Progetto Reddit

Link GitHub

Autore: Tatarenko Nikita

Email: n.tatarenko@studenti.uniba.it
Matricola: 758475

Università degli Studi di Bari Aldo Moro

Indice

1 Introduzione

L'obiettivo del progetto è quello di sviluppare un sistema per l'analisi e la classificazione automatizzata dei post di Reddit. Le principali funzionalità che saranno sviluppate sono le seguenti:

- **Clustering dei posts**. Applicazione di algoritmi di clustering per scoprire pattern e raggruppamenti nascosti all'interno del dataset.
- **Classificazione dei posts**. Definizione di una etichetta (virale/non virale) basata sulle caratteristiche dei Cluster della fase precedente. Il sistema dovrà classificare i nuovi post come virali o non virali.
- **Classificazione Probabilistica dei posts**. Un approccio alternativo per la classificazione dei post. Durante questa fase, vengono identificate in modo automatico nuove connessioni tra le features, e su questa base si effettua una classificazione differente dalla fase precedente.

1.1 Introduzione a Reddit

Reddit è un social network ma con alcune differenza rispetto ai classici come Instagram, Facebook o TikTok.

Proprio per questo appartiene di più alla categoria social news nella quale gli utenti posso pubblicare anche semplice contenuto di testo, link, immagini, articoli ecc. Questo non è permesso in tutti i social: per esempio, Instagram si limita alla pubblicazione di sole foto, video o reel di conseguenza hanno un funzionamento di rilevanza dei post differente.

1.1.1 Funzionamento Reddit

Gli utenti interagiscono attraverso i post, questi sono valutati tramite il criterio "su" o "giù" (solitamente chiamate *upvote* e *dawnvote*). Questo funzionamento determina la posizione in cui appare il post nella homepage o altre sezioni della piattaforma.

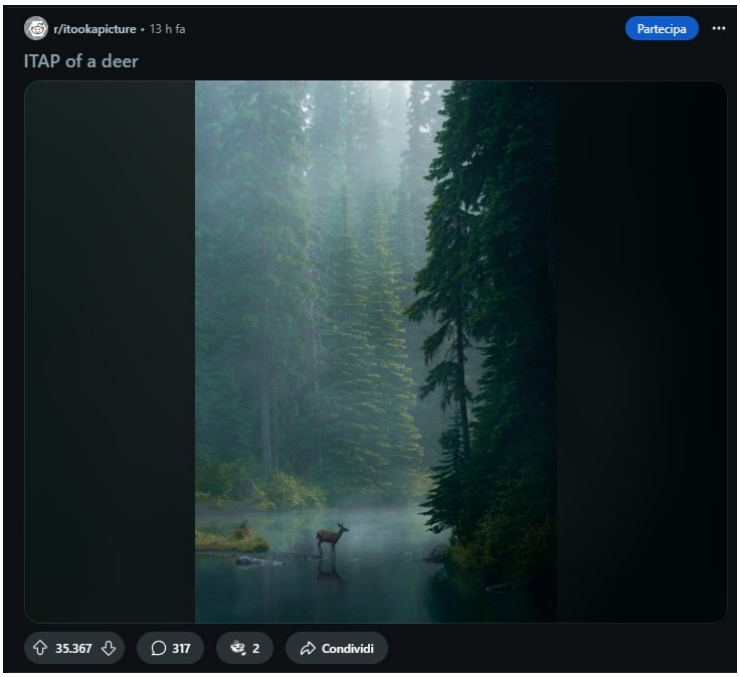

Figure 1: Esempio di un post su Reddit

I post con un numero di upvote elevato vengono posizionati nella homepage prima di quelli con un punteggio inferiore. Inoltre si hanno anche dei *subreddit* cioè forum tematici in base ad un determinato argomento, notizia, hobby o interesse spesso gestiti da utenti moderatori. In questo progetto andremo comunque a lavorare con i post in generale, non saranno considerati i subreddit specifici.

1.2 Ambiente di Sviluppo

Il progetto è stato sviluppato in Python, in quanto rappresenta oggi lo strumento principale per lo sviluppo di sistemi in ambito di machine learning e offre numerose librerie per ogni tipo di apprendimento. Di seguito è riportato l'elenco delle librerie utilizzate nel progetto:

- **pandas**: Libreria per la manipolazione e l'analisi dei dati strutturati (usato per il dataset).
- **numpy**: Libreria per il calcolo matematico.
- **scikit-learn**: Libreria per il machine learning, include diverse implementazioni di algoritmi di apprendimento supervisionato e non supervisionato con strumenti di valutazione.
- **matplotlib**: Libreria per la visualizzazione dei grafici.
- **seaborn**: Libreria che lavora con matplotlib per una visualizzazione migliore dei grafici.
- **pgmpy**: Libreria per modelli probabilistici.
- **networkx**: Libreria per la manipolazione dei grafi e reti complesse.
- **imbalanced-learn**: Libreria per il bilanciamento del dataset.
- **praw**: Libreria per l'accesso all'API di Reddit.
- **prawcore**: Modulo di supporto per praw.
- **tqdm**: Libreria per la creazione di progress bar.
- **scipy**: Libreria per il calcolo scientifico avanzato.
- **joblib**: Libreria per il salvataggio dei modelli.
- **threadpoolctl**: Libreria per controllare i thread.
- **pyarrow**: Libreria per il formato Apache Arrow, necessario per evitare problemi di compatibilità.

Questo elenco comprende le librerie non installate in Python base, sono utilizzate altre librerie comuni già disponibili per altre operazioni semplici.

2 Dataset

Il dataset è stato ottenuto tramite l'API ufficiale di Reddit e utilizzato tramite la libreria `praw` su Python. Il file `reddit_scanner.py` è un script che si occupa di recuperare i post più in tendenza al momento, in questo caso sono raccolte le informazioni del 07/08/2025. Sono stati ricavati 5000 post totali con le seguenti informazioni:

- `subreddit`: nome del subreddit
- `title`: titolo del post
- `text`: testo del post (fino a 500 caratteri)
- `url`: URL del post
- `created_utc`: data e ora di creazione in formato ISO
- `hour`: ora di creazione
- `weekday`: giorno della settimana di creazione (0 = lunedì)

- **Indicatori di viralità**
  - `score`: punteggio (upvotes - downvotes)
  - `num_comments`: numero di commenti
  - `upvote_ratio`: rapporto upvote
  - `awards`: numero totale di premi ricevuti
  - **Dati autore**
    - `author_link_karma`: karma link autore (se disponibile)[1]
    - `author_comment_karma`: karma comment autore (se disponibile)[2]
- **Flag reddit-specifici**
  - `is_original_content`: contenuto originale (booleano)
  - `spoiler`: spoiler (booleano)
  - `nsfw`: contenuto NSFW (booleano)
  - `stickied`: post fissato (booleano)
  - `locked`: post bloccato (booleano)
  - `edited`: post modificato (booleano)
  - `distinguished`: stato speciale del post (es. "mod", "admin", "none")
- `content_type`: tipo di contenuto

Sono stati scaricati più dati possibili per poter capire al meglio cosa può influire nel far diventar un post virale, non tutti i campi saranno utilizzati nel progetto.

Si raccomanda di non avviare il file `reddit_scanner.py`, dato che sovrascrive i dati su cui sono state effettuate le fasi successive. Proprio per questo motivo, il file è eseguibile solo da terminale e non è stato incluso nel `main.py` dove è prevista l'esecuzione vera del progetto. Inoltre, non è possibile avviare la scansione se non si effettua la registrazione sul Reddit Developer Portal. Solo dopo questa fase sono forniti un ID e una password, necessari per ottenere l'autorizzazione per la scansione. Questi saranno omessi nel codice per motivi di privacy.

Per maggiori informazioni, consultare la documentazione ufficiale dell'API di Reddit.

[1] `author_link_karma` punteggio ottenuto dai voti positivi ricevuti dai post pubblicati dall'utente.

[2] `author_link_karma` somma dei voti positivi ricevuti dai commenti dell'utente in tutto Reddit.

2.1 Preprocessing del dataset

Dalla fase pretendente abbiamo ottenuto il file `reddit_viral_posts.json`, che contiene dati grezzi in formato JSON. Ora è necessario trasformare i dati in un dataset strutturato e pulito (in formato Parquet) in modo che sia pronto per il machine learning. Il file `data_cleaner.py` si occupa proprio di questo.

Come già detto in precedenza è stata prelevata la top 5000 dei post di reddit, di conseguenza è introdotto un primo controllo che verifica l'esistenza di 5 campi cruciali:

`["title", "score", "num_comments", "upvote_ratio", "created_utc"]`

Se questi dati mancano, significa che ci è stato un problema nel download dei dati dall'API di Reddit, dato che non è possibile che un post compaia in questa classifica senza avere uno dei campi elencati. Naturalmente questo problema non è stato rilevato ma il controllo è comunque stato lasciato per completezza.

