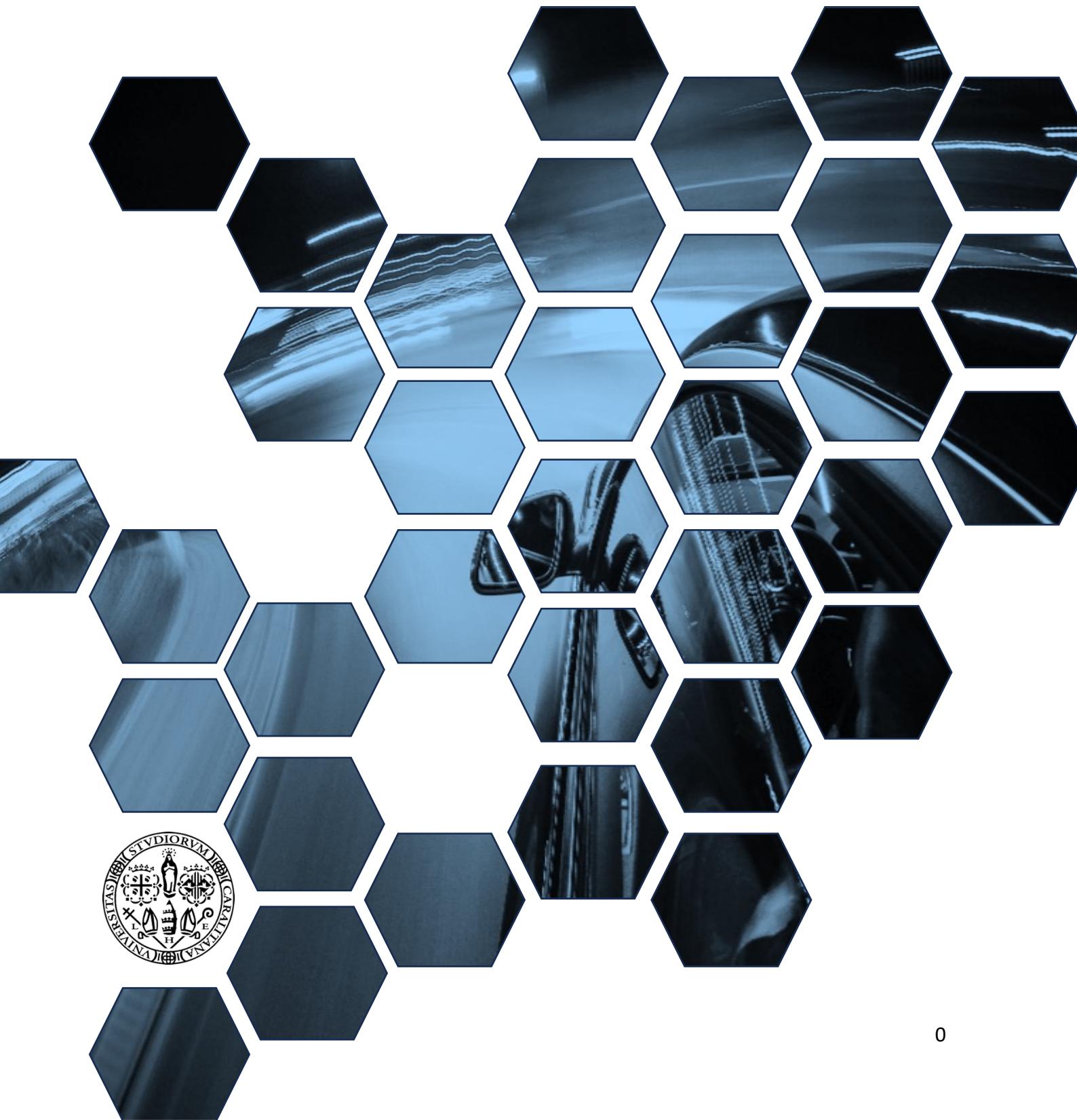


DriveFeelings :

Cosa pensano gli utenti di Twitter su
BMW, Renault e Tesla?



Progetto svolto per scopi accademici ed al fine del superamento dell'esame di Web Analytics e Analisi Testuale 2023/2024 del Corso Magistrale DSBAI :

Christian Putzu 11/82/00333

Alberto Mancosu 11/82/00357

SOMMARIO DEL PROGETTO

DRIVEFEELINGS :	0
INTRODUZIONE	5
BRANCH MASTER	9
README.MD	9
FASE 1: BRANCH TWITTERSCRAPPER	9
ACCOUNT.DB	10
TWSCRAPER.PY	11
TWEETS_ESTRATTI	15
FASE 2: BRANCH PRE-PROCESSING	17
PRE-PROCESSING.PY	17
TWEETS_PULITI	21
FASE 3: BRANCH TOPICMODELING:	22
TOPIC-MODELING.PY	22
FASE 4: BRANCH RoBERTA SENTIMENTANALYSIS	39
FINE-TUNING-MODEL.PY	39
VALIDATION-MODEL.PY	44
CLASSIFICATION-MODEL.PY	49
TWEETS_SENTIMENT	52
RISULTATI E CONCLUSIONI	54

INDICE DELLE FIGURE

FIGURA 1. ACCOUNT.DB	11
FIGURA 2. GESTIONE DELLE LIMITAZIONI.	11
FIGURA 3. CLASSE TWITTERSCRAPER	11
FIGURA 4. LOGIN.....	12
FIGURA 5. STAMPA DI DEBUG CON GLI ACCOUNTS AGGIUNTI.	12
FIGURA 6. SCRAPE_TWEETS.....	13
FIGURA 7. FILE ESTRATTI CON IL PROCESSO DI SCRAPING.	13
FIGURA 8. WRITE_TWEETS_TO_CSV.	14
FIGURA 9. FILE ESTRATTI.	15
FIGURA 10. MAIN: TWSCRAPE.PY.	16
FIGURA 11. CLASS TEXTPREPOCESSOR.	17
FIGURA 12. FUNZIONE PREPROCESS_TEXT.	18
FIGURA 13. STOPWORDS AGGIUNTIVE.	19
FIGURA 14. CLASS DATAPROCESSOR	19
FIGURA 15. PREPROCESS_CVS:	20
FIGURA 16. FUNZIONI: PREPROCESS_CVS, PREPROCESS_ALL.....	20
FIGURA 17. FILE PREPROCESSATI: BMW, TESLA, RENAULT.....	21
FIGURA 18. MAIN: PRE-PROCESSING.PY.	21
FIGURA 19. CLASS TOPICMODELINGANALYZER.	22
FIGURA 20. GET_INPUT_FILES()	23
FIGURA 21. PERFORM_TOPIC_MODELING(INPUT_FILENAME).....	23
FIGURA 22. DISPLAY_TOPICS(TF_VECTORIZER, LDA, N_WORDS=10).....	24
FIGURA 23. CREATE_WORDCLOUD(INPUT_FILENAME)	24
FIGURA 24. MAIN: TOPIC-MODELING.PY.....	26
FIGURA 25. WORDCLOUD PER BMW.	27
FIGURA 26. WORDCLOUD PER TESLA.	27
FIGURA 27. WORDCLOUD PER RENAULT.....	27
FIGURA 28. GRAFICO DELLA VERO SIMIGLIANZA LOGARITMICA DI BMW.	29
FIGURA 29. STAMPA DELL'OUTPUT DELLA TOPIC MODELING PER BMW.....	29
FIGURA 30. GRAFICO DELLA VERO SIMIGLIANZA LOGARITMICA DI TESLA.	30
FIGURA 31. STAMPA DELL'OUTPUT DELLA TOPIC MODELING PER TESLA.	30
FIGURA 32. GRAFICO DELLA VERO SIMIGLIANZA LOGARITMICA DI RENAULT.....	31
FIGURA 33. STAMPA DELL'OUTPUT DELLA TOPIC MODELING PER RENAULT.....	32
FIGURA 34. RISULTATI DELLA PAGINA HTML DI BMW (PARTE 1).	33
FIGURA 35. RISULTATI DELLA PAGINA HTML DI BMW (PARTE 2).....	34
FIGURA 36. RISULTATI DELLA PAGINA HTML DI TESLA (PARTE 1).....	35

FIGURA 37. RISULTATI DELLA PAGINA HTML DI TESLA (PARTE 2)	36
FIGURA 38. RISULTATI DELLA PAGINA HTML DI RENAULT (PARTE 1)	37
FIGURA 39. RISULTATI DELLA PAGINA HTML DI RENAULT (PARTE 2)	38
FIGURA 40. FINE-TUNING.....	40
FIGURA 41. CLASS TWITTERDATASET.	41
FIGURA 42. MODELLO SU HUGGING FACE.....	42
FIGURA 43. CLASS TWITTER_SENTIMENT_ANALYSIS.....	43
FIGURA 44. MAIN: FINE-TUNING-MODEL.PY.....	44
FIGURA 45. VALIDATE_MODEL_FINE_TUNED.....	45
FIGURA 46. VALIDATE_MODEL_PRE_TRAINED	46
FIGURA 47. MAIN: VALIDATION-MODEL.PY.....	47
FIGURA 48. DATASET: TRAIN_AIRLINES-SENTIMENT.CSV	47
FIGURA 49. MATRICE DI CONFUSIONE E CLASSIFICATION REPORT DEL MODELLO FINE-TUNED.	48
FIGURA 50. MATRICE DI CONFUSIONE E CLASSIFICATION REPORT DEL MODELLO PRE-TRAINED.	48
FIGURA 51. CLASS SENTIMENTANALYZER.	49
FIGURA 52. ANALYZE_SENTIMENT_ROBERTA.....	50
FIGURA 53. MAIN: CLASSIFICATION-MODEL.PY.....	51
FIGURA 54. TWEETS_SENTIMENT: BMW, TESLA E RENAULT.	52
FIGURA 55. GRAFICO A BARRE BMW.....	52
FIGURA 56. GRAFICO A BARRE TESLA.....	53
FIGURA 57. GRAFICO A BARRE RENAULT.	53

Introduzione

Alla base della scelta di questo progetto, c'è la nostra grande passione per i motori, in particolare per le automobili. Questa motivazione ci ha spinto ad indagare su quello che è il pensiero degli utenti del social network Twitter riguardo ad alcune delle case automobilistiche più famose. In particolare, abbiamo scelto una casa automobilistica preferita a testa privilegiando determinate caratteristiche distintive:

- BMW, scelta da Alberto per la sportività delle proprie automobili.
- Tesla, scelta da Christian per l'innovazione tecnologica nel settore elettrico.

Ed una terza casa automobilistica:

- Renault scelta per il mix tra le due, in quanto produce veicoli elettrici ed allo stesso partecipa a sport motoristici come la Formula 1.

Questo progetto, quindi rappresenta un'opportunità per soddisfare la nostra curiosità su questa tematica e grazie alle tecniche utilizzate di sentiment analysis e topic modeling possiamo inoltre capire cosa gli utenti maggiormente pensano e provano, così da poter confrontare i soggetti dell'analisi e confrontare il grado di apprezzamento degli utenti.

Fonti:

- Materiale didattico e script forniti dal docente
- Stackoverflow
- Github
- Kaggle
- Huggingface.co
- Twitter

Strumenti:

- Git e Bitbucket
- Python e Pycharm
- Chatgpt3.5
- Microsoft 365
- Microsoft Teams
- Microsoft Word
- Microsoft PowerPoint

Librerie utilizzate:

- **asyncio** per la gestione delle operazioni asincrone.
- **twscreape.API** per l'accesso all'API twscreape per lo scraping dei dati twitter.
- **csv** per il salvataggio dei dati su file csv.
- **sys** per l'uscita dal programma.
- **os** per gestire i percorsi dei file e la creazione di directory.
- **contextlib.aclosing** per garantire la chiusura quando il contesto è terminato.
- **pandas** per importare, manipolare ed elaborare i dati.
- **nltk** per l'elaborazione del linguaggio naturale.
- **sklearn** per sviluppare e addestrare modelli di machine learning e contenente le librerie feature_extraction.text per CountVectirzer, decomposition per LatentDirichletAllocation, e model_selection per GridSearchCV e train_test_split, metrics per accuracy_score, confusion_matrix e classification_report.
 - **CountVectorizer** per convertire i testi dei tweet in una rappresentazione numerica, contando la frequenza delle parole in ciascun tweet.
 - **LatentDirichletAllocation** è un algoritmo di topic modeling utilizzato per scoprire argomenti nascosti all'interno di un insieme di documenti.
 - **GridSearchCV** per la ricerca dei migliori parametri di un modello di machine learning.
 - **train_test_split** per la divisione del dataset in training set e test set.
 - **accuracy_score** per il calcolo del tasso di accuracy di classificazione del modello.
 - **confusion_matrix** per il calcolo della matrice di confusione.
 - **classification_report** per il calcolo del report di classificazione metriche come accuracy F1 score, Precision, Recall e Support.
- **scipy.special** per la risoluzione di problemi matematici avanzati che coinvolgono funzioni speciali.
- **trasformers** per accedere a modelli preaddestrati come RoBERTa.
 - **RobertaTokenizer** si occupa della tokenizzazione dei testi per l'uso con i modelli Roberta.
 - **RobertaForSequenceClassification** è un modello pre-addestrato specifico per compiti di classificazione di sequenze.
 - **Pipeline** questo modulo permette di combinare diverse operazioni, come tokenizzazione, predizione e altre attività NLP.
 - **TrainingArguments** contiene tutti gli argomenti e i parametri necessari per configurare e gestire l'addestramento di un modello.
 - **Trainer** è un'interfaccia ad alto livello che semplifica il ciclo di addestramento di un modello.
- **warnings** per ignorare gli errori non rilevanti.
- **tqdm** per mostrare barre di avanzamento durante i processi lunghi.
- **Matplotlib.pyplot** per la visualizzazione grafica dei risultati.
- **Torch** fornisce strumenti e risorse per creare e addestrare reti neurali.
- **PyLDSavis.Ida_model** per la generazione del panel interattivo del modello LDA e per l'esportazione su pagina html.
- **wordcloud** per creare grafici Word Cloud che mostrano le parole più frequenti nei documenti.
- **seaborn** per generare una rappresentazione visiva più accattivante e informativa dei dati statistici.

Il progetto si compone di quattro fasi a cui corrisponde un branch dedicato:

Raccolta dei Dati: BRANCH TwitterScrapper

Abbiamo strutturato il progetto in modo da raccogliere un ampio campione di tweet contenenti le parole chiave BMW, Renault e Tesla. La raccolta dei dati riguarda l'estrazione di 10000 tweets per ciascuna casa automobilistica per un totale di 30000. Questo numero di dati è stato scelto pensando al compromesso tra la nostra capacità computazionale e un'adeguata dimensione del campione, che a nostro avviso ci consentirà di ottenere una panoramica abbastanza precisa e completa delle discussioni online riguardo a queste case automobilistiche.

Preparazione dei Dati: BRANCH PreProcessing

Una volta effettuata l'estrazione dei dati, abbiamo effettuato la loro pulizia e preparazione. Abbiamo dunque rimosso: URL, hashtag, menzioni, caratteri speciali e parole aggiuntive dai tweet, allo scopo di mantenere solo il testo rilevante. Questa fase di pulitura dei tweets è stata effettuata per la predisposizione all'analisi di topic modeling e sentiment analysis.

Analisi dei Dati: BRANCH TopicModeling

In questo branch abbiamo eseguito un'analisi delle parole chiave al fine di individuare i principali temi e argomenti che emergono nei tweet relativi a BMW, Renault e Tesla. Questa fase ci ha permesso analizzare a fondo le conversazioni su Twitter riguardo a tali marchi automobilistici e di categorizzare i tweet estratti in diverse aree tematiche.

Classificazione del Sentiment: BRANCH RoBERTa SentimentAnalysis

Abbiamo condotto un'analisi comparativa del sentiment, considerando le categorie (Positivo, Neutro, Negativo) e le varie case automobilistiche. Utilizzando un modello pre-addestrato di RoBERTa, lo abbiamo ulteriormente addestrato attraverso il processo di fine-tuning. Un aspetto centrale del nostro progetto è stato l'implementazione di un classificatore di sentiment, la cui scelta è stata determinata dal confronto dell'accuracy tra il modello pre-addestrato RoBERTa "out of box" e il modello fine-tuned su un dataset di validazione esterno. Abbiamo quindi scelto di utilizzare il modello pre-addestrato di RoBERTa per l'analisi.

Interpretazione dei risultati e conclusioni

Una volta ottenuti i risultati, abbiamo esaminato attentamente le opinioni degli utenti su ciascuna delle case automobilistiche, cercando di identificare i motivi alla base dei sentiment positivi e negativi e di comprenderne i principali argomenti di discussione.

Presentazione dei Risultati:

Report_Tecnico_Putzu_Mancosu.pdf, Presentazione_Putzu_Mancosu.pdf.

Infine, abbiamo condiviso i nostri risultati in un rapporto dettagliato dove abbiamo utilizzato grafici e risultati per illustrare le tendenze e i punti salienti emersi dal nostro studio, consentendo a chi legge di ottenere una chiara visione delle opinioni degli utenti su BMW, Renault e Tesla.

Branch Master

Questo branch è destinato soltanto a contenere i file:

- README.md,
- Report_Tecnico_Putzu_Mancosu.pdf,
- Presentazione_Putzu_Mancosu.pdf.

README.md

In breve, il progetto riguarda l'analisi dei tweet degli utenti di Twitter relativi a tre case automobilistiche: BMW, Renault e Tesla. Gli obiettivi principali includono la comprensione delle opinioni degli utenti su queste marche automobilistiche, l'identificazione dei sentimenti associati a ciascuna marca e l'individuazione degli argomenti trattati nei tweet. Il progetto prevede anche la creazione di un classificatore di sentiment basato su architetture BERT. Le quattro fasi del progetto includono l'estrazione dei tweet, la pulizia dei dati, il raggruppamento in categorie di argomenti, l'analisi dei sentiment e l'addestramento del classificatore di sentiment. Ogni fase è trattata in dettaglio in branch separati del progetto.

DriveFeelings: Cosa pensano gli utenti di Twitter su BMW, Renault e Tesla?

Il progetto analizza i testi dei tweets degli utenti del social network Twitter che contengono le parole chiave di tre case automobilistiche scelte casualmente tra le più rilevanti del settore:

1. BMW
2. Renault
3. Tesla

L'obiettivo del progetto è quello di capire cosa pensano gli utenti riguardo queste auto, quale è la più apprezzata o la meno apprezzata, ma anche gli argomenti a cui fanno riferimento ed infine implementare un classificatore di sentiment con il modello pre-addestrato di RoBERTa.

Analisi da svolgere

Il progetto si compone di 4 fasi e ad ogni fase corrisponde un branch dedicato:

1. TwitterScraper: Estrazione 10.000 tweets per ogni casa automobilistica, per un Totale 30.000.
 2. PreProcessing: Pulitura dei tweets per la predisposizione all'analisi di sentiment e topic.
 3. TopicModeling: Per raggruppare i tweets estratti in 5 categorie di argomenti o topic.
 4. RoBERTa SentimentAnalysis: sentiment analysis effettuata con modello RoBERTa considerando ogni etichetta (positive/neutral/negative) per ogni casa automobilistica e confrontando l'accuracy tra i modelli pre-trained e fine-tuned, così da etichettare i dati estratti utilizzando il modello più performante.
-

Fase 1: BRANCH TwitterScrapper

Nella prima fase di questo progetto, è stato sviluppato uno scraper per estrarre dati da Twitter utili per le analisi successive. In seguito ad una valutazione delle opzioni di scraping a disposizione tra cui selenium, snscreape e nitter, abbiamo optato per l'unica opzione a nostro avviso funzionante, ovvero, la libreria twscrape, che gestisce il login di account Twitter e i time-out di estrazione in modo da bypassare i nuovi limiti all'attività di scraping imposti da Twitter a partire dal 1° luglio 2023:

Link: <https://twitter.com/elonmusk/status/1675187969420828672>

Elon Musk:

"To address extreme levels of data scraping & system manipulation, we've applied the following temporary limits: - Verified accounts are limited to reading 6000 posts/day - Unverified accounts to 600 posts/day - New unverified accounts to 300/day".

Di seguito sono presentati i file contenenti lo script e i file di input e di output del branch:

- Account.db
- Twscraper.py
- Tweets_estratti .csv

Account.db

Il file "account.db" è un file database utilizzato per archiviare e gestire le credenziali degli account Twitter che possono essere utilizzati per l'accesso e l'autenticazione durante il processo di web scraping.

Questo database contiene informazioni specifiche relative agli account Twitter come il nome utente, la password, l'indirizzo-email e la corrispondente password, necessari per accedere a un account Twitter rispettando i requisiti di sicurezza.

