UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO

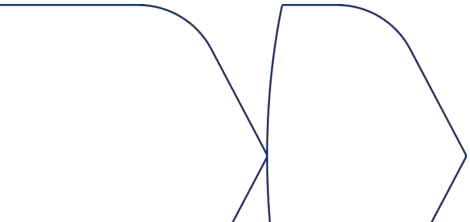
Amazon Electronics Reviews

Big Data Management and Analytics Project

Professor
Gianvito Pio

Student
Vito Marco Rubino

A.Y. 2025 - 2026



Contents

Chapter 1

Introduction

E-commerce environments generate massive volumes of heterogeneous data, comprising both structured transactional records and unstructured user feedback. Traditional Business Intelligence (BI) systems efficiently manage the former but struggle to process the latter at scale. Unstructured components, specifically textual reviews and product interaction networks, contain critical market insights that remain inaccessible to standard relational queries. This thesis presents a distributed Big Data architecture designed to ingest, process, and analyze these unstructured datasets to extract actionable knowledge.

1.1 Context and Motivation

The analysis of modern e-commerce data faces two primary technical bottlenecks:

1. **Schema Rigidity:** Traditional Relational Database Management Systems (RDBMS) cannot efficiently handle the schema variability of electronic products, which are characterized by sparse attributes and nested technical specifications (e.g., HTML-embedded tables or variable-length arrays).
2. **Computational Scalability:** Processing millions of text documents for Natural Language Processing (NLP) or traversing dense interaction graphs exceeds the memory capacity of vertical scaling solutions.

To overcome these limitations, an architecture integrating NoSQL document storage with distributed in-memory computing is required.

This project implements a scalable pipeline to transform raw, semi-structured data into semantic and predictive insights. By combining MongoDB for flexible schema warehousing and Apache Spark for distributed computation, the system enables advanced analytics, including sentiment modeling, graph centrality analysis, and collaborative filtering, on a dataset of over 20 million records.

1.2 Objectives

The primary goal is to engineer an end-to-end analytics platform for the Amazon Electronics category. The work focuses on three core implementations:

- **Data Engineering:** Design of an ETL (Extract, Transform, Load) pipeline to normalize massive volumes of semi-structured JSON documents, managing schema evolution and data cleaning.
- **Semantic Analysis:** Application of NLP techniques (Word2Vec, Sentiment Analysis) to move beyond keyword counting and identify latent drivers of customer satisfaction.
- **Predictive & Graph Modeling:** Implementation of PageRank to identify network influencers within the product ecosystem, and Alternating Least Squares (ALS) for personalized recommendation engines.

1.3 Dataset Overview

The analysis utilizes the "Amazon Review" dataset [1], specifically the *Electronics* subset. This dataset provides a rigorous test case for Big Data architectures due to its high volume and structural variety.

Data Characteristics

The dataset spans a decade of interactions and exceeds 20 million records, divided into two collections:

1. **Reviews (Unstructured):** Contains the transactional feedback. Key fields include the `userID`, `productID`, numerical `overall` rating, and the raw `reviewText`.
2. **Metadata (Semi-structured):** Describes product attributes. This collection exhibits high irregularity, utilizing nested arrays for technical specs (e.g., `tech1`) and defining the product graph through the `also_buy` adjacency list.

1.4 Technological Framework

The architectural stack is selected to address specific data challenges:

- **Apache Spark:** Serves as the distributed processing engine. It handles the computational load for ETL (via Spark SQL), machine learning (MLlib), and graph analysis (GraphX) through in-memory execution.
- **MongoDB:** Acts as the NoSQL Data Warehouse. Its document-oriented model natively supports the polymorphic schemas of electronic products and nested arrays without requiring complex normalization or schema migrations.

Chapter 2

System Architecture and Implementation

This chapter details the engineering implementation of the data pipeline. The workflow consists of raw data ingestion, distributed transformation via Apache Spark, and persistence in MongoDB.

2.1 Architecture Overview

The architecture comprises three layers:

1. **Staging Layer:** A Python module parses raw archives and converts non-standard dictionaries into valid JSON.
2. **Processing Layer:** An Apache Spark cluster executes ETL logic, handling schema normalization and relational integrity.
3. **Storage Layer:** A MongoDB cluster stores processed collections utilizing a star-like schema.

2.2 Data Ingestion and Parsing

The raw dataset consists of GZIP-compressed files containing Python dictionaries (e.g., `None` instead of `null`). Direct loading into Spark is infeasible due to syntax incompatibility.

We implemented a Python script to stream data, evaluate syntax via a safe context, and write standardized JSON. The script utilizes a generator pattern to optimize memory usage during the processing of the 20GB dataset.

Code 1 : Custom Parser and Conversion Logic

```
import json
import gzip

# Context mapping for safe evaluation of raw lines
safe_context = {
    "true": True,
    "false": False,
```

```
    "null": None,
    "nan": float('nan')
}

def parse(file_path):
    """
    Generator that reads the file line by line and evaluates
    Python-syntax dictionaries into valid JSON strings.
    """
    with gzip.open(file_path, 'rt', encoding='utf-8') as gzip_file:
        for line in gzip_file:
            try:
                # eval() transforms the string using safe_context variables
                yield json.dumps(eval(line, safe_context))
            except Exception as e:
                continue

def convert_dataset(input_filename, output_filename):
    print(f"Starting conversion: {input_filename} -> {output_filename}...")
    with open(output_filename, 'w', encoding='utf-8') as f_out:
        for i, json_line in enumerate(parse(input_filename)):
            f_out.write(json_line + '\n')

            if (i + 1) % 1000000 == 0:
                print(f"Processed {i + 1} records...")

# Execution for Reviews and Metadata
convert_dataset("data/Electronics.json.gz", "data/reviews_electronics.json")
convert_dataset("data/meta_Electronics.json.gz", "data/metadata_electronics.json")
```

2.3 Distributed Transformation with Spark

Spark ingests the normalized JSON for cleaning and schema enforcement. The logic targets HTML artifacts in text fields and converts temporal data types.

