twitter.com/{tweet.user.screen_name}/status/{
25             tweet.id}"
26         list_tweets.append([tweet.id, tweet.full_text, tweet.user.
27             screen_name, tweet.user.location, tweet.created_at, tweet.
28             retweet_count, media, url])
29
30     date_start += datetime.timedelta(days=1)
31     date_end += datetime.timedelta(days=1)
32
33 df_raw = pd.DataFrame(data=list_tweets, columns=[ 'id ', 'text ', 'user ',
34                           'location ', 'created_at ', 'retweet_count ', 'image ', 'url '])
35 df_raw

```

7.6 NLP sui testi

Prima di applicare gli algoritmi di clustering sui testi è necessario effettuare qualche operazione preliminare su di essi:

- rimuovere gli hashtag, i link e le menzioni;
- rimuovere eventuali retweet, cioè tweet che iniziano con *RT*;
- rimuovere le *stopwords*, ossia parole che non aggiungono informazioni ai fini dell'analisi e pertanto vengono rimosse. Possono essere articoli, congiunzioni, preposizioni, ecc.;
- rimuovere le *profanity*, ossia parole scurrili;
- rimuovere tutti i simboli che non sono lettere, quindi numeri e caratteri speciali.

La pulizia del contenuto dei testi avviene grazie alla funzione *clean_tweets* che dato il testo *raw* di un tweet applica tutte le operazioni elencate poc'anzi e lo restituisce *clean*. In seguito vengono filtrati i tweet per lingua con il metodo **detect** della libreria *langdetect*, che permette di riconoscere la lingua di un dato testo e quindi di escludere quelli di altre lingue.

Infine il testo viene passato alla funzione *simili* insieme alla lista dei tweet precedentemente salvati e calcola il grado di somiglianza tra il testo e i testi degli altri tweet già in lista: se la somiglianza supera il 90% con un singolo tweet, scarta il nuovo testo. Il calcolo della somiglianza viene effettuato con il metodo **SequenceMatcher** della libreria *difflib*.

```

1 # funzione che dato il testo di un tweet e la lista dei tweet
  attualmente puliti, confronta il testo con il testo degli altri
  tweet: se la somiglianza con uno supera il 90%, lo scarta
2 def simili(testo, lista):
3     for tw in lista:
4         sim = SequenceMatcher(testo, tw[1]).ratio()
5         if sim > 0.90:
6             return True
7     return False
8
9 # funzione che dato un tweet, lo "pulisce"
10 def clean_tweets(tweet):
11     if type(tweet) == float:
12         return ""
13     tweet = tweet.lower()
14     r = []      # stringa finale
15

```

```

16     for word in tweet.split(" "):
17         if word.startswith("http") or word.startswith("#") or word.
18             startswith("@") or word.startswith("https"):
19             continue
20         if "http" in word:
21             word1 = re.split('http', word)
22             r.append(word1[0])
23             continue
24         if "#" in word:
25             word1 = re.split('#', word)
26             r.append(word1[0])
27             continue
28         if "@" in word:
29             word1 = re.split('@', word)
30             r.append(word1[0])
31             continue
32         if word in (stop):
33             continue
34         if word.startswith("rt"):
35             continue
36         r.append(word)
37
38     r = " ".join(r)
39
40     r = re.sub("[^A-Za-z ]", " ", r)      # rimuovo tutti i caratteri con
41     non sono lettere, numeri o spazi
42     r = profanity.censor(r)
43     r = re.sub("\*", "", r)    # rimuovo i * dopo la censura
44     r = r.strip()
45
46     return r

```

I tweet finali con il testo pulito vengono salvati nel dataframe *df_clean* e poi salvati in un file su Google Drive.

```

1 cleaned_tweets = []
2 for i in range(len(df_raw)):
3     id = df_raw["id"][i]
4     text = df_raw["text"][i]
5     user = df_raw["user"][i]
6     location = df_raw["location"][i]
7     created_at = df_raw["created_at"][i]
8     retweet_count = df_raw["retweet_count"][i]
9     image = df_raw["image"][i]
10    url = df_raw["url"][i]
11
12
13    testo = clean_tweets(df_raw["text"][i])    # pulisco il testo
14
15    # se il testo e' vuoto non appendo il tweet

```