2.1.1 Pulizia del testo

Nella funzione `clean_text` è utilizzata una espressione regolare per pulire il testo rimuovendo i caratteri alfanumerici, underscore e spazi bianchi.

`re.sub(r'[^\w\s]', '', text).strip()`

In questo modo il testo con cui lavoreremo successivamente avrà una forma uniforme e sarà utile a diminuire il "rumore" non informativo per i modelli ML.

2.1.2 Gestione datetime

Il campo `created_utc`, che rappresenta la data e l'ora di creazione del post in formato timestamp è trasformato in un oggetto datetime di Python, in modo da poter estrarre le componenti temporali. Sono ricavate 2 feature principali `post_hour` e `post_weekday` che rappresentano rispettivamente l'ora e il giorno della settimana della creazione del post(0 = Lunedì, 6 = Domenica).

2.1.3 Gestione dell'autore

In questa fase cerchiamo di gestire le due features grezze ottenute in precedenza sull'autore.

- `author_impact:` questa feature dovrà rappresentare la rilevanza dell'autore del post nel farlo diventare virale, di conseguenza si è pensato di unire i campi relativi a `link_karma` e `comment_karma`. Questi due valori molto spesso hanno scale molto diverse quindi è stata applicata la seguente formula:

$$0.6 \cdot \ln(\texttt{link\_karma} + 1) + 0.4 \cdot \ln(\texttt{comment\_karma} + 1)$$

  Ad entrambi i parametri si applica il logaritmo naturale sommato a 1 per ridurre l'impatto dei valori grandi e evitare la divergenza a $-\infty$. Dopo la trasformazione logaritmica ai risultati è applicata la combinazione lineare con i pesi rispettivamente 0.6 e 0.4, dando così più rilevanza a `link_karma`.

2.1.4 Risultati finali

In fine è creato un dizionario `cleaned_posts` che in base alla modifiche descritte prima conserva i dati per le fasi successive. Quindi sono conservate le seguenti feature:

- `id`: id del post convertito in stringa
- `subreddit`: nome del subreddit; se manca è impostato a `unknown`
- `title` e `text`: feature già discusse in precedenza
- `upvotes`: recupera lo score e lo converte in intero
- `num_comments`: numero dei commenti convertito in intero
- `upvotes_ratio`: rapporto upvote/downvote (es. 0.95 significa il 95% di upvote), conservato come float, di default 0
- `title_length`: lunghezza del titolo
- `post_hour` e `post_weekday`: feature già discusse in precedenza
- `nsfw`: flag per i contenuti "Not Safe For Work".
- `content_type`: tipo di contenuto come stringa
- `author_impact`: metrica discussa nel paragrafo precedente

Non tutte le features elencate sono utilizzate nella fasi successive di apprendimento, dato che alcune possono non essere utili nell'apprendimento di determinate fasi, si è deciso comunque di pulire tutti i dati per scopi di test e debugging. In seguito per ogni tipo di apprendimento saranno specificate le feature con cui si andrà a lavorare.

Tutti questi risultati in fine sono salvati nel file `cleaned_posts.parquet`.

2.2 Grafici

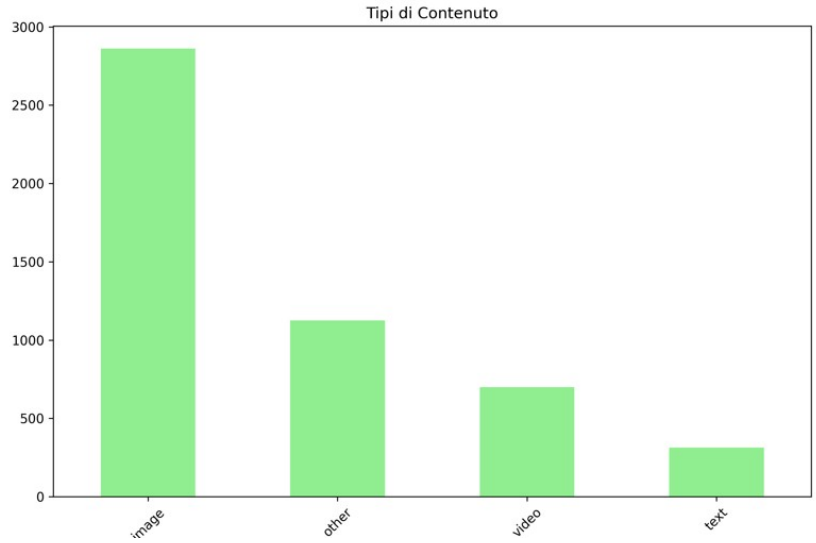

Figure 2: Grafico sui contenuti dei posts

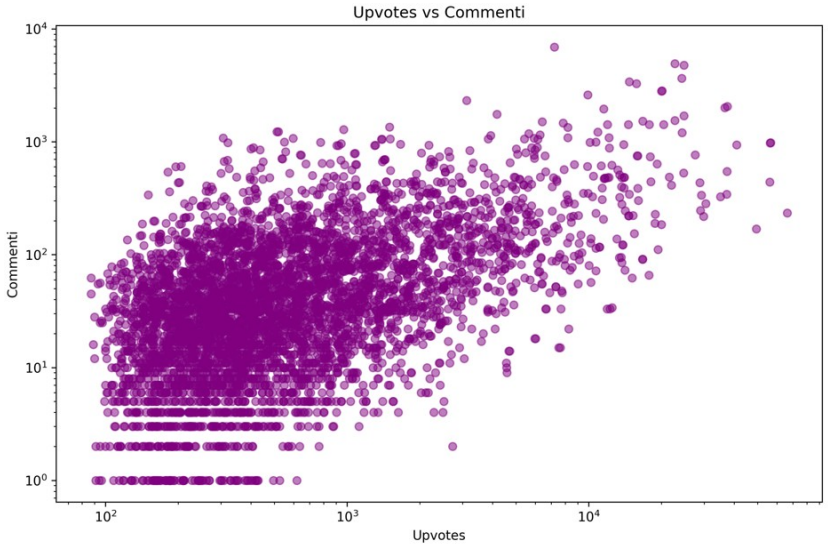

Figure 3: Grafico upvotes/commenti

3 Apprendimento non Supervisionato

L'apprendimento non supervisionato è una tecnica dell'apprendimento automatico in cui la macchina analizza e raggruppa insieme di dati non etichettati. L'obbiettivo è trovare pattern nascosti senza l'intervento umano. Gli approcci principali nell'apprendimento non supervisionato sono i seguenti:

- **Riduzione della dimensionalità:** L'obiettivo è semplificare i dati riducendo il numero di feature cercando di mantenere le informazioni più significative.

  Il **PCA**(Principal Component Analysis) è una delle tecniche più usate in questo ambito. Crea nuove feature, chiamate componenti principali, che siano combinazione lineare delle feature originali in base alla varianza. L'idea è che le componenti principali con varianza maggiore catturano più informazioni sul dataset, vogliamo considerarle e trascurare le componenti con varianza bassa.

  Nel nostro caso, il PCA permette la visualizzazione dei Cluster sotto forma di grafico, riducendo la dimensionalità delle feature.

- **Clustering:** In questa caso invece l'obiettivo è di raggruppare i dati in base alle loro somiglianze. Dopo averli analizzati si dividono in gruppi, detti Cluster, cercando di trovare pattern nascosti. Esistono due tipi di clustering: hard clustering, in cui per ogni esempio è assegnato un cluster specifico, e soft clustering, in cui per ogni esempio invece è assegnata una probabilità di appartenenza ad un determinato cluster.

  Il **K-means** è un algoritmo di clustering hard, che divide i dati in k cluster. Ogni cluster ha un centroide, e ogni elemento del dataset viene assegnato al centroide più vicino. Si prosegue aggiornando centroide e assegnazioni finchè i gruppi diventano stabili e quindi non cambiano più.

3.1 Feature Utilizzate

Le feature da utilizzare sono caricate dal file creato dalla fase precedente, e le features selezionate sono:

- `upvote_ratio`
- `author_impact`
- `post_hour`
- `title_length`
- `upvotes`
- `num_comments`

Tutte le feature sono salvate nel dataframe `df`.

3.2 Standardizzazione

In questa fase bisogna rendere le feature confrontabili, dato che molte hanno ancora scale diverse. In questo modo possiamo migliorare la convergenza degli algoritmi che saranno usati successivamente.

1. Come prima cosa sono selezionate le colonne delle feature corrette specificate nella lista `feature`.
2. Successivamente i dati sono ordinati e avviene la sostituzione dei valori mancanti con `NaN` (comando `.fellina(...)`).