L'obiettivo principale è di aggirare le limitazioni temporali degli account Twitter durante il processo di scraping dei tweet in modo da poter estrarre un gran numero di tweet.

Login accounts:

```
INFO | twscrape.accounts_pool:add_account:97 - Account [REDACTED] added successfully (active=False)
INFO | twscrape.accounts_pool:add_account:97 - Account [REDACTED] added successfully (active=False)
INFO | twscrape.accounts_pool:add_account:97 - Account [REDACTED] added successfully (active=False)
INFO | twscrape.accounts_pool:add_account:97 - Account [REDACTED] added successfully (active=False)
INFO | twscrape.accounts_pool:add_account:97 - Account [REDACTED] added successfully (active=False)
INFO | twscrape.accounts_pool:login_all:155 - [1/5] Logging in [REDACTED] - [REDACTED]
INFO | twscrape.accounts_pool:login:138 - Logged in to [REDACTED] successfully
INFO | twscrape.accounts_pool:login_all:155 - [2/5] Logging in [REDACTED] - [REDACTED]
INFO | twscrape.accounts_pool:login:138 - Logged in to [REDACTED] successfully
INFO | twscrape.accounts_pool:login_all:155 - [3/5] Logging in [REDACTED] - [REDACTED]
INFO | twscrape.accounts_pool:login:138 - Logged in to [REDACTED] successfully
INFO | twscrape.accounts_pool:login_all:155 - [4/5] Logging in [REDACTED] - [REDACTED]
INFO | twscrape.accounts_pool:login:138 - Logged in to [REDACTED] successfully
INFO | twscrape.accounts_pool:login_all:155 - [5/5] Logging in [REDACTED] - [REDACTED]
INFO | twscrape.accounts_pool:login:138 - Logged in to [REDACTED] successfully
```

Figura 1. Account.db

Gestione delle limitazioni temporali:

```
INFO | twscrape.accounts_pool:get_for_queue_or_wait:260 - No account available for queue "SearchTimeline". Next available at 23:46:50
```

Figura 2. Gestione delle limitazioni.

Twscraper.py

Questo script trae ispirazione dal modulo twscrape, ai quali sviluppatori dedichiamo un grosso ringraziamento. Il nostro team ha implementato il codice messo in documentazione al link <https://github.com/vladkens/twscrape> scegliendo le funzioni utili allo scopo del progetto e riscrivendo il codice ad oggetti per mezzo della creazione della classe **TwitterScraper** implementata per la memorizzazione dei dati estratti su file csv.

Il primo step in questo script è stato definire la classe **TwitterScraper** per contenere le seguenti tre funzioni che gestiscono le funzionalità del programma:

```
class TwitterScraper:
    """
    Classe per gestire l'estrazione dei tweets da twitter.
    """

    # Chiamata API alla libreria twscrape
    def __init__(self):
        """
        Inizializza l'oggetto TwitterScraper e crea un'istanza di API per l'accesso alle funzionalità di twscrape.
        """
        self.api = API()
```

Figura 3. Classe TwitterScraper

Login: questa funzione si occupa di aggiungere cinque account Twitter ad una "pool" ovvero, l'insieme di account che verranno utilizzati per l'estrazione dati su Twitter. Successivamente, chiamando **await self.api.pool.login_all()**, vengono eseguiti i tentativi di accesso per ciascun account aggiunto alla "pool". Questo processo consente di effettuare il login simultaneamente su tutti gli account registrati. Le credenziali degli account sono passate come argomenti al metodo **add_account** nella forma (**nome utente, password, email, password**).

```
async def login(self):
    """
    Esegue il login a tutti gli account Twitter salvati nella pool.

    Nota: È necessario aggiungere gli account utilizzando il metodo `add_account` prima di eseguire il login.
    """
    # await self.api.pool.add_account("USERNAME", "USERNAME_PSW", "INDIRIZZO_EMAIL_ACCOUNT_TWITTER", "PSW_INDIRIZZO_EMAIL_ACCOUNT_TWITTER")
    # await self.api.pool.add_account("USERNAME", "USERNAME_PSW", "INDIRIZZO_EMAIL_ACCOUNT_TWITTER", "PSW_INDIRIZZO_EMAIL_ACCOUNT_TWITTER")

    await self.api.pool.login_all()
```

Figura 4. Login.

Una volta eseguito lo script verrà creato definitivamente il file accounts.db. Ad ogni successiva esecuzione del programma verrà riutilizzato il file ed apparirà l'avviso '*Account already exists*'.

```
WARNING | twscrape.accounts_pool:add_account:76 - Account ██████████ already exists
```

Figura 5. Stampa di debug con gli accounts aggiunti.

scrape_tweets: Questa funzione **async def scrape_tweets(self, query, max_tweets=10000)** è usata per l'estrazione dei tweet basati su una query specifica. In particolare, utilizza un ciclo asincrono per ottenere i tweet uno alla volta e li salva in una lista inizializzata vuota, chiamata tweets_list. La funzione prende in ingresso la query di ricerca e tiene conto del numero di tweets estratti.

```

async def scrape_tweets(self, query, max_tweets=10000):
    """
    Esegue lo scraping dei tweet in base a una query specifica con un limite massimo di tweets.
    """

    # Lista principale tweets
    tweets_list = []

    # Contatore quantità tweets
    tweet_count = 0

    # Uscita dal programma quando raggiunge quantità tweets richiesto
    async with aclosing(self.api.search(query)) as gen:

        async for tweet in gen:

            # Stampa debug per ogni tweet estratto
            print(tweet.id, tweet.user.username, tweet.rawContent)

            # Tweet appeso in coda in lista tweets
            tweets_list.append([tweet.user.username, tweet.rawContent])

            # Contatore incrementato +1 per ogni tweet appeso in lista
            tweet_count += 1

            # Controllo condizione di uscita numero tweets
            if tweet_count == max_tweets:
                break

    return tweets_list

```

Figura 6. Scrape_tweets.

async with aclosing(self.api.search(query)) as gen: esegue la ricerca di ogni query su Twitter utilizzando l'API (**self.api**). Inoltre, utilizziamo l'espressione **aclosing** per assicurarci che il contenitore (**gen**) venga chiuso correttamente quando il contesto asincrono viene terminato.

Nella lista **tweets_list** precedentemente inizializzata, viene appeso ogni username e contenuto di ogni tweet col metodo **.append()** che appende in coda i nuovi tweets. Ad ogni tweet estratto, il contatore **tweet_count** viene incrementato di +1. Ogni tweet estratto viene stampato a video con una stampa di debug, mostrando id tweet, username e contenuto del tweet.

```

1716571135184126150 Bmw167914322234 @Partisangirl @IDF I want this video deleted just for the sake of the casualties!
The wrold does not need those footage because theu already know that Israel is the killer
1716570743964672047 Valkence POV: A BMW driver on Forza Motorsport ↗ (TT/spectreXgaming) https://t.co/Y4VXXkD2w0
1716570722863051246 Alhaji_Griot BMW and reliable in the same sentence?☺
1716561793252769839 aliyumalan If you have money get this car, it is one of the most reliable BMW I know
1716570441438159198 carsandbids Now live on Cars & Bids: 2011 BMW 335i Sedan with NO RESERVE! https://t.co/HPnR0wMFJs
1716570376497512727 aaahhhkkkay @SMR_VISION @BMW Can't see the lidar so it's ____??
1716571847108841962 TimEdwardsSF @bmwfremont so forget it BMW of Fremont. I change the oil myself and you can keep your service contracts and promotions. At
1716571285000736785 TimEdwardsSF @bmwfremont I have had to fight for every appt and decent service experience that I have ever gotten from a BMW dealership.
1716570295098544208 BeauRivage8 @Lead_Flinger An option for a BMW https://t.co/fhcliII1ZR
1716570017116897584 willstone_uk @joncoupland Please do share. We lost a whole industry pretty much.

```

My Dad had a Rover 600 in BRG. Believe it had a Honda engine.

Then the 75 diesel (BMW?)

Figura 7. File estratti con il processo di scraping.

write_tweets_to_csv: Attraverso questa funzione abbiamo trascritto i tweets estratti per ogni query su un file CSV dedicato. Nello specifico, questa funzione utilizza il decoratore '@' per attivare uno @staticmethod giustificato dal fatto che è una funzione ausiliaria che non dipende da nessuna istanza specifica della classe **TwitterScraper**, ma è piuttosto una funzione generica per scrivere i dati estratti con lo scraping su un file CSV al raggiungimento della condizione del massimo numero di tweet estratti e senza arrestare l'esecuzione di scraping per le altre query inserite. La funzione in questione prende come parametri di input la lista tweet ed un generico filename su cui vengono scritti i dati.

```
@staticmethod
def write_tweets_to_csv(tweets_list, filename):
    """
    Scrive i dati estratti su un file CSV.

    """
    with open(filename, 'w', newline='') as f:
        # Creazione oggetto per scrivere ogni tweet
        writer = csv.writer(f)

        # Scrittura etichette colonne username e contenuto tweet
        writer.writerow(["Username", "Content"])

        # Ciclo for per scrivere ogni tweet estratto su CSV
        for tweet in tweets_list:

            # Scrittura tweet
            writer.writerow(tweet)
```

Figura 8. Write_tweets_to_csv.

Al termine dell'esecuzione del programma per ogni query (BMW, Renault e Tesla) abbiamo generato tre file csv rinominati secondo la query obiettivo ed alla lingua di traduzione dei tweet, nel nostro caso 'en' per english:

tweets-BMW-en.csv
tweets-Renault-en.csv
tweets-Tesla-en.csv

Tweets_estratti

La cartella "**tweets_estratti**" presente nel branch e richiamata nello script è adibita a contenitore dei tre file CSV di output relativi ai dati estratti da Twitter correlati alle query. Ogni file CSV rappresenta una raccolta di 10.000 tweet per ciascuna query e contiene dati testuali allo stato grezzo suddivisi per colonne Username e Content. La colonna Content in particolar modo contiene il testo di ogni tweet e rappresenta il dato più importante per le nostre analisi.

```
Username,Content
SurajKarof5966,"@M_Classpyurl Both of u are consumers, but still I prefer d BMW"
Jl____Jl____Jl,@MacRumors BMW - Break My Wireless
OVHcloud_IT,"👉 Il dominio è un elemento importante dell'identità di qualsiasi azienda.

💡 Sapevi che registrare un dominio per più anni ti permette di proteggere al meglio l'immagine del marchio da eventuali minacce?"  
bankyuk1,@InfKiller_TMH @BMW @rollsroycecars @MINI @UCRblxx @AbcGlitched @CarterDaCar @qk_62 @PearlWrap @DEmpireLeaks literally eve  
BMWxAustin,"Feel the wind in your hair and luxury at your fingertips with the BMW 840i convertible, now available at BMW of Austin.  
LRIRw,"#ForzaHorizon5 #FH5 #Forza #ForzaShare
#XboxShare #xbox
#BlueArchive #ブルーアカイブ #ブルーアカ
#阿慈谷ヒフミ
#痛車 #itasha
Car : 2023 BMW M2
Design by @KamikazeS2K https://t.co/TcvJkPgGz2"
a_a_mahmood_,I know it's a BMW. 😊
```

Figura 9. File estratti.

if __name__ == "__main__": gestisce il flusso del programma e le chiamate di funzioni in successione per l'estrazione dei tweet da Twitter e la scrittura dei risultati in file CSV.

Innanzitutto, definiamo l'oggetto "scraper" che rappresenta l'interfaccia per il web scraping su Twitter, istanziato dalla classe "TwitterScraper".

Successivamente, tramite **asyncio.run(scraper.login())**, eseguiamo il login su Twitter utilizzando l'oggetto "scraper". Quindi, popoliamo una lista chiamata "queries" con 3 stringhe, ognuna delle quali rappresenta il nome di una casa automobilistica di interesse, seguito dalla specifica "lang:en" per indicare la lingua inglese dei tweet che desideriamo estrarre.

Definiamo anche la cartella di output per i file CSV come "tweets_estratti" utilizzando **os.makedirs(output_folder, exist_ok=True)**. Per ogni query nel ciclo for, chiamiamo la funzione **scraper.scrape_tweets(query)** utilizzando **asyncio.run()** per eseguire il web scraping dei tweet associati alla query di ricerca. I risultati vengono memorizzati nella lista "tweets_list" ed il nome del file CSV in cui verranno salvati i tweet è generato in base al nome della query, rimuovendo però gli spazi, i due punti e la parola "lang" dalla query originale. Per tenere traccia dell'operazione, facciamo una stampa di debug che indica il percorso del file CSV in cui verranno salvati i tweet estratti.

Infine, chiamiamo la funzione **scraper.write_tweets_to_csv(tweets_list, filename)** per scrivere i tweet estratti nel file CSV specifico.

Questi passaggi vengono eseguiti in un ciclo for per ogni query, permettendo di estrarre e salvare i tweet per tutte le case automobilistiche obiettivo.

```

if __name__ == "__main__":
    """
    Gestione del flusso di esecuzione del programma.
    """

    # Creazione oggetto scraper contenente la classe TwitterScraper
    scraper = TwitterScraper()

    # Chiamata alla funzione login
    asyncio.run(scraper.login()) # Attendere che il login venga completato

    # Lista contenente le Query dei nomi delle case automobilistiche obiettivo seguito dalla lingua
    queries = ["BMW lang:en", "Renault lang:en", "Tesla lang:en"]

    # Cartella di destinazione per i file CSV di output
    output_folder = "tweets_estratti"
    os.makedirs(output_folder, exist_ok=True)

    # Ciclo for attraverso le query
    for query in queries:

        # Chiamata alla funzione di Scrapping scrape_tweets per ogni Query inserita
        tweets_list = asyncio.run(scraper.scrape_tweets(query))

        # Riconominazione filename
        filename = os.path.join(output_folder, f"tweets-{query.replace(' ', '-').replace(':', '')}.csv")

        # Stampa debug per salvataggio file csv
        print(f"Salvataggio CSV: {filename}")

        # Chiamata di funzione per salvataggio tweets estratti in 3 file csv, uno per ogni casa automobilistica obiettivo
        scraper.write_tweets_to_csv(tweets_list, filename)

    # Uscita dal programma
    sys.exit()

```

Figura 10. Main: Twscrape.py.

Fase 2: BRANCH Pre-processing

I tweets estratti con lo scraper contengono rumore, informazioni inutili e testo sporco allo stato grezzo. In questo branch chiamato pre-processing, eseguiamo la pulizia dei tweets eseguendo le seguenti operazioni:

- Rimozione di tweet duplicati
- Rimozione di caratteri speciali, punteggiatura e emoticon
- Tokenizzazione del testo
- Rimozione delle stopwords
- Lemmatizzazione

Di seguito sono presentati i file contenenti lo script e i file di input e di output del branch:

- Pre-processing.py
- Tweets_estratti
- Tweets_puliti

Pre-processing.py

All'interno di questo programma, abbiamo definito la classe “**TextPreprocessor**”, responsabile della pulizia dei testi presenti nei tweet. La classe esegue il download dei pacchetti punkt, stopwords, wordnet necessari per il funzionamento di nltk. Inoltre, inizializza il lemmatizzatore WordNetLemmatizer, le stopwords inglesi default ed i filename_tokens ovvero una lista di stopwords aggiuntive individuate in corso d'opera. La classe textProcessor contiene le seguenti funzioni:

```
class TextPreprocessor:

    """Questa classe è responsabile del pre-processing dei testi presenti nei tweet."""
    def __init__(self):
        """ Inizializza l'oggetto TextPreprocessor."""
        # Download dei dati necessari da NLTK
        nltk.download('punkt')
        nltk.download('stopwords')
        nltk.download('wordnet')

        # Inizializzazione del lemmatizzatore e delle stopwords in lingua inglese di WordNet
        self.lemmatizer = WordNetLemmatizer()
        self.stop_words = set(stopwords.words('english'))

        # Stopwords aggiuntive non influenti da eliminare
        self.filename_tokens = ["bmw", "renault", "tesla", "new", "like", "one", "know",
                               "u", "musk", "na", "yes", "baby", "amp", "car", "time"]
```

Figura 11. Class TextPreprocessor.

preprocess_text: accetta un testo come input e lo sottopone a una serie di trasformazioni.

```
def preprocess_text(self, text):

    """Esegue il pre-processing del testo del tweet."""

    # Trasformazione di tutti i caratteri in minuscolo
    text = text.lower()

    # Rimozione delle occorrenze di 'http' e 'https' per eliminare i link
    text = text.replace('http', '').replace('https', '')

    # Rimozione delle parole che iniziano con '@' per eliminare i tag degli utenti
    text = ' '.join(word for word in text.split() if not word.startswith('@'))

    # Tokenizzazione del testo in parole
    tokens = word_tokenize(text)

    # Rimozione di punteggiature, caratteri speciali e numeri
    tokens = [word for word in tokens if word.isalpha()]

    # Rimozione delle stopwords in lingua inglese e conversione in minuscolo
    tokens = [word for word in tokens if word.lower() not in self.stop_words]

    # Rimozione delle parole presenti nella lista filename_tokens
    tokens = [word for word in tokens if word.lower() not in self.filename_tokens]

    # Lemmatizzazione delle parole
    tokens = [self.lemmatizer.lemmatize(word) for word in tokens]

    # Ricostruzione del testo pre-processato
    preprocessed_text = ' '.join(tokens)

    return preprocessed_text
```

Figura 12. Funzione preprocess_text.

Con “**text = text.lower()**” effettuiamo la trasformazione di tutti i caratteri in minuscolo.

Con il metodo “**text = text.replace('http', '').replace('https', '')**” rimuoviamo le occorrenze di "http" e "https" dal testo che indicano la presenza di link esterni di reindirizzamento, in quanto queste informazioni non sono rilevanti al fine dell'obiettivo dell'analisi di sentiment e topic modeling.

Con **text = ' '.join(word for word in text.split() if not word.startswith('@))** invece, abbiamo eliminate tutte le parole che iniziano con "@" in quanto risultano essere le menzioni degli utenti e non vogliamo che vengano prese in considerazione tra le parole più frequenti.

Con “**tokens = word_tokenize(text)**” abbiamo tokenizzato il testo, ovvero, suddiviso in parole individuali.

Con “**tokens = [word for word in tokens if word.isalpha()]**” abbiamo rimosso i segni di punteggiatura, caratteri speciali e numeri.

Con “**tokens = [word for word in tokens if word.lower() not in self.stop_words]**” abbiamo rimosso le stop words di default di nltk, ovvero, parole comuni che spesso non esprimono un significato specifico.

Infine, per migliorare ulteriormente la qualità dei dati, con “**tokens = [word for word in tokens if word.lower() not in self.filename_tokens]**” abbiamo rimosso anche un set di parole aggiuntive ed arbitrarie presenti nella lista “**filename_tokens**”:

```
# Stopwords aggiuntive non influenti da eliminare
self.filename_tokens = ["bmw", "renault", "tesla", "new", "like", "one", "know",
                      "u", "musk", "na", "yes", "baby", "amp", "car", "time"]
```

Figura 13. Stopwords aggiuntive.

Con “**tokens = [self.lemmatizer.lemmatize(word) for word in tokens]**” il testo viene sottoposto a lemmatizzazione, che riduce le parole alla loro forma base, garantendo una maggiore coerenza nei dati pre-processati.

Inizialmente, avevamo inserito uno stemmer, utile per ridurre la forma di una parola alla sua radice. Abbiamo effettuato delle prove sia con il metodo di Porter, sia con il metodo di Lancaster, ma ci è stato subito evidente che questi metodi abbassassero le prestazioni tagliando le parole in maniera inopportuna, e quindi alterano anche i risultati dell’analisi. Per questo motivo abbiamo optato per non adottare questo metodo. Infine, “**preprocessed_text = ' '.join(tokens)**”, ci ha permesso di ricostruire l’intero testo pre-processato.

