Colombo Lorenzo 885895

Corso Large scale data management:

BENCHMARKING E SENTIMENT ANALYSIS SU CLUSTER SPARK

Sommario

1. Introduzione

Il presente progetto si pone l’obiettivo di progettare, implementare e testare una pipeline di Big Data Analytics utilizzando Apache Spark in un ambiente distribuito su Cloud Azure. Il caso d'uso selezionato riguarda l'analisi del sentiment di un dataset massivo eterogeneo composto da recensioni utente di 20 applicazioni distinte, estratte dal Google Play Store.

L'elaborato non si limita all'estrazione di insight di business, ma pone un forte accento sugli aspetti ingegneristici del Large Scale Data Management come:

- Gestione del Volume: Vengono applicate tecniche di espansione sintetica dei dati per raggiungere la soglia dei 20 milioni di record, stressando così le capacità di memoria e I/O del cluster;
- Analisi di Scalabilità: Viene effettuato un Benchmarking rigoroso delle prestazioni del cluster al variare delle risorse computazionali, confrontando i tempi di esecuzione su configurazioni hardware differenti.

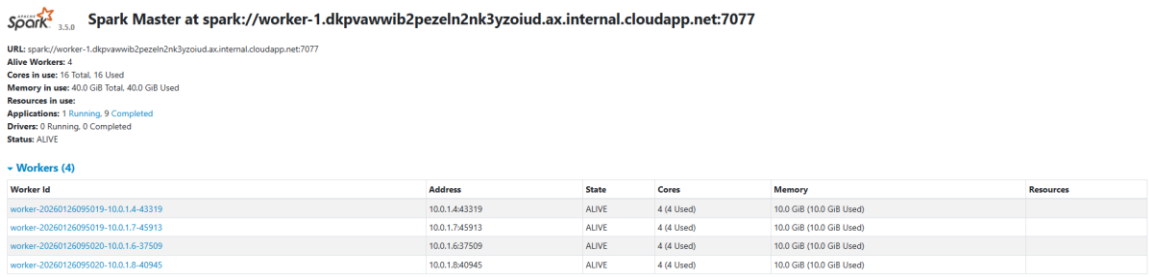
2. Architettura del Sistema

Per simulare un ambiente di produzione reale, è stata predisposta un'infrastruttura distribuita basata su macchine virtuali ospitate su Microsoft Azure.

2.1 Configurazione del Cluster su Azure

Il cluster è composto da cinque nodi (una vm master e quattro vm worker) equipaggiati con sistema operativo Linux. La topologia di rete è stata configurata per permettere la comunicazione diretta tra i nodi tramite IP privati all'interno della stessa Virtual Network, garantendo bassa latenza.

- Nodo Master (10.0.1.5): Responsabile della gestione delle risorse e dell'orchestrazione dei job.
- Nodi Worker (10.0.1.4, 10.0.1.6, 10.0.1.7, 10.0.1.8): Responsabili dell'esecuzione materiale dei task di calcolo.



2.2 Stack Software

È stato scelto di utilizzare uno stack Big Data standard:

- Apache Spark 3.5.0 in modalità Standalone Cluster Manager.
- Apache Hadoop 3.3.6 (HDFS) per la gestione distribuita dei file.
- Python per la definizione delle pipeline.

2.3 Strategia di Storagee

Per garantire affidabilità, scalabilità e gestione dei guasti, è stato installato e configurato Hadoop Distributed File System (HDFS). L'architettura di storage riflette la topologia del cluster di calcolo:

- NameNode: Gestisce i metadati del file system;
- DataNodes (Workers): Gestiscono l'archiviazione fisica dei blocchi dati sui dischi locali delle singole macchine.

È stato configurato un Replication Factor pari a tre. Questo significa che ogni blocco di dati viene replicato automaticamente su 3 nodi diversi. Questa scelta architetturale garantisce due vantaggi chiave:

1. Fault Tolerance: Se un nodo cade, i dati non vengono persi perché esistono altre copie nel cluster.
2. Data Locality: Spark, interrogando il NameNode, è in grado di schedulare i task di calcolo esattamente sul nodo che ospita il dato fisico, minimizzando il traffico di rete (Network Shuffle).

3. Data Engineering e Preparazione

La fase di ingestione e preparazione dei dati è stata progettata per gestire la Varietà e il Volume del dataset sfruttando il file system distribuito.