   A questo punto abbiamo una serie di valori per le feature e si ha la possibilità di riscontrare i valori nulli, di conseguenza viene calcolata la mediana per ogni colonna.
3. Il comando `.median()` si occupa proprio di questo, in fine i valori `NaN` sono sostituiti con la mediana della feature di cui fanno parte.

   Si è deciso di considerare la mediana perchè la media sarebbe stata compromessa dai valori `NaN`.
4. In fine applichiamo la standardizzazione.

$$X_{\text{scaled}} = \frac{X - \mu}{\sigma}$$

   - μ: media della colonna
   - σ: deviazione standard della colonna
   - X: valore originale

In questo modo ogni feature avrà media 0 e derivazione standard 1.

```
# Feature selezionate
features = [
    'upvote_ratio',
    'author_impact',
    'post_hour',
    'title_length',
    'upvotes',
    'num_comments'
]

# Standardizzazione
X = df[features].fillna(df[features].median())
X_scaled = StandardScaler().fit_transform(X)
```

Figure 4: Codice della Standardizzazione

3.3 Clustering

Adesso siamo in grado di applicare un algoritmo di Clustering, ma prima bisogna trovare il numero di cluster ottimale per il nostro caso.

3.3.1 Ricerca k ottimale

Per trovare il numero di cluster ottimale si è pensato di avviare l'algoritmo K-means in base alle componenti che abbiamo, e tramite lo score di confrontare i vari risultati. La metrica che si usa per confrontare e valutare la qualità del clustering è il **Silhouette Score**. Sono eseguiti i seguenti passi:

- Per prima cosa si esegue il K-means all'interno di un ciclo, facendo variare k da 2 a 9, ovvero $k = 2, 3, \ldots, 9$.
- Successivamente si calcola il Silhouette Score su `X_scaled` e `labels`, dove:
  - `X_scaled`: è un array multidimensionale in questo caso a 6 dimensioni che contiene le coordinate dei punti nel piano.
  - `labels`: è un array di interi in cui ogni elemento è l'indice di cluster assegnato dal K-means.

  Per ogni punto i in `X_scaled`, `silhouette_score` esegue i seguenti passaggi:

- Calcola $a(i)$: la distanza media tra i e gli altri punti del suo cluster.
- Calcola $b(i)$: la distanza media tra i e i punti vicini al suo cluster ma diversi dal suo.
- Lo score è ottenuto da:

$$s(i) = \frac{b(i) - a(i)}{\max\big(a(i), b(i)\big)}$$

- Lo score complessivo è ottenuto facendo la media di tutti gli $s(i)$.

Nel ciclo sono conservati gli score complessivi e sono confrontati tra loro, dopo aver completato tutte le iterazioni è restituito il k ottimale.

Inizialmente la ricerca del k ottimale era stata implementata tramite la regola del gomito; il risultato ottenuto era di 4 cluster, uno dei quali ricopriva la categoria dei post molto virali, in totale 98 su 5000 post. Nelle successive fasi di apprendimento, i risultati ottenuti da questa fase saranno utilizzati per definire la feature target `is_viral` basata sulle caratteristiche dei cluster dei post virali Definzione is_viral (4.1).

Nelle fasi successive, l'obiettivo è classificare un post come virale/non virale, quindi una classificazione binaria, con un numero così basso di esempi per la classe target, l'apprendimento sarebbe stato impossibile. Proprio per questa ragione è stato scelto il Silhouette Score, che spesso preferisce pochi cluster ma ben separati, mentre la regola del gomito cerca il punto in cui la riduzione dell'errore rallenta, se la curva non ha un gomito netto, potrebbe restituire più cluster anche se non sono ben separati.

3.3.2 K-Means

Ottenuto il k ottimale si esegue K-means finale che andrà a creare i nostri cluster. In questo caso il k ottimale ottenuto precedentemente è 2, di conseguenza l'algoritmo è impostato a quel valore in automatico.

Avvengono i seguenti passaggi:

- Sono inizializzati 2 punti, chiamati centroidi, in modo casuale.
- Per ogni punto i in `X_scaled`, viene calcolata la distanza dai centroidi.
- Ogni punto è assegnato al cluster con il centroide più vicino.
- Adesso avviene il ricalcolo del centroide di ogni cluster come media dei punti assegnati.

I due passaggi finali sono ripetuti fino a quando i centroidi non cambiano o si è raggiunto un numero massimo di iterazioni.

3.3.3 Risultati Finali

I 2 cluster individuati sono i seguenti:

- **Cluster 0: "Contenuti Non Virali"**
  - **Caratteristiche principali:**
    - `upvote_ratio` = 0.98 → Tasso di approvazione molto alto (98% upvote)
    - `upvotes` = 637.63 (media) / 374 (mediana) → Engagement contenuto
    - `title_length` = 45.61 caratteri (media) → Titoli relativamente brevi
    - `num_comments` = 47.36 (media) → Discussioni limitate
    - `author_impact` = 8.92 → Autori mediamente influenti
    - `post_hour` = 17:00 (mediana)
    - `count` = 4193
  - **Interpretazione:**
    Gli utenti apprezzano questa tipologia di post ma interagiscono poco, sia in termini di commenti che upvotes.

- **Cluster 1: "Contenuti Virali"**
  - **Caratteristiche principali:**
    - `upvote_ratio` = 0.92 → Alto tasso di approvazione (92% upvote)
    - `upvotes` = 4623.87 (media) / 2205 (mediana) → Engagement circa 7.3× superiore al Cluster 0
    - `title_length` = 73.94 caratteri (media) → Titoli significativamente più lunghi
    - `num_comments` = 341.99 (media) → Discussioni molto attive (7.2× Cluster 0)
    - `author_impact` = 9.81 → Autori più influenti
    - `post_hour` = 13:00 (mediana)
    - `count` = 807
  - **Interpretazione:**
    Questo cluster rappresenta contenuti virali, con un engagement molto elevato e una partecipazione molto attiva nella sezione dei commenti.

3.3.4 Grafici

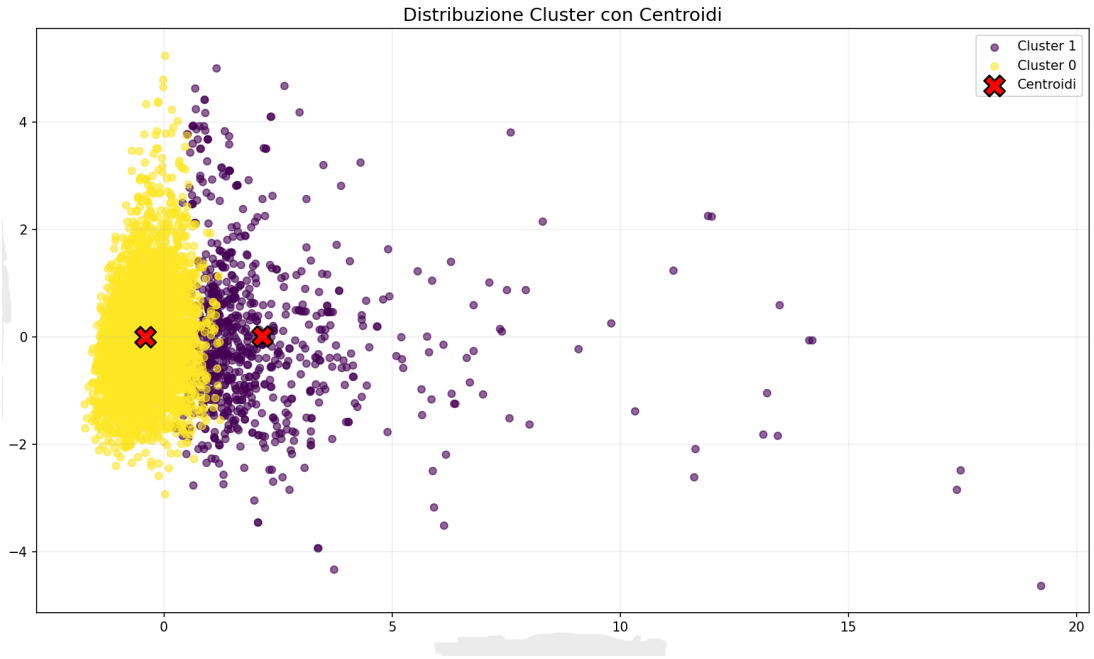

Figure 5: Distribuzione dei due cluster

3.3.5 Conclusioni

Nello svolgimento dell'apprendimento non supervisionato, ci sono stai dei problemi dovuti all'inserimento delle feature che determinavano il tipo dei posts. Infatti, l'algoritmo si limitava semplicemente a raggruppare contenuti dello stesso tipo in un unico cluster, senza ricavare pattern utili. Il problema è stato risolto escludendo queste feature, e siccome si voleva comunque vedere la distribuzione delle categorie nei clusters, i dati sono stati estratti successivamente e sono i seguenti:

Tipo di contenuto	**Cluster 0**	**Cluster 1**
`image`	$0.59 \rightarrow 59\%$	$0.48 \rightarrow 48\%$
`video`	$0.14 \rightarrow 14\%$	$0.16 \rightarrow 16\%$
`other`	$0.23 \rightarrow 23\%$	$0.22 \rightarrow 22\%$
`text`	$0.05 \rightarrow 5\%$	$0.13 \rightarrow 13\%$

Table 1: Distribuzione dei tipi di contenuto nei due clusters

Anche la feature riguardante il giorno di pubblicazione è stato evitato sempre per le motivazioni precedenti.