Successivamente abbiamo creato un’altra classe chiamata “**DataPreprocessor**”, responsabile del pre-processing dei file CSV contenenti i tweets. Questa classe è inizializzata con i file di input ed output, le rispettive cartelle e la classe TextProcessor. All’interno di DataProcessor sono contenute le seguenti funzioni:

```
class DataProcessor:

    """Questa classe è responsabile del pre-processing dei file CSV contenenti tweets."""

    def __init__(self, input_files, output_files, input_folder, output_folder):

        """ Inizializza l’oggetto DataProcessor.

        self.input_files = input_files
        self.output_files = output_files
        self.input_folder = input_folder
        self.output_folder = output_folder
        self.text_preprocessor = TextPreprocessor()
```

Figura 14. Class DataProcessor

preprocess_csv: legge il file CSV di input dalla cartella "tweets_estratti" (tweets-BMW-en.csv, tweets-Renault-en.csv, tweets-Tesla-en.csv) attraverso “**pd.read_csv**”, e utilizzando la funzione “**preprocess_text**” per ciascun testo nella colonna “Content” del file in input, viene effettuato il pre-processing dei dati attraverso:

“**df['Content'] = df['Content'].apply(self.text_preprocessor.preprocess_text)**”.

```
def preprocess_csv(self, input_filename, output_filename):

    """Esegue il pre-processing di un file CSV di input e salva il risultato in un nuovo file."""

    # Legge il file CSV in un DataFrame
    input_path = os.path.join(self.input_folder, input_filename)
    df = pd.read_csv(input_path)

    # Applicazione del pre-processing alla colonna 'Content' del DataFrame
    df['Content'] = df['Content'].apply(self.text_preprocessor.preprocess_text)

    # Salva il DataFrame pre-processato in un nuovo file CSV
    output_path = os.path.join(self.output_folder, output_filename)
    df.to_csv(output_path, index=False)
```

Figura 15. *preprocess_csv*:

Infine, con “**output_path = os.path.join(self.output_folder, output_filename)**” salviamo i dati puliti in tre diversi file CSV nella cartella "tweets_puliti":

tweets_puliti-BMW-en.csv,
tweets_puliti-Renault-en.csv,
tweets_puliti-Tesla-en.csv.

preprocess_all: la funzione automatizza il pre-processing tramite un ciclo for. Nello specifico, scorre attraverso ciascun file di input csv e chiama la funzione “**preprocess_csv**” per elaborare i dati e salvarli come file csv puliti.

```
def preprocess_all(self):

    """Esegue il pre-processing per tutti i file di input."""

    for input_file, output_file in zip(self.input_files, self.output_files):
        self.preprocess_csv(input_file, output_file)
```

Figura 16. Funzioni: *preprocess_csv*, *preprocess_all*

Tweets_puliti

La cartella "tweets_puliti" serve ad archiviare i dati puliti dei tweets. All'interno di questa cartella troviamo i tre file CSV che serviranno successivamente per l'analisi di sentiment e topic modeling. Nelle seguenti immagini è riportata la struttura dei file CSV contenenti i dati ripuliti.

Username (1)	Content (2)	Username (1)	Content (2)	Username (1)	Content (2)
1 Username	Content	1 Username	Content	1 Username	Content
2 LI3AModels	bought drive	2 SurajKarof5966	u consumer still prefer	2 cblaymire	collected zoe gt line plus reg second
3 KarenV35692848	big thing taken consideration calculation worker get stock many	3 JL_____JL_____JL	break wireless	3 muckyfellrunner	zoe wife month love question trying
4 KureWellness	unplan known wanthu bina park biden would say year	4 OVHcloud_IT	il dominio è un elemento importante di qualsiasi azienda sapevi che	4 karma_shopping	remembering brother used decrepit era
5 yramkee	game whether ev incentive ev mostly win better car	5 bankyuk1	literally everyone see	5 colinwalker79	think would fun race around
6 hikingskiing	predict ultimately able assemble vehicle lt hour	6 BMWAustin	convertible available austin experience unmatched elegance	6 renault_uk	overpay amazon ever amazon shopping
					hack prime day steroid
					top notch trolling
					vive electric revolution

Figura 17. File preprocessati: BMW, Tesla, Renault

if __name__ == "__main__": all'interno del main definiamo le liste di file di input sporchi e dei file di output puliti, quindi definiamo le directory di lavoro da cui vengono archiviati i file **input_folder = "./tweets_estratti"** e **output_folder = "./tweets_puliti"**. Infine, creiamo l'oggetto della classe **Data_preprocessor** con ed eseguiamo il processo di pulizia del testo con la funzione **data_processor.preprocess_all()**, al cui termine viene stampato a video il messaggio di debugging: "Pre-processing completato." per indicare che l'intero processo di pre-processing è stato eseguito con correttamente.

```

if __name__ == "__main__":
    # Files di input sporchi
    input_files = [
        "tweets-BMW-en.csv",
        "tweets-Renault-en.csv",
        "tweets-Tesla-en.csv"
    ]

    # Files di output puliti
    output_files = [
        "tweets_puliti-BMW-en.csv",
        "tweets_puliti-Renault-en.csv",
        "tweets_puliti-Tesla-en.csv"
    ]

    # Directory input
    input_folder = "./tweets_estratti"

    # Directory output
    output_folder = "./tweets_puliti"

    # Creazione di un oggetto DataPreprocessor
    data_processor = DataProcessor(input_files, output_files, input_folder, output_folder)

    # Esecuzione del pre-processing per tutti i file di input
    data_processor.preprocess_all()

    # Stampa debug pre-processing eseguito correttamente
    print("Pre-processing completato.")

```

Figura 18. Main: Pre-processing.py.

Fase 3: BRANCH TopicModeling:

L'obiettivo dell'analisi di topic modeling è quello di identificare i principali argomenti o "topic" da un insieme di documenti. Nel nostro caso specifico questo tipo di analisi è stata condotta per identificare le principali discussioni dai tweet e condurre un'analisi avanzata dei dati testuali relativi ai marchi automobilistici obiettivo.

A questo scopo è stato utilizzato il modello Latent Dirichlet Allocation (LDA), perché il suo principale obiettivo è quello di scoprire i temi o le tematiche nascoste all'interno di un insieme di documenti, inoltre è un modello probabilistico particolarmente adatto per identificare automaticamente le tematiche principali all'interno di grandi quantità di testo, come i tweet.

Una volta addestrato il modello LDA, è infatti possibile esplorare e analizzare i topic per ottenere una comprensione dettagliata delle tematiche più frequenti discusse dagli gli utenti di Twitter. Questo approccio consente di ottenere insights preziosi sulle conversazioni e le tendenze presenti sulla piattaforma Twitter relative alle query di ricerca aiutandoci ad individuare argomenti rilevanti.

In seguito sono presentati i file contenenti lo script e i file di input e di output del branch:

- Topic-modeling.py
- Tweets_puliti
- *LDA_panel_tweets_puliti-BMW-en.html*
- *LDA_panel_tweets_puliti-Tesla-en.html*
- *LDA_panel_tweets_puliti-Renault-en.html*

Topic-modeling.py

Per il funzionamento di questo programma, abbiamo definito la classe "**TopicModelingAnalyzer**", responsabile della gestione dell'intero processo di topic modeling. Questa classe inizializza l'input_folder che contiene i file csv di input.

```
class TopicModelingAnalyzer:  
    """Classe che gestisce la Topic Modeling"""  
  
    def __init__(self, input_folder):  
        # Inizializzazione directory di input  
        self.input_folder = input_folder
```

Figura 19. Class TopicModelingAnalyzer.

La classe TopicModeling contiene le seguenti funzioni:

get_input_files() che esplora la cartella di input "tweets_puliti" e identifica i file CSV delle case automobilistiche, appendendo ogni file nella lista di file di inputs e restituendo la lista stessa.

```
def get_input_files(self):  
  
    """Funzione che restituisce una lista dei file CSV nella cartella di input."""  
  
    input_files = []  
    for root, dirs, files in os.walk(self.input_folder):  
        for file in files:  
            if file.endswith(".csv"):  
                input_files.append(os.path.join(root, file))  
    return input_files
```

Figura 20. `get_input_files()`

La funzione **perform_topic_modeling(input_filename)** accetta un file CSV come input utilizzando la libreria pandas e avvia il processo di elaborazione dei dati testuali contenuti al suo interno. Nel caso in cui siano presenti valori NA, gestiamo questa situazione sostituendoli con stringhe vuote tramite l'istruzione `df['Content'].fillna("", inplace=True)`.

Successivamente, procediamo all'inizializzazione del vettorizzatore **CountVectorizer**, che svolge il ruolo di convertire il testo in una rappresentazione numerica calcolando le frequenze delle parole. In questo contesto, è stato scelto di utilizzare il TF (Term Frequency) per valutare l'importanza di ciascuna parola all'interno dei file csv in base alla frequenza di ciascuna parola. Questa scelta si riflette nella linea di codice `tf = tf_vectorizer.fit_transform(df['Content'])` in cui il vettorizzatore **tf_vectorizer** trasforma il testo contenuto nella colonna 'Content' del DataFrame df in una rappresentazione numerica. Di conseguenza, ogni riga nel DataFrame viene convertita in un vettore di frequenze assolute delle parole.

```
def perform_topic_modeling(self, input_filename):  
  
    """Funzione che esegue la topic modeling su ogni file CSV specifico."""  
  
    # Lettura del file CSV  
    df = pd.read_csv(input_filename)  
  
    # Gestione dei valori NaN nella colonna 'Content'  
    df['Content'].fillna('', inplace=True) # Sostituisci i valori NaN con stringhe vuote  
  
    # Inizializzazione del vettorizzatore CountVectorizer per le frequenze delle parole (tf)  
    tf_vectorizer = CountVectorizer(max_df=0.95, min_df=2, stop_words='english')  
    tf = tf_vectorizer.fit_transform(df['Content'])  
  
    return tf_vectorizer, tf, tf_vectorizer.transform(df['Content'])
```

Figura 21. `perform_topic_modeling(input_filename)`

display_topics(tf_vectorizer, lda, n_words=10) è la funzione responsabile della generazione e della restituzione dei topic individuati dal modello LDA. Il processo coinvolge l'estrazione delle parole chiave più rilevanti per ciascun topic.

```

def display_topics(self, tf_vectorizer, lda, n_words=10):

    """Funzione che genera e restituisce i topic individuati dal modello LDA."""

    feature_names = tf_vectorizer.get_feature_names_out()
    topics = {}

    for topic_idx, topic in enumerate(lda.components_):
        top_n_words = [feature_names[i] for i in topic.argsort()[:-n_words - 1:-1]]
        topics[f"Topic {topic_idx + 1}"] = top_n_words

    return topics

```

Figura 22. `display_topics(tf_vectorizer, lda, n_words=10)`

Infine, la funzione `create_wordcloud(input_filename)` produce grafici Word Cloud basati sulle parole più frequenti in ciascun file, fornendo un'illustrazione visiva delle parole chiave più rilevanti e rappresentative di ciascun file csv. Questo tipo di grafici sono visualizzati creando l'oggetto WordCloud e facendo il plot con l'opzione imshow di matplotlib. Abbiamo optato per l'utilizzo di questi grafici data la loro semplicità e immediatezza nell'effettuare una prima analisi esplorativa.

```

def create_wordcloud(self, input_filename):

    """Funzione che crea e visualizza i grafici word cloud basati sul contenuto del file CSV specifico."""

    # Lettura del file CSV
    df = pd.read_csv(input_filename)

    # Gestione dei valori NaN nella colonna 'Content'
    df['Content'].fillna('', inplace=True) # Sostituzione dei valori NaN con stringhe vuote

    # Join dei testi
    long_string = ','.join(df['Content'].to_list())

    # Creazione oggetto WordCloud per generare i grafici
    wordcloud = WordCloud(background_color="white", max_words=5000, contour_width=3, contour_color='steelblue')
    wordcloud.generate(long_string)

    # Grafici Word Cloud
    plt.figure(figsize=(10, 10))
    plt.imshow(wordcloud, interpolation="bilinear")
    plt.axis("off")
    plt.title("Word Cloud")
    plt.show()

```

Figura 23. `create_wordcloud(input_filename)`

Al termine di questa procedura sono stati creati come output tre file html:

LDA_panel_BMW.html,
LDA_panel_Tesla.html,
LDA_panel_Renault.html

Che sono stati inseriti all'intero della cartella **pagine_html_LDA** come unica directory di output.

`if __name__ == "__main__":` all'interno del main definiamo le directory di input e di output. Creiamo l'oggetto `topic_modeling_analyzer = TopicModelingAnalyzer(input_folder)` legato alla classe TopicModelingAnalyzer per poter richiamare le funzioni al suo interno.

Eseguiamo un ciclo for che itera attraverso ogni elemento nella lista `input_files`.

Il ciclo permette l'estrazione dei nomi di ogni file all'interno della lista.

Stampiamo un messaggio del debug per verificare quale file CSV è in elaborazione attraverso `print(f"\nTopic Modeling per il file input '{file_name}' con numero di topics ottimale della GridSearchCV:\n")`.

Eseguiamo l'LDA con `lda = LatentDirichletAllocation()`: abbiamo creato un oggetto LDA (Latent Dirichlet Allocation) vuoto. Questo oggetto rappresenta il modello LDA che verrà addestrato durante l'analisi dei topic.

La chiamata di funzione `model = GridSearchCV(lda, param_grid=searchParams, verbose=3, n_jobs=-1)` ci ha permesso di utilizzare la GridSearchCV che esegue una ricerca esaustiva su tutte le combinazioni dei parametri specificati in `searchParams`, ovvero, il numero ottimale di topic ed il learning decay per identificare il valore ottimale che massimizzi le prestazioni del modello, tenendo conto di come l'adattamento del modello influisca sui risultati del topic modeling.

Con la funzione `model.fit(tf)` avviamo il processo di ricerca dei migliori parametri del modello LDA. Il modello viene addestrato su `tf`, che è la frequenza assoluta del termine precedentemente calcolata dal metodo `perform_topic_modeling`.

Con la funzione `best_lda_model = model.best_estimator_` dopo la ricerca, estraiamo il miglior modello LDA, identificato come `best_lda_model`. Questo modello avrà i parametri ottimali determinati durante la ricerca.

Dopo l'analisi effettuiamo delle stampe di debug sia dei risultati del GridSearchCV:

- Miglior punteggio di verosimiglianza logaritmica,
- Parametri del miglior modello,
- Perplexity del modello,
- Numero di topic ottimale.

Effettuiamo anche la stampa di debug per visualizzare l'indice dei topic e le parole più frequenti associate.

Infine, salviamo la cronologia della gridsearch per i parametri in due variabili `n_topics` e `log_likelihoods`. La lista `n_topics` contiene il numero di topic testati, mentre `log_likelihoods` raccoglie gli score di verosimiglianza logaritmica.

Attraverso questi parametri, creiamo un grafico utilizzando la libreria Matplotlib per visualizzare i punteggi di log-likelihood in funzione del numero di topic e del parametro "learning decay".

Utilizzando il modello LDA ottimale `best_lda_model` con la funzione `pyLDAvis.Lda_model.prepare`, creiamo un pannello interattivo che consente di esplorare i topic in modo approfondito. Il pannello fornisce una varietà di informazioni e visualizzazioni, tra cui diversi grafici utili per esplorare e comprendere meglio i risultati.

Abbiamo quindi salvato i risultati del modello LDA in una pagina HTML che contiene il panel pyLDAvis. Il nome del file HTML è creato utilizzando il nome del file di input, e successivamente viene salvato nella directory di output `pagine_html_LDA`.

L'ultimo step di questo programma è la chiamata di funzione a `create_wordcloud` per generare i grafici esplorativi Wordcloud in cui le parole più frequenti nei topic vengono visualizzate in modo più evidente.

```

if __name__ == "__main__":
    # Directory di input
    input_folder = "./tweets_puliti"

    # Directory di output
    output_folder = "pagine_html_LDA"

    # Creazione di un oggetto TopicModelingAnalyzer
    topic_modeling_analyzer = TopicModelingAnalyzer(input_folder)

# Ciclo for per automatizzare l'esecuzione del programma per ogni file
input_files = topic_modeling_analyzer.get_input_files()
for input_file in input_files:

    # Estrazione solo del nome del file CSV dal percorso completo
    file_name = os.path.basename(input_file)

    # Stampa di debug per vedere su quale file CSV stiamo lavorando
    print(f"\nTopic Modeling per il file input '{file_name}' con numero di topics ottimale della GridSearchCV:\n")

    # Esecuzione GridSearchCV per trovare il numero di topic ottimale e LDA
    vectorizer, dtm, tf = topic_modeling_analyzer.perform_topic_modeling(input_file)
    searchParams = {'n_components': [5, 10, 15, 20, 25, 30], 'learning_decay': [.5, .7, .9]}
    lda = LatentDirichletAllocation()
    model = GridSearchCV(lda, param_grid=searchParams, verbose=3, n_jobs=-1)
    model.fit(tf)
    best_lda_model = model.best_estimator_
    num_topics = best_lda_model.n_components

    # Stampa debug dei risultati del grid search
    print("\nMiglior Punteggio di verosimiglianza logaritmica (Log Likelihoods):", model.best_score_)
    print("Parametri del modello migliore:", model.best_params_)
    print("Perplexity del modello:", best_lda_model.perplexity(tf))
    print(f"Numero di Topics Ottimale: {num_topics}\n")

    # Stampa di debug che mostra i topic e le keyword associate
    topics = topic_modeling_analyzer.display_topics(vectorizer, best_lda_model)
    for topic, words in topics.items():
        print(f"{topic}: {', '.join(words)}")

    # Parametri per grafici
    n_topics = searchParams['n_components']
    log_likelihoods = [round(model.cv_results_['mean_test_score'][index]) for index, gscore in
                      enumerate(model.cv_results_['params'])]

    # Grafici matplotlib dei modelli della GridSearchCV
    plt.figure(figsize=(12, 8))
    for decay in [0.5, 0.7, 0.9]:
        log_likelihoods = [log_likelihoods[i] for i, gscore in enumerate(model.cv_results_['params']) if
                           gscore['learning_decay'] == decay]
        plt.plot(n_topics, log_likelihoods, label=f'Learning Decay {decay}')

    plt.title("Scelta del modello LDA ottimale")
    plt.xlabel("Numero di Topics")
    plt.ylabel("Punteggi di verosimiglianza logaritmica (Log Likelihoods)")
    plt.legend(loc='best')
    plt.show()

    # Generazione del pannello pyLDAvis
    panel = pyLDAvis lda_model.prepare(lda_model=best_lda_model, dtm=dtm, vectorizer=vectorizer)

    # Creazione pagina HTML del modello LDA
    input_file_name = os.path.splitext(os.path.basename(input_file))[0]
    output_html_path = os.path.join(output_folder, f'LDA_panel_{input_file_name}.html')
    pyLDAvis.save_html(panel, output_html_path)

    # Grafici Wordcloud
    topic_modeling_analyzer.create_wordcloud(input_file)

```

Figura 24. Main: Topic-modeling.py.

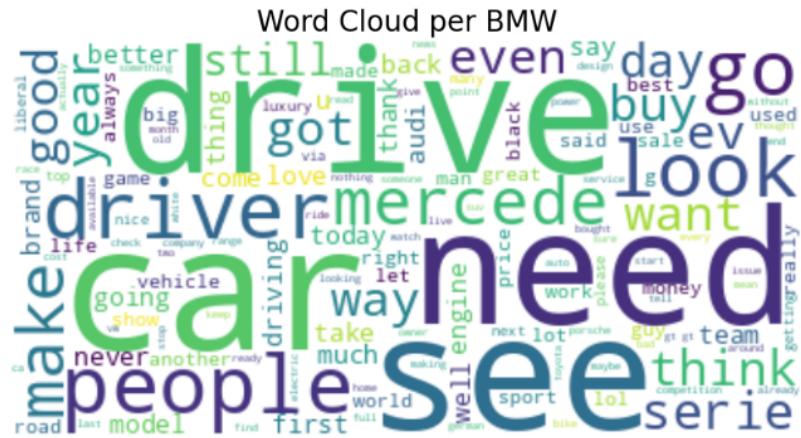


Figura 25. WordCloud per BMW.

Le parole più rilevanti nel grafico Wordcloud per BMW sono: drive, car, see, need.

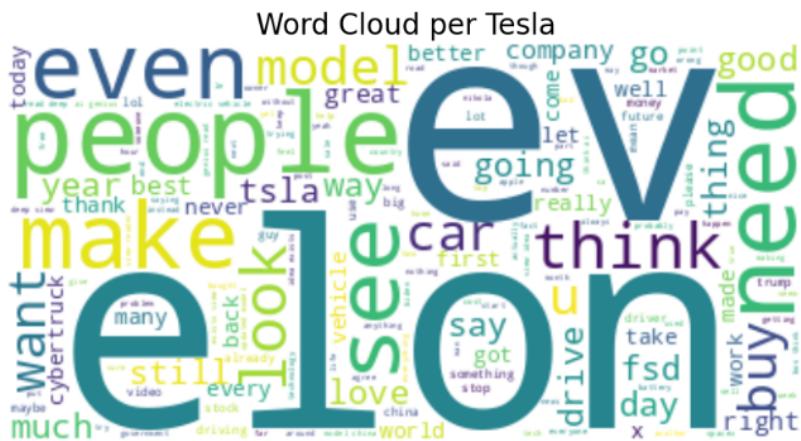


Figura 26. WordCloud per Tesla.