3.1 Ingestione

l punto di partenza è costituito da 20 file CSV separati. Lo script di ingestione interagisce direttamente con il protocollo HDFS per:

- Caricare i file raw nella directory /input del cluster HDFS.
- Leggere in parallelo i blocchi dati distribuiti sui DataNodes.
- Unificarli in un unico DataFrame Spark distribuito.

3.2 Data Augmentation

Per testare efficacemente le capacità di Spark su grandi moli di dati, la somma dei record originali non era sufficiente. È stato quindi implementato un modulo di Data Augmentation che utilizza le funzioni native di Spark explode e array_repeat. Questo approccio permette di moltiplicare i record esistenti per un fattore N calcolato dinamicamente, fino a raggiungere il target prefissato di > 20 milioni di righe. L'operazione di esplosione dei record ha comportato la duplicazione dei reviewId originali, rendendolo inadatto a identificare univocamente le righe nel dataset espanso. Per questo motivo, è stata introdotta una nuova colonna unique_id generata tramite la funzione monotonically_increasing_id(), garantendo così una chiave primaria unica per ciascuno dei 20 milioni di record generati.

```
 GNU nano 6.2                                                    prepare_data.py
# --- CONFIGURAZIONE ---
APP_NAME = "1_Data_Preparation_Augmentation"
INPUT_CSV_PATH = "hdfs://10.0.1.5:9000/progetto/data/*.csv"
OUTPUT_PARQUET_PATH = "hdfs://10.0.1.5:9000/progetto/data_parquet"

TARGET_ROWS = 20000000

spark = SparkSession.builder.appName(APP_NAME).getOrCreate()
spark.sparkContext.setLogLevel("WARN")

print(f"AVVIO PREPARAZIONE DATI")

# 1. INGESTIONE E TAGGING
print("\n[1] Ingestione Dati Grezzi")
try:
    df_base = spark.read.csv(INPUT_CSV_PATH, header=True, inferSchema=True)
    df_raw = df_base.withColumn("full_path", input_file_name()) \
        .withColumn("app_name",
                    regexp_replace(element_at(split(col("full_path"), "/"), -1), "\.csv", "")
                   ) \
        .drop("full_path")
    current_rows = df_raw.count()
    print("Anteprima")
    df_raw.select("app_name").distinct().show(5, truncate=False)
except Exception as e:
    print(f"ERRORE: {e}")
    spark.stop()
    exit(1)

# 2. DATA AUGMENTATION
print("\n[2] Data Augmentation")
FACTOR = int(TARGET_ROWS / current_rows) + 1
print(f"   -> Moltiplicazione x{FACTOR} per raggiungere target {TARGET_ROWS}")
df_expanded = df_raw.withColumn("dummy", explode(array_repeat(lit(0), FACTOR))) \
                .drop("dummy")
df_final = df_expanded.withColumn("unique_id", monotonically_increasing_id())
total_rows = df_final.count()

# 4. SALVATAGGIO
print("\n[3] Scrittura su Disco")
start_pq = time.time()
df_final.write.mode("overwrite").parquet(OUTPUT_PARQUET_PATH)

time_pq = time.time() - start_pq

spark.stop()
```

3.3 Ottimizzazione Storage

I dati aumentati non vengono salvati nuovamente in CSV, ma convertiti in Apache Parquet in quanto è un formato di storage colonnare che offre vantaggi critici per l'analisi successiva:

1. Compressione: Riduzione dello spazio su disco grazie ad algoritmi di compressione efficienti applicati per colonna.
2. Pushdown Projection: Spark è in grado di leggere dal disco solo le colonne necessarie all'analisi, ignorando colonne pesanti ma inutili, riducendo drasticamente l'I/O.

4. Pipeline di Analisi NLP

Il cuore applicativo del progetto è lo script di analisi, che implementa una logica MapReduce per l'estrazione di pattern semantici dai testi delle recensioni.

4.1 Pre-processing e Cleaning

La fase di pulizia, implementata con trasformazioni Spark SQL, normalizza il testo per renderlo analizzabile.

Sono state implementate le seguenti trasformazioni:

- Conversione in minuscolo;
- Rimozione di caratteri speciali e punteggiatura tramite Regex;
- Normalizzazione degli spazi bianchi;
- Rimozione preventiva di porzioni di frasi che inquinerebbero l'analisi.