Tutto questo avviene perchè queste feature anche dopo la normalizzazione non assumano valori continui e quindi il sistema tende a raggrupparli insieme con pattern banali e già noti.

Dal risultato possiamo vedere che entrambi i cluster sono formati per la maggior parte da immagini., il tipo di contenuto non sembra che abbia un ruolo importante nella determinazione dei contenuti virali, ma il contenuto immagine è il più diffuso nel dataset (come abbiamo già visto in precedenza).

4 Apprendimento Supervisionato

L'apprendimento supervisionato è una tecnica del Machine Learning in cui l'agente viene addestrata su un insieme di dati etichettati. Questo tipo di apprendimento di solito è utilizzato per risolvere problemi di:

- **Classificazione:** L'obbiettivo è predire un'etichetta, che deve assumere valori finiti e sono definiti inizialmente. Avendo le feature di input, il modello sarà in grado di classificarle in una delle categorie.
- **Regressione:** In questo caso invece l'etichetta assume valori continui, di conseguenza l'obbiettivo è prevedere questo valore.

Nel progetto è affrontato il problema della classificazione binaria, cioè un caso speciale in cui le classi possono assumere due valori (`'true'`,`'false'`). Più specificatamente nel nostro caso questi valori andranno a determinare se un post è virale/non virale.

La fase di apprendimento non supervisionato ci ha fornito dei parametri per i post che potrebbero essere virali, naturalmente non è una regola sempre valida ma si vuole comunque considerare i dati ottenuti per le definizione dei post virale.

4.1 Definzione is_viral

Il sistema dovrà predire l'etichetta `is_viral`, come già detto in precedenza queste soglie vanno a considerare le caratteristiche che ha prodotto il cluster 1 nella fase precedente, quindi un post per essere virale dovrà avere almeno 3 delle seguenti condizioni vere:

$$\begin{aligned} \texttt{upvote\_ratio} &\geq 0.92 \\ \texttt{author\_impact} &\geq 9.8 \\ \texttt{upvotes} &\geq 2200 \\ \texttt{num\_comments} &\geq 180 \\ \texttt{title\_length} &\geq 40 \end{aligned}$$

4.2 Feature Utilizzate

In questa fase di apprendimento consideriamo anche altre feature che sono state escluse, e sono le seguenti:

- **Feature base**

  Queste feature catturano le metriche base dell'engagement di un post.

  - `upvotes`
  - `num_comments`
  - `upvote_ratio`

- **Feature autore**

  Unica feature che cattura l'impatto di un autore, la gestione è stata già descritta in precedenza nel preprocessing del dataset.

  - `author_impact`

- **Feature testo**

  Queste feature descrivono le caratteristiche testuali del titolo del post.

  - `title_length`
  - `has_question_mark`

- **Feature tempo**

  Queste feature rappresentano informazioni temporali legate alla pubblicazione del post.

  - `post_hour_sin, post_hour_cos`
    La feature `post_hour` rappresenta l'ora di pubblicazione del post (0 - 23). Questo può creare problemi per i modelli che saranno utilizzati, abbiamo la possibilità di interpretare erroneamente che 23 e 0 siano molti distanti ma questo in realtà non è vero perchè in questo caso abbiamo una variabile ciclica. Per evitare questa discontinuità e rappresentare la ciclicità di solito l'ora è gestita attraverso le seguenti formule:

$$\text{post\_hour\_sin} = \sin\left(\frac{2\pi \cdot \text{post\_hour}}{24}\right)$$

$$\text{post\_hour\_cos} = \cos\left(\frac{2\pi \cdot \text{post\_hour}}{24}\right)$$

In questo modo l'ora è formata dalla coppia $(\sin, \cos)$ in radianti su un cerchio e di conseguenza due ore vicine avranno vettori vicini nello spazio Euclideo, infatti:

$$\begin{aligned} 0 &\rightarrow (0.0, 1.0) \\ 6 &\rightarrow (1.0, 0.0) \\ 12 &\rightarrow (0.0, -1.0) \\ 18 &\rightarrow (-1.0, 0.0) \\ 23 &\rightarrow (-0.26, 0.97) \end{aligned}$$

- `is_weekend` L'engagement sui social è spesso maggiore nel weekend, quindi è stata aggiunta una feature booleana che controlla se la feature grezza `post_weekday == 5 o 6`; in tal caso, il post è pubblicato di sabato o domenica e `is_weekend = 1`.

- **Feature tipo di contenuto**

  In fine le ultime feature rappresentano il tipo di contenuto del post. Sono ricavate nella fase di preprocessing del dataset, questo perchè i modelli che andremo ad utilizzare richiedono input numerici quindi la feature grezza `content_type` è processata in 4 colonne binarie.

  - `content_image`
  - `content_video`
  - `content_text`
  - `content_other`

4.3 Implementazione Modelli

Per classificare i post virali sono stati scelti e analizzati 3 modelli per avere punti di vista diversi e capire il modello più adatto al nostro caso.

4.3.1 SMOTE

Ogni modello è stato testato con e senza l'utilizzo di **SMOTE**(Synthetic Minority Oversampling Technique).

Il dataset, per natura del problema che stiamo cercando di risolvere sarà sempre sbilanciato, ovvero i post virali sono molto meno frequenti di quelli non virali, i modelli tendono ad imparare poco su quella classe causando così prestazioni molto basse.

SMOTE è una tecnica che si utilizza per diminuire lo sbilanciamento nella classe minoritaria: seleziona casualmente uno o più vicini più prossimi al campione della classe minoritaria e crea nuovi esempi posizionati lungo la linea che li unisce, l'importante che questo nuovo punto non sia già esistente. Il nuovo punto creato sarà l'esempio sintetico che aiuterà al sistema ad apprendere. Questa tecnica funziona solo se nella classe minoritaria ci sono almeno due esempi, questo dipende dal train-test-split, quindi in certe iterazioni SMOTE può essere saltato.

4.4 Suddivisione Dataset

Nel processo di addestramento, validazione e test il dataset è suddiviso più volte, questa suddivisione rimane la stessa per ogni modello.

1. **Suddivisione iniziale del dataset:** Dopo il caricamento delle features descritte in precedenza il dataset è diviso in **Training Set** e **Test Set**. La prima parte si occuperà di addestrare i modelli mentre la seconda di testare le prestazioni dei modelli. Più specificatamente la suddivisione dei dati è la seguente:

   - **80%:** : Utilizzati per il training set (3996 campioni).
   - **20%:** Utilizzati per il test set (999 campioni).

   Tramite il parametro `stratify = y` è garantita la distribuzione della feature target (`is_viral`) sia nel training set che nel test set.

2. **Suddivisione del Training Set nella Nested Cross-Validation:** Ogni modello ha i suoi iperparametri, e scegliere quelli giusti non è affatto semplice; questa decisione influisce notevolmente sulle prestazioni del sistema. Per trovare la configurazione ottimale dei modelli, si utilizza la Nested Cross-Validation.

   Questa tecnica divide il training set ulteriormente seguente modo:

   - **Outer Loop:** Tramite la `StratifiedKFold` sono creati k (nel nostro caso 5) fold di valutazione esterna, ogni fold viene utilizzato come test mentre gli altri addestrano il modello.

- **Inner Loop:** Per ogni fold rimanente dall'outer loop esegue una `GridSearchCV` per ottimizzare gli iperparametri.

La `GridSearchCV` implementa al suo interno la `K-Fold Cross-Validation`, sono algoritmi complementari ma con scopi diversi:

- **K-Fold Cross-Validation:** Si occupa di valutare le prestazioni del modello su diverse divisione del dataset. Divide il dataset in k fold, in questo caso 5, addestra il modello sul primo fold e lo testa sull'ultimo, ripete il processo per tutti i fold e calcola la media in base ad un metrica di valutazione.

  In questo caso è stata utilizzata la metrica **Accuracy**, siccome stiamo lavorando sui fold nell'inner loop questo valore è calcolato per ogni fold nel seguente modo:

  $$\text{Accuracy}_{\text{fold}} = \frac{\text{Numero di predizioni corrette nel fold}}{\text{Numero totale di campioni nel fold}}$$

  Viene calcola la media dell'accuracy su tutti i fold per ogni combinazione di iperparametri.

