SENTIMENT ANALYSIS DELLA SERIE TV

“THE GENTLEMEN”

1. OBIETTIVO

L'obiettivo di questo progetto è raccogliere e analizzare le recensioni della nuova serie TV "The Gentlemen" utilizzando la sentiment analysis. La sentiment analysis è una tecnica di elaborazione del linguaggio naturale (NLP) che ci aiuta a capire l'orientamento emotivo dei testi. In parole semplici, ci dice se i commenti degli spettatori sono positivi, negativi o neutri. Questa analisi è molto utile per comprendere cosa pensano gli spettatori della serie e può essere usata per diversi scopi.

1. RACCOLTA DATI

Lo script inizia con il processo di raccolta dei dati delle recensioni della serie televisiva "The Gentlemen" utilizzando tecniche di web scraping. Per questa attività ho utilizzato le librerie Selenium e BeautifulSoup, per automatizzare la raccolta delle recensioni dal sito IMDb. Per iniziare, ho configurato il WebDriver di Chrome utilizzando la libreria Selenium. Il WebDriver è essenziale per automatizzare l'interazione con il browser e navigare sul sito IMDb. Ho aperto la pagina delle recensioni su IMDb e preparato il browser per l'operazione di scraping. IMDb carica inizialmente solo una parte delle recensioni e richiede di cliccare sul pulsante "Load More" per caricarne altre. Per ottenere un numero sufficiente di recensioni, ho automatizzato questo processo con un ciclo che clicca ripetutamente sul pulsante fino a raggiungere almeno 200 recensioni. Durante questo processo, ho utilizzato un breve ritardo tra un clic e l'altro per permettere il caricamento dei nuovi contenuti. Una volta caricate tutte le recensioni necessarie, ho estratto le informazioni rilevanti da ciascuna recensione. Le informazioni estratte includono : nome dell'utente, numero di stelle assegnate, data della recensione, testo della recensione e titolo dellarecensione. Dopo aver raccolto tutte le recensioni, ho archiviato i dati in un file CSV.

1. CARICAMENTO E PULIZIA DEL DATASET

Innanzitutto ho importato le librerie necessarie come: “pandas” per la manipolazione dei dati e la gestione dei DataFrame e “nltk” per l'elaborazione del linguaggio naturale. Il primo passo nel trattamento del dataset consiste nel caricamento e nella pulizia dei dati. Ho caricato il file CSV contenente le recensioni e verificato la presenza di valori mancanti.

Dopo aver identificato le colonne con valori mancanti, ho rimosso le righe contenenti tali valori per assicurare l'integrità del dataset.

Dopo la pulizia, il dataset non conteneva più valori mancanti, come evidenziato dalle verifiche effettuate.

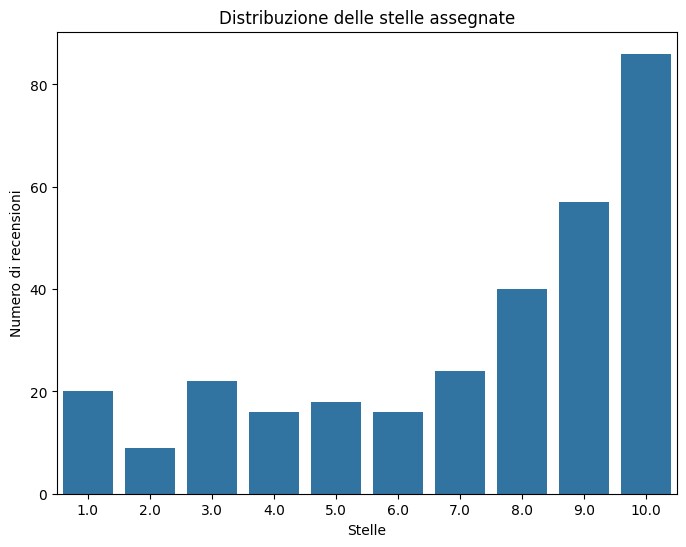
La verifica del dataset infatti prima e dopo la pulizia ha mostrato che, sebbene inizialmente fossero presenti 7 valori NaN nella colonna "Stars", dopo la pulizia tutte le colonne risultavano prive di valori mancanti.

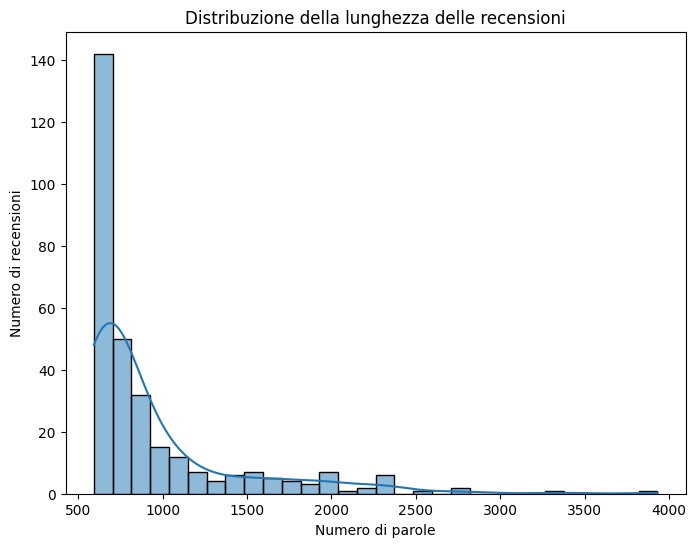
1. ANALISI ESPLORATIVA DEI DATI

Dopo aver caricato e pulito il dataset, ho svolto una piccola analisi esplorativa.

Ho analizzato la distribuzione delle stelle assegnate dagli utenti per le loro recensioni. Questo mi ha permesso di visualizzare come gli spettatori hanno valutato la serie "The Gentlemen". La distribuzione delle stelle è stata rappresentata graficamente per mostrare il numero di recensioni per ciascun punteggio in stelle.

Un altro aspetto analizzato è stata la lunghezza delle recensioni. Ho calcolato la lunghezza di ogni recensione in termini di numero di parole e ho creato una distribuzione per visualizzare questa caratteristica. Questo mi ha aiutato a capire se le recensioni tendono ad essere brevi o dettagliate.





1. Pre-elaborazione dei Dati

Dopo l'analisi esplorativa, ho proceduto con la fase di pre-elaborazione dei dati. Questo passaggio è cruciale per preparare le recensioni per l'analisi del sentiment, garantendo che i testi siano uniformi e privi di elementi non necessari.

Ho iniziato scaricando i dati necessari per l'elaborazione del linguaggio naturale utilizzando la libreria nltk, come WordNet, le stopwords e altri strumenti di tokenizzazione.