4.2 Feature Extraction

L'analisi a singola parola si è rivelata insufficiente per catturare il sentiment reale perché, ad esempio, la parola "like" appariva spesso in recensioni negative perché preceduta da "don't" o "didn't". Per risolvere questa ambiguità semantica, è stata implementata la generazione di Bi-grammi tramite la classe NGram di Spark ML. Questo ha permesso di preservare il contesto locale, distinguendo chiaramente termini come "really like" da "dont like".

4.3 Strategia di Stopwords Removal

La rimozione delle parole inutili è stata raffinata distinguendo tra:

- Rumore Generico: Articoli, preposizioni "the", "and".
- Rumore di Dominio: Termini specifici molto frequenti ma poco informativi in questo contesto, come "app", "game", "play", "update", oppure i nomi delle varie applicazioni.

4.4 Aggregazione MapReduce

La fase finale sfrutta il paradigma distribuito di Spark:

1. **Map:** Appiattimento della lista di bi-grammi in singole righe.
2. **Shuffle & Reduce:** Aggregazione per chiave composta e conteggio delle occorrenze.

5. Benchmarking e Scalabilità

L'obiettivo sperimentale del progetto era verificare la legge di scalabilità di Spark al variare delle risorse. Perciò sono state implementate alcune tecniche di benchmarking per raccogliere dati da analizzare.

5.1 Metodologia di Test

Sono stati definiti ed eseguiti due scenari di test, mantenendo costante il carico di lavoro e variando le risorse di calcolo disponibili:

- Scenario A: Esecuzione su 4 Nodi;

```
Colomboadmin@worker-1:~/progetto$ spark-submit --master spark://10.0.1.5:7077 --total-executor-cores 16 \
  --executor-cores 4 --executor-memory 10g analysis_pipeline.py
```

- Scenario B: Esecuzione su 3 Nodi;

```
Colomboadmin@worker-1:~/progetto$ spark-submit --master spark://10.0.1.5:7077 --total-executor-cores 12 \
  --executor-cores 4 --executor-memory 10g analysis_pipeline.py
```

- Scenario C: Esecuzione su 2 Nodi;

```
Colomboadmin@worker-1:~/progetto$ spark-submit --master spark://10.0.1.5:7077 --total-executor-cores 8 \
  --executor-cores 4 --executor-memory 10g analysis_pipeline.py
```

- Scenario D: Esecuzione su 1 Nodo.

```
Colomboadmin@worker-1:~/progetto$ spark-submit --master spark://10.0.1.5:7077 --total-executor-cores 4 \
  --executor-cores 4 --executor-memory 10g analysis_pipeline.py
```

5.2 Setup di test

I test di benchmark sono stati eseguiti su un cluster Spark Standalone composto da quattro nodi fisici, con la seguente configurazione hardware e software per nodo:

- CPU: 4 vCPU
- RAM Totale: 10 GB
- Storage: SSD Locale

Il dataset utilizzato comprende 20 Milioni di record.

5.3 Risultati ottenuti

Configurazione	Core Totali	RAM Totale	Load (s)	NLP (s)	Calcolo (s)	Totale (s)
1 Nodo	4 CPU	10 GB	5.19	0.83	237.54	243.96
2 Nodi	8 CPU	20GB	5.78	0.77	135.75	142.63
3 Nodi	12 CPU	30 GB	5.78	0.78	83.2	90.08
4 Nodi	16 CPU	40 GB	5.66	0.78	73.40	80.12

Completed Applications (13)