- **GridSearch:** Il suo scopo invece è ottimizzare gli iperparametri del modello e valutare con K-Fold Cross-Validation. Effettua una ricerca esaustiva su una griglia (`params`) di iperparametri, per ogni combinazione esegue K-Fold Cross-Validation e seleziona la combinazione con il migliore punteggio medio ottenuto sui fold.

Il risultato della GridSearchCV viene testato sul fold riservato in precedenza (outer loop) e il processo deve essere eseguito nuovamente sul fold successivo fino all'ultimo. Alla fine sarà utilizzato il migliore risultato ottenuto dalle vari fasi della Nasted Cross-Validation.

Questa tecnica viene utilizzata sui vari modelli per la ottimizzazione degli iperparametri descritti in seguito.

Inoltre nella Nested Cross-Validation è calcolata la media e la deviazione standard:

- **Media:** La media è una stima affidabile delle prestazioni del sistema dato che è calcolata nell'outer loop con dati non utilizzati per l'addestramento e per l'ottimizzazione degli iperparametri. Un valore alto della media indica che il modello generalizza bene.

- **Deviazione Standard:** Misura quanto sono distanti i risultati ottenuti dell'outer loop dalla media. Un valore basso della deviazione standard indica che il modello è stabile e ha buone prestazioni.

4.4.1 Random Forest

La Random Forest è un modello dell'Ensemble Learning,in particolare appartiene alla categoria degli algoritmi di Bagging. In questo approccio sono combinati più modelli deboli (weak learner) per ottenere un nuovo modello più robusto. In questo caso sono utilizzati più alberi di decisione addestrati su parti diverse del dataset, ogni esempio avrà una predizione e la predizioni sono unite insieme per creare la predizione finale per l'esempio in questione.

4.4.1.1 Iperparametri Gli iperparametri sono configurazioni che non vengono apprese automaticamente dai dati ma devono essere gestiti manualmente e impostati prima dell'addestramento.

```
models_config = {
    'RandomForest': {
        'model': RandomForestClassifier(random_sta
        'params': {
            'n_estimators': [100, 150],
            'max_depth': [4, 6, 8],
            'min_samples_split': [10, 20, 30],
            'max_features': ['sqrt', 0.5]
        }
    },
```

Figure 6: Random Forest Codice

- `n_estimators`: [100, 150] — Numero di alberi.
- `max_depth`: [4, 6, 8] — Massima profondità degli alberi.
- `min_samples_split`: [10, 20, 30] — Minimo numero di esempi di training per dividere un nodo.
- `max_features`: [sqrt, 0.5] — Numero di feature considerate per ogni split.

I valori elencati in precedenza sono configurazioni di base, tramite il processo descritto in precedenza sono ottimizzati e impostati i migliori per il modello finale.

4.4.1.2 Risultati

I risultati ottenuti durante l'addestramento sono i seguenti:

Table 2: Confronto tra modelli con e senza SMOTE

Metriche	**Con SMOTE**	**Senza SMOTE**
AUC-ROC (TEST)	0.9998	0.9996
Accuracy (TEST)	0.9930	0.9940
Nested CV Accuracy	0.9906 ± 0.0038	0.9945 ± 0.0020

Table 3: Report con SMOTE

Classe	**Precision**	**Recall**	**F1-score**	**Support**
0	0.9986	0.9915	0.9950	706
1	0.9799	0.9966	0.9882	293

Table 4: Report senza SMOTE

Classe	**Precision**	**Recall**	**F1-score**	**Support**
0	0.9986	0.9929	0.9957	706
1	0.9832	0.9966	0.9898	293

Entrambi i modelli hanno delle prestazioni ottime, con accuratezza e AUC-ROC che superano il 99%. Esaminando in più le metriche nella tabella 2 possiamo vedere che il modello senza SMOTE ha un'accuratezza leggermente migliore sia nel test set (0.9940 vs 0.9930) che nella Nested CV Accuracy (0.9945 ± 0.0020 vs 0.9906 ± 0.0038). L'ultimo dato è molto più significativo perchè offre una stima più affidabile delle prestazioni sui nuovi dati. Analizzando anche la deviazione standard più bassa (± 0.0020 vs ± 0.0038) possiamo concludere che il modello senza SMOTE è più stabile e generalizza meglio.

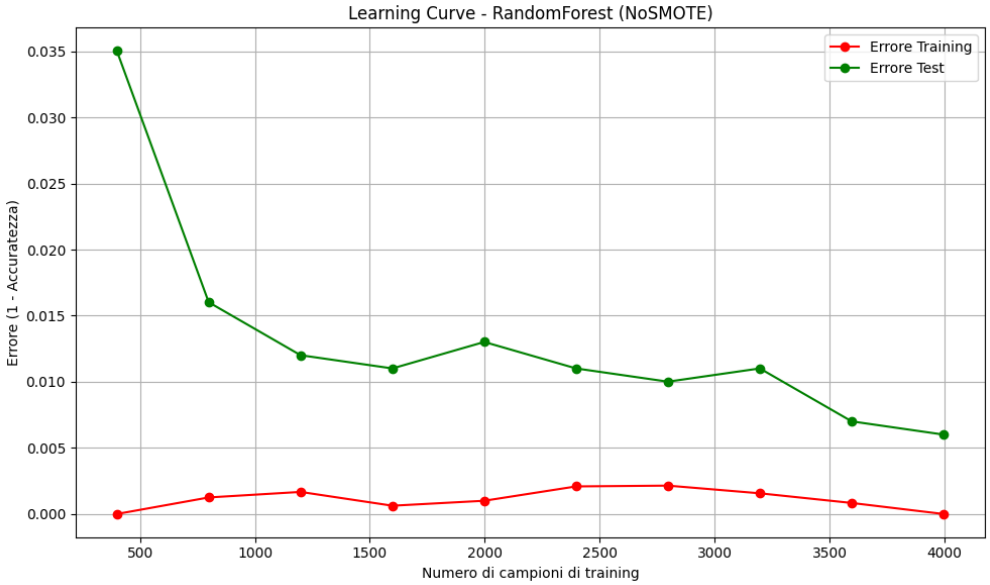

Figure 7: Learning Curve Senza SMOTE

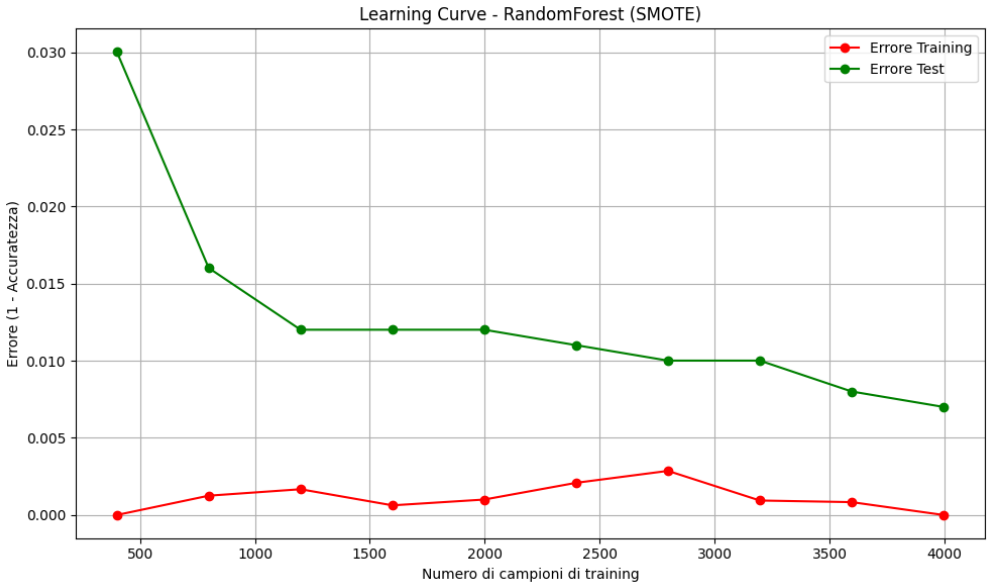

Figure 8: Learning Curve Con SMOTE

4.4.2 AdaBoost

Come RandomForest anche AdaBoost è un modello dell'Ensemble Learning, ma in questo caso appartiene alla categoria degli algoritmi di Boosting. Il funzionamento è simile al caso precedente, quindi l'idea è sempre combinare dei modelli deboli per creare un modello finale più robusto ma questi weak learners sono combinati in sequenza, e ogni modello successivo si concentra sugli errori dei modelli precedenti, assegnando pesi più grandi ai campioni difficili da classificare. La predizione finale è la somma ponderata delle predizioni dei modelli deboli.