Ho ricaricato il dataset originale contenente le recensioni ed ho applicato una funzione per rimuovere la punteggiatura da tutte le recensioni. Questo passaggio è importante per eliminare caratteri non necessari che possono interferire con l'analisi del testo.

Ho poi convertito tutte le recensioni in lettere minuscole per garantire l'uniformità e facilitare il confronto delle parole.

Ho rimosso tutti i caratteri non alfabetici dalle recensioni per concentrare l'analisi solo sulle parole.

Infine ho rimosso tutte le stopwords, ossia le parole comuni che non portano significato rilevante (come "the", "and", "is"). Questo passaggio aiuta a ridurre il rumore nel testo e a focalizzarsi sui termini significativi.

Dopo aver applicato tutte queste trasformazioni, ho verificato alcune recensioni per assicurarmi che la pulizia fosse avvenuta correttamente. Le recensioni risultanti erano più omogenee e pronte per ulteriori analisi.

Infine, ho salvato il dataset pulito in un nuovo file CSV: “RECENSIONI\_pulite.csv”, che sarà utilizzato nelle fasi successive dell'analisi del sentiment.

1. DEFINIZIONE DELLE SOGLIE PER IL SENTIMENT

Per analizzare il sentiment delle recensioni, ho definito delle soglie basate sul numero di stelle assegnate dagli utenti. Ho creato una funzione per etichettare il sentiment delle recensioni in base al numero di stelle. Questa funzione assegna un'etichetta di "positivo" (1) alle recensioni con un punteggio pari o superiore a 7, e un'etichetta di "negativo" (0) alle recensioni con un punteggio pari o inferiore a 6. Le recensioni con valori di punteggio fuori dai range specificati vengono etichettate come "None" per gestire eventuali anomalie.

Utilizzando questa funzione, ho aggiunto una nuova colonna denominata 'Sentiment' al dataset ed ho verificato visivamente alcune righe del DataFrame per assicurarmi che l'etichettatura fosse corretta. Prima di salvare il DataFrame aggiornato, ho eseguito un conteggio dei valori nella colonna 'Sentiment' per verificare la distribuzione dei sentiment nel dataset. Questo ha mostrato che c'erano 207 recensioni positive e 101 recensioni negative. Ho salvato il DataFrame aggiornato in un nuovo file CSV e l'ho ricaricato per verificare l'integrità dei dati. La verifica ha confermato che non c'erano valori NaN nella colonna 'Sentiment' dopo la rilettura del file.

Ho deciso di utilizzare solo due classi (positive e negative) e di non includere una classe neutra per evitare un'ineguale distribuzione dei dati tra le classi. L'inclusione di una classe neutra avrebbe potuto creare un disequilibrio significativo, rendendo l'analisi meno robusta e l'addestramento del modello meno efficace.

1. Suddivisione del Dataset

Dopo aver preparato il dataset con le etichette di sentiment, ho proceduto con la suddivisione dei dati in set di addestramento e di test.

Ho suddiviso il dataset in due parti: un set di addestramento e un set di test. Ho utilizzato il 20% dei dati per il test e l'80% per l'addestramento, mantenendo la casualità della selezione con un seed di randomizzazione fisso per garantire la riproducibilità dei risultati.

Dopo la suddivisione, ho verificato la distribuzione delle classi (positive e negative) nei due set. Questa verifica è importante per assicurarsi che entrambi i set siano rappresentativi della distribuzione originale dei dati e che non ci sia uno squilibrio significativo tra le classi.

Nel set di addestramento, ci sono 167 recensioni positive e 79 recensioni negative.

Nel set di test, ci sono 40 recensioni positive e 22 recensioni negative.

Questo è un passaggio cruciale per garantire che il modello di analisi del sentiment possa essere addestrato e testato in modo efficace. Mantenere una distribuzione equilibrata delle classi in entrambi i set aiuta a migliorare la generalizzazione del modello e la sua capacità di predire correttamente il sentiment delle recensioni future.

1. Tokenizzazione e Padding

Per preparare i dati testuali per l'analisi, ho utilizzato un Tokenizer[[1]](#footnote-1) per convertire i testi delle recensioni in sequenze di token numerici. Questo processo serve per poter utilizzare i dati testuali nei modelli di machine learning.

Ho quindi inizializzato il Tokenizer e l'ho adattato sui testi del set di addestramento per costruire il vocabolario ed ho calcolato il numero totale di parole uniche nel vocabolario, aggiungendo una parola in più per il padding.

Dopo aver adattato il Tokenizer, ho trasformato i testi delle recensioni in sequenze di token numerici sia per il set di addestramento che per il set di test. Per garantire che tutte le sequenze abbiano la stessa lunghezza, ho trovato la lunghezza massima delle sequenze nel set di addestramento. Questa lunghezza massima viene utilizzata per il padding.

Ho applicato il padding[[2]](#footnote-2) alle sequenze per uniformare la lunghezza di tutte le recensioni. Il padding viene aggiunto alla fine di ciascuna sequenza (padding='post') e le sequenze più lunghe vengono troncate alla lunghezza massima stabilita (truncating='post').

1. Vettorizzazione

La vettorizzazione serve a trasformare dati testuali in vettori numerici che possono essere elaborati da algoritmi di machine learning e deep learning. Questo processo consente di rappresentare testi in una forma matematica, facilitando l'analisi, il confronto e l'addestramento dei modelli su dati testuali.

Ho fatto 3 tipi di vettorizzazioni: Bag of words (BoW) , Term Frequency-Inverse Document Frequency (TF-IDF) e Word2Vec.

**BoW[[3]](#footnote-3):** Ho inizializzato il CountVectorizer, uno strumento che crea una rappresentazione BoW delle recensioni. Ho poi adattato il vettorizzatore BoW ai dati di addestramento per costruire il vocabolario. Dopo l'adattamento, ho trasformato sia i dati di addestramento che quelli di test in matrici BoW, dove ogni riga rappresenta una recensione e ogni colonna rappresenta una parola del vocabolario. Infine, ho verificato le dimensioni dei dati vettorizzati per assicurarmi che la procedura sia stata eseguita correttamente.

Dopo la vettorizzazione BoW, il set di addestramento ha la forma di una matrice con 246 righe (recensioni) e un numero di colonne pari al numero di parole uniche nel vocabolario. Analogamente, il set di test è rappresentato come una matrice con 62 righe e lo stesso numero di colonne del set di addestramento, garantendo coerenza tra i due set.

**TF-IDF[[4]](#footnote-4):** Ho inizializzato il TfidfVectorizer, uno strumento che calcola i pesi TF-IDF per le parole nelle recensioni. Ho adattato il vettorizzatore TF-IDF ai dati di addestramento per costruire il vocabolario e calcolare i pesi TF-IDF.