Le parole più rilevanti nel grafico Wordcloud per Tesla sono: elon, ev, need, people, make.

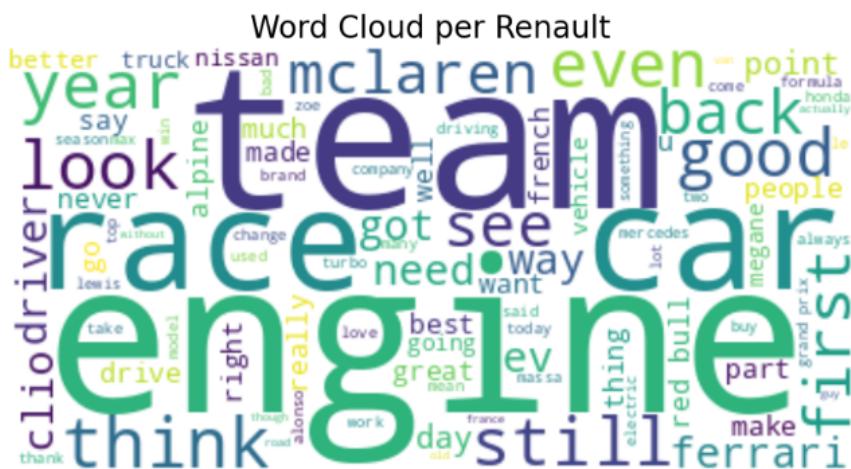


Figura 27. WordCloud per Renault.

Le parole più rilevanti nel grafico Wordcloud per Renault sono: engine, team, race, car.

Grafici con matplotlib:

Nel grafico sottostante è rappresentata la cronologia della GridSearchCV, mostrando i valori di verosimiglianza logaritmica (Log Likelihood), che rappresentano la misura della qualità del modello nella scoperta dei topic, in relazione al numero di argomenti e al parametro "learning decay".

La verosimiglianza logaritmica è una misura comune utilizzata per valutare quanto bene un modello probabilistico si adatta ai dati. Più il valore di verosimiglianza logaritmica è alto, migliore è la performance del modello nella generazione dei topic coerenti. Il parametro "learning decay" indica la velocità con cui il modello si adatta ai nuovi documenti durante l'iterazione dell'algoritmo di apprendimento. Valori più bassi di "learning decay" indicano un adattamento più rapido, mentre valori più alti indicano un adattamento più lento. Questo parametro influisce sulla flessibilità del modello nell'adattarsi a nuovi dati. Ogni colore nel grafico rappresenta un diverso valore di "learning decay" - blu per 0.5, arancione per 0.7 e verde per 0.9. Si può notare che il grafico traccia una costante tendenza al declino dei punteggi di verosimiglianza all'aumentare del numero di argomenti, con un abbassamento inesorabile a partire da 5 argomenti in poi.

La visualizzazione grafica è stata realizzata utilizzando la libreria Matplotlib al fine di presentare in modo chiaro l'andamento dei punteggi di verosimiglianza logaritmica in relazione al numero di argomenti e al parametro "learning decay". L'obiettivo principale di questa rappresentazione è valutare l'effetto congiunto di entrambi i fattori sulla qualità complessiva del modello LDA (Latent Dirichlet Allocation) nell'analisi dei topic.

BMW

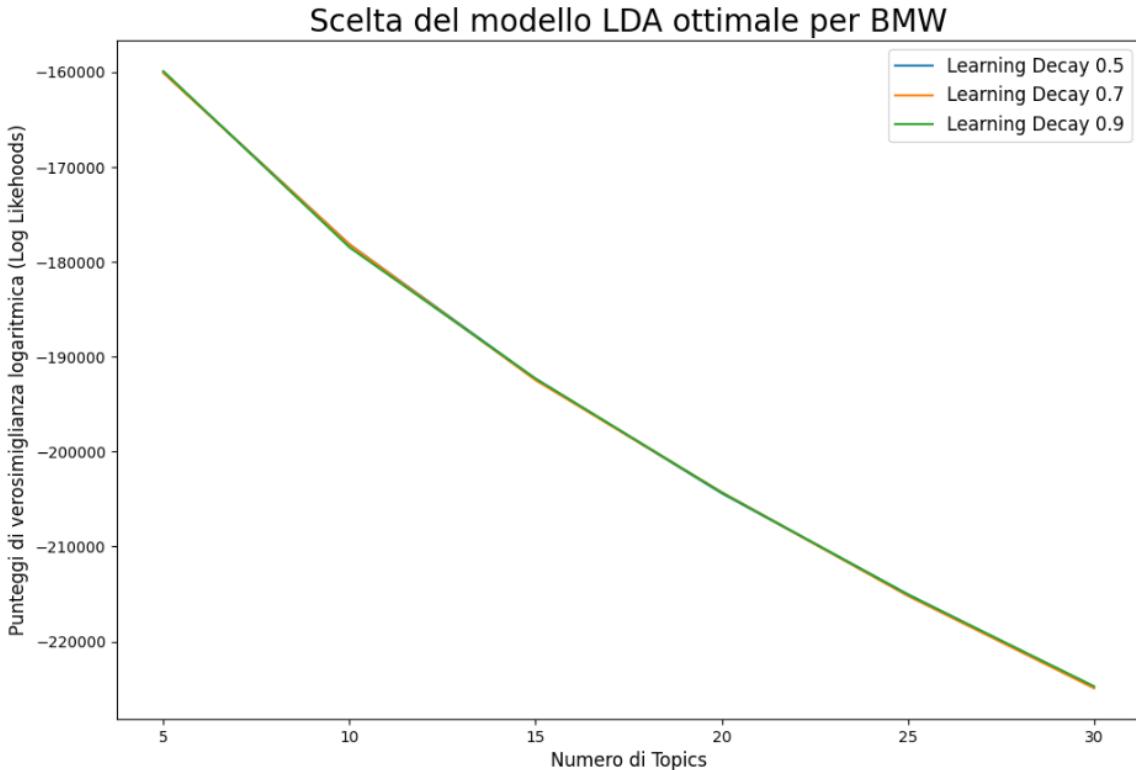


Figura 28. Grafico della verosimiglianza logaritmica di BMW.

Nel grafico sono rappresentati i parametri del miglior modello legato al soggetto di analisi Tesla. Il miglior modello trovato per l'analisi di topic in questo caso è quello con:

Topic Modeling per il file input di 'BMW' con numero di topics ottimale della GridSearchCV:

```
Fitting 5 folds for each of 18 candidates, totalling 90 fits
[CV 3/5] END learning_decay=0.5, n_components=10;, score=-193221.051 total time= 18.2s
[CV 2/5] END learning_decay=0.5, n_components=10;, score=-172474.018 total time= 18.7s
[CV 1/5] END learning_decay=0.5, n_components=10;, score=-169580.486 total time= 18.9s
[CV 2/5] END learning_decay=0.5, n_components=5;, score=-155215.009 total time= 21.6s
```

Miglior Punteggio di verosimiglianza logaritmica (Log Likehoods): -160005.30410910514

Parametri del modello migliore: {'learning_decay': 0.7, 'n_components': 5}

Perplexity del modello: 4340.666633156796

Numero di Topics Ottimale: 5

```
Topic 1: gt, car, series, competition, year, company, sale, today, drive, day
Topic 2: driver, got, guy, big, game, look, going, driving, fan, nice
Topic 3: want, people, good, need, drive, audi, car, make, driving, love
Topic 4: engine, mercedes, model, thank, benz, motorcycle, design, think, come, owner
Topic 5: electric, ev, suv, india, year, black, range, luxury, vehicle, drive
```

Figura 29. Stampa dell'output della Topic Modeling per BMW.

I topic più frequenti dall'analisi del file `tweets_puliti-BMW-en.csv` sono rappresentati nell'indice dei topic ed è emerso che i termini suggeriscono che le conversazioni e le discussioni rilevanti possano

spaziare da tematiche prettamente incentrate su tecnologia e prestazione delle auto BMW ad argomenti relativi alla soddisfazione dei clienti, al design ed a opinioni su specifici modelli di auto paragonate anche con altre case automobilistiche.

Tesla

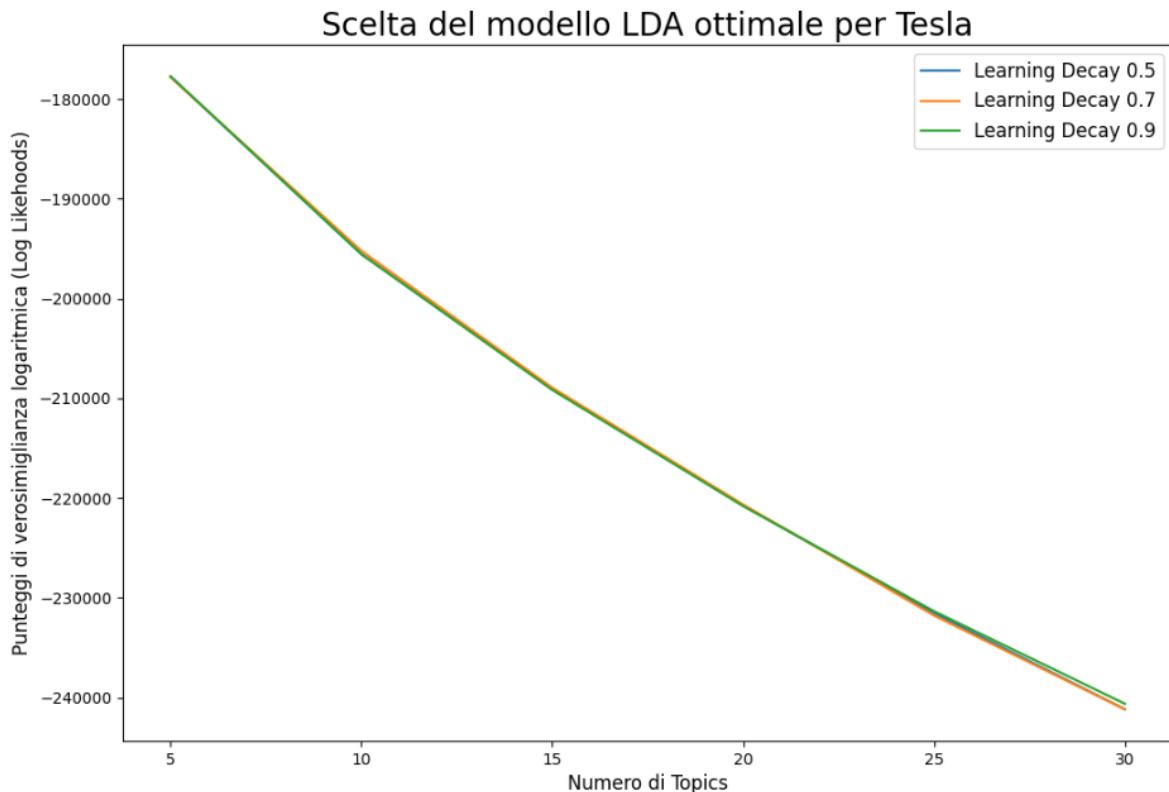


Figura 30. Grafico della verosimiglianza logaritmica di Tesla.

Nel grafico sono rappresentati i parametri del miglior modello legato al soggetto di analisi Tesla. Il miglior modello trovato per l'analisi di topic in questo caso è quello con:

Topic Modeling per il file input di 'Tesla' con numero di topics ottimale della GridSearchCV:

```
Fitting 5 folds for each of 18 candidates, totalling 90 fits
[CV 1/5] END learning_decay=0.5, n_components=10;, score=-203460.179 total time= 24.0s
[CV 3/5] END learning_decay=0.5, n_components=10;, score=-210350.947 total time= 25.1s
[CV 2/5] END learning_decay=0.5, n_components=10;, score=-203188.905 total time= 25.6s
[CV 2/5] END learning_decay=0.5, n_components=5;, score=-185573.141 total time= 27.9s
[CV 5/5] END learning_decay=0.5, n_components=5;, score=-166820.127 total time= 28.8s
```

Miglior Punteggio di verosimiglianza logaritmica (Log Likelihoods): -177669.40126407205

Parametri del modello migliore: {'learning_decay': 0.9, 'n_components': 5}

Perplexity del modello: 3357.2634947761535

Numero di Topics Ottimale: 5

```
Topic 1: view, ai, think, idea, fsd, read, deep, genius, bos, good
Topic 2: model, china, stock, ev, price, tsla, elon, cybertruck, cost, work
Topic 3: electric, ev, vehicle, buy, battery, elon, need, way, good, truck
Topic 4: love, company, got, make, delivery, car, year, tsla, ev, day
Topic 5: people, need, want, elon, model, thing, ev, day, going, think
```

Figura 31. Stampa dell'output della Topic Modeling per Tesla.

L'analisi di topic modeling condotta sul file tweets_puliti-Tesla-en.csv ha offerto una visione dettagliata delle tematiche emergenti all'interno dei dati. Sono stati individuati diversi argomenti chiave che riflettono le opinioni e le conversazioni degli utenti sulla casa automobilistica. Tra i temi rilevanti si evidenziano le discussioni sulle innovazioni tecnologiche, mettendo in luce il ruolo di Tesla nell'evoluzione dell'industria automobilistica. Si sono inoltre evidenziate tematiche legate all'elettrificazione del settore, all'autonomia delle batterie e alla sostenibilità ambientale. Altri argomenti trattano aspetti finanziari, come le azioni e gli investimenti, evidenziando il continuo interesse degli utenti per la dimensione economica dell'azienda.

Renault

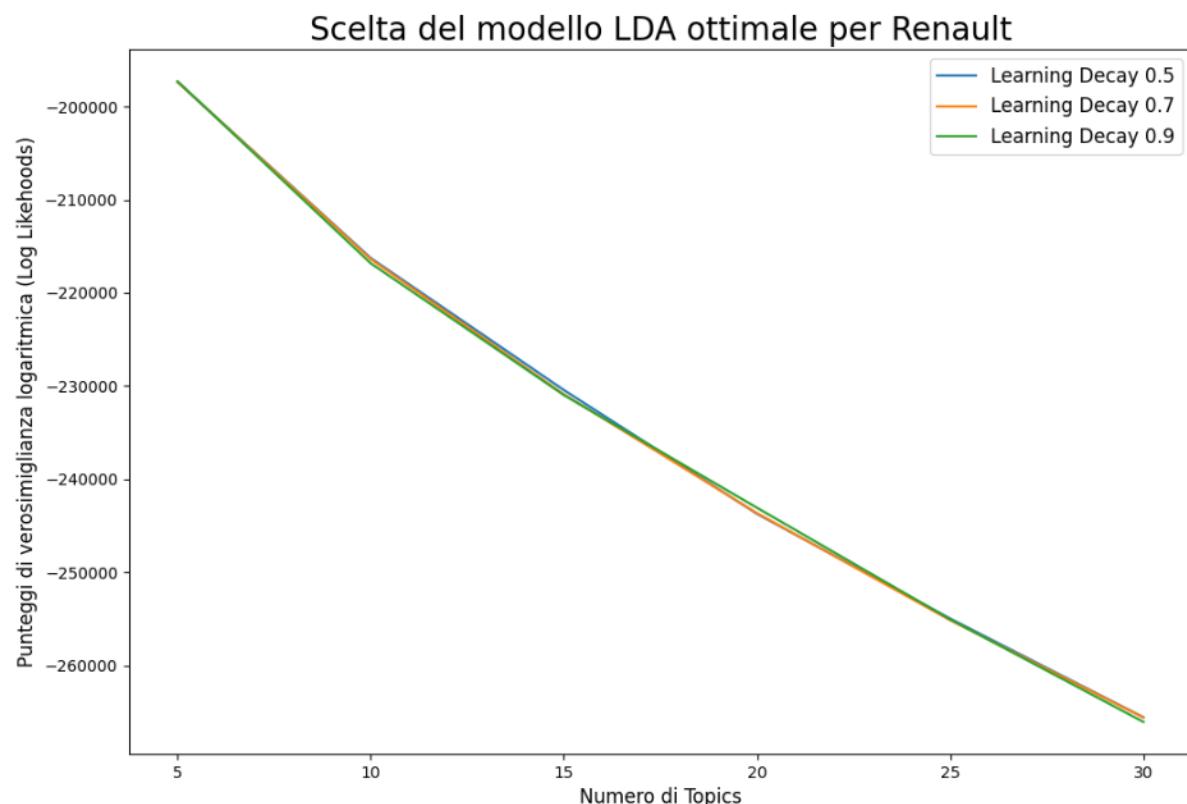


Figura 32. Grafico della verosimiglianza logaritmica di Renault.

Nel grafico sono rappresentati i parametri del miglior modello legato al soggetto di analisi Renault. Il miglior modello trovato per l'analisi di topic in questo caso è quello con:

```
Topic Modeling per il file input di 'Renault' con numero di topics ottimale della GridSearchCV:
```

```
Fitting 5 folds for each of 18 candidates, totalling 90 fits
[CV 3/5] END learning_decay=0.5, n_components=10;, score=-223005.556 total time= 25.1s
[CV 2/5] END learning_decay=0.5, n_components=10;, score=-211765.961 total time= 25.2s
[CV 1/5] END learning_decay=0.5, n_components=10;, score=-215233.484 total time= 25.7s
[CV 5/5] END learning_decay=0.5, n_components=5;, score=-203367.134 total time= 27.3s
```

```
Miglior Punteggio di verosimiglianza logaritmica (Log Likelihoods): -197360.97782693803
Parametri del modello migliore: {'learning_decay': 0.7, 'n_components': 5}
Perplexity del modello: 3697.502678420452
Numero di Topics Ottimale: 5
```

```
Topic 1: engine, team, mclaren, ferrari, red, bull, good, mercedes, got, better
Topic 2: win, race, year, think, want, point, truck, lewis, man, got
Topic 3: ev, scenic, electric, car, year, look, range, nissan, battery, vehicle
Topic 4: alonso, race, massa, grand, prix, make, people, driver, championship, hamilton
Topic 5: alpine, team, gt, car, clio, drive, old, wheel, electric, van
```

Figura 33. Stampa dell'output della Topic Modeling per Renault.

I temi rilevati dall'analisi di topic condotta sul file tweets_puliti-Renault-en.csv, in contrasto con le altre due case automobilistiche, evidenziano discussioni degli utenti di Twitter orientate verso competizioni automobilistiche e sport motoristici, come ad esempio la Formula 1. Tali argomenti testimoniano l'interesse di Renault in tali settori, evidenziando le prestazioni dei piloti e le innovazioni tecnologiche che l'azienda ha introdotto in questi contesti.

Risultati grafici della pagina html con LDA:

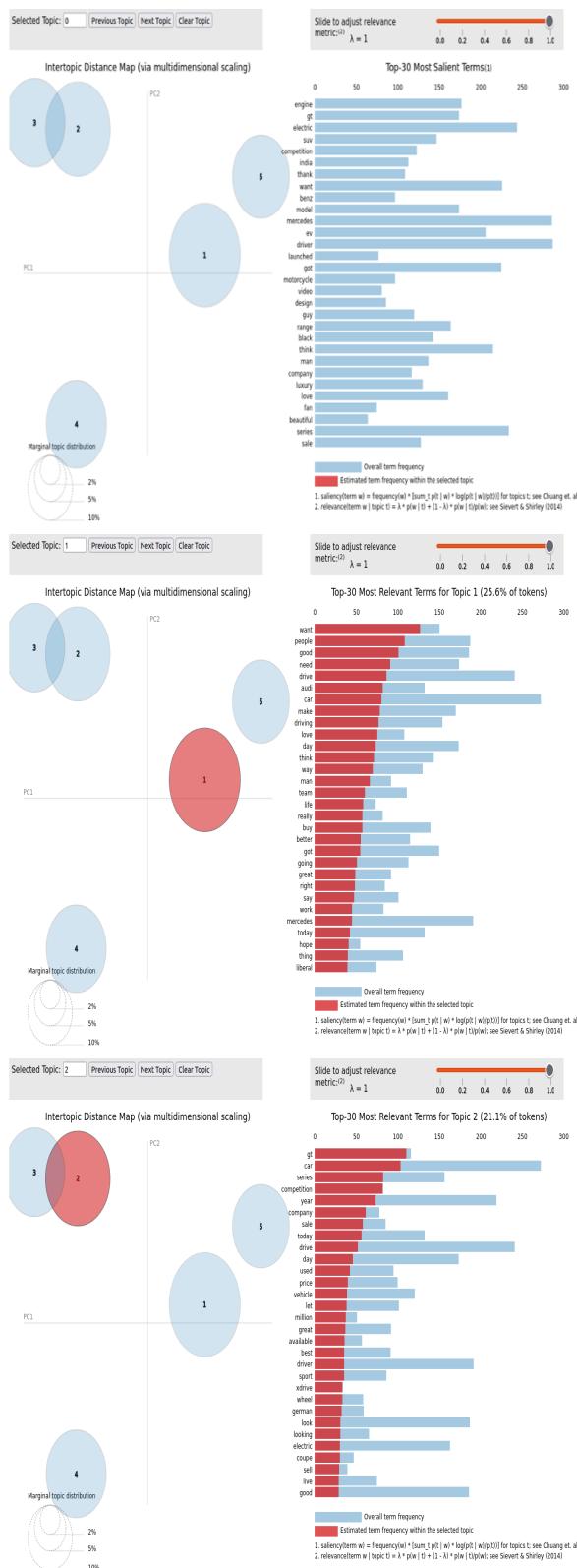


Figura 34. Risultati della pagina html di BMW (parte 1).