4.4.2.1 Iperparametri

```
},
'AdaBoost': {
    'model': AdaBoostClassifier(random_state=RANDOM_STATE),
    'params': {
        'n_estimators': [50, 100],
        'learning_rate': [0.1, 0.5],
        'estimator': [
            DecisionTreeClassifier(max_depth=2, random_state=RANDOM_STATE),
            DecisionTreeClassifier(max_depth=3, random_state=RANDOM_STATE)
        ]
    }
```

Figure 9: AdaBoost Codice

- `n_estimators`: [100, 150] — Numero di alberi.
- `learning_rate`: [0.1, 0.5] — Tasso di apprendimento.
- `estimator`: Specifica il modello base per creare gli alberi deboli, in questo caso sono alberi decisionali di profondità massima 2 o 3, in questo modo ogni albero è semplice e diminuisce l'overfitting.

Table 5: Confronto tra modelli AdaBoost con e senza SMOTE

Metriche	**Con SMOTE**	**Senza SMOTE**
AUC-ROC (Test Set)	0.9999	1.0000
Accuracy (Test Set)	0.9950	0.9990
Nested CV Accuracy	0.9954 ± 0.0015	0.9992 ± 0.0010

Table 6: Classification Report AdaBoost con SMOTE

Classe	**Precision**	**Recall**	**F1-score**	**Support**
0	1.0000	0.9929	0.9964	706
1	0.9832	1.0000	0.9915	293

Table 7: Classification Report AdaBoost senza SMOTE

Classe	**Precision**	**Recall**	**F1-score**	**Support**
0	1.0000	0.9986	0.9993	706
1	0.9966	1.0000	0.9983	293

4.4.2.2 Risultati Entrambi i modelli mostrano performance eccellenti, accuratezza sul test set del 99.5% (con SMOTE) e del 99.9% (senza SMOTE) lo stesso vale per AUC-ROC di 0.9999 e 1.0000. Osservando la deviazione standard della Nested CV nei due modelli possiamo vedere che il modello senza SMOTE non è solo più accurato ma anche più stabile.

In conclusione, entrambi i modelli hanno performance quasi perfette, ma il modello addestrato senza SMOTE si dimostra superiore in ogni metrica. L'uso di SMOTE in questo non è necessario e risulta solo controproducente. I learning curve mostrate in seguito, confermano che il modello con SMOTE è in overfitting, con un errore di training bassissimo e un errore di test molto più alto che non converge, mentre il modello senza SMOTE mostra un errore di test basso e stabile che converge con quello di training.

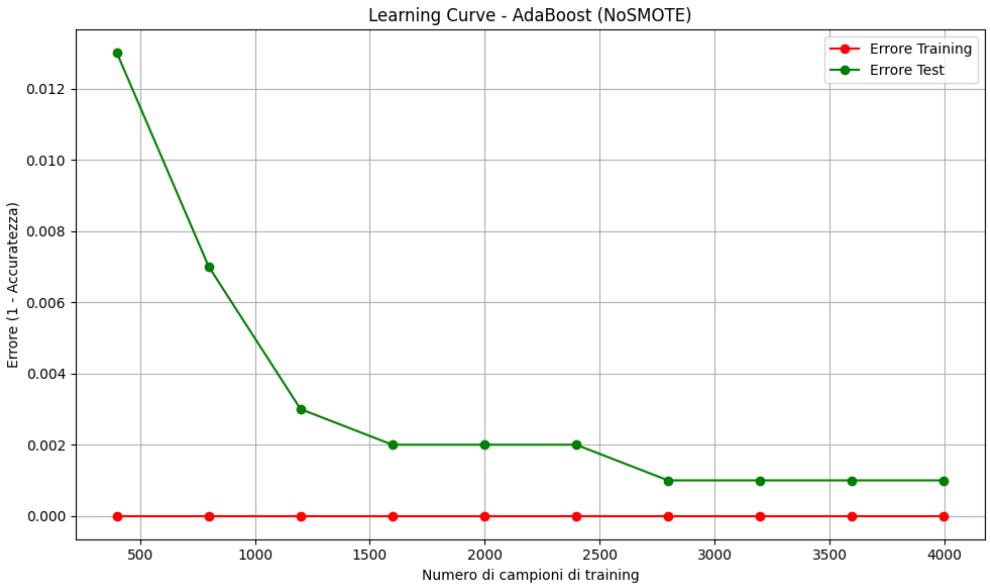

Figure 10: Learning Curve Senza SMOTE

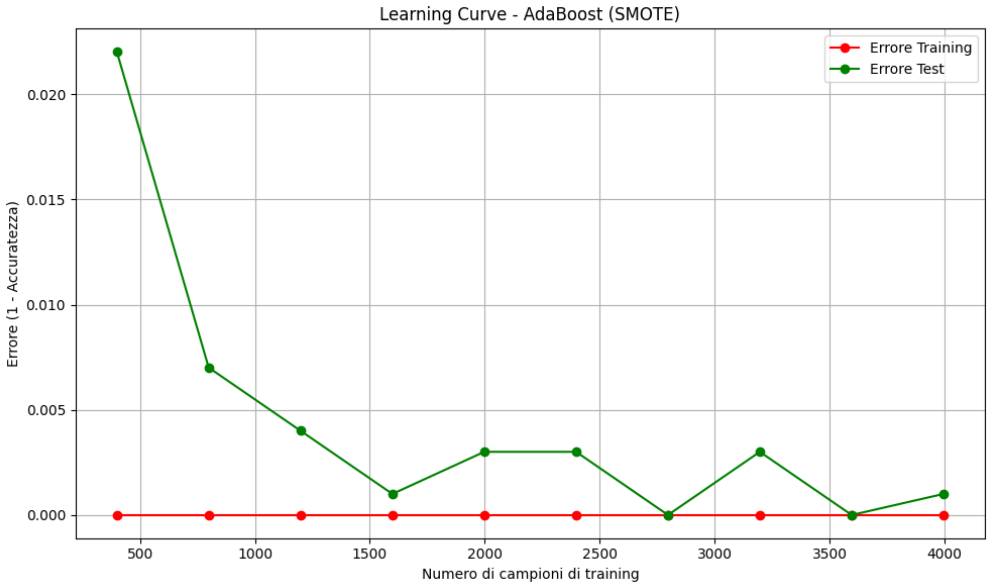

Figure 11: Learning Curve Con SMOTE

4.4.3 Support Vector Classifier

L'SVC è un algoritmo di apprendimento supervisionato, fa parte della famiglia dei modelli basati su Support Vector Machines. Può essere esteso anche per problemi di regressione (Support Vector Regression), ma in questo caso per ovvie ragioni utilizziamo la versione per problemi di classificazione.

Questo algoritmo ha un approccio completamente diverso dai due modelli precedenti. Cerca un iperpiano che separa le classi nel dataset. Questo iperpiano è scelto in modo da massimizzare il margine tra i punti delle classi più vicine all'iperpiano anche chiamate support vectors. Se i dati non sono linearmente separabili allora utilizza un kernel (in questo caso RBF) i modo da trasformare i dati in uno spazio di dimensioni maggiore. Dopo l'addestramento i nuovi dati sono classificati in base alla loro posizione rispetto all'iperpiano.

4.4.3.1 Iperparametri

```
},
'SVC': {
    'model': SVC(probability=True, kernel='rbf', class_weight='balanced',
    'params': {
        'C': [0.1, 1, 10],
        'gamma': ['scale', 0.01, 0.1, 1]
    }
```

Figure 12: SVC Codice

- `C`: [0.1, 1, 10] — Rappresenta il trade-of tra la massimizzazione e la minimizzazione degli errori di classificazione. Un valore altro di C cercherà di classificare correttamente tutti i punti ma può portare ad overfitting.
- `gamma`: ['scale', 0.01, 0.1, 1] — Verifica la forma del kernel RBF (Radial Basis Function) per mappare i dati in uno spazio di dimensione superiore in modo da renderli separabili.

Table 8: Confronto tra modelli SVC con e senza SMOTE

Metriche	**Con SMOTE**	**Senza SMOTE**
AUC-ROC (Test Set)	0.8499	0.8684
Accuracy (Test Set)	0.7858	0.8058
Nested CV Accuracy	0.8438 ± 0.0065	0.8071 ± 0.0077

Table 9: Classification Report SVC con SMOTE

Classe	**Precision**	**Recall**	**F1-score**	**Support**
0	0.9256	0.7578	0.8333	706
1	0.5938	0.8532	0.7003	293

Table 10: Classification Report SVC senza SMOTE

Classe	**Precision**	**Recall**	**F1-score**	**Support**
0	0.8596	0.8669	0.8632	706
1	0.6725	0.6587	0.6655	293

4.4.3.2 Risultati I risultati ottenuti sono pessimi, modello addestrato con SMOTE mostra chiari segni di overfitting le curve de learning mostrano un errore nel training set basso me la curva sul test set è molto più alta e non converge. Questo significa che il modello ha memorizzato i dati di training set ma non generalizza sui nuovi dati.