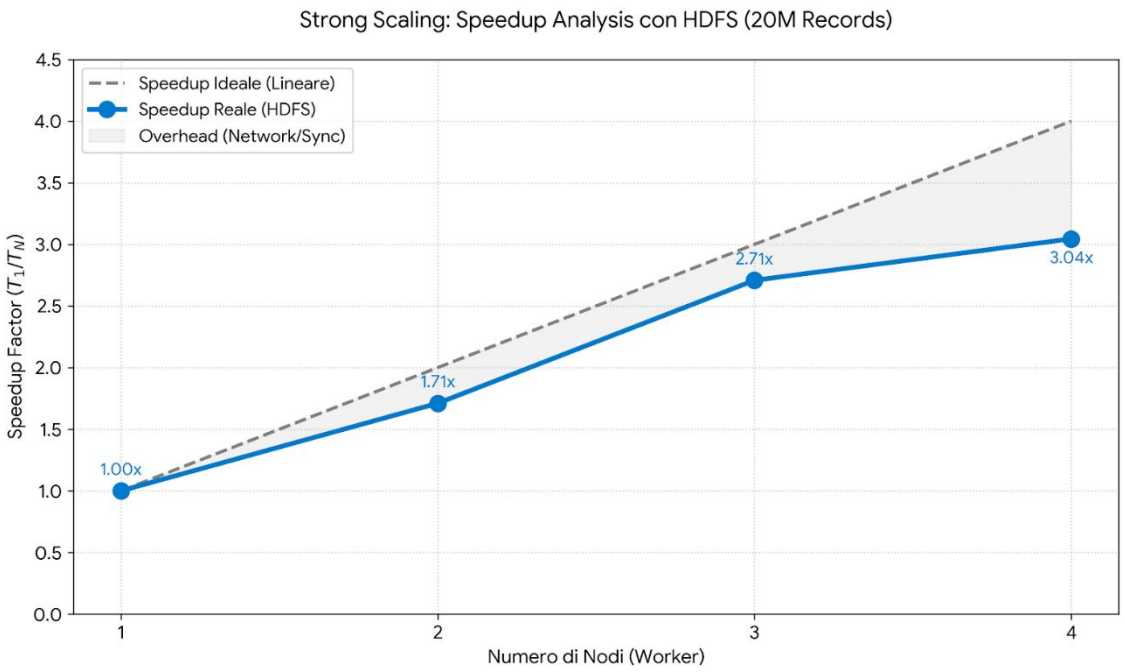
Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20260126140711-0012	Analysis_Pipeline_Parquet_Optimized	4	10.0 GiB		2026/01/26 14:07:11	Colomboadmin	FINISHED	4.1 min
app-20260126140132-0011	Analysis_Pipeline_Parquet_Optimized	8	10.0 GiB		2026/01/26 14:01:32	Colomboadmin	FINISHED	2.4 min
app-20260126135904-0010	Analysis_Pipeline_Parquet_Optimized	12	10.0 GiB		2026/01/26 13:59:04	Colomboadmin	FINISHED	1.5 min
app-20260126135616-0009	Analysis_Pipeline_Parquet_Optimized	16	10.0 GiB		2026/01/26 13:56:16	Colomboadmin	FINISHED	1.3 min

5.3 Analisi dati ottenuti

Per valutare la scalabilità, sono stati calcolati lo **Speedup** (Tbase / TN) e l'**Efficienza** (Speedup / N) focalizzandosi sul tempo totale.

Configurazione	Spedup ideale	Speedup reale	Efficienza	Totale (s)
1 Nodo	1x	1x	100%	243.96
2 Nodi	2x	1.71x	86%	142.63
3 Nodi	3x	2.71x	90%	90.08
4 Nodi	4x	3.04x	76%	80.12

I dati sperimentali mostrano una notevole capacità di scaling del sistema, con una riduzione del tempo di esecuzione complessivo del 67% nel passaggio dalla configurazione a singolo nodo a quella a quattro nodi. L'andamento dello speedup reale segue fedelmente la curva ideale fino ai tre nodi, dove si registra il picco di efficienza parallela pari all'85%. Sebbene l'aggiunta del quarto nodo garantisca il tempo di esecuzione assoluto più basso (80 secondi), si osserva una flessione dell'efficienza che scende al 76%, indicando che il guadagno marginale ottenuto dall'aumento delle risorse computazionali inizia a ridursi leggermente rispetto alle configurazioni precedenti.


Strong Scaling: Speedup Analysis con HDFS (20M Records)
Speedup Ideale (Lineare)
Speedup Reale (HDFS)
Overhead (Network/Sync)
Speedup Factor (T_1/T_N)
Numero di Nodi (Worker)
1.00x
1.71x
2.71x
3.04x
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
1
2
3
4

6. Analisi del Sentiment

Oltre alla valutazione prestazionale dell'infrastruttura, il progetto ha previsto un modulo di Big Data Analytics finalizzato all'estrazione di valore informativo dal Data Lake. L'obiettivo è trasformare i dati grezzi in dati utili per il business.

Per massimizzare la profondità dell'analisi, è stata progettata un'architettura a due livelli di granularità, implementata tramite due pipeline Spark distinte ma complementari:

1. Analisi aggregata su tutto il dataset per identificare tendenze trasversali e problematiche comuni.
2. Analisi partizionata per singola applicazione, per identificare le specificità di ogni prodotto.