Dopo l'adattamento, ho trasformato sia i dati di addestramento che quelli di test in matrici TF-IDF, dove ogni riga rappresenta una recensione e ogni colonna rappresenta una parola del vocabolario con il relativo peso TF-IDF. Dopo la trasformazione, ho verificato le dimensioni dei dati vettorizzati per assicurarmi che la procedura sia stata eseguita correttamente.

Il set di addestramento ha la forma di una matrice con 246 righe (recensioni) e un numero di colonne pari al numero di parole uniche nel vocabolario.

Mentre set di test è rappresentato come una matrice con 62 righe e lo stesso numero di colonne del set di addestramento, garantendo coerenza tra i due set.

**Word2Vec[[5]](#footnote-5):** Prima di addestrare il modello Word2Vec, ho preparato i dati convertendo il testo delle recensioni in una forma più semplice utilizzando “simple\_preprocess” dalla libreria Gensim. Questo processo suddivide ogni recensione in una lista di token.

Ho addestrato il modello Word2Vec sui dati di addestramento. Il modello è stato configurato con una dimensione vettoriale di 300, una finestra di contesto di 5 parole, e considera tutte le parole presenti nei testi (min\_count=1).

Per convertire le recensioni in vettori numerici, ho definito una funzione “document\_vector” che calcola il vettore del documento come la media dei vettori delle parole presenti nel documento. Questo approccio cattura il significato complessivo del documento in base ai vettori delle parole che lo compongono.

Dopo la vettorizzazione con Word2Vec, il set di addestramento è rappresentato da una matrice con 246 righe (recensioni) e 300 colonne (dimensione vettoriale).

Il set di test è invece rappresentato da una matrice con 62 righe e 300 colonne, mantenendo lo stesso numero di dimensioni vettoriali del set di addestramento.

1. SMOTE

Poiché le due classi non sono bilanciate perfettamente in quanto vi è una disparità tra il numero di campioni nelle classi, ho deciso di applicare la tecnica di sovracampionamento SMOTE[[6]](#footnote-6) ma esclusivamente ai dati di addestramento per mantenere l'integrità del processo di addestramento e valutazione del modello, garantendo che i dati di test rimangano rappresentativi e indipendenti. Ciò permette al modello di apprendere meglio da entrambe le classi durante l'addestramento e di essere più preciso quando si confronta con nuovi dati.

1. Valutazione performance dei modelli

11.1 Valutazione senza applicazione di SMOTE: Ho iniziato la valutazione tramite la cross-validation stratificata per ogni metodo di vettorizzazione. La stratificazione garantisce che ogni fold abbia una distribuzione simile delle classi di target, il che può portare a stime più affidabili delle prestazioni del modello. Considerando che sto lavorando con dati di recensioni, che sono soggetti a sbilanciamento delle classi, l'uso della cross-validation stratificata è una scelta appropriata e può portare a stime più accurate delle prestazioni del modello.

Ho usato quindi questo approccio suddividendo il dataset in cinque fold mantenendo la proporzione tra le classi in ciascun fold.

Successivamente, ho addestrato e valutato diversi modelli di machine learning, tra cui regressione logistica, Naive Bayes, SVM, Random Forest e KNN, utilizzando misure di performance come l'accuratezza, la precisione, il richiamo e l'F1-score. Per ogni combinazione di modello e metodo di vettorizzazione, ho calcolato la media e la deviazione standard delle misure di performance attraverso i fold di cross-validation. Questi risultati forniscono una stima della capacità dei modelli di generalizzare su nuovi dati, considerando variazioni dovute alla suddivisione del dataset in fold diversi.

Guardando i risultati ottenuti, possiamo trarre alcune osservazioni:

1. **Vettorizzazione BoW**: **Logistic Regression** [[7]](#footnote-7)e **Naive Bayes** [[8]](#footnote-8)hanno ottenuto i migliori risultati, con alte precision(rispettivamente **81%** e **82%**) , recall(**94**% e **91%**) e F1-score(**87%** e **86%**), suggerendo che sono efficaci nel rilevare la classe di interesse.

**SVM**[[9]](#footnote-9) ha mostrato un recall molto alto(**99%**), ma una precision(**0.0082**) e un’accuracy(**67%**) inferiori.

**Random Forest** [[10]](#footnote-10)e **KNN**[[11]](#footnote-11) hanno ottenuto risultati discreti, ma con maggiore variabilità nelle prestazioni.

1. **Vettorizzazione TF-IDF**: **Logistic Regression**, **Naive Bayes** [[12]](#footnote-12)e **SVM** hanno mostrato risultati identici con un’accuracy (**67%**)e F1-score moderate e un recall perfetto, suggerendo una mancanza di discriminazione tra le classi.

**Random Forest** ha mostrato una migliore accuracy (**71%)** e precision (**70%**), con un recall perfetto.

**KNN** ha ottenuto i migliori risultati complessivi, con una buona accuracy (**76%**), precision (**76%**), recall (**95%**) e F1-score (**84%**), suggerendo una buona capacità di bilanciare tra precision e recall.

1. **Vettorizzazione Word2Vec**: **Logistic Regression** e **SVM** hanno mostrato risultati identici, con precision(**67%**) e F1-score(**80%**) moderate e un recall perfetto, suggerendo una mancanza di discriminazione tra le classi.

**Random Forest** ha mostrato una migliore accuracy(**68%**) e precision(**70%**) rispetto a Logistic Regression e SVM, con un buon recall(**93%**), suggerendo una migliore capacità di discriminazione tra le classi.

Infine **KNN** ha ottenuto i risultati meno performanti, con una minore accuracy (**63%**) e precision(**70%**) rispetto agli altri modelli.

In generale, sembra che la vettorizzazione BoW abbia prodotto i risultati migliori, con precision, recall e F1-score più elevati rispetto alle altre vettorizzazioni. TF-IDF ha mostrato prestazioni inferiori, con Logistic Regression, Naive Bayes e SVM che hanno avuto difficoltà a discriminare tra le classi. Word2Vec ha prodotto risultati simili, ma generalmente leggermente inferiori rispetto a BoW..