La precisione sulla classe 1 (0.59) cioè i post virali, è molto bassa il modello sbaglia quasi una volta su due questo rende questa soluzione quasi inutilizzabile per il nostro scopo.

Al contrario, il modello addestrato senza SMOTE è molto più affidabile, le learning curve non sono perfette, ma mostrano un comportamento normale senza overfitting, questo indica una buona capacità di generalizzazione. Le sue performance sul test set sono migliori e più equilibrate (Accuracy: 80.58%, AUC: 0.8684). Ma anche in questa soluzione la precisione per la classe 1 è troppo bassa.

Report completo dei vari modelli nel file `supervised_report.txt`

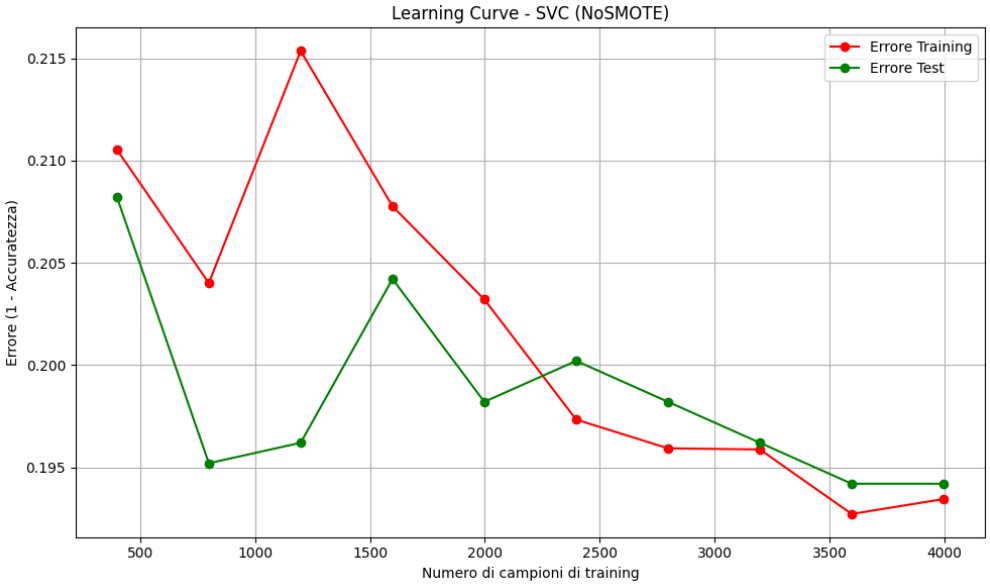

Figure 13: Learning Curve Senza SMOTE

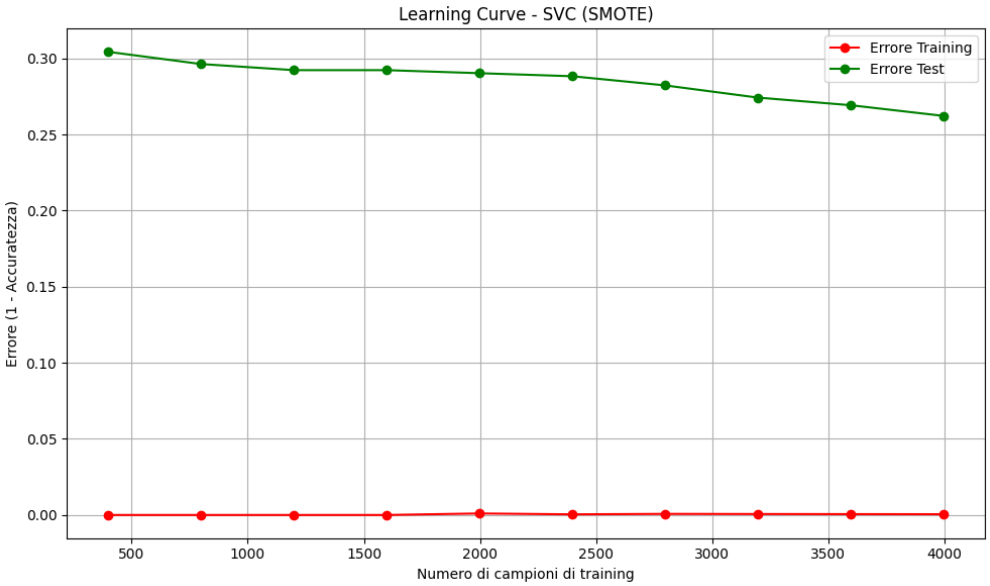

Figure 14: Learning Curve Con SMOTE

4.5 Conclusioni

Abbiamo visto diverse soluzioni, il modello migliore per il nostro problema è sicuramente AdaBoost senza SMOTE, inoltre la tecnica SMOTE non ha aiutato nessun modello ad apprendere meglio, anzi ha sempre peggiorato le prestazioni.

Il modello RandomForest mostra anche prestazioni ottime nel rilevare i post virali ma non paragonabili con le performance di AdaBoost.

5 Ragionamento Probabilistico

Il ragionamento probabilistico è un approccio usato in machine learning per gestire l'incertezza presente nei dati e nelle decisioni. Come abbiamo visto dalle fasi precedenti i dati non sono sempre perfetti e possono essere incompleti, ambigui o in presenza di rumore.

Questo tipo di ragionamento si basa sulla teoria della probabilità, in particolare sfrutta le varie dipendenze e indipendenze tra variabili. Una della applicazioni più note sono le reti Bayesiane, che si basano sul teorema di Bayes.

Dalle fasi precedenti abbiamo ricavato una definizione di `is_viral` con questo tipo di ragionamento si vuole provare ad effettuare una classificazione binaria probabilistica, più specificatamente si calcola probabilità che l'etichetta `is_viral == 1`, cioè la probabilità che il post sia virale in base alle sue caratteristiche.

5.1 Reti Bayesiane

Le reti Bayesiane sono rappresentate tramite dei grafi orientati aciclici, dove ogni nodo rappresenta una variabile e gli archi indicano le relazioni di dipendenza probabilistica con le altre variabili.

Una delle tecniche più utilizzate per apprendere in modo automatico la struttura di una rete Bayesiana è **Hill Climb Search**. Questo è un algoritmo di ricerca locale che parte da un grafo iniziale (vuoto o casuale), e ad ogni iterazione valuta le mosse locali (aggiunta, rimozione o inversione di un arco) in base ad una funzione di punteggio, si sceglie la mossa migliore, in fine si ferma quando è raggiunto un ottimo locale.

5.1.1 Hill Climb Search

Per l'implementazione di questo algoritmo si utilizza la libreria `pgmpy`, in generale dalle implementazioni fornite questo algoritmo non supporta direttamente variabili continue ma lavora con variabili discrete/categoriali.

Per come è composto il dataset senza una fase di discretizzazione delle variabili l'algoritmo necessita di troppo tempo e memoria per essere eseguito. La discretizzazione è il processo che trasforma le variabili continue in categorie o intervalli, se questa fase non è eseguita nel modo corretto potrebbe portare ad errori o al fallimento dell'algoritmo.

L'idea iniziale era di avviare automaticamente l'algoritmo subito dopo la discretizzazione per apprendere la struttura della rete in modo automatico ma i problemi di tempo e memoria non sono stati risolti da questo processo. A causa di queste problematiche è stata presa la scelta di implementare una prima versione che non prevede l'implementazione di Hill Climb Search, in modo da poter dichiarare manualmente la rete Bayesiana.

5.1.2 Modello Base

Anche in questo tipo di apprendimento vogliamo considerare la stessa definzione di `is_viral` riportata in Definzione is_viral (4.1), in modo da avere coerenza con gli altri modelli.

Naturalmente le variabili che hanno sicuramente relazioni con la viralità di un post sono quelle utilizzate per la sua definizione, sono le seguenti con la relativa discretizzazione:

```python
def discretize_variables(df):
    # Discretizzazione delle variabili
    bins_dict = {
        'upvotes': [0, 2000, 4000, np.inf],
        'num_comments': [0, 180, 500, np.inf],
        'upvote_ratio': [0, 0.5, 0.75, 0.92, 1.0],
        'title_length': [0, 40, 80, np.inf],
        'author_impact': [0, 5, 9.8, np.inf]
    }
```

Figure 15: Discretizzazione delle variabili

A questo punto sono state create manualmente due tipi di reti Bayesiane per verificare se le ipotesi sulle relazioni sono valide.