BMW:

In questo grafico sono rappresentati i “Top 30 most salient terms” del file “LDA_panel_BMW.html”.

I risultati evidenziano la frequenza assoluta dei termini più utilizzati nelle discussioni su Twitter degli utenti.

I termini più frequenti in questa analisi sono: engine, gt, eletric, suv e competition.

Nell’Intertopic Distance Map sono presenti dei cerchi che indicano i cluster dei 5 diversi topic trovati. La posizione e la distanza dei cerchi ci fa comprendere come alcuni topic possano essere correlati tra loro e perché hanno in comune alcuni termini.

Questo aspetto può essere interpretato sulla base della coerenza tematica o semantica.

Dall’analisi visiva di questa mappa, è evidente che i topic 2 e 3, data la loro stretta vicinanza, presentino termini simili e fortemente correlati tra di loro.

Al contrario, i topic 1, 4 e 5, essendo più distanti, mostrano una correlazione minore ed una maggiore indipendenza.

La vicinanza dei primi due cluster è spiegata dalla predominanza, evidenziata nei grafici dei seguenti termini più frequenti:

Per il topic 2:

- Gt
- Car
- Series
- Competition
- Year

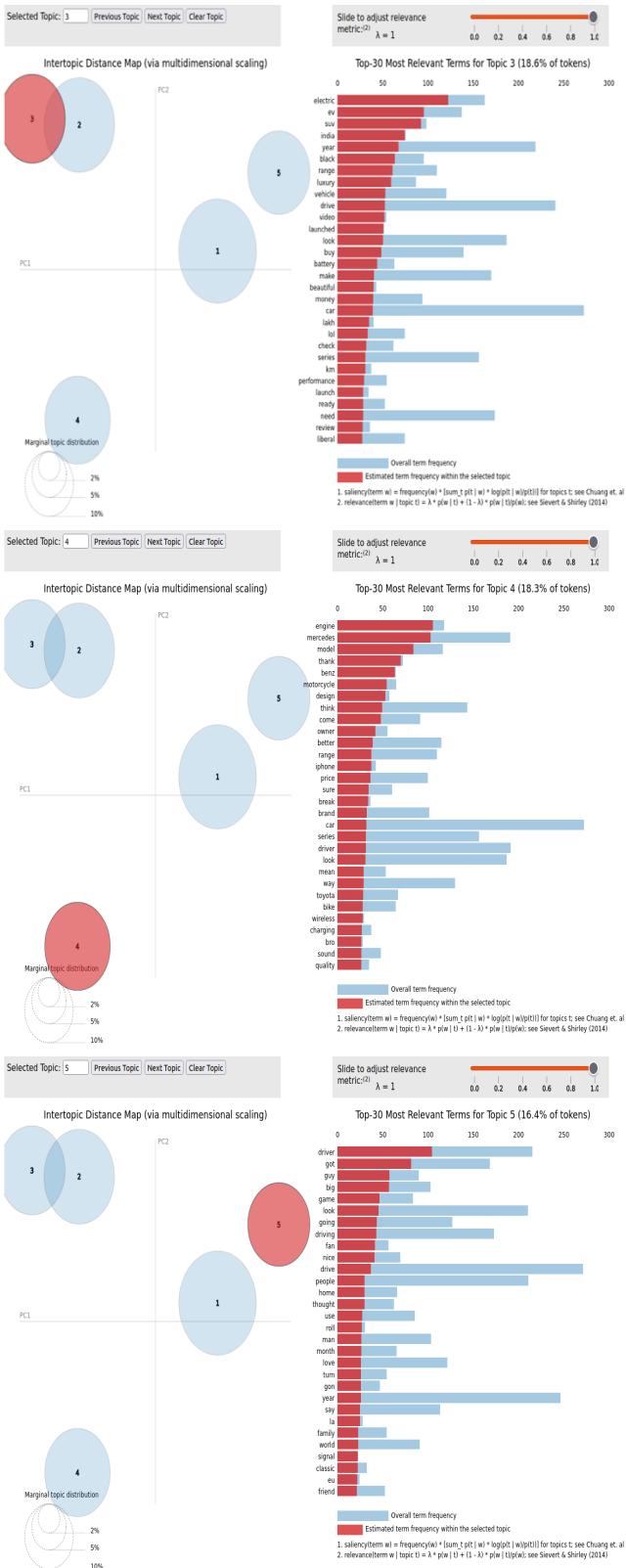


Figura 35. Risultati della pagina html di BMW (parte 2).

tra di loro e una distanza maggiore sulla mappa delle distanze intertopiche. Questo suggerisce che trattino argomenti diversi o non condividono termini significativi in modo evidente. Ad esempio, il Topic 1 sembra affrontare aspetti più generali o desideri delle persone, il Topic 4 sembra essere associato all'ingegneria automobilistica con termini come "engine" e "Mercedes," mentre il Topic 5 sembra trattare di conduzione e giochi.

Per il topic 3:

- Eletric
- Ev
- Suv
- India
- Year

Mentre per i topic 1, 4 e 5 troviamo:

Per il topic 1:

- Want
- People
- Good
- Need
- Drive

Per il topic 4:

- Engine
- Mercedes
- Model
- Thank
- Benz

Per il topic 5:

- Driver
- Got
- Guy
- Big
- Game

Commento:

Prima di tutto, i risultati evidenziano i cinque diversi topic individuati e la loro relativa distanza nella mappa delle distanze intertopiche. La vicinanza dei cluster dei topic 2 e 3 indica una forte correlazione tra questi due topic, il che è supportato dai termini condivisi come "gt," "year," e "competition" che possono essere rilevanti sia per le auto sportive (Topic 2) che per i veicoli elettrici SUV in India (Topic 3). D'altro canto, i topic 1, 4 e 5 mostrano una maggiore indipendenza

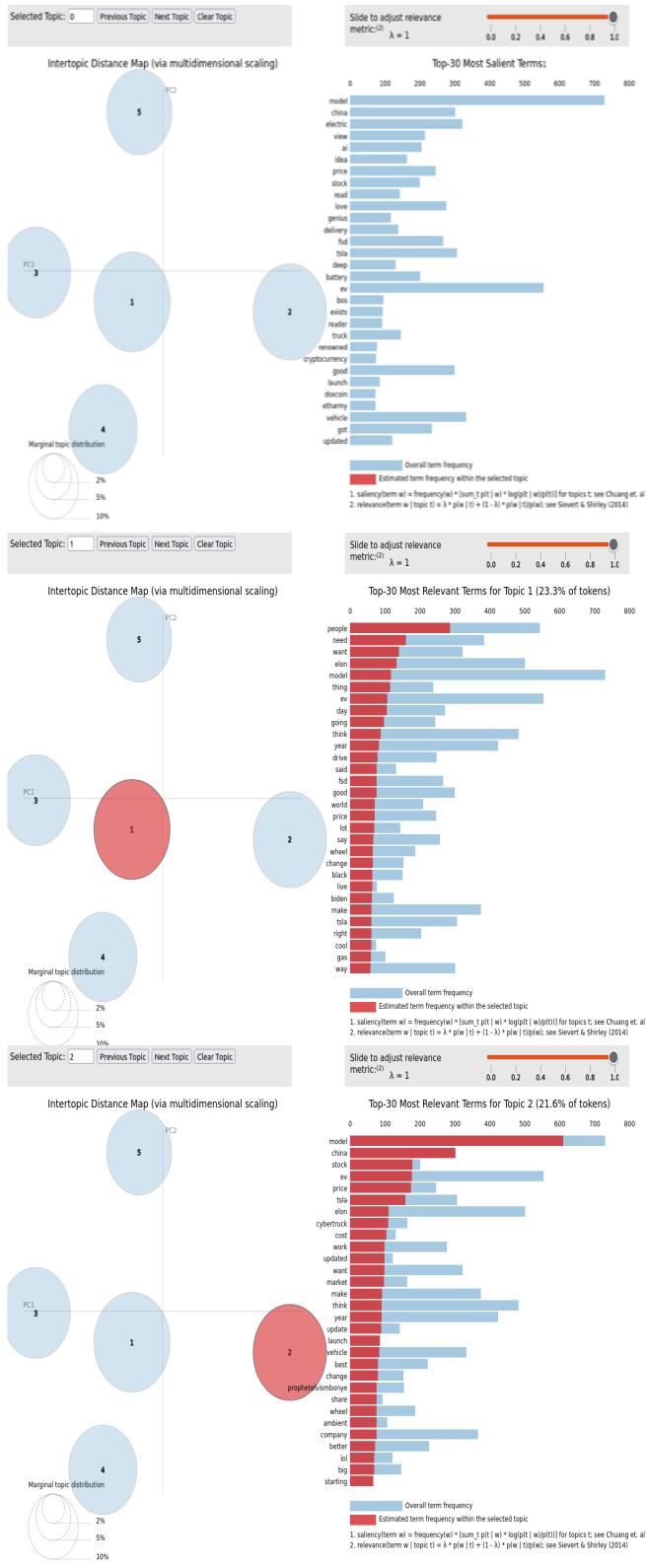


Figura 36. Risultati della pagina html di Tesla (parte 1).

Tesla:

In questo grafico sono rappresentati i “Top 30 most salient terms” del file *LDA_panel_Tesla.html*.

I risultati evidenziano la frequenza assoluta dei termini più utilizzati nelle discussioni su Twitter degli utenti.

I termini risultati più frequenti da questa analisi di topic modeling sono: model, ev, electric, china, vehicle, tsla, good.

Analizzando graficamente questa mappa si può notare che i topic sono tutti molti indipendenti l’uno dall’altro.

Prendendo i grafici dei singoli topic possiamo dire che i termini più frequenti sono:

Per il topic 1:

- People
- Need
- Want
- Elon
- Model
- Thing
- Ev

Per il topic 2:

- Model
- China
- Stock
- Ev
- Price
- Tsla
- Elon
- Cybertruck

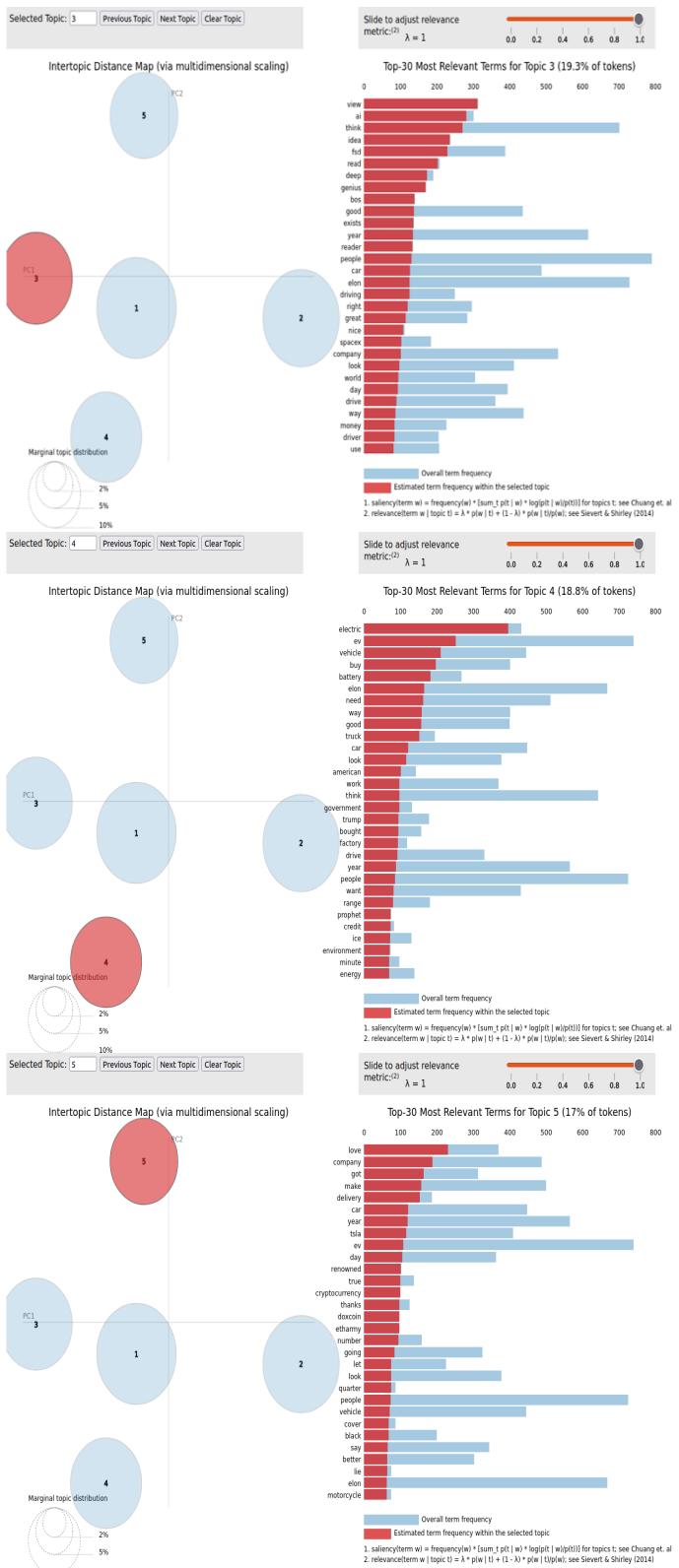


Figura 37. Risultati della pagina html di Tesla (parte 2).

il "Cybertruck". Nel topic 3, si rileva un interesse per l'intelligenza artificiale e le idee innovative legate al settore automobilistico. Il topic 4 evidenzia un'attenzione particolare verso aspetti come l'acquisto di veicoli elettrici e la tecnologia delle batterie. Infine, nel topic 5, emergono discussioni incentrate sull'apprezzamento per il marchio e sul processo di consegna.

Per il topic 3:

- View
- Ai
- Think
- Idea
- Fsd
- Read

Per il topic 4:

- Electric
- Ev
- Vehicle
- Buy
- Battery
- Elon

Per il topic 5:

- Love
- Company
- Got
- Make
- Delivery

Commento:

Analizzando i "Top 30 most salient terms" nel file "LDA_panel_.html" dedicato a Tesla, emergono termini chiave come model, ev, electric, china, vehicle, tsla e good, evidenziando una discussione ampia e diversificata che spazia dalla tecnologia alla percezione del marchio. La mappa dei topic mostra un livello significativo di indipendenza tra i vari cluster, suggerendo una pluralità di temi di discussione relativi a Tesla. Approfondendo l'analisi dei singoli topic, emerge un insieme eterogeneo di temi e concetti. Il topic 1, ad esempio, riflette una discussione focalizzata sulle preferenze e le esigenze dei consumatori, con menzioni specifiche riguardanti Elon Musk. Nel topic 2, il focus si sposta su argomenti come il mercato azionario, i prezzi delle azioni e i modelli specifici come

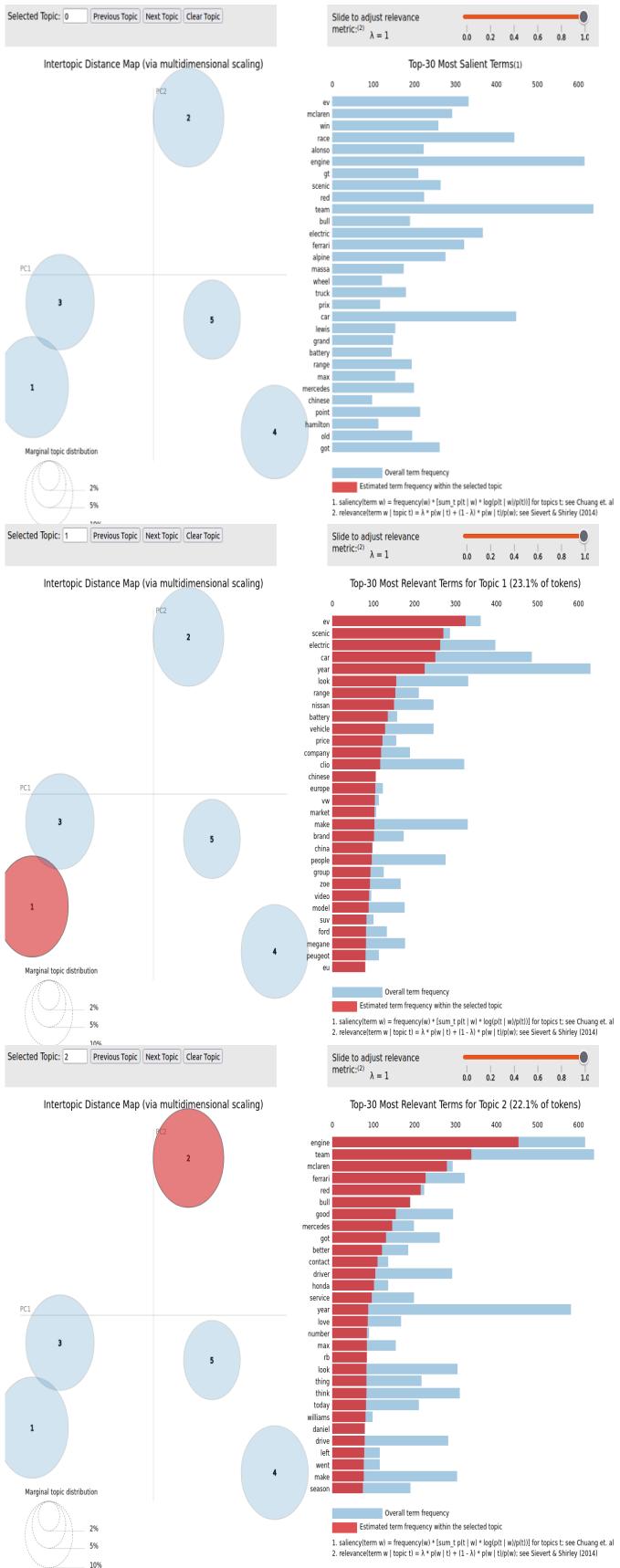


Figura 38. Risultati della pagina html di Renault (parte 1).

Renault

In questo grafico sono rappresentati i "Top 30 most salient terms" del file *LDA_panel_.html*.

I risultati evidenziano la frequenza assoluta dei termini più utilizzati nelle discussioni su Twitter degli utenti. I termini risultati più frequenti da questa analisi di topic modeling sono: team, engine, race, car, ev, electric.