6.1 Preprocessing e NLP Pipeline

Entrambi i livelli di analisi condividono una pipeline di Natural Language Processing (NLP) comune, definita per pulire il rumore e normalizzare il testo.

La pipeline è composta dai seguenti stage:

1. Text Normalization: Conversione in minuscolo, rimozione di caratteri non alfanumerici e rimozione degli spazi bianchi multipli.
2. Tokenization: Segmentazione delle frasi in singole parole.
3. Rimozione delle stopwords: È stata definita una Custom StopWords List per rimuovere il rumore di fondo. Oltre alle classiche stopwords inglesi come articoli e preposizioni, sono stati rimossi:
    - Nomi delle App: la loro presenza falserebbe la frequenza, essendo ovvio che appaiano nelle recensioni della loro stessa app;
    - Termini Generici di Dominio: L'obiettivo è far emergere solo i concetti qualitativi;

- N-Gram Extraction: Invece di analizzare singole parole, si è scelto di estrarre Bigrammi coppie di parole consecutive. Questo è fondamentale per catturare il contesto del sentiment, inquanto ad esempio la parola "login" da sola potrebbe portare ad un’interpretazione ambigua, mentre le parole "login error" o "easy login" chiariscono meglio il sentiment che vi è dietro la recensione.

6.2 Script 1: Analisi aggregata su tutto il dataset

Il primo script tratta il Data Lake come un unico corpus indistinto. Questa vista permette di rispondere a domande di livello infrastrutturale o di piattaforma, indipendenti dalla specifica applicazione. Ad esempio, analizzando i bigrammi più frequenti associati al Rating 1 stella su tutto il dataset, emergono pattern di insoddisfazione trasversali. Questo livello di analisi è utile per gli store manager, come google play o app store, per capire il sentiment generale che ha il pubblico riguardo le applicazioni presenti in store.

6.3 Script 2: Analisi partizionata per singola applicazione

Il secondo script introduce la dimensione dell'Application Name, permettendo una diagnosi mirata. Dal punto di vista tecnico, la sfida di parallelismo è stata risolta utilizzando le Window Functions di Spark SQL, che permettono di calcolare classifiche locali all'interno di partizioni logiche.

La logica implementata è la seguente:

1. Raggruppamento: Conteggio dei bigrammi per App e per Rating.
2. Window Partitioning: Definizione di una finestra logica per ogni applicazione.
3. Ranking: Estrazione dei Top 10 bigrammi per ogni partizione.

Questo approccio risolve il problema che si ha in un'analisi globale, in quanto le app con milioni di recensioni oscurerebbero le app più piccole. Con le Window Function, ogni app ha la sua classifica dedicata. Ciò permette di distinguere la natura dei problemi.

6.4 Analisi dei risultati

Per validare l'efficacia del Data Lake, l'analisi dei risultati è stata condotta seguendo un approccio architetturale a doppio livello di granularità. Questa metodologia ha permesso di distinguere tra i problemi sistemici della piattaforma e le criticità specifiche del singolo software.

A. Livello Macro: Global Ecosystem Analysis

In questa fase, l'intero dataset è stato trattato come un unico corpus per identificare le tendenze trasversali. L'analisi frequenziale dei bigrammi (Rating = 1) ha isolato tre cluster di criticità universali:

- Cluster 1: Technical Failure
    - Keywords: "doesn work" (52.8k), "won let" (32.3k), "keeps crashing";
    - La stabilità tecnica rappresenta oltre il 40% delle cause di recensione negativa o abbandono, indipendentemente dal tipo di app.
- Cluster 2: Customer Support Gap
    - Keywords: "customer service" (29.3k), "support team".
    - Gli utenti ricorrono alle recensioni pubbliche come canale per segnalare l'inefficacia dei canali di supporto tradizionali.
- Cluster 3: Dark Patterns & Billing
    - Keywords: "cancel subscription" (15.9k), "free trial".
    - Forte insofferenza verso modelli di monetizzazione poco trasparenti come rinnovi automatici non voluti.

B. Livello Micro: Drill-Down Application Analysis

Utilizzando le Window Functions per partizionare l'analisi, è stato possibile confutare l'ipotesi che non tutte le app riportano gli stessi problemi. Il confronto tra categorie rivela driver di insoddisfazione radicalmente diversi.