5.1.2.1 Rete Simple Questa è la soluzione più banale dove semplicemente andiamo ad mettere in relazione le quattro variabili precedenti con la variabile target `is_viral`.

```python
def define_bayesian_model_simple():
    model = DiscreteBayesianNetwork([
        ('upvotes', 'is_viral'),
        ('num_comments', 'is_viral'),
        ('upvote_ratio', 'is_viral'),
        ('title_length', 'is_viral'),
        ('author_impact', 'is_viral')
    ])
    return model
```

Figure 16: Rete Simple

5.1.2.2 Rete Full In questa soluzione invece abbiamo una struttura più complessa in cui oltre alle relazioni dirette con `is_viral` vengono aggiunti anche archi tra le feature stesse.

```
def define_bayesian_model_full():
    model = DiscreteBayesianNetwork([
        ('upvotes', 'is_viral'),
        ('num_comments', 'is_viral'),
        ('upvote_ratio', 'is_viral'),
        ('title_length', 'is_viral'),
        ('author_impact', 'is_viral'),

        ('upvotes', 'num_comments'),
        ('upvotes', 'upvote_ratio'),
        ('author_impact', 'upvotes'),
        ('author_impact', 'num_comments'),
        ('num_comments', 'upvote_ratio'),
        ('title_length', 'num_comments')
    ])
    return model
```

Figure 17: Rete Full

Adesso possiamo confrontare le due soluzioni per vedere se le relazioni in più del modello full sono utili.

5.1.2.3 Risultati Per valutare i due modelli è stato adottato un approccio simile a quello della fase precedente: il dataset è stato suddiviso in training set e test set, e i due modelli sono stati addestrati sullo stesso training set e valutati sullo stesso test set.

Table 11: Confronto dei risultati tra il modello Full e il modello Simple

Metrica	**Modello Full**	**Modello Simple**
Accuracy	0.9810	0.9810
Precision	0.9757	0.9757
Recall	0.9590	0.9590
F1-score	0.9673	0.9673
Binary Cross-Entropy	0.0851	0.0851

Abbiamo anche l'introduzione di una nuova metrica utilizzata per valutare questa tipologia di modelli probabilistici:

- **Binary Cross-Entropy (o log-loss):** è una funzione di costa utilizzata di solito per valutare quanto bene un modello probabilistico di classificazione binaria stima la probabilità della classe target.

  Per ogni esempio si calcola:

  $$\text{BCE} = -\big[y \cdot \log(p) + (1 - y) \cdot \log(1 - p)\big]$$

  dove:

  - $y \in \{0, 1\}$ è l'etichetta vera del campione.
  - $p \in [0, 1]$ è la probabilità predetta dal modello che la classe sia 1.

  In fine si effettua la media su tutti gli esempi del test set.

  Se il modello è corretto, la BCE assume valori bassi; altrimenti, se la BCE è alta, il modello non è sicuro e le sue previsioni molto probabilmente sono errate.

  Nel nostro caso non è stata rilevata nessuna differenza tra i due modelli, inoltre il valore assunto dalla BCE è molto basso. Questo probabilmente è dovuta alla semplicità della rete che è stata dichiarata e all'utilizzo delle stesse feature per la dichiarazione della feature target.

  Con questa soluzione non abbiamo ottenuto risultati utili; ricordiamo che `is_viral` è una stima ottenuta dalle fasi precedenti e con questa fase, l'obiettivo è esplorare nuove relazioni per aggiornare eventualmente l'ipotesi e renderla più vicino alla realtà.

5.1.3 Modello Finale

In questa implementazione finale andiamo ad implementare l'algoritmo Hill Climb Search in modo da esplorare in modo automatico delle nuove relazioni.

5.1.3.1 Aggiunta Feature Sono state create due nuove feature, `engagement_score` e `title_complexity`, combinando quelle già esistenti per catturare meglio la loro importanza e dare più informazioni al modello.

```python
def add_new_features(df):
    df = df.copy()
    df['engagement_score'] = df['upvotes'] + df['num_comments']
    df['title_complexity'] = df['title_length'] / df['title_length'].mean()
    return df
```

Figure 18: Aggiunta Feature

5.1.3.2 Discretizzazione delle Variabili La discretizzazione per le variabili già note è la stessa, mentre per quelle nuove:

```python
def discretize_variables(df):
    df = df.copy()
    bins_dict = {
        'upvotes': [0, 2000, 4000, float('inf')],
        'num_comments': [0, 100, 300, float('inf')],
        'upvote_ratio': [0, 0.5, 0.75, 1.0],
        'title_length': [0, 40, 80, float('inf')],
        'author_impact': [0, 5, 10, float('inf')],
        'engagement_score': [0, 2000, 5000, float('inf')],
        'title_complexity': [0, 0.5, 1.0, float('inf')]
    }
```

Figure 19: Discretizzazione

5.1.3.3 Risultati Con la precedente discretizzazione delle feature siamo riusciti a risolvere il problema del tempo e della memoria riuscendo ad avviare l'algoritmo.

La rete Bayesiana appresa è la seguente:

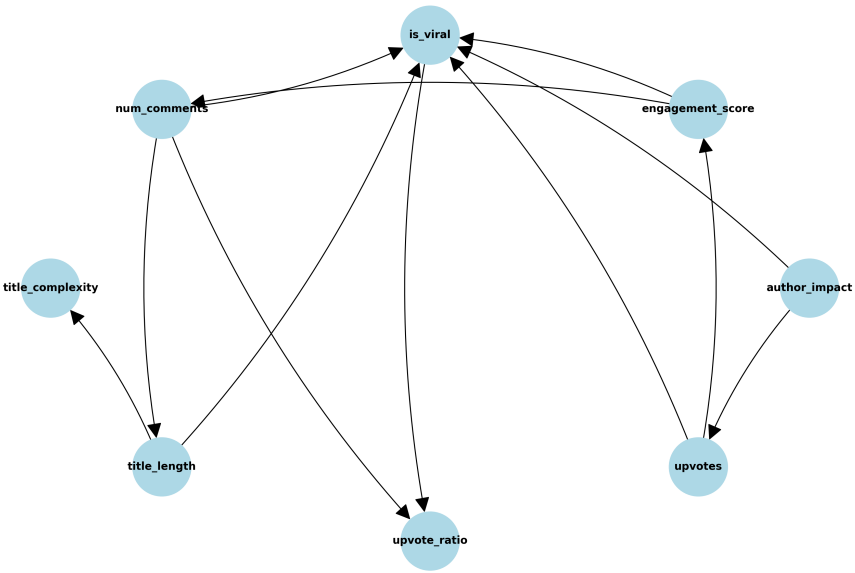

Figure 20: Rete Bayesiana

- upvotes → engagement_score
- upvotes → is_viral
- engagement_score → is_viral
- engagement_score → num_comments
- is_viral → upvote_ratio
- num_comments → title_length
- num_comments → upvote_ratio
- num_comments → is_viral
- title_length → title_complexity
- title_length → is_viral
- author_impact → is_viral
- author_impact → upvotes

Il risultato ottenuto è molto più complesso delle due soluzioni precedenti e fornisce informazioni aggiuntive per la predizione della feature target.

5.1.3.4 Prestazioni del Modello Utilizziamo le stesse metriche delle due soluzioni precedenti e sono le seguenti:

Metrica	**Valore**
Accuracy	0.9560
Precision	0.9560
Recall	0.8908
F1-score	0.9223
Binary Cross-Entropy	0.2753

Table 12: Risultati del modello

I risultati mostrano che il modello è molto efficiente nel predire la viralità del post; probabilmente valori così alti sono dovuti anche alla semplicità del problema, dato che lavora su feature con cui abbiamo definito `is_viral`. In un contesto più realistico, i valori potrebbero risultare più bassi.

6 Conclusioni Finali

Nello sviluppo del progetto abbiamo fatto delle ipotesi sulle quali è stato definito se un post è virale/non virale. Questa stima è molto semplice dove semplicemente cerchiamo di combinare le features tra loro e stabiliamo delle soglie, in uno studio più reale la classificazione di un post prevede sicuramente informazioni più complesse che necessitano dell'intervento di un esperto del settore, nel nostro caso magari un admin che ci fornisce dati aggiuntivi sulle stime.

Uno sviluppo futuro potrebbe prevedere il download di un dataset ancora più grande con informazioni più utili che magari potrebbero permettere una classificazione ancora più discriminante dove possiamo analizzare anche i post molto virali.

Inoltre abbiamo visto che la maggior parte dei post sono immagini quindi l'implementazione di strumenti avanzati per l'analisi di queste potrebbe essere anche molto utile per stabilire altri pattern nascosti.

L'API di Reddit è utilizzabile da tutti in modo gratuito, proprio per questo i tempi di download sono molto lunghi e i dati da scaricare sono limitati. Altri social network interessanti come X (Twitter) prevedono un pagamento per l'utilizzo dell'API ma probabilmente forniscono informazioni aggiuntive che potrebbero essere utili per la classificazione o altri scopi.