Analizzando graficamente questa mappa si può notare che i cluster più vicini tra loro sono quelli dei topic 1 e 3 che possiedono alcuni termini in comune tra i topic, mentre i più distanti ed indipendenti sono i topic 2, 4 e 5. Analizzando i grafici dei singoli topic possiamo dire che i termini più frequenti nei topic tra essi più correlati sono:

Per il topic 1:

- Company
- Ev
- Scenic
- Electric
- Car
- Year

Per il topic 3:

- Alpine
- Team
- Gt
- Car
- Clio

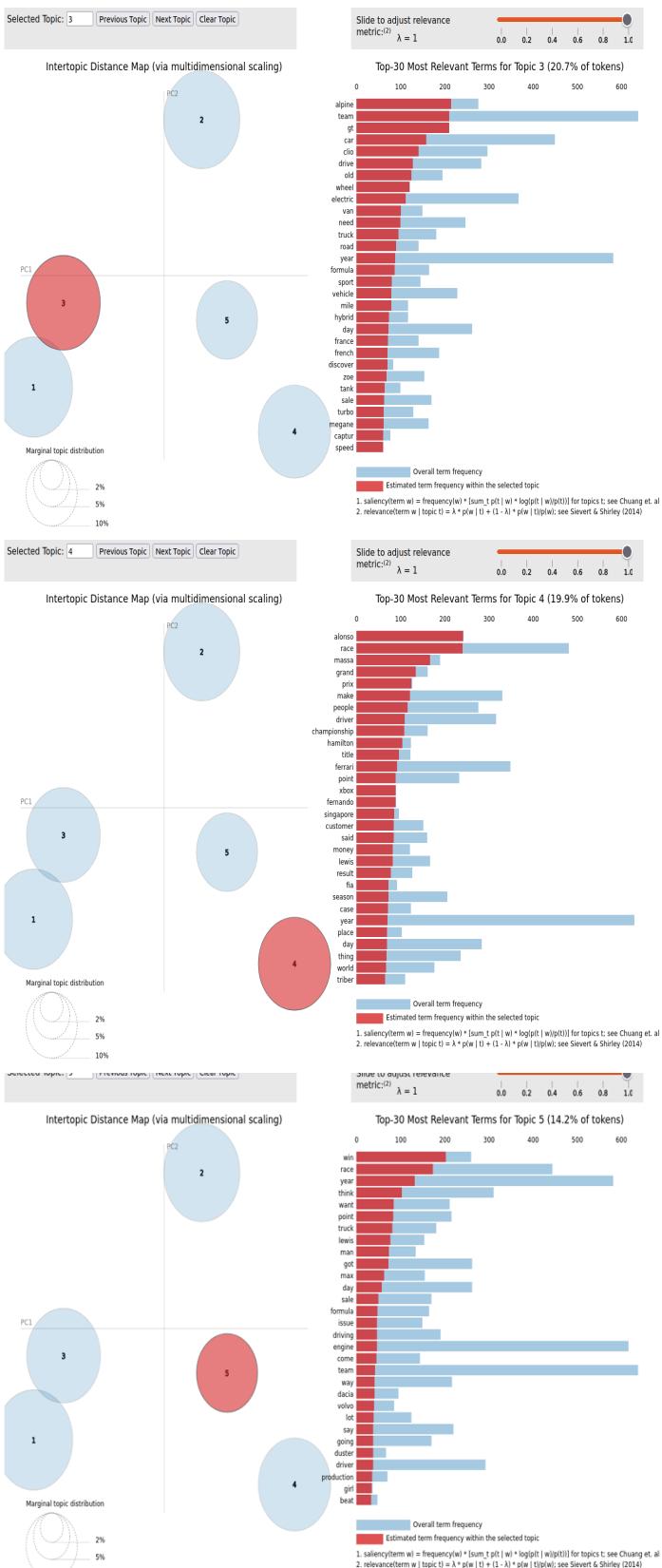


Figura 39. Risultati della pagina html di Renault (parte 2).

competizioni automobilistiche, come nomi di team e piloti, insieme a concetti legati alla vittoria e alla competizione.

Invece, i termini più frequenti all'interno dei topic tra loro più distanti sono:

Per il topic 2:

- Engine
- Team
- McLaren
- Ferrari
- Redbull

Per il topic 4:

- Alonso
- Race
- Massa
- Grand
- Prix

Per il topic 5:

- Win
- Race
- Year
- Think
- Want

Commento:

Analizzando i "Top 30 most salient terms" nel file "LDA_panel_.html" relativo a Renault, emergono termini chiave come team, engine, race, car, ev ed electric, indicando un focus predominante sulle discussioni legate alle competizioni automobilistiche e alle prestazioni dei veicoli elettrici. La mappa mostra una chiara vicinanza tra i cluster dei topic 1 e 3, suggerendo una correlazione tematica tra i due. D'altra parte, i topic 2, 4 e 5 si distinguono per la loro indipendenza e distanza reciproca. Nel dettaglio, il topic 1 riflette discussioni su aziende automobilistiche, veicoli elettrici e aspetti legati alla sostenibilità, mentre il topic 3 pone l'attenzione su squadre automobilistiche specifiche e modelli di auto come "Alpine" e "Clio". In contrasto, i topic 2, 4 e 5 sono caratterizzati da un linguaggio incentrato su aspetti specifici delle

Fase 4: BRANCH RoBERTa SentimentAnalysis

In questo branch abbiamo eseguito la sentiment analysis sui dati pre-elaborati per determinare il sentimento generale dei tweet nei confronti delle case automobilistiche target. Abbiamo utilizzato un modello di apprendimento automatico per classificare i tweet in tre categorie principali: Positivo, Neutro e Negativo.

Di seguito presentiamo i file utilizzati, relativi allo script, nonché i file di input e output del branch di lavoro:

- fine-tuning-model.py
- validation-model.py
- classification-model.py
- tweets_puliti
- tweets_sentiment

fine-tuning-model.py

Il nostro obiettivo principale in questo programma è eseguire una sentiment analysis sui tre file CSV che contengono i tweet estratti e ripuliti. Per quest'analisi, stiamo impiegando una rete neurale come classificatore di deep learning, utilizzando la versione più recente di RoBERTa. Questo modello RoBERTa, è un modello di NLP (Natural Language Processing) avanzato che viene utilizzato per analizzare il testo e rappresenta una potente variante di BERT, ottimizzata e pre-addestrata su un vasto corpus di tweets di Twitter. La nostra scelta di adottare questo metodo è motivata dal fatto che il modello RoBERTa è ampiamente utilizzato per le sue eccezionali capacità in diverse attività linguistiche, tra cui la sentiment analysis.

Il modello RoBERTa specifico che abbiamo utilizzato è stato integrato seguendo la documentazione a disposizione in questo link: <https://huggingface.co/cardiffnlp/twitter-roberta-base-sentiment-latest>

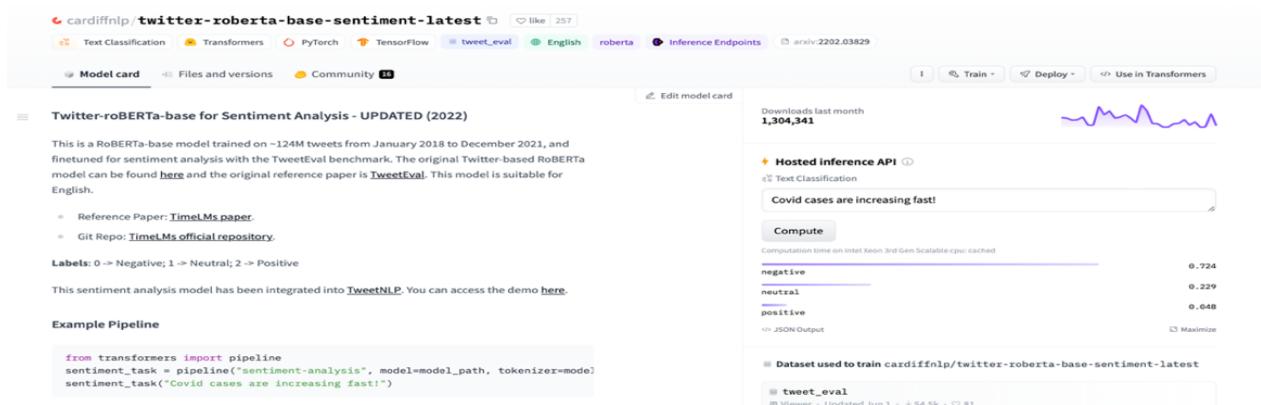


Figura 40. Modello RoBERTa pre-addestrato.

RoBERTa è un modello di linguaggio basato sull'architettura Transformer, che è stato addestrato su enormi quantità di testi, provenienti da varie fonti online. Questo addestramento massiccio effettuato su 124 milioni di tweets ha consentito a RoBERTa di acquisire una comprensione approfondita della struttura e del significato del linguaggio, nonché di catturare relazioni semantiche complesse tra le parole e le frasi.

Data la sua complessità e la quantità estesa di dati utilizzati durante il pre-addestramento, è generalmente considerato non necessario o addirittura controproducente eseguire il fine-tuning su RoBERTa per molte applicazioni.

Tuttavia, abbiamo ritenuto utile effettuare il fine-tuning sul modello perché permette di ottimizzare le prestazioni del modello e di adattare le capacità di comprensione del linguaggio in maniera specifica per riconoscere e classificare correttamente il sentimento nei testi dei tweet.

Il fine-tuning di un modello di RoBERTa è un processo vantaggioso per diversi motivi:

- per l'adattamento a specifici compiti, permettendo al modello di apprendere le sfumature e le caratteristiche del task specifico, migliorando le prestazioni in quel compito;
- per il miglioramento delle prestazioni, consentendo al modello di apprendere da un insieme più ristretto e specifico di dati relativi al task che si vuole svolgere. Questo porta a prestazioni migliori in termini di accuratezza e precisione;
- per il risparmio di risorse, cioè che rispetto all'addestramento da zero di un modello, il fine-tuning richiede meno risorse computazionali e tempo, in quanto, si parte da un modello pre-addestrato con conoscenze linguistiche generali. Utilizzando minori costi computazionali e una più rapida scalabilità per adattare il modello a nuovi compiti;
- il fine-tuning offre inoltre la flessibilità di personalizzare il modello secondo le esigenze specifiche dell'utente. Si possono regolare iperparametri, dimensioni del modello e strategie di ottimizzazione per ottenere prestazioni ottimali nel contesto specifico.
- Ed infine: Il fine-tuning è particolarmente utile per il miglioramento delle prestazioni su dati limitati. Utilizzando un modello pre-addestrato e aggiornandolo per un compito specifico, si possono ottenere prestazioni migliori anche con quantità ridotte di dati di addestramento.

Il fine tuning è stato implementato al modello utilizzando il dataset Twitter_data.csv, scaricato da Kaggle al link: <https://www.kaggle.com/datasets/saurabhshahane/twitter-sentiment-dataset>

Questo dataset nella totalità presenta 162980 tweets pre-etichettati, ma noi abbiamo optato di utilizzare solo i primi 30000 perché ci siamo resi conto che computazionalmente è molto oneroso, di fatto ci ha messo circa 8 ore.



Figura 41. Fine-tuning.

All'interno dello script abbiamo eseguito l'addestramento del modello.

Per l'addestramento del modello di RoBERTa attraverso il fine-tuning abbiamo effettuato i passaggi descritti in seguito.

Definiamo la classe **TwitterDataset**, ereditata dalla classe **torch.utils.data.Dataset**. Questa classe funge da struttura per la gestione dei dati. Contiene un dataframe, un tokenizer e la lunghezza massima (**max_length**) come parametri al suo interno. All'interno di questa classe, inoltre estraiamo i testi dal dataframe, così come le etichette corrispondenti **clean_text** e **category** e li prepariamo in un formato adatto per l'addestramento del modello.

La funzione **len** all'interno della classe TwitterDataset restituisce la lunghezza dei dati.

La funzione **getitem** definisce come ottenere un singolo elemento del dataset. Estraie il testo e le etichette relativi a un indice specifico, applica il processo di tokenizzazione utilizzando il tokenizer fornito e restituisce un dizionario contenente gli **input_ids** (identificatori degli input), le **attention_mask** (maschere di attenzione, utilizzata per indicare quali parti del testo sono rilevanti per il modello durante l'elaborazione) e le etichette corrispondenti.

```
class TwitterDataset(torch.utils.data.Dataset):
    """
    Classe che rappresenta un dataset personalizzato per l'analisi del sentimento su Twitter
    """

    def __init__(self, dataframe, tokenizer, max_length):
        # Inizializzazione delle variabili necessarie
        self.tokenizer = tokenizer
        self.data = dataframe
        self.text = dataframe["clean_text"].tolist() # Estrazione dei testi puliti
        self.labels = dataframe["category"].tolist() # Estrazione delle categorie
        self.max_length = max_length

    def __len__(self):
        # Ritorna la lunghezza del dataset
        return len(self.text)

    def __getitem__(self, index):
        # Estrazione dei dati per un determinato indice
        text = str(self.text[index])

        # Tokenizzazione e formattazione dei dati di input
        inputs = self.tokenizer(text, padding="max_length", truncation=True, max_length=self.max_length, return_tensors="pt")
        label = self.labels[index]

        # Restituzione dei dati nel formato atteso per il modello
        return {
            "input_ids": inputs["input_ids"].squeeze(),
            "attention_mask": inputs["attention_mask"].squeeze(),
            "labels": torch.tensor(label, dtype=torch.long)
        }
```

Figura 42. Class TwitterDataset.

Abbiamo creato la classe “**TwitterSentimentAnalysis**” che si occupa dell'addestramento di un modello per l'analisi del sentimento sui dati provenienti da Twitter, utilizzando RoBERTa. Vediamo nel dettaglio come funziona questa classe e quali funzioni sono contenute al suo interno.

__init__: questa funzione è il costruttore della classe. Al momento della creazione di un'istanza di TwitterSentimentAnalysis, vengono eseguite una serie di operazioni per inizializzare gli attributi e configurare l'ambiente di addestramento del modello.

La funzione `load_training_data` si occupa del caricamento dei dati di addestramento. Legge il file CSV `train_file` utilizzando la libreria pandas e ne estrae le prime 30.000 righe. Successivamente eseguiamo la conversione delle etichette preesistenti, rappresentate dalla colonna "category", che abbiamo trasformato per adattarle al modello di analisi del sentimento. Questo processo di adattamento prevede una sostituzione dei valori originali (1, 0, -1) con nuovi valori (2, 1, 1) nella colonna denominata `category`. Ciò mira a uniformare la rappresentazione dei sentimenti secondo gli standard richiesti dal modello RoBERTa.

In `prepare_model_and_tokenizer`, prepariamo il modello pre-addestrato e il tokenizer da utilizzare per l'analisi del sentimento. Specificiamo il modello pre-addestrato "`cardiffnlp/twitter-roberta-base-sentiment-latest`" e inizializziamo un tokenizer specifico per il modello RoBERTa. Inoltre, creiamo un'istanza del modello di classificazione di sequenze RoBERTa configurato con il numero di etichette passato come argomento.

La funzione `setup_trainer` configura il Trainer che gestirà l'addestramento effettivo del modello. Configuriamo quindi gli argomenti di addestramento attraverso l'oggetto `TrainingArguments`. Questi includono la directory di output per i risultati, il numero di epoche di addestramento (5), le dimensioni dei batch per dispositivo sia durante l'addestramento che la valutazione, il numero di passaggi di "warmup" (500) e il decadimento del peso (weight decay) pari a 0.01. Inoltre, è specificata una directory per il logging. Viene poi inizializzato un oggetto `Trainer` che addestrerà il modello. Passiamo il modello da addestrare, gli argomenti di addestramento, nonché i dataset di addestramento e di valutazione preparati in precedenza. Infine, eseguiamo il fine-tuning vero e proprio che viene eseguito chiamando il metodo `train()` sull'oggetto `trainer`, il che avvia il processo di addestramento del modello utilizzando i dati forniti e seguendo le configurazioni specificate. In seguito, la funzione `push_to_hub()` carica il modello addestrato su Hugging Face Hub, effettuando il push.

Il modello creato con l'addestramento del fine-tuning è contenuto all'interno del link:

<https://huggingface.co/bibbia/DriveFeelings-Roberta-sentiment-analyzer-for-twitter>

Category	Probability
positive	1.000
neutral	0.000
negative	0.000

Figura 43. Modello fine-tuned su Hugging Face.

```

class TwitterSentimentAnalysis:

    """ Classe che rappresenta l'analisi del sentimento su Twitter """

    def __init__(self, train_file, max_length=100, num_labels=3):

        # Caricamento dei dati di addestramento
        self.load_training_data(train_file)

        # Preparazione del modello e del tokenizzatore
        self.prepare_model_and_tokenizer(num_labels)

        # Creazione del dataset di addestramento e del dataset di test
        self.dataset_train = TwitterDataset(self.train_data, self.tokenizer, max_length)
        self.dataset_test = TwitterDataset(self.test_data, self.tokenizer, max_length)

        # Configurazione del trainer
        self.setup_trainer()

    def load_training_data(self, train_file):

        """ Funzione che carica i dati da un file CSV e li divide in set di addestramento e test set """

        # Caricamento dei dati da un file CSV e suddivisione in set di addestramento e test
        df = pd.read_csv(train_file)[:30000]

        # Mappatura delle categorie in un formato specifico
        df['category'] = df['category'].map({1: 2, 0: 1, -1: 0})

        # Divisione in set di addestramento e test
        self.train_data, self.test_data = train_test_split(df, test_size=0.33, random_state=13)

    def prepare_model_and_tokenizer(self, num_labels):

        """ Caricamento del modello e del tokenizzatore preaddestrati """

        # Caricamento del modello e del tokenizzatore preaddestrati
        self.MODEL = "cardiffnlp/twitter-roberta-base-sentiment-latest"
        self.tokenizer = RobertaTokenizer.from_pretrained(self.MODEL)
        self.model = RobertaForSequenceClassification.from_pretrained(self.MODEL, num_labels=num_labels)

    def setup_trainer(self):

        """ Funzione per configurare ed eseguire il training """

        # Configurazione degli argomenti di addestramento
        self.training_args = TrainingArguments(
            output_dir='./DriveFeelings-Roberta-sentiment-analyzer-for-twitter',
            num_train_epochs=5,
            per_device_train_batch_size=6,
            per_device_eval_batch_size=2,
            warmup_steps=500,
            weight_decay=0.01,
            logging_dir='./logs',
            logging_steps=10,
            evaluation_strategy='epoch'
        )

        # INIZIALIZZAZIONE DEL TRAINER PER L'ADDESTRAMENTO DEL MODELLO
        self.trainer = Trainer(
            model=self.model,
            args=self.training_args,
            train_dataset=self.dataset_train,
            eval_dataset=self.dataset_test,
            tokenizers=self.tokenizer,
        )

        # Avvio dell'addestramento del modello
        self.trainer.train()

        # Pubblicazione del modello su directory cloud HuggingFace.co
        self.trainer.push_to_hub()

```

Figura 44. Class Twitter_Sentiment_analysis.

```
if __name__ == "__main__":
```

Quando lo script viene eseguito direttamente, questo blocco di codice avvierà il processo di analisi del sentiment su dati provenienti da Twitter, utilizzando il file CSV specificato. Creerà un'istanza della classe `TwitterSentimentAnalysis` e inizierà l'intero flusso di lavoro per l'addestramento del modello, il tutto partendo dal caricamento dei dati fino all'addestramento effettivo e al caricamento del modello addestrato su Hugging Face Hub.

La funzione viene utilizzata per avviare il processo di analisi del sentiment su dati `Twitter_Data.csv`. Con `input_file = 'Twitter_Data.csv'` assegna il nome del file CSV contenente i dati provenienti da Twitter.

Con `sentiment_analysis = TwitterSentimentAnalysis(input_file)` inizializziamo un'istanza della classe `TwitterSentimentAnalysis`, passando come argomento il nome del file CSV contenente i dati di input per l'analisi del sentiment su Twitter. Questo avvia il processo di inizializzazione della classe.

```
if __name__ == '__main__':  
  
    # File di input per il fine-tuning  
    input_file = 'Twitter_Data.csv'  
  
    # Chiamata di funzione per l'analisi del sentiment su Twitter  
    sentiment_analysis = TwitterSentimentAnalysis(input_file)
```

Figura 45. Main: fine-tuning-model.py

validation-model.py

Con la funzione `validate_model_fine_tuned` abbiamo effettuato la validazione delle performance di un modello specifico di classificazione delle sequenze, focalizzato sull'analisi del sentiment.

Iniziamo caricando un modello pre-addestrato noto come "**bibbia/DriveFeelings-Roberta-sentiment-analyzer-for-twitter**", appartenente alla categoria **RobertaForSequenceClassification**.

Questo modello è stato precedentemente addestrato su un corpus di testo per identificare e classificare automaticamente il sentiment associato a diverse sequenze di testo.

Successivamente, abbiamo utilizzato la libreria Hugging Face's **pipeline** per creare un flusso di lavoro di analisi del sentiment. Questo flusso incorpora il modello appena caricato e il tokenizer associato, indispensabile per la preparazione dei testi e la loro comprensione da parte del modello. Abbiamo importato i dati di test dal file CSV `Test_airlines-sentiment.csv`, specificato come argomento.

Abbiamo usato il file che contiene la colonna chiamata `text`, che rappresenta i testi su cui volevamo eseguire le previsioni, e la colonna `airline_sentiment`, che contiene i sentimenti di riferimento associati a ciascun testo.

Dopodiché, abbiamo utilizzato il modello per effettuare previsioni sui testi presenti nel dataset di test. Le previsioni sono state estratte per valutare il sentimento predetto per ciascun testo.

Successivamente, abbiamo calcolato l'accuratezza delle previsioni confrontandole con i sentimenti reali presenti nel dataset di test. Questo calcolo è stato eseguito attraverso l'utilizzo della funzione

accuracy_score proveniente dal modulo **sklearn.metrics**. Inoltre, abbiamo creato e calcolato una matrice di confusione utilizzando la funzione **confusion_matrix** dallo stesso modulo. Questa matrice fornisce un quadro dettagliato delle previsioni corrette e scorrette fatte dal modello per ciascuna classe di sentimento presente nel dataset di test.