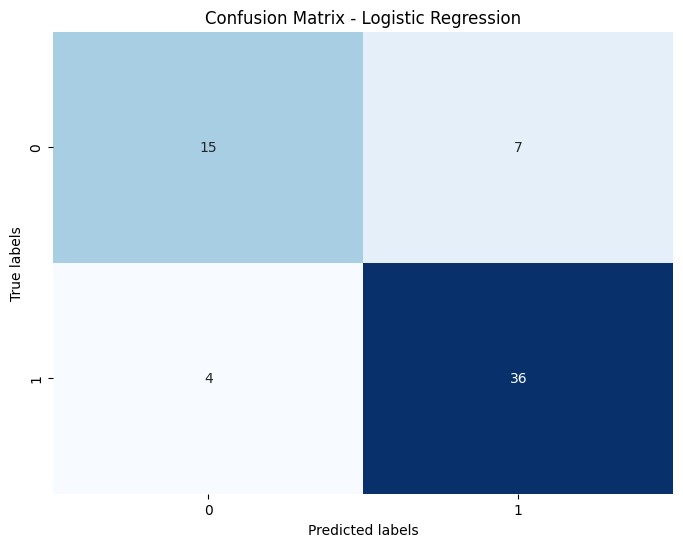
11.2 Valutazione con applicazione di SMOTE: Ho valutato quindi i modelli utilizzando una tecnica di cross-validation stratificata per ogni metodo di vettorizzazione, come prima, ma considerando sta volta anche i dati sovracampionati con SMOTE. Come prima per ciascun metodo di vettorizzazione (Bag of Words e TF-IDF) ho addestrato e valutato i modelli di machine learning (Logistic Regression, Naive Bayes, SVM, Random Forest e KNN) utilizzando misure di performance come l'accuratezza, la precisione, il richiamo e l'F1-score

1. **Vettorizzazione BoW**: **Logistic Regression** e **Naive Bayes** hanno ottenuto risultati molto simili in termini di accuracy (**84%**) e F1-score (**83%**). Entrambi i modelli sembrano avere una buona capacità predittiva e un buon equilibrio tra precisione e recall.

**SVM** ha mostrato un’accuracy leggermente inferiore (**83%**), ma ha ottenuto un recall molto alto (**97%**). Questo indica che il modello SVM è molto bravo a identificare correttamente i casi positivi.

**Random Forest** ha prodotto risultati accettabili con un’accuracy del **81%** e un recall del **75%.** Anche se l'accuracy è leggermente inferiore rispetto a Logistic Regression e Naive Bayes, il modello ha una buona capacità di identificare i casi positivi.

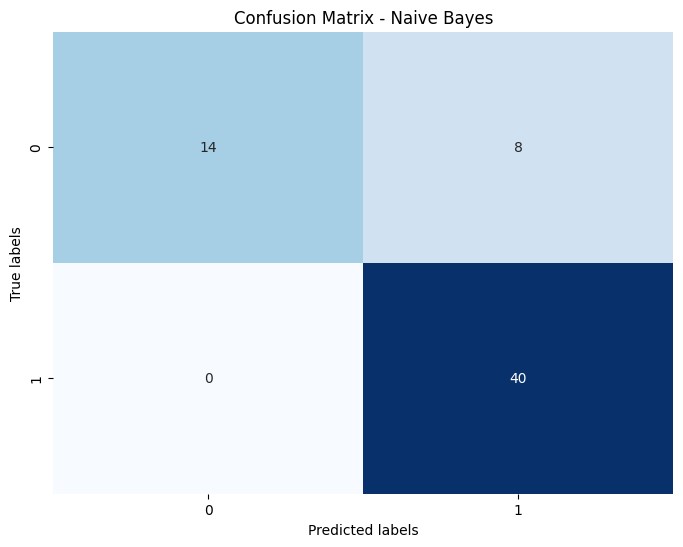
**KNN** ha avuto prestazioni molto basse, con un'accuracy e un F1-score del **50%**. Questo indica che il modello KNN non è stato in grado di adattarsi bene ai dati e ha ottenuto risultati casuali o poco migliori di un modello casuale

Matrice di confusione con vettorizzazione BoW+ SMOTE:

Guardando la matrice di confusione, possiamo notare che il modello di regressione logistica si comporta bene nelle previsioni.

Infatti, ha predetto correttamente 15 casi negativi veri e solo 7 sono stati erroneamente classificati come positivi.

Inoltre, ha predetto correttamente 36 casi positivi veri, mentre solo 4 sono stati erroneamente classificati come negativi.



Anche il classificatore Naive Bayes ha ottenuto risultati simili.

Ha correttamente predetto 14 casi di negativi veri e solo 8 sono stati erroneamente classificati come positivi.

Inoltre, ha correttamente classificato 40 casi di positivi veri senza alcun errore, ossia nessun positivo è stato erroneamente predetto come negativo.

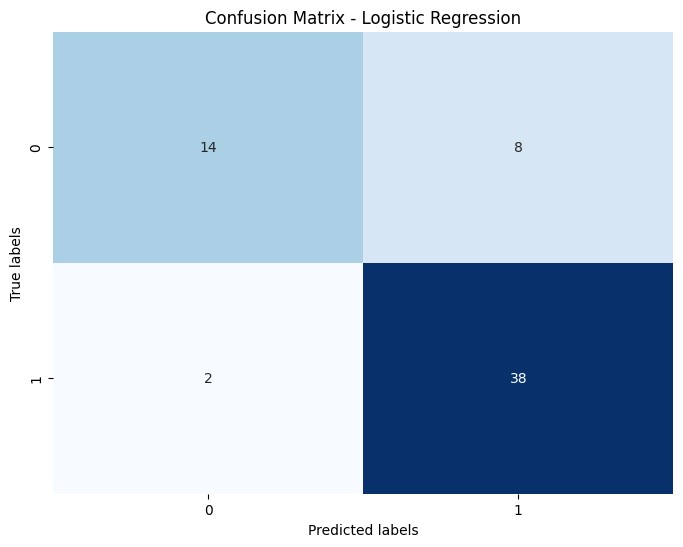
2. **Vettorizzazione TF-IDF**: **Logistic Regression** e **Naive Bayes** hanno ottenuto risultati molto simili in termini di accuracy (circa **93%**) e F1-score (**93%**). Entrambi i modelli sembrano avere una buona capacità predittiva e un buon equilibrio tra precisione e recall.

**SVM** ha mostrato un'accuracy leggermente inferiore (**90%**), ma ha ottenuto un recall molto alto (**100%**). Questo indica che il modello SVM è molto bravo a identificare correttamente i casi positivi.

**Random Forest** ha prodotto risultati accettabili con un'accuracy dell'**87%** e un recall del **98%.** Anche se l'accuracy è leggermente inferiore rispetto a Logistic Regression e Naive Bayes, il modello ha una buona capacità di identificare i casi positivi.

**KNN** ha avuto prestazioni molto basse, con un'accuracy, una precision, un recall e un F1-score del **50%, 0%, 0% e 0%,** rispettivamente. Questo indica che il modello KNN non è stato in grado di adattarsi bene ai dati e ha ottenuto risultati casuali o poco migliori di un modello casuale.