```
16     if testo == None or testo == "":
17         continue
18
19     DetectorFactory.seed = 0
20     if detect(text) != "en": # se il testo non e' in lingua inglese,
21         lo scarto
22         continue
23
24     # se ho un tweet molto simile a uno già salvato, lo scarto
25     if len(cleaned_tweets) > 0 and simili(testo, cleaned_tweets):
26         continue
27
28     cleaned_tweets.append([id, testo, user, location, created_at,
29                           retweet_count, image, url])
30 df_cleaned = pd.DataFrame(data=cleaned_tweets, columns=['id', 'text',
31                            'user', 'location', 'created_at', 'retweet_count',
32                            'image', 'url'])
```


Capitolo 8

Conclusione e sviluppi futuri

Nei capitoli precedenti è stato descritta una procedura di clustering per effettuare *data mining* sui tweet riguardanti il tema della guerra in Ucraina, analizzando dati di diversa natura (testi e immagini) nel tentativo di estrarre topic più dettagliati in modo da avere un andamento nel tempo del fenomeno. Innanzitutto si è visto come creare diversi dataset sufficientemente grandi contenenti i dati di interesse, poi sono state mostrate delle tecniche di pre-elaborazione per permettere al computer di "leggere" i dati e di applicarvi gli algoritmi di clustering. Di questi ne sono stati descritti diversi, per arrivare infine alla conclusione, tramite alcuni test e osservazioni, quale adatta meglio allo studio di un fenomeno di attualità come la guerra in Ucraina.

Il lavoro riportato pone solo le basi ad un'analisi che può sicuramente essere realizzata in modo più ampio e dettagliato. Per esempio si può pensare di continuare a estendere l'insieme dei tweet a disposizione, continuando a importarli in futuro di volta in volta, per applicare una procedura simile e avere un confronto tra i risultati a lungo termine. Inoltre le limitazioni della api premium *search_full_archive* hanno impedito di avere dei dataset negli intervalli temporali di febbraio-giugno e luglio-novembre tanto grandi quanto quello di dicembre 2022-febbraio 2023. Anche il numero delle immagini ne ha risentito, che nei primi due archi temporali era decisamente troppo basso. L'assenza di queste limitazioni (ottenibile elevando il livello del proprio account sviluppatore di Twitter tramite un abbonamento) avrebbe consentito sia di avere sicuramente molti più dati a disposizione, sia di estrarre i dati in modo più dettagliato, magari scegliendo intervalli temporali più ridotti per effettuare un'analisi più mirata.

Nel lavoro riportato, per avere a disposizione una quantità di dati abbastanza ampia, è stato scelto di importare solamente tweet in lingua inglese. Per avere una visione più estesa dell'argomento trattato, un futuro studio potrebbe prevedere la raccolta di tweet di diverse lingue oppure filtrati per paese di origine, in modo da analizzarli separatamente e confrontarli tra loro per vedere eventuali differenze.

Bibliografia

- [1] AI4Business. *Cos'è BERT, l'algoritmo che cambia il mondo del Natural Language Processing.* <https://www.ai4business.it/intelligenza-artificiale/cose-bert-lalgoritmo-che-cambia-il-mondo-del-natural-language-processing/>. [Online; accessed 07-Mar-2023]. 2019.
- [2] AnalytixLabs. *Types of Clustering Algorithms.* Accessed: March 7, 2023. URL: <https://www.analytixlabs.co.in/blog/types-of-clustering-algorithms/>.
- [3] Ritik Bhatia. *Clustering with K-means.* 2022. URL: <https://towardsdatascience.com/clustering-with-k-means-1e07a8bfb7ca>.
- [4] Jason Brownlee. *The Curse of Dimensionality.* <https://towardsdatascience.com/the-curse-of-dimensionality-50dc6e49aa1e>. [Online; accessed 07-Mar-2023]. 2017.
- [5] Andrew Davies. «A review of unsupervised feature learning and deep learning for time-series modeling». In: *Journal of Big Data* 2.1 (2015), pp. 1–34. URL: <https://link.springer.com/article/10.1007/s40745-015-0040-1>.
- [6] UMAP-learn Developers. *UMAP-learn Documentation.* <https://umap-learn.readthedocs.io/en/latest/>. [Online; accessed 07-Mar-2023]. 2021.
- [7] Fernando Fernandez. «Natural Language Processing: What It Is, How It Works, and Applications». In: *MonkeyLearn* (2021). URL: <https://monkeylearn.com/natural-language-processing/>.
- [8] Fernando Fernandez. «Text Cleaning: A Guide to Preprocessing Text in Python». In: *MonkeyLearn* (2021). URL: <https://monkeylearn.com/blog/text-cleaning/>.
- [9] Fernando Fernandez. «Word Embeddings: Transforming Text to Numbers». In: *MonkeyLearn* (2021). URL: <https://monkeylearn.com/blog/word-embeddings-transform-text-numbers/>.