Infine, abbiamo stampato a schermo i risultati ottenuti. Abbiamo visualizzato l'accuratezza del modello, espressa in percentuale, e la matrice di confusione.

La matrice è stata ulteriormente rappresentata con la mappa di calore per avere una visualizzazione grafica più significativa.

Questi risultati ci hanno fornito un'analisi dettagliata delle performance del modello nell'analisi del sentimento basandoci sui dati di test forniti.

```
def validate_model_fine_tuned(test_file):
    """
    Funzione per validare il modello fine tuned DriveFeelings
    """

    # Caricamento del modello per modello fine-tuned
    tokenizer = RobertaTokenizer.from_pretrained("bibbia/DriveFeelings-Roberta-sentiment-analyzer-for-twitter")
    model = RobertaForSequenceClassification.from_pretrained("bibbia/DriveFeelings-Roberta-sentiment-analyzer-for-twitter")
    pipe = pipeline("sentiment-analysis", model=model, tokenizer=tokenizer)

    # Caricamento dei dati per modello fine-tuned
    test_file = pd.read_csv(test_file)[:10000]

    # Calcolo predizioni per modello fine-tuned
    predictions = [pipe(text) for text in test_file['text']]
    extracted_predictions = [result[0]['label'] for result in predictions]

    # Calcolo accuratezza per modello fine-tuned
    true_sentiments = test_file['airline_sentiment']
    accuracy = accuracy_score(true_sentiments, extracted_predictions)
    matrix_c = confusion_matrix(true_sentiments, extracted_predictions)
    print(f"Accuracy su per modello {fine_tuned}: {round(accuracy * 100, 2)}%")
    print(f"Confusion Matrix per modello {fine_tuned}: \n{matrix_c}")

    # Creazione del report di classificazione con ROBERTa modello fine-tuned
    roberta_classification_report = classification_report(true_sentiments, extracted_predictions)
    print(f"ROBERTa Classification Report per {fine_tuned}:")
    print(roberta_classification_report)

    # Grafico Matrice di confusione con mappa di calore per modello fine-tuned
    plt.figure(figsize=(8, 6))
    hmap = sns.heatmap(matrix_c, annot=True, fmt='d', cmap='Blues')
    class_names = ['Negativo', 'Neutro', 'Positivo']
    hmap.set_xticklabels(class_names, rotation=0, fontsize=10)
    hmap.set_yticklabels(class_names, rotation=0, fontsize=10)
    plt.ylabel('Sentimento Vero', fontsize=14)
    plt.xlabel('Sentimento Predetto', fontsize=14)
    plt.title(f'Matrice di confusione per {fine_tuned}', fontsize=16)
    plt.show()
```

Figura 46. Validate_model_fine_tuned.

La funzione **validate_model_pre_trained** esegue le stesse elaborazioni della precedente con l'unica differenza data dal modello utilizzato. In questo caso il modello che abbiamo utilizzato è "**cardiffnlp/twitter-roberta-base-sentiment-latest**".

```

christian_putzu
def validate_model_pre_trained(test_file):

    """ Funzione per validare il modello pre-addestrato """

    # Caricamento del modello pre-trained
    tokenizer = RobertaTokenizer.from_pretrained("cardiffnlp/twitter-roberta-base-sentiment-latest")
    model = RobertaForSequenceClassification.from_pretrained(
        "cardiffnlp/twitter-roberta-base-sentiment-latest")
    pipe = pipeline("sentiment-analysis", model=model, tokenizer=tokenizer)

    # Caricamento dei dati
    test_file = pd.read_csv(test_file)[:10000]

    # Calcolo predizioni per modello pre-trained
    predictions = [pipe(text) for text in test_file['text']]
    extracted_predictions = [result[0]['label'] for result in predictions]

    true_sentiments = test_file['airline_sentiment']
    accuracy = accuracy_score(true_sentiments, extracted_predictions)
    matrix_c = confusion_matrix(true_sentiments, extracted_predictions)
    print(f"Accuracy su per modello {pre_trained}: {round(accuracy * 100, 2)}%")
    print(f"Confusion Matrix per modello {pre_trained}: \n{matrix_c}")

    # Creazione del report di classificazione con RoBERTa modello pre-trained
    roberta_classification_report = classification_report(true_sentiments, extracted_predictions)
    print(f"RoBERTa Classification Report per {pre_trained}:")
    print(roberta_classification_report)

    # Grafico Matrice di confusione con mappa di calore per modello pre-trained
    plt.figure(figsize=(8, 6))
    hmap = sns.heatmap(matrix_c, annot=True, fmt='d', cmap='Blues')
    class_names = ['Negativo', 'Neutro', 'Positivo']
    hmap.set_xticklabels(class_names, rotation=0, fontsize=10)
    hmap.set_yticklabels(class_names, rotation=0, fontsize=10)
    plt.ylabel('Sentimento Vero', fontsize=14)
    plt.xlabel('Sentimento Predetto', fontsize=14)
    plt.title(f'Matrice di confusione per {pre_trained}', fontsize=16)
    plt.show()

```

Figura 47. Validate_model_pre_trained

```
if __name__ == "__main__":

```

Questo codice è stato progettato per eseguire una validazione dei modelli di analisi del sentimento (un modello fine-tuned e un modello pre-addestrato) utilizzando il file CSV specificato.

La funzione si occupa di validare un modello pre-addestrato utilizzando un file CSV di test specificato.

Definiamo il file di test con **test_file = 'Test_airlines-sentiment.csv'** assegna il nome del file CSV, contenente i dati di test relativi al sentimento delle compagnie aeree, alla variabile denominata **test_file**.

Poi, definiamo due stringhe, **fine_tuned** e **pre_trained**, che rappresentano i modelli che useremo, definiti in precedenza.

Successivamente, richiamiamo due funzioni:

validate_model_fine_tuned() e **validate_model_pre_trained()** passando il file di test come argomento. Queste funzioni contengono il codice necessario per la validazione dei modelli.

```

if __name__ == '__main__':
    # File di validazione
    test_file = 'Test_airlines-sentiment.csv'

    # Variabili per stampe debug e grafici
    fine_tuned = 'fine_tuned'
    pre_trained = 'pre_trained'

    # Chiamata di funzione per validare modello fine tuned
    validate_model_fine_tuned(test_file)

    # Chiamata di funzione per validare modello pre-trained
    validate_model_pre_trained(test_file)

```

Figura 48. Main: validation-model.py

VALIDAZIONE DEL MODELLO ROBERTA: calcolo dell'accuracy

Per validare l'attendibilità del calcolo del sentiment con il modello RoBERTa abbiamo utilizzato un dataset pre etichettato scaricato da Kaggle al seguente link:

<https://www.kaggle.com/datasets/crowdflower/twitter-airline-sentiment>

Il nome del dataset è Test_airlines-sentiment.csv, contenente 10000 tweets etichettati.

Il dataset è stato scelto per la sua data di creazione, risalente a febbraio 2015 che ci rende sicuri del fatto che non sia stato etichettato con il modello di RoBERTa da noi scelto in quanto la sua data di creazione risale ad un periodo compreso tra gennaio 2018 e dicembre 2021.

Per verificare quale dei due modelli fosse più accurato abbiamo effettuato una validazione con il dataset descritto in precedenza.

Abbiamo confrontato:

- il modello fine-tuned di RoBERTa (validate_model_fine_tuned, **bibbia/DriveFeelings-Roberta-sentiment-analyzer-for-twitter**),
- il modello pre-addestrato di RoBERTa (validate_model_pre_trained, **cardiffnlp/twitter-roberta-base-sentiment-latest**).

Per entrambi i modelli abbiamo eseguito:

abbiamo inserito il file all'interno dei modelli permettendoci di creare una colonna chiamata Prediction_twitter contenente le nuove etichette di sentiment calcolate con RoBERTa.

Per la validazione abbiamo calcolato l'accuracy dei modelli confrontando le etichette di sentiment già presenti all'interno del file Test_airlines_sentiment.csv, utilizzate come test, con le etichette di

tweet_id	airline_sentiment	airline_sentiment_confidence	airline
570306133677760513	neutral	1.0	,,Virgin America,,cairdin,
570301130888122368	positive	0.3486	,0.0,Virgin America,,
570301083672813571	neutral	0.6837	,,Virgin America,,yvoni
570301031407624196	negative	1.0	,Bad Flight,0.7033,Virgin
570300817074462722	negative	1.0	,Can't Tell,1.0,Virgin Am
570300767074181121	negative	1.0	,Can't Tell,0.6842,Virgin
	it's really the only bad thing about flying VA",		it's really the only bad thing about flying VA",,2015-02-
570300616901320704	positive	0.6745	,0.0,Virgin America,,
570300248553349120	neutral	0.634	,,Virgin America,,pilot,
570299953286942721	positive	0.6559	,,Virgin America,,dhej
570295459631263746	positive	1.0	,,Virgin America,,Yupits'
570294189143031808	neutral	0.6769	,0.0,Virgin America,,ic
570289724453216256	positive	1.0	,,Virgin America,,HyperC
570289584061480960	positive	1.0	,,Virgin America,,HyperC
570287408438120448	positive	0.6451	,,Virgin America,,moll

Figura 49. Dataset: Train_airlines-sentiment.csv

sentiment calcolate con il modello di RoBERTa utilizzato per calcolare il sentiment dei nostri dati sulle case automobilistiche.

Abbiamo applicato questo metodo di validazione per avere un ulteriore conferma sull'attendibilità del modello nella predizione.

- L'accuracy del modello **bibbia/DriveFeelings-Roberta-sentiment-analyzer-for-twitter** è 49,3%.

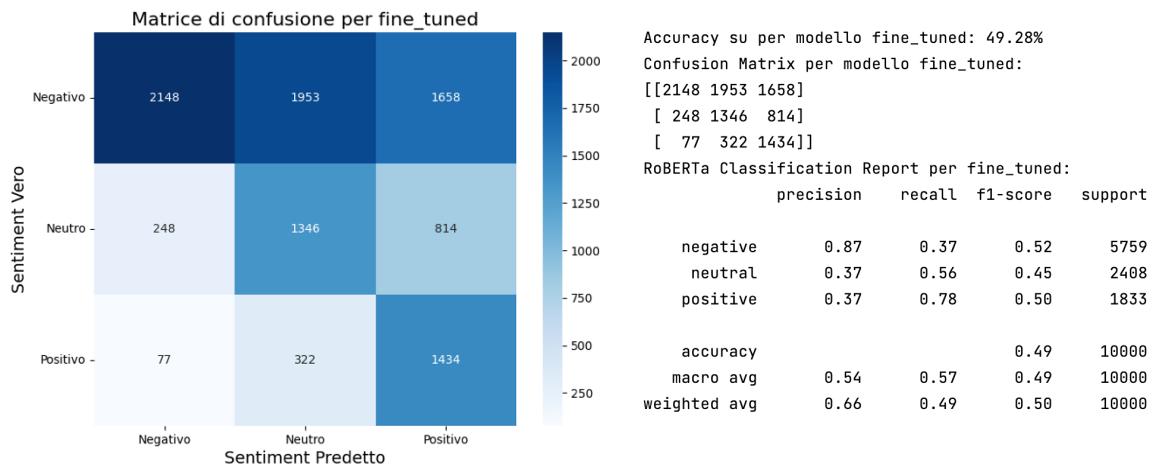


Figura 50. Matrice di confusione e classification report del modello fine-tuned.

- L'accuracy del modello **cardiffnlp/twitter-roberta-base-sentiment-latest** è 79,4% permettendoci di definire che il modello ottiene degli ottimi risultati nel calcolo delle etichette di sentiment.

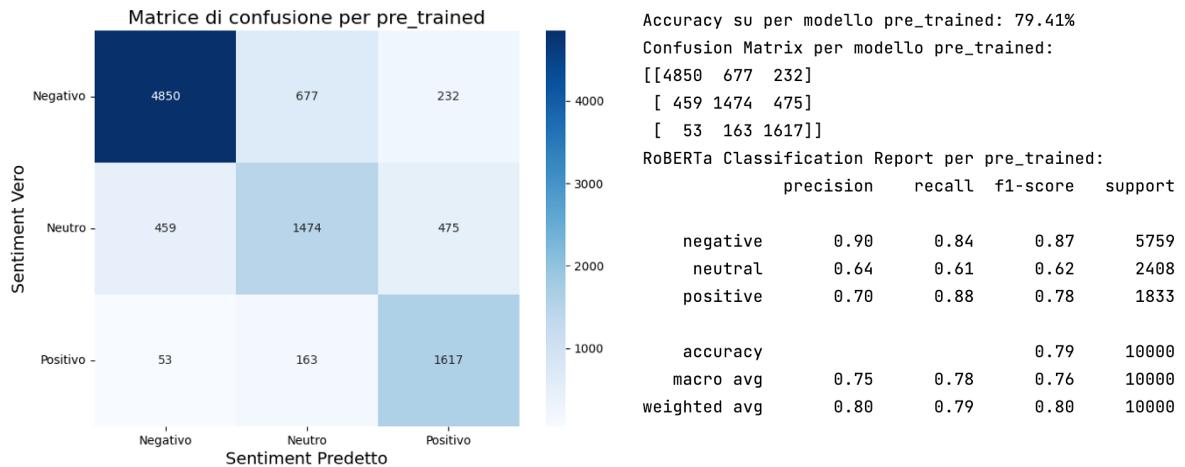


Figura 51. Matrice di confusione e classification report del modello pre-trained.

La validazione ha confermato che il modello pre-addestrato di RoBERTa per la sentiment analysis ci permette di ottenere dei risultati di classificazione migliori rispetto a quello su cui è stato eseguito un ulteriore addestramento attraverso il fine-tuning.

Sulla base dei risultati dell'accuratezza dei due modelli abbiamo preferito utilizzare il modello pre-addestrato di RoBERTa per il processo di sentiment analysis.

I nostri dati relativi alle case automobilistiche sono stati classificati utilizzando un modello di classificazione che ha dimostrato un'accuratezza molto promettente. Questo modello ha dimostrato di essere affidabile nel suo compito, fornendo risultati precisi nella categorizzazione dei dati

classification-model.py

Definiamo la classe **SentimentAnalyzer** che permette di analizzare il sentimento dei testi utilizzando il modello RoBERTa pre-addestrato, restituendo le etichette di sentiment associate a ciascun testo fornito come input. Questa classe contiene le seguenti funzioni:

```
christian_putzu
class SentimentAnalyzer:

    """Classe per la sentiment analysis con RoBERTa."""

    christian_putzu
    def __init__(self, input_files):
        self.input_files = input_files
```

Figura 52. Class SentimentAnalyzer.

Inizialmente importiamo **input_files** che prende i dati in input.

La funzione principale all'interno di questa classe è **analyze_sentiment_roberta**, utilizzata per analizzare il sentimento dei testi forniti come input.

Questa sfrutta una pipeline per il **text classification** utilizzando il modello RoBERTa pre-addestrato "**cardiffnlp/twitter-roberta-base-sentiment-latest**", in maniera "out of the box" cioè che non necessita di ulteriori configurazioni o modifiche. Abbiamo implementato un tokenizer del modello e definito la lunghezza massima con **max_length=512**.

La variabile **texts** rappresenta l'input, ovvero una lista di testi da analizzare.

All'interno della funzione, viene istanziata una pipeline con il modello RoBERTa per la classificazione del testo. Successivamente, questa pipeline viene utilizzata per processare i testi forniti come input. Il risultato restituito da **analyze_sentiment_roberta** è una lista contenente le etichette di sentiment associato a ciascun testo nell'input.

Per verificare le tempistiche di analisi abbiamo inserito un ciclo for che ci permette di stampare a schermo, tramite la libreria **tqdm** una barra di caricamento che conta tutti i file su cui viene calcolato il sentimento. Alla fine il risultato viene appeso su **sentiment_roberta**.

```

christian_putzu *
def analyze_sentiment_roberta(self, texts):

    """Funzione che analizza il sentiment dei testi utilizzando RoBERTa e restituisce etichette."""

    # Pipeline modello per analizzare il sentiment dei testi
    pipe = pipeline("text-classification",
                    model="cardiffnlp/twitter-roberta-base-sentiment-latest",
                    tokenizer="cardiffnlp/twitter-roberta-base-sentiment-latest",
                    max_length=512)

    sentiment_roberta = []

    for text in tqdm(texts, desc=f"Calcolo sentiment per file {input_file}", unit="testo"):
        result = pipe([text])
        sentiment_roberta.append(result[0]['label'])

    # Return del calcolo delle etichette di sentiment
    return sentiment_roberta

```

Figura 53. Analyze_sentiment_roberta

if __name__ == "__main__":

All'inizio del codice, definiamo le directory di lavoro di input e di output.

Creiamo l'oggetto **sentiment_analyzer** della classe **SentimentAnalyzer** precedentemente definita. Questo oggetto verrà utilizzato per condurre l'analisi del sentiment.

Successivamente, effettuiamo un'iterazione sui file nella directory di input, creando un ciclo che attraversa tutti i file. Per ciascun file, eseguiamo le seguenti operazioni:

- Lettura del file di input: se il file è un file CSV, leggiamo e memorizziamo i dati nel data frame **df**.
- Verifichiamo se il dataframe **sentiment_df** è vuoto: se "sentiment_df" è vuoto, il data frame "df" viene assegnato direttamente al data frame. Se invece "sentiment_df" contiene già dei dati, "df" viene unito al dataframe lungo l'asse delle righe utilizzando la funzione **pd.concat**.
- L'opzione **ignore_index=True** assicura che l'indice delle righe nel dataframe risultante sia riorganizzato in modo continuo, ignorando gli indici dei dataframe originali. Alla fine di questo blocco di codice, "sentiment_df" conterrà i dati combinati da tutti i dataframe "df" considerati.
- Utilizziamo l'oggetto **sentiment_analyzer** per eseguire l'analisi del sentiment su tutti i testi presenti nel file. I risultati vengono poi scritti in un nuovo file CSV di output.
- Le etichette di sentiment vengono aggiunte come una nuova colonna al Data Frame.
- Il Data Frame aggiornato con le etichette di sentiment viene salvato come un nuovo file CSV nella directory di output.
- Infine, stampiamo i risultati dell'analisi del sentiment, tra cui le distribuzioni del sentiment sotto forma di grafici a barre e il conteggio dei tweet per etichetta di sentiment. Questa visualizzazione fornisce un'idea chiara delle distribuzioni dei sentiment nei dati.

- Al termine del programma ci vengono forniti come output tre file CSV che vengono inseriti nella cartella Tweet_sentiment.

```

if __name__ == "__main__":
    # Directory di input e output
    input_folder = "./tweets_puliti"
    output_folder = "./tweets_sentiment"

    # Oggetto Classe utilizzato per le predizioni
    sentiment_analyzer = SentimentAnalyzer(input_folder)

    # Elaborazione dei file nella directory di input
    for input_file in os.listdir(input_folder):

        if input_file.endswith('.csv'):
            target_name_input = input_file.split('-')[-2]
            input_filepath = os.path.join(input_folder, input_file)
            df = pd.read_csv(input_filepath)

            # Converti la serie 'Content' in una lista di stringhe
            texts = df['Content'].astype(str).tolist()

            # Analisi del sentiment per i testi nel DataFrame
            sentiments = sentiment_analyzer.analyze_sentiment_roberta(texts)

            # Aggiunta delle etichette di sentiment al DataFrame
            df['Sentiment_Roberta'] = sentiments

            # Salva i risultati in un nuovo file CSV nella cartella di output
            output_filename = f"roberta-sentiment-analysis-{target_name_input}.csv"
            output_filepath = os.path.join(output_folder, output_filename)
            df.to_csv(output_filepath, index=False)

            # Conteggio delle etichette di sentiment
            roberta_counts = df['Sentiment_Roberta'].value_counts()

            # Grafici a barre per visualizzare le distribuzioni del sentiment per tutti i file,
            plt.figure(figsize=(12, 10))
            sns.barplot(x=roberta_counts.index, y=roberta_counts.values, palette="Set2")
            plt.title(f"Sentiment con RoBERTa per {target_name_input}", fontsize=30)
            plt.ylabel('Frequenza assoluta', fontsize=24)
            plt.xlabel('Sentiment RoBERTa', fontsize=24)
            plt.xticks(fontsize=20)
            plt.yticks(fontsize=20)
            plt.show()

            # Stampe debug percentuali tweets per etichetta di sentiment
            totale = float(roberta_counts['positive'] + roberta_counts['neutral'] + roberta_counts['negative'])
            print(f"\nFrequenze relative per le etichette di sentiment di {target_name_input}:")
            print(f"Sentiment Positivi: {round((roberta_counts['positive'] / totale * 100), 2)}%\n")
            print(f"Sentiment Neutri: {round((roberta_counts['neutral'] / totale * 100), 2)}%\n")
            print(f"Sentiment Negativi: {round((roberta_counts['negative'] / totale * 100), 2)}%\n")

```

Figura 54. Main: classification-model.py

Tweets_sentiment

I tre file csv di output sono contenuti dentro la cartella “tweets_sentiment” e sono:

sentiment-analysis-roberta-tweets_puliti-BMW-en.csv,
sentiment-analysis- roberta-tweets _puliti-Tesla-en.csv,
sentiment-analysis- roberta-tweets_puliti-Renault-en.csv

Questi nuovi file CSV contengono le colonne originali del file di input a cui sono state aggiunte delle colonne aggiuntive che rappresentano il sentiment calcolato.