In conclusione, con la vettorizzazione Bag-of-Words (BoW), Logistic Regression e Naive Bayes offrono performance solide ma meno elevate rispetto a TF-IDF, dove Logistic Regression e Naive Bayes mantengono prestazioni elevate. SVM e Random Forest sono buone alternative in entrambi i casi, con Random Forest che si distingue per la sua robustezza nell'identificare i casi positivi. KNN mostra prestazioni inferiori in entrambe le configurazioni. La scelta dipende dalle priorità specifiche e dalla complessità del problema.

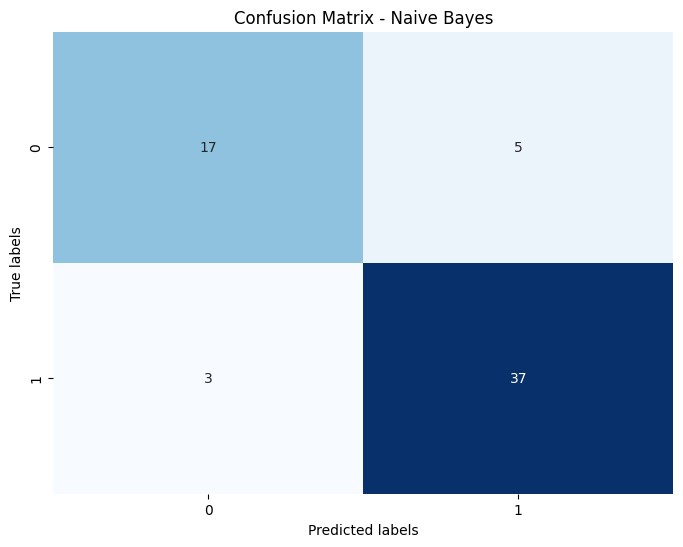
Matrice di confusione con vettorizzazione TF-IDF+ SMOTE:

In questo caso la matrice di confusione ci mostra che 14 veri negativi sono stati predetti correttamente, mentre 8 erroneamente.

D’altro canto, ha correttamente predetto 38 casi di positivi, sbagliandone solo 2 .

Inizio modulo

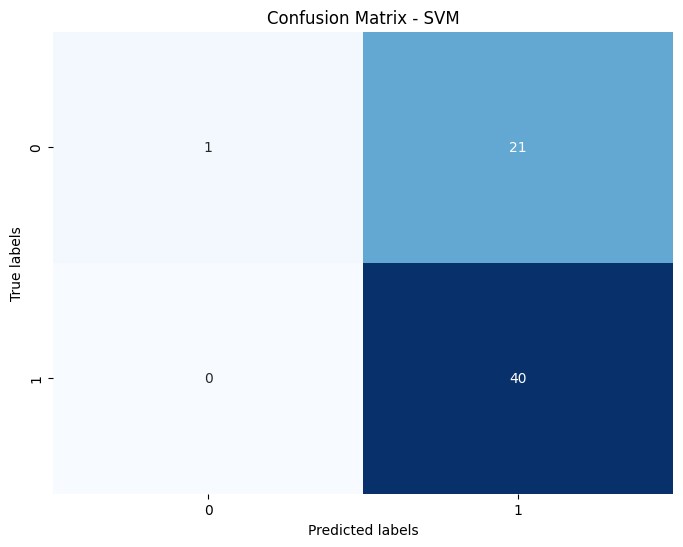
Fine modulo



Anche il modello Naive Bayes ha ottenuto risultati simili.

Infatti 17 veri negativi sono stati predetti correttamente, mentre 5 erroneamente.

Inoltre, ha correttamente classificato 37 casi di veri positivi , sbagliandone solo 3 .



Per quanto riguarda il modello SVM, invece, sbaglia molto a classificare i veri negativi.

Infatti, ha correttamente predetto solo 1 vero negativo, mentre 20 li ha predetti erroneamente.

Della classe dei positivi,invece, li ha predetti tutti correttamente, senza alcun errore, ossia nessun positivo è stato erroneamente predetto come negativo.

1. Vettorizzazione Word2Vec: Innanzitutto ho dovuto fare il Reshape e solo allora ho potuto applicare SMOTE perché bisogna adattare il formato dei dati per garantire la compatibilità con l'algoritmo SMOTE. Ho quindi poi svolto la valutazione.

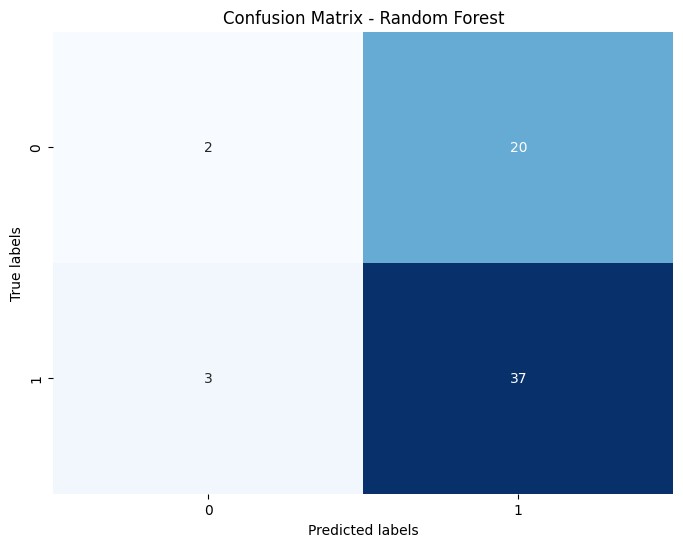
**Logistic Regression** nonostante un'accuracy del **54%,** si osserva una precision del **71%** e un recall del **54%.** Tuttavia, l'F1-score medio del **47%** suggerisce che il modello ha difficoltà nel bilanciare precisione e recall. Questo potrebbe indicare una scarsa capacità di discriminazione dei dati. **SVM** pur avendo un'accuracy simile (**55%**) a Logistic Regression, mostra una precision del **58%** e un recall del **32%,** con un F1-score medio del **41%.** Anche se SVM riesce a identificare alcuni casi positivi, il suo basso recall suggerisce che molti casi positivi vengono ignorati.

**Random Forest** si distingue per un'accuracy dell'**83%,** una precision dell'**84%** e un recall dell'**81%.** L'F1-score dell'**82%** indica che il modello ha una buona capacità di bilanciare precisione e recall. Questo potrebbe essere dovuto alla sua capacità di utilizzare un insieme di alberi decisionali per migliorare la predizione.