- [10] Michael Galarnyk. *All About Feature Scaling - Towards Data Science*. <https://towardsdatascience.com/all-about-feature-scaling-bcc0ad75cb35>. [Online; accessed 07-Mar-2023]. 2019.
- [11] Ritu Garg. «Understanding the Concept of Hierarchical Clustering Technique». In: *Towards Data Science* (2021). URL: <https://towardsdatascience.com/understanding-the-concept-of-hierarchical-clustering-technique-c6e8243758ec>.
- [12] Google. *Overview of Clustering in Machine Learning*. Accessed: March 7, 2023. URL: <https://developers.google.com/machine-learning/clustering/overview>.
- [13] Alina-Larisa Hilge. *Artificial Intelligence*. Accessed: March 7, 2023. URL: <https://towardsdatascience.com/artificial-intelligence-d1e45efc99b4>.
- [14] IBM. *API*. 2023. URL: <https://www.ibm.com/topics/api>.
- [15] IBM. *Unsupervised Learning*. Accessed: March 7, 2023. URL: <https://www.ibm.com/topics/unsupervised-learning>.
- [16] Anil K Jain, M Narasimha Murty e Patrick J Flynn. «Data clustering: 50 years beyond K-means». In: *Pattern recognition letters* 31.8 (2010), pp. 651–666. URL: <https://www.sciencedirect.com/science/article/pii/S0167865510000585>.
- [17] Alboukadel Kassambara. *Cluster Validation Statistics: Must-Know Methods*. Accessed: March 7, 2023. URL: <https://www.datanovia.com/en/lessons/cluster-validation-statistics-must-know-methods>.
- [18] Kinsta. *Cos'è un API Endpoint?* <https://kinsta.com/it/knowledgebase/api-endpoint/>. Accessed on March 8, 2023. 2023.
- [19] Prateek Kumar. «BIRCH in Data Mining». In: *JavaTpoint* (2021). URL: <https://www.javatpoint.com/birch-in-data-mining>.
- [20] James Melville Leland McInnes John Healy. *How HDBSCAN Works - HDBSCAN 0.8.27 documentation*. https://hdbSCAN.readthedocs.io/en/latest/how_hdbSCAN_works.html. [Online; accessed 07-Mar-2023]. 2021.
- [21] Yanchi Liu et al. «Understanding of internal clustering validation measures». In: *2010 IEEE international conference on data mining*. IEEE. 2010, pp. 911–916.
- [22] Sayan Mazumdar. «Topology and Density Based Clustering». In: *Domino Data Lab* (2019). URL: <https://www.dominodatalab.com/blog/topology-and-density-based-clustering>.

- [23] Tara Mullin. *DBSCAN Parameter Estimation*. <https://medium.com/@tarammullin/dbscan-parameter-estimation-ff8330e3a3bd>. [Online; accessed 07-Mar-2023]. 2019.
- [24] Gokul Nair. *A Brief Introduction to Supervised Learning*. Accessed: March 7, 2023. URL: <https://towardsdatascience.com/a-brief-introduction-to-supervised-learning-54a3e3932590>.
- [25] Gokul Nair. *Introductory Guide to Artificial Intelligence*. Accessed: March 7, 2023. URL: <https://towardsdatascience.com/introductory-guide-to-artificial-intelligence-11fc04cea042>.
- [26] Gokul Nair. *Machine Learning: An Introduction*. Accessed: March 7, 2023. URL: <https://towardsdatascience.com/machine-learning-an-introduction-23b84d51e6d0>.
- [27] Rinkesh Patel. «Word Embeddings: CBOW vs Skip-Gram». In: *Baeldung* (2021). URL: <https://www.baeldung.com/cs/word-embeddings-cbow-vs-skip-gram>.
- [28] F. Pedregosa et al. «Scikit-learn: Machine Learning in Python». In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.
- [29] Parth Sharma. *Dimension Reduction Techniques with Python*. 2022. URL: <https://towardsdatascience.com/dimension-reduction-techniques-with-python-f36ca7009e5c>.
- [30] Kuldeep Singh. *Keyword Extraction with BERT*. <https://towardsdatascience.com/keyword-extraction-with-bert-724efca412ea>. [Online; accessed 07-Mar-2023]. 2020.
- [31] John Smith e Sarah Johnson. «A Study of Convolutional Neural Networks». In: *Journal of Machine Learning Research* (2022).
- [32] TechTarget. *BERT (Bidirectional Encoder Representations from Transformers) language model definition*. <https://www.techtarget.com/searchenterpriseai/definition/BERT-language-model>. [Online; accessed 07-Mar-2023]. 2018.
- [33] Twitter. *Twitter API*. 2023. URL: <https://help.twitter.com/it/rules-and-policies/twitter-api>.
- [34] Wikipedia. *Elbow method (clustering) - Wikipedia*. [https://en.wikipedia.org/wiki/Elbow_method_\(clustering\)](https://en.wikipedia.org/wiki/Elbow_method_(clustering)). [Online; accessed 07-Mar-2023]. 2022.
- [35] Wikipedia contributors. *Cluster analysis*. Accessed: March 7, 2023. 2023. URL: https://en.wikipedia.org/wiki/Cluster_analysis.

- [36] Phil Yang. «What Is Text Vectorization in NLP?» In: *Deepset* (2020). URL: <https://www.deepset.ai/blog/what-is-text-vectorization-in-nlp>.