Di seguito vengono riportati alcuni esempi:

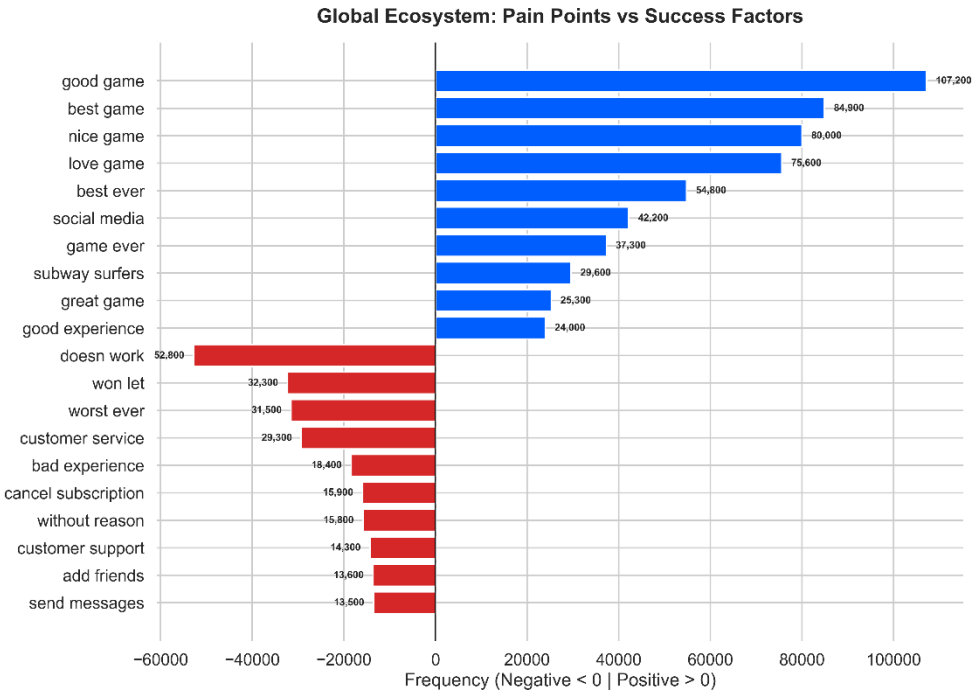
Categoria App	**Nome**	**Top negative**	**Causa identificata**
Produttività	Dropbox	“cancel subscription”	Frustrazione per le policy di rinnovo e trial.
Gaming	Candi crush	“gold bars”, “next level”	Difficoltà artificiale per indurre microtransazioni
Social	Facebook	“keeps crashing”	Pesantezza dell'app e instabilità su dispositivo datati
Produttività	Ms Word	“fetching files”	Problema nel recupero dati

7. Visualizzazione dei dati

7.1 Analisi Globale del Sentiment

Questo grafico a barre divergenti offre una visione macroscopica dello stato di salute dell'intero ecosistema di app analizzato.