Username (1)	Content (2)	Sentiment_Roberta (3)	Username (1)	Content (2)	Sentiment_Roberta (3)	Username (1)	Content (2)	Sentiment_Roberta (3)
Username	Content	Sentiment_Roberta	Username	Content	Sentiment_Roberta	Username	Content	Sentiment_Roberta
SurajKaroF5966	u consumer still prefer	Neutro	1 Li3AModelS	ezebron1 bought drive	Neutro	2 columvire	whatcar collected zed gt line plus reg second zed wife mom love	Neutro
JL_____JL_____JL	macrumors break wireless	Neutro	3 KarenV35692848	ajtourville big thing taken	Positivo	3 muckyfellrunner	silverstoneuk remembering brother used decrepit era think would fun	Positive
DVHcloud_IT	il dominio è un elemento importante di qualsiasi azienda sapevi che registrare un dominio non ci vu anni	Neutro	4 KureWellness	consideration calculation worker get	Neutro	4 karma_shopping	overpay amazon ever amazon shopping hack prime day steroid	Negativo
bankyuk1	rollsroycecars mini ucblxx abglitched carterdacar pearlwrap	Neutro	5 yramkee	rondesantis game whether ev incentive	Neutro	5 colinwalker79	top notch trolling	Negativo
BMWAustin	feel with hair lung finetip convertible available austin experience unmatched elegance	Positivo	6 hikingskiing	predict ultimately able assemble	Neutro	6 renault_uk	vive electric revolution	Neutro
LRIRw	forza forzashare xboxshare xbox bluerarchive ブルーアーカイブ ブラック 阿慈谷 レフミ 阿慈谷	Neutro	7 BeneluxTesla	tbh think wait enable local law say point failure want collect much data	Negativo	7 QatarRenault	october mark beginning special year u year celebrate year year count tot	Positivo

Figura 55. Tweets_sentiment: BMW, Tesla e Renault.

Qui sotto sono forniti i risultati dell’analisi di sentiment con RoBERTa per i tre brand automobilistici:
BMW

Analisi di /Users/albertomancosu/pythonProject7/pythonProjects7/tweets_puliti/tweets_puliti-BMW-en.csv con Roberta: 100% | 10000/10000

Risultati dell’analisi del sentiment scritti in sentiment-analysis-textblob_vs_roberta-tweets_puliti-BMW-en.csv

Sentiment Positivi: 21.34%

Sentiment Neutri: 60.02%

Sentiment Negativi: 18.64%

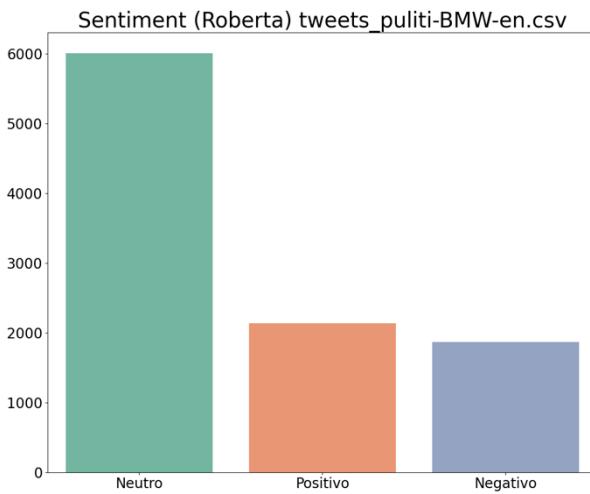


Figura 56. Grafico a barre BMW.

- Nella stampa di debug sono mostrate le percentuali delle etichette dei sentiment.
- In questo grafico a barre sono mostrate le frequenze del numero di etichette (neutro, positivo e negativo) determinate per il file di BMW.
- Le percentuali delle etichette per la casa automobilistica BMW:
 - Positivi: 21.34%
 - Neutri: 60.02%
 - Negativi: 18.64%

Tesla

Analisi di /Users/albertomancosu/pythonProject7/pythonProjects7/tweets_puliti/tweets_puliti-Tesla-en.csv con Roberta: 100%|██████████| 10000/10000

Risultati dell'analisi del sentiment scritti in sentiment-analysis-textblob_vs_roberta-tweets_puliti-Tesla-en.csv

Sentiment Positivi: 21.04%

Sentiment Neutri: 58.78%

Sentiment Negativi: 20.18%

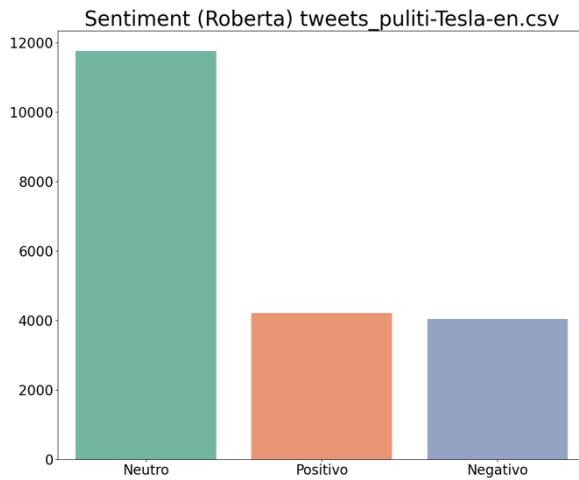


Figura 57. Grafico a barre Tesla.

- Nella stampa di debug sono mostrate le percentuali delle etichette dei sentiment.

- In questo grafico a barre sono mostrate le frequenze del numero di etichette (neutro, positivo e negativo) determinate per il file di Tesla. Le percentuali delle etichette per la casa automobilistica Tesla:

- Positivi: 21.04%
- Neutri: 58.78%
- Negativi: 20.18%

Renault

Analisi di /Users/albertomancosu/pythonProject7/pythonProjects7/tweets_puliti/tweets_puliti-Renault-en.csv con Roberta: 100%|██████████| 10000/10000

Risultati dell'analisi del sentiment scritti in sentiment-analysis-textblob_vs_roberta-tweets_puliti-Renault-en.csv

Sentiment Positivi: 20.06%

Sentiment Neutri: 59.74%

Sentiment Negativi: 20.2%

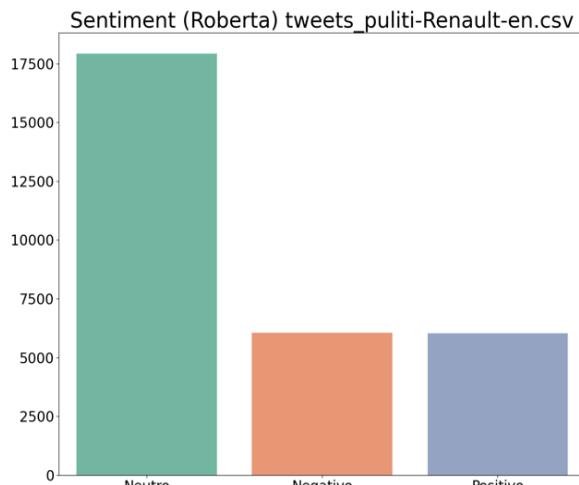


Figura 58. Grafico a barre Renault.

- Nella stampa di debug sono mostrate le percentuali delle etichette dei sentiment

- In questo grafico a barre sono mostrate le frequenze del numero di etichette (neutro, positivo e negativo) determinate per il file di Renault.

Le percentuali delle etichette per la casa automobilistica Renault:

- Positivi: 20.06%
- Neutri: 59.74%
- Negativi: 20.2%

Interpretazione dei risultati e conclusioni

Il progetto ha fornito tante informazioni relative alle discussioni degli utenti di Twitter in ambito automobilistico, legate alle query che gli sono state inserite come input.

Il branch di **pre-processing** ha avuto un ruolo fondamentale per l'affinamento dell'analisi, perché attraverso il suo debugging e la sua modifica siamo riusciti ad ottenere dei risultati che ci potessero soddisfare.

Nella scelta delle stopwords da eliminare, abbiamo effettuato un lavoro di selezione secondo un criterio di identificazione dell'importanza dei termini. Un esempio è quello delle parole "elon" e "musk" che inizialmente sono state eliminate perché molto frequenti e successivamente è stata mantenuta "elon" perché ritenuta importante per le analisi di topic e sentiment. Abbiamo invece eliminato la keyword "time" che si ripeteva in tanti topic e non arricchiva l'analisi.

I branches delle analisi di topic modeling e di sentiment analysis ci hanno restituito un ampio numero di risultati, che abbiamo interpretato per comprendere quelli che sono i pareri sulle case automobilistiche prese come soggetti per l'analisi.

I risultati forniti dall'analisi di **topic modeling** sono i grafici delle pagine html prodotte.

All'interno questi presentano l'Intertopic Distance Map, ossia una mappa in cui sono presenti dei cerchi che indicano i cluster dei 5 diversi topic e un grafico a barre con la term-frequency che indica la frequenza delle parole più utilizzate all'interno di ogni topic.

Per quanto riguarda l'interpretazione della mappa delle distanze tra i topic: la posizione e la distanza dei cluster ci hanno indicato se i topic sono correlati e se gli argomenti principalmente trattati siano in relazione tra loro.

Mentre, il grafico della frequenza delle parole ci indica quali sono gli argomenti di discussione generali più importanti per ogni casa.

BMW

I termini più salienti evidenziati offrono un quadro significativo riguardo alle discussioni prevalenti all'interno dei cinque topic individuati.

I termini più comuni come "engine", "gt", "eletric", "suv" e "competition" emergono come indicatori chiave all'interno di queste conversazioni. Questi termini rappresentano argomenti specifici e rilevanti all'interno dei vari topic, delineando tematiche come motori, auto sportive, veicoli elettrici, SUV e competizioni automobilistiche, che sono centrali nelle discussioni degli utenti di Twitter.

La mappa delle distanze tra i topic, chiaramente illustrata dai cerchi rappresentativi dei cluster, mette in luce la relazione e la distanza semantica tra i vari argomenti. I topic 2 e 3 mostrano una notevole vicinanza, indicando una forte correlazione tra di loro, il che è supportato dalla condivisione di termini simili e rilevanti come "gt", "car", "competition", "eletric", "ev", "suv" e "India". Al contrario, i topic 1, 4 e 5 si presentano come più distanti tra loro, suggerendo una minor

correlazione e una maggiore indipendenza. Questo è riflesso nei termini predominanti di ciascun topic, evidenziando la diversità degli argomenti trattati.

I topic 1, 4 e 5 trattano argomenti più generici o distinti rispetto ai topic 2 e 3. Ad esempio, il Topic 1 sembra concentrarsi su concetti come desideri, bisogni e opinioni personali delle persone, il Topic 4 è fortemente orientato all'ingegneria automobilistica, mentre il Topic 5 sembra affrontare argomenti più generali e distanti dall'ambito automobilistico come la guida e i giochi.

Tesla

I termini più frequenti evidenziati nell'analisi, come "model", "ev", "electric", "china", "vehicle", "tsla" e "good" forniscono una panoramica significativa degli argomenti trattati dagli utenti su Twitter in relazione a Tesla.

Esaminando la mappa delle distanze tra i topic, emerge che i vari topic sono generalmente indipendenti l'uno dall'altro. Questa indipendenza suggerisce una diversità di argomenti trattati nelle discussioni, ognuno con un proprio focus distintivo.

Analizzando i singoli topic, emerge un quadro più dettagliato:

1. Topic 1: presenta termini che riflettono bisogni e desideri delle persone, insieme a menzioni di Elon Musk e concetti come "ev" e "model".
2. Topic 2: sembrerebbe affrontare argomenti riguardanti il modello Tesla, con particolare enfasi su Tesla in Cina, il prezzo delle azioni, il modello Cybertruck e Elon Musk.
3. Topic 3: è orientato verso concetti più astratti come "view", "ai", "think" e "fsd" (Full Self-Driving). Questo potrebbe suggerire una discussione legata a prospettive e idee riguardo al futuro dell'azienda e della tecnologia.
4. Topic 4: concentrato su termini legati all'aspetto tecnologico e all'offerta di Tesla, inclusi "electric", "ev", "vehicle", "buy", "battery" e "Elon".
5. Topic 5: contiene termini che suggeriscono un'opinione positiva o emozionale, come "love", "company", "got", "make" e "delivery." Questo potrebbe indicare conversazioni sull'apprezzamento per l'azienda o sulle esperienze legate alla ricezione e all'utilizzo dei veicoli Tesla.

Quest'analisi evidenzia una diversità di argomenti trattati nelle conversazioni su Twitter riguardo a Tesla. Dai desideri e le prospettive dei consumatori, alle questioni di mercato, ai dettagli tecnici e all'esperienza degli utenti, le discussioni riflettono un'ampia gamma di argomenti che interessano la comunità che discute di Tesla su questa piattaforma.

Renault

L'analisi dei "Top 30 most salient terms" derivanti dalla modellazione dei topic nelle discussioni su Twitter riguardo Renault offre un'interessante panoramica delle tematiche trattate dagli utenti.

I termini più frequenti come "team", "engine", "race", "car", "ev" e "electric" delineano le principali aree di discussione. Questi termini riflettono una gamma diversificata di argomenti, che vanno dalla performance delle auto, alle competizioni, alla tecnologia e alla mobilità elettrica.

Osservando la mappa delle distanze tra i topic, emerge una correlazione tra i topic 1 e 3, evidenziata dalla loro vicinanza. Questi topic condividono alcuni termini come "car", "ev" e "electric" tra di loro. Al contrario, i topic 2, 4 e 5 risultano più distanti e indipendenti, suggerendo una minore correlazione tra di loro.

Esaminando i singoli topic, emergono dettagli interessanti:

1. Topic 1: Include termini come "company", "ev", "scenic", "electric", "car" e "year", Sembra concentrarsi su aspetti legati all'azienda Renault e alla sua offerta di veicoli, includendo la mobilità elettrica.

3. Topic 3: Contiene termini come "alpine", "team", "gt", "car" e "clio". Questo potrebbe riguardare principalmente i modelli e le attività legate alle corse o alle divisioni specifiche all'interno dell'azienda.

Invece, i topic più distanti presentano distinte aree di interesse:

2. Topic 2: Include termini come "engine", "team", "mclaren", "ferrari" e "redbull". Sembrerebbe concentrarsi sulle prestazioni dei motori e sulle competizioni sportive, enfatizzando i team di diverse scuderie di Formula 1.

4. Topic 4: Contiene termini come "alonso", "race", "massa", "grand" e "prix".

Si riferisce principalmente a piloti e competizioni automobilistiche di alto livello come la Formula 1.

5. Topic 5: Include termini come "win", "race", "year", "think" e "want". Si concentra su emozioni e desideri legati alle competizioni e alle vittorie.

L'analisi dei topic sull'argomento Renault riflette una gamma diversificata di aree di interesse, che spaziano dalle prestazioni delle auto, alle competizioni sportive, alla mobilità elettrica, enfatizzando aspetti legati all'azienda e alle emozioni dei fan riguardo a vittorie e piloti.

In generale quindi, con i risultati di analisi di topic modeling siamo riusciti ad individuare cinque topic che ci hanno permesso di raggruppare quelli che sono i principali temi di dibattito di ogni brand.

BMW sembra attrarre discussioni su una vasta gamma di aspetti legati all'automobilismo, inclusi gli aspetti ludici e le opinioni personali, dimostrando una forte presenza in termini di varietà di interesse. Tesla, al contrario, sembra avere una platea di utenti altamente coinvolta e diversificata, che si concentra su una gamma ampia di temi, dalla tecnologia ai desideri dei consumatori. Infine, Renault sembra avere una base di appassionati molto orientata verso gli sport automobilistici, ma anche interessata alla mobilità elettrica e alle prestazioni dei motori.

In generale, mentre ogni azienda automobilistica attrae una vasta gamma di discussioni e coinvolge gli utenti su diverse tematiche, Tesla emerge come l'azienda con una maggiore varietà di temi e coinvolgimento, seguita da BMW e poi da Renault, che mostra un orientamento più specifico verso le competizioni automobilistiche.

Analizzando invece i risultati ottenuti nel branch **RoBERTa Sentiment analysis** per le case automobilistiche BMW, Tesla e Renault si notano delle similitudini e delle leggere differenze nei risultati:

- Per **BMW**, RoBERTa ha identificato che il 21.34% delle opinioni esaminate sono state classificate come positive, il 60.02% come neutre e il 18.64% come negative.
- Per **Tesla**, il 21.04% delle opinioni sono state valutate come positive, il 58.78% come neutre e il 20.18% come negative.
- Per **Renault**, RoBERTa ha classificato il 20.06% delle opinioni come positive, il 59.74% come neutre e il 20.2% come negative.

Questi dati riflettono le percentuali dei sentiment rilevate e classificate dal modello RoBERTa. Innanzitutto, la predominanza del sentimento neutro è evidente in tutte e tre le marche automobilistiche, rappresentando la categoria più ampia in ciascun caso, indicando che la maggior parte dei testi analizzati non manifesta un forte sentimento positivo o negativo verso le case automobilistiche considerate. Invece evidenzia che i termini e le discussioni più frequenti sono di natura tecnica e legata ai motori, come si evince dall'analisi della topic modeling.

Un'ulteriore considerazione sulla causa della presenza della dominanza del sentimento neutro è il fatto che molti tweets siano di natura pubblicitaria o comunque prodotti da bot.

La presenza dei bot è confermata dalle politiche che Elon Musk sta inserendo per il social network Twitter, cercando di limitare la presenza di profili non verificati e che producono spam inserendo delle licenze a pagamento per disincentivare la creazione di profili di questo tipo.

(https://www.ilmessaggero.it/social/instagram_meta_twitter_bot_contenuti_falsi-7355744.html?refresh_ce).

Mentre i sentiment neutri dominano, le leggere variazioni nei sentimenti positivi e negativi tra le case automobilistiche indicano una sottile differenza nelle percezioni delle persone riguardo a queste marche. Nonostante le differenze minime, sembra che Tesla e Renault possano ricevere una percentuale leggermente maggiore di feedback negativi rispetto a BMW.

D'altra parte, se si confrontano i sentimenti positivi e negativi, si nota una differenza minima.

Tutte e tre le marche hanno valori percentuali simili di sentimenti positivi, che oscillano attorno al 20-21%. BMW è la casa che presenta più etichette di sentiment positive, comunque di poco superiori rispetto alle altre due.

Una differenza un po' più marcata si evidenzia nei sentimenti negativi, in cui Renault ha la percentuale più alta (20.2%), seguita da Tesla (20.18%) e infine BMW (18.64%).

Questo potrebbe indicare che, sebbene tutte e tre le marche abbiano una presenza simile di sentimenti positivi, è interessante notare che Tesla e Renault sono soggette a discussioni e argomentazioni più critiche rispetto a BMW.

Tuttavia, nel complesso, l'andamento generale dei sentimenti positivi, neutri e negativi è sostanzialmente simile tra le tre marche.

In conclusione, il progetto ha fornito una visione approfondita delle opinioni e delle discussioni online relative alle case automobilistiche BMW, Renault e Tesla. Sono stati identificati i sentimenti prevalenti e i principali argomenti di discussione.

Tuttavia, un aspetto importante che dobbiamo considerare è la presenza di classi sbilanciate all'interno dei dati. Questo significa che potrebbero esserci discrepanze significative nel numero di dati rappresentativi per ciascuna casa automobilistica. Affrontare questo problema è cruciale per garantire risultati più accurati e bilanciati. Una strategia per far fronte alle classi sbilanciate potrebbe essere l'utilizzo di tecniche di bilanciamento dei dati, come l'oversampling o l'undersampling. Queste metodologie consentono di equilibrare il dataset, garantendo che ciascuna casa automobilistica abbia un peso paritario nell'analisi.