**KNN** con un'accuracy del **64%** e un F1-score del **47%**, KNN mostra una leggera miglioramento rispetto a Logistic Regression e SVM. Tuttavia, il suo F1-score ancora inferiore rispetto a Random Forest suggerisce che KNN potrebbe soffrire di una scarsa capacità di generalizzazione e una sensibilità agli outlier.

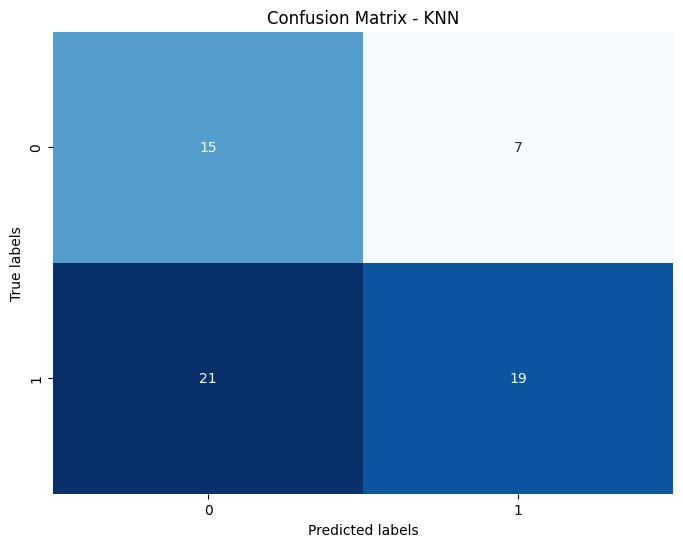
In sintesi, Random Forest si distingue come la scelta migliore con Word2Vec, mostrando un'accuratezza e un F1-score medi superiori e una capacità di bilanciare precisione e recall. SVM e KNN hanno prestazioni inferiori, mentre Logistic Regression è il modello meno performante tra quelli considerati, con difficoltà nel bilanciare precisione e recall.

Matrice di confusione con vettorizzazione Word2Vec+ SMOTE:

Come possiamo vedere, infatti, il modello Random Forest è quello migliore con la vettorizzazione Word2Vec anche se è palese anche qui una tendenza a classificare erroneamente troppi esempi come positivi.

Infatti, solo 2 veri negativi sono stati predetti correttamente, 20 invece sono stati predetti come positivi erroneamente.

Mentre della classe dei veri positivi, solo 3 sono stati predetti male.



Il modello KNN invece, non è molto performante nonostante.

Come possiamo vedere nei positivi, infatti, sono più i dati predetti erroneamente che correttamente.

Per quanto riguarda i veri negativi, sono stati predetti correttamente 15, sbagliando a predirne solo 7.

Invece della classe dei veri positivi, 19 sono stati predetti correttamente, mentre 21 sono stati predetti male.

1. Le Reti Neurali

Ho poi definito e addestrato, con vettorizzazioneWord2Vec , tre modelli neurali[[13]](#footnote-13) : RNN, CNN e LSTM. Per valutare le prestazioni di ciascun modello, ho utilizzato misure come l'accuratezza e l’ F1-score.

Per ciascun modello, ho specificato la sua architettura e i parametri di addestramento, inclusi i layer neurali, le funzioni di attivazione e i livelli di dropout per evitare l'overfitting. Successivamente, ho compilato i modelli specificando la funzione di perdita, l'ottimizzatore e le metriche da monitorare durante l'addestramento. Infine, ho addestrato i modelli utilizzando i dati di addestramento e valutato le loro prestazioni utilizzando i dati di test.

Non ho utilizzato SMOTE qui perché in teoria puoi applicare SMOTE anche alle reti neurali ma l'uso di dati sintetici può influenzare la velocità e la stabilità con cui la rete neurale impara dai dati. Se i dati sintetici non riflettono accuratamente la vera distribuzione dei dati, la rete neurale potrebbe impiegare più tempo o avere difficoltà a "capire" come adattarsi ai dati durante l'addestramento. Ciò potrebbe comportare una maggiore incertezza nel processo di apprendimento.

I modelli sono stati addestrati e valutati con le seguenti metriche:

**RNN Model - Accuracy**: 0.59, **F1-score**: 0.74

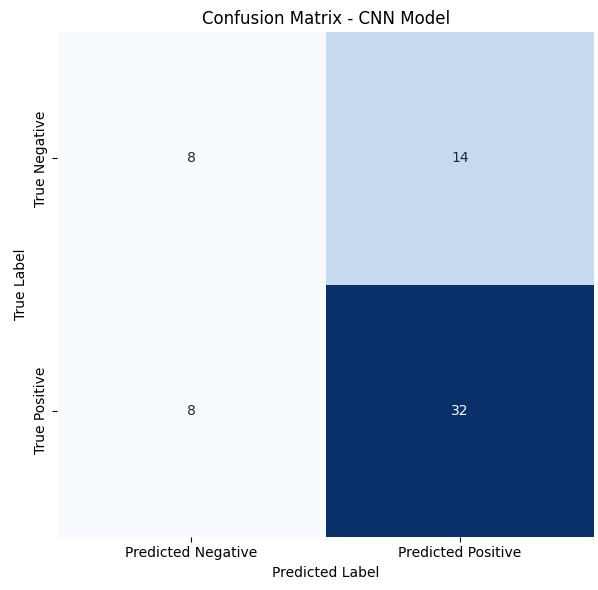
**CNN Model - Accuracy:** 0.64, **F1-score**: 0.74

**LSTM Model - Accuracy:** 0.64, **F1-score**: 0.78

La **RNN** ha ottenuto un punteggio decente, ma potrebbe non essere stata ottimale nel catturare le relazioni a lungo termine nei dati.

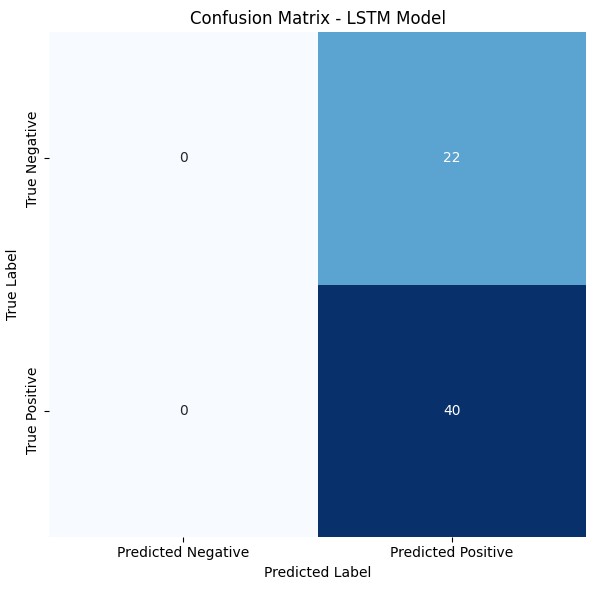
La **CNN** ha mostrato un miglioramento, suggerendo una maggiore stabilità nei risultati grazie alle sue capacità di riconoscere pattern.