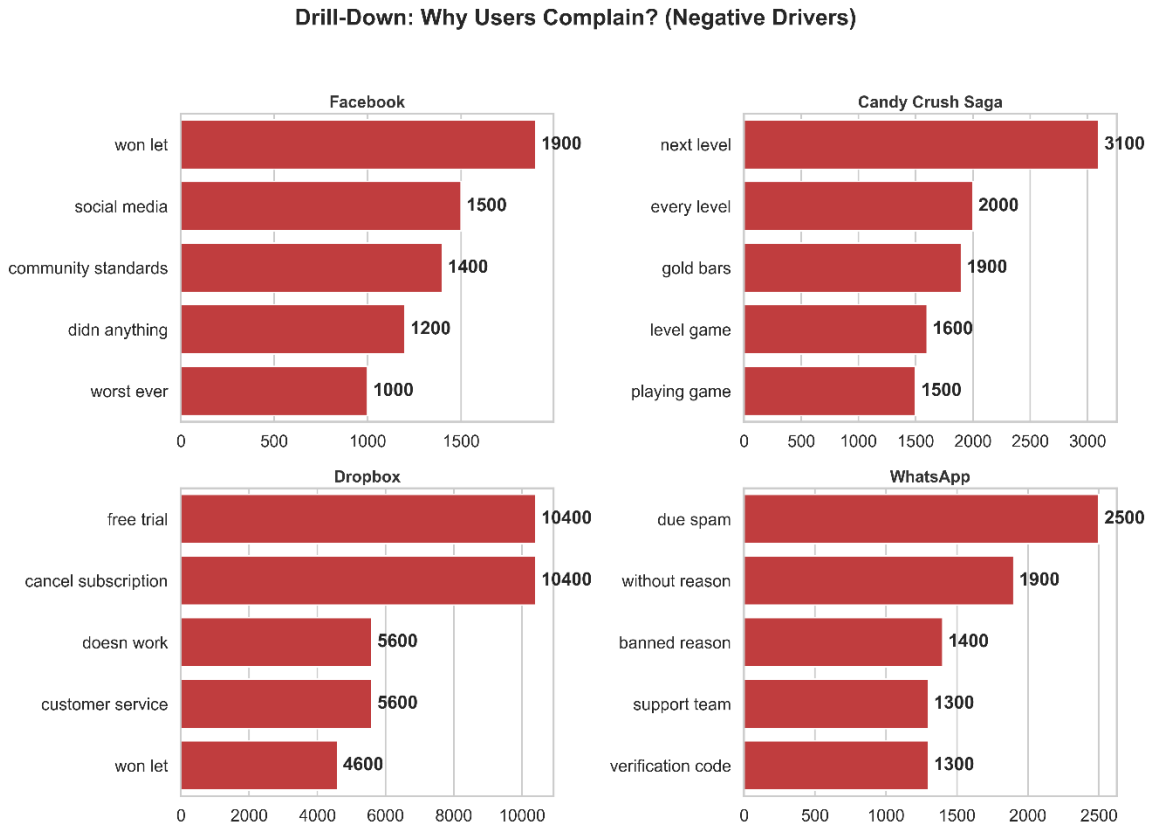
- A sinistra (Rosso - Rating 1): Sono visualizzati i principali driver di insoddisfazione. Si nota immediatamente la predominanza di problemi tecnici trasversali: il bigramma "doesn work" (52.8k), "won let" e "customer service".
- A destra (Azzurro - Rating 5): Sono visualizzati i driver di successo. I termini "good game", "best game" e "love game" saturano la classifica positiva, suggerendo che l'esperienza ludica genera un volume di feedback entusiasti nettamente superiore rispetto alle app di utilità o i social media.



7.2 Driver di Insoddisfazione per singola app

Le cause di fallimento sono specifiche per dominio. Di seguito vengono riportati alcuni esempi:

- Candy Crush Saga (Gaming): Gli utenti non lamentano bug, ma criticano il Game Design. Termini come "next level" e "gold bars" indicano frustrazione per la difficoltà artificiale mirata alla monetizzazione;
- Dropbox (Productivity): I bigrammi "cancel subscription" e "free trial" evidenziano la percezione di problematiche nei rinnovi automatici;
- Facebook (Social): Termini come "keeps crashing" e "won let" riflettono la pesantezza dell'app.
- WhatsApp (Communication): Parole come "Verification code" segnalano un collo di bottiglia specifico nel processo di autenticazione SMS.