**L'LSTM** ha registrato risultati simili alla CNN ma con un F1-score più alto, indicando una migliore capacità nel bilanciare precisione e recall grazie alla sua capacità di gestire informazioni a lungo termine.

Matrice di confusione con vettorizzazione Word2Vec delle reti neurali:

Per quanto riguarda il modello CNN[[14]](#footnote-14), i veri positivi, sono stati predetti correttamente 32, sbagliando a predirne solo 8.

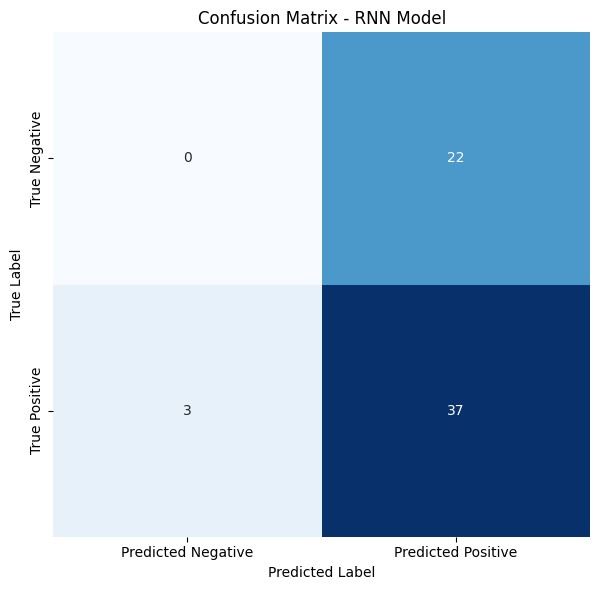
Invece della classe dei veri negativi, solo 8 sono stati predetti correttamente, mentre 14 sono stati predetti sbagliando.



Per quanto riguarda il modello LSTM[[15]](#footnote-15), invece, sbaglia molto a classificare i veri negativi.

Infatti, ha correttamente predetto 0 veri negativo, mentre 22 li ha predetti erroneamente.

Della classe dei positivi,invece, li ha predetti tutti correttamente, senza alcun errore, ossia nessun positivo è stato erroneamente predetto come negativo.



Praticamente stessa cosa di LSTM è stata per il modello RNN[[16]](#footnote-16).

Infatti, anche qui, i veri negativi sono stati predetti tutti come positivi; mentre i positivi sono stati predetti quasi tutti correttamente tranne 3 .

Questo, perciò, ci fa capire che c'è una tendenza nel predire il positivo al posto del negativo.

1. I TRANSFORMERS

Sono quindi poi passata ai transformers con dei modelli pre-addestrati quali Bert e GPT-2.

**BERT**[[17]](#footnote-17): ho caricato i dati di addestramento e l’ho suddivisi in input (testi) e output (etichette di sentimento). Ho quindi tokenizzato i testi utilizzando il tokenizer di BERT, ho convertito in input\_ids (id dei token) e attention\_masks (maschere di attenzione) e diviso in set di addestramento e di test.

Ho caricato un modello BERT pre-addestrato per la classificazione di sequenze di testo utilizzando “BertForSequenceClassification” dalla libreria Hugging Face Transformers. Ed ho definito il numero di etichette (per il sentiment positivo e negativo) e altri parametri relativi all'output del modello

Il modello è stato addestrato per 3 epoche, e durante ogni epoca, itera sui dati di addestramento tramite il DataLoader, calcolando le perdite, aggiornando i pesi del modello attraverso la retropropagazione del gradiente e valutando il modello sui dati di validazione. Dopo ogni epoca, ho stampato le perdite di addestramento e di validazione, così come l'accuratezza del modello sui dati di validazione.

Il modello ha raggiunto una validation accuracy di 64% sul set di dati di validazione dopo tutte e tre le epoche di addestramento.

**GPT-2[[18]](#footnote-18):** Ho caricato i dati dal file CSV e li hai suddivisi in insiemi di addestramento e test utilizzando “train\_test\_split”. Ho utilizzato il tokenizer di GPT-2 per tokenizzare e codificare le recensioni, aggiungendo anche un token di padding alla fine di ciascuna sequenza. Ho poi creato DataLoader per i set di dati di addestramento e test utilizzando “TensorDataset” e “DataLoader” di PyTorch. Dopo ho addestrato il modello GPT-2 utilizzando un ciclo di addestramento in cui ho iterato sui DataLoader, calcolando le perdite e aggiornando i pesi del modello tramite ottimizzatore. Infine, dopo l'addestramento, ho valutato le prestazioni del modello sui dati di test.

Dai risultati della valutazione, il modello ha ottenuto un'accuratezza del 100% sui dati di test, con una precisione, richiamo e F1-score di 1.0 per la classe positiva ("1"). Questo indica che il modello ha funzionato molto bene nella classificazione delle recensioni positive nel dataset di test ma male in quelle negative.

Ho pensato ad utilizzare SMOTE anche qui , ma non compatibile con i modelli basati su trasformer perché: questi modelli richiedono dati tokenizzati e non matrici di dati come fanno i modelli tradizionali; potrebbe non riflettere accuratamente la distribuzione dei dati reali presenti nel corpus di addestramento; ed infine, l’aggiunta di dati sintetici potrebbe influenzare negativamente la convergenza del modello durante l'addestramento.

1. Conclusioni

Dai dati forniti, possiamo trarre alcune conclusioni:

Utilizzando SMOTE : per gestire lo sbilanciamento delle classi la combinazione di vettorizzazione TF-IDF con modelli come Logistic Regression, Naive Bayes e SVM ha mostrato prestazioni complessivamente migliori in termini di accuratezza, precisione, recall e F1-score rispetto alla vettorizzazione BoW e Word2Vec.

Logistic Regression e Naive Bayes con vettorizzazione TF-IDF hanno ottenuto le migliori prestazioni in termini di accuratezza media e F1-score medio, seguiti da SVM.

Random Forest ha prodotto risultati accettabili in entrambe le vettorizzazioni, ma tende a migliorare leggermente con TF-IDF.

Senza l'utilizzo di SMOTE: Anche qui, TF-IDF mostra prestazioni migliori rispetto a BoW e Word2Vec, con una variazione minore tra le misure di prestazione dei vari modelli.