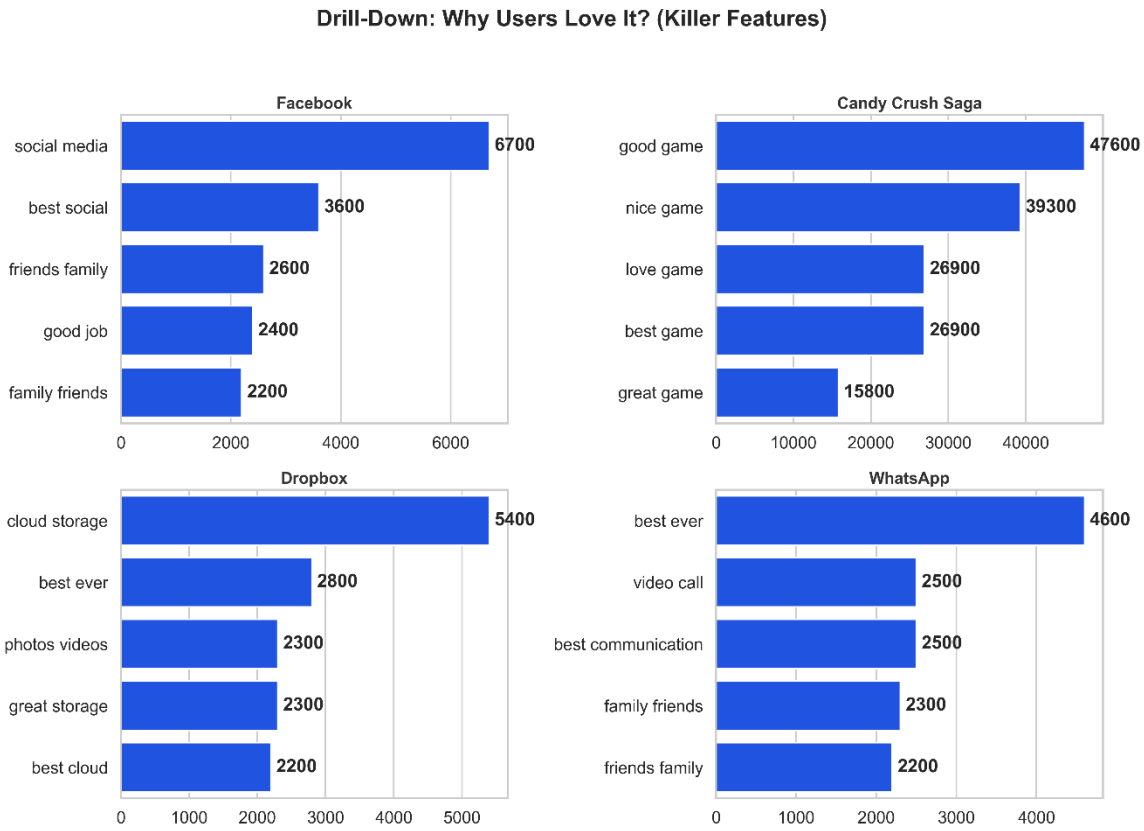


7.3 Fattori di Successo per singola app

Speculare alla precedente, questa vista isola le killer features percepite dagli utenti felici.

Di seguito alcuni esempi:

- Candy Crush Saga: L'apprezzamento è puramente emozionale/ludico;
- Dropbox: Il valore percepito è funzionale. "Cloud storage" e "photos videos" indicano che l'app è apprezzata principalmente come archivio sicuro;
- Facebook: Nonostante le critiche tecniche, vince la rete sociale, "Friends family" e "connect friends" lo confermano come un ottimo social network;
- WhatsApp: La killer feature è la semplicità di comunicazione visuale. "Video call" e "easy use" sono i driver principali, suggerendo che la qualità delle chiamate e la facilità d'uso sono i fattori di maggior gradimento.



8. Conclusioni

Questo progetto ha permesso di progettare, implementare e validare un’architettura Big Data basata su Apache Spark e HDFS, dimostrando concretamente come le moderne tecnologie distribuite siano in grado di trasformare grandi moli di dati grezzi in valore informativo tangibile per il business.

Dal punto di vista infrastrutturale, i test condotti sul dataset aumentato a venti milioni di record hanno confermato la piena efficacia del paradigma Scale-Out nell'affrontare carichi di lavoro onerosi. L'analisi di benchmarking ha evidenziato una scalabilità solida, con il cluster che ha raggiunto uno speedup superiore a 3x nella configurazione a quattro nodi, abbattendo drasticamente i tempi di esecuzione del 67% rispetto alla configurazione a singolo nodo. Particolarmente interessante è emerso il ruolo della configurazione a tre nodi, che si è rivelata il punto di ottimo tra costi infrastrutturali e benefici prestazionali, mantenendo un'efficienza del 90%. Al contrario, l'aggiunta del quarto nodo ha mostrato i primi, fisiologici segni di saturazione, confermando come i costi di coordinamento e la latenza di rete inizino a incidere marginalmente quando il tempo di calcolo puro si riduce troppo. Inoltre, l'integrazione di un vero file system distribuito come HDFS, configurato con un fattore di replicazione pari a tre, si è rivelata determinante per garantire la tolleranza ai guasti.

L'analisi globale ha fatto emergere come cause sistemiche di malcontento degli utenti: malfunzionamenti tecnici generici e l'insoddisfazione verso il supporto clienti, identificando tendenze trasversali a tutte le categorie. Distinguendo poi tra le diverse applicazioni, è stato possibile diagnosticare problemi radicalmente diversi nonostante il medesimo punteggio negativo o positivo.

In definitiva, l'elaborato dimostra che l'utilizzo di framework come Spark, abbinati a storage distribuito e formati colonnari ottimizzati come Parquet, rappresenta una soluzione industriale estremamente valida.