Tra i modelli, Logistic Regression e Naive Bayes con vettorizzazione BoW hanno ottenuto risultati simili, ma generalmente inferiori rispetto alla vettorizzazione TF-IDF.

SVM, Random Forest e KNN mostrano miglioramenti significativi quando applicati a TF-IDF rispetto a BoW e Word2Vec.

Perciò dopo aver esaminato i risultati dei vari modelli con diverse vettorizzazioni, e vedendo che il sentiment della serie tv “The Gentlemen” è positivo, possiamo identificare **le migliori tre opzioni**(Considerando sia i risultati con SMOTE che senza):

**Logistic Regression con vettorizzazione TF-IDF:** Ha ottenuto ottime prestazioni in termini di accuracy, precision, recall e F1-score. La Logistic Regression è un modello lineare che si adatta bene ai dati con alta dimensionalità come quelli derivanti dalla vettorizzazione TF-IDF. Inoltre, TF-IDF cattura l'importanza delle parole nei documenti, il che può essere particolarmente efficace per la Logistic Regression nel prendere decisioni basate su questa rilevanza.

**Naive Bayes con vettorizzazione TF-IDF:** Anche Naive Bayes ha ottenuto risultati eccellenti in termini di accuracy , precision, recall e F1-score. Questo modello è noto per la sua semplicità e velocità computazionale, ed è particolarmente efficace quando ci sono molte feature come nel caso della vettorizzazione TF-IDF. TF-IDF fornisce informazioni sulla frequenza e l'importanza delle parole nei documenti, che sono facilmente assimilate dal modello Naive Bayes.

**SVM con vettorizzazione TF-IDF:** Anche il modello SVM ha ottenuto prestazioni notevoli, specialmente per recall e F1-score. SVM è un modello che può trovare iperpiani di separazione ottimali in spazi ad alta dimensione, il che lo rende adatto per i dati derivati dalla vettorizzazione TF-IDF. Inoltre, le caratteristice di TF-IDF , possono essere sfruttate efficacemente da SVM per classificare correttamente i documenti.

Inizio modulo

Fine modulo

1. Tokenizzazione: processo di suddividere un testo in unità più piccole, chiamate token, come parole, frasi o simboli. Questo passaggio è fondamentale per trasformare il testo in una forma che può essere elaborata da modelli di elaborazione del linguaggio naturale (NLP) [↑](#footnote-ref-1)
2. Padding: tecnica utilizzata nel deep learning per rendere uniformi le dimensioni delle sequenze di input, aggiungendo elementi di riempimento (solitamente zeri) alle sequenze più corte. Questo consente di processare batch di dati con dimensioni diverse in modo efficiente. [↑](#footnote-ref-2)
3. BoW: rappresenta i testi come vettori di frequenza delle parole. [↑](#footnote-ref-3)
4. TF-IDF:misura l'importanza di una parola rispetto a un documento, considerando anche la frequenza delle parole in tutti i documenti. [↑](#footnote-ref-4)
5. Word2Vec: fornisce una rappresentazione numerica delle parole catturando le relazioni semantiche tra di esse e consentendo di comprendere il contesto in cui sono utilizzate. [↑](#footnote-ref-5)
6. **SMOTE**:tecnica utilizzata per affrontare il problema dello sbilanciamento delle classi nei dati, generando artificialmente nuovi esempi per le classi di minoranza [↑](#footnote-ref-6)
7. **Logistic Regression**: è un metodo di classificazione che stima la probabilità che un'osservazione appartenga a una determinata classe utilizzando una funzione logistica. Viene utilizzato per risolvere problemi di classificazione binaria e può essere esteso a classificazioni multiclasse. [↑](#footnote-ref-7)
8. **Naive Bayes**: è un metodo di classificazione che si basa sul teorema di Bayes, assumendo che le caratteristiche siano indipendenti tra loro [↑](#footnote-ref-8)
9. **SVM**: metodo di classificazione che trova l'iperpiano ottimale che separa le classi in uno spazio ad alta dimensione, massimizzando il margine tra i dati delle diverse classi [↑](#footnote-ref-9)
10. **Random Forest**: metodo di classificazione ed ensemble learning che costruisce una "foresta" di alberi decisionali, utilizzando campionamenti casuali dei dati e delle feature, per migliorare la precisione e ridurre l'overfitting. Le previsioni vengono effettuate prendendo la modalità (classificazione) o la media (regressione) delle previsioni di tutti gli alberi. [↑](#footnote-ref-10)
11. **K-Nearest Neighbors**: metodo di classificazione che assegna a un'osservazione la classe più comune tra i suoi k vicini più prossimi, basandosi su una misura di distanza (tipicamente la distanza euclidea). [↑](#footnote-ref-11)
12. [↑](#footnote-ref-12)
13. Le reti neurali sono modelli computazionali ispirati dal funzionamento del cervello umano, composti da unità di calcolo chiamate neuroni e organizzate in strati. Servono per l'apprendimento automatico da dati complessi, consentendo ai computer di riconoscere pattern e fare previsioni su dati non lineari o ad alta dimensionalità. [↑](#footnote-ref-13)
14. **CNN:** Algoritmo di deep learning che analizza dati complessi come immagini o testo, identificando automaticamente pattern e caratteristiche significative, utile per compiti come classificazione e riconoscimento di oggetti. [↑](#footnote-ref-14)
15. **LSTM:** tipo di architettura di rete neurale ricorrente (RNN) che possono mantenere e utilizzare informazioni a lungo termine, rendendole efficaci nel modellare sequenze di dati, come testo o serie temporali, per compiti come previsione e generazione di linguaggio. [↑](#footnote-ref-15)
16. **RNN:** tipo di architettura di rete neurale progettata per elaborare dati sequenziali, come testo, audio o serie temporali, mantenendo una memoria interna che consente loro di considerare informazioni precedenti nel contesto di attuali input, rendendole utili per compiti di previsione, classificazione e generazione di sequenze [↑](#footnote-ref-16)
17. **BERT:** modello di linguaggio basato su trasformatori che utilizza l'auto-apprendimento per creare rappresentazioni di parole contestualizzate. Questo significa che considera il contesto circostante di una parola durante l'elaborazione del testo, rendendolo particolarmente efficace per compiti di comprensione del linguaggio naturale, come la traduzione, la classificazione del testo e la risposta alle domande. [↑](#footnote-ref-17)
18. **GPT-2:** modello di linguaggio basato su trasformatori sviluppato da OpenAI. È pre-addestrato su un'ampia quantità di testo da Internet e può essere utilizzato per una varietà di compiti legati al linguaggio naturale, come la generazione di testo, la traduzione automatica, la risposta alle domande e altro ancora [↑](#footnote-ref-18)