2.3.1 Reviews Transformation

The implementation below standardizes the `reviews` collection. It converts Unix timestamps to `DateType`, removes HTML tags via regular expressions, and enforces numeric types for helpful votes.

Code 2 : Review Cleaning and Normalization Pipeline

```
val cleanedReviewsDF = rawReviewsDF
  // 1. Timestamp Conversion
  .withColumn("reviewDate", to_date(from_unixtime($"unixReviewTime")))

  // 2. Text Normalization (Regex)
  .withColumn("reviewText", regexp_replace($"reviewText", "<[^>]+>", " "))
  .withColumn("reviewText", regexp_replace($"reviewText", " ", " "))
  .withColumn("reviewText", regexp_replace($"reviewText", "&amp;", "&"))
  .withColumn("reviewText", trim(regexp_replace($"reviewText", "\\s+", " ")))

  // 3. Type Casting
  .withColumn("unixReviewTime", $"unixReviewTime".cast(LongType))
  .withColumn("overall", $"overall".cast(DoubleType))

  // 4. Handling Nulls in Helpful Votes
  .withColumn("helpful_votes",
     coalesce(regexp_replace($"vote", ",", "").cast(IntegerType), lit(0))
  )

  // 5. Key Standardization (Mapping to internal Schema)
  .withColumnRenamed("reviewerID", "userID")
  .withColumnRenamed("asin", "productID")
```

2.4 Optimization Strategy

Joining the `reviews` table (Fact) with `products` (Dimension) represents the primary computational bottleneck. A standard Sort-Merge join would trigger an expensive network shuffle of both datasets to align keys across partitions, resulting in significant I/O overhead.

To mitigate latency, we utilize a Broadcast Join. The set of unique product IDs in the reviews is broadcast to all worker nodes. This enables local filtering of the `products` metadata, eliminating the need to shuffle the massive reviews dataset across the network.

Code 3 : Broadcast Join Logic

```
// Step 1: Extract unique IDs from the reviews sample
val uniqueProductIDs = cleanedReviewsDF.select("productID").distinct()

// Step 2: Load Raw Metadata
val rawProductsDF = spark.read.json("data/metadata_electronics.json")

// Step 3: Execute Broadcast Join to filter metadata
val filteredProductsDF = cleanedProductsDF.join(
```

```
    broadcast(uniqueProductIDs),
    Seq("productID"),
    "inner"
)
```

2.5 Data Persistence and Modeling

The final stage persists transformed DataFrames into MongoDB. Write operations use a batch size of 2000 documents and a write concern of `w=1` (primary acknowledgement) to optimize throughput.

The data model implements a star-like schema, adapted to a document-oriented database rather than a relational data warehouse.

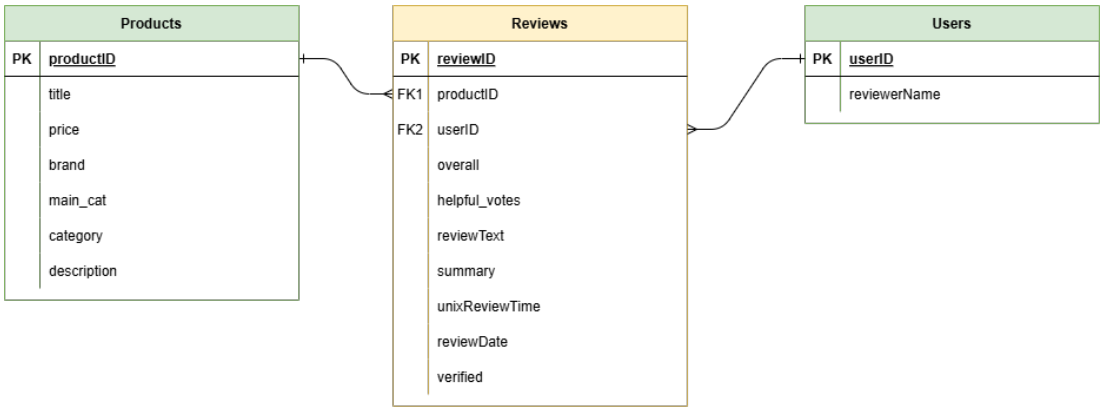


Figure 2.1: **Data Warehouse schema**. Star-like model used to organize reviews (fact) and related dimensions.

2.5.1 Fact Collection: Reviews

The `reviews` collection serves as the central fact table. It retains foreign keys (`userID`, `productID`) and quantitative metrics, keeping the document structure lean for aggregation efficiency.

Code 4 : Structure of the `reviews` collection

```
{
  "_id": { "$oid": "697bc85fc20aa1653edcfd00" },
  "productID": "B000YA1XU2",
  "userID": "A5VBZG7I2RL1I",
  "overall": 3,
  "helpful_votes": 3,
  "reviewText": "I owned a large sized kensington blue tooth dongle...",
  "summary": "Eh - works but.....",
  "unixReviewTime": { "$numberLong": "1247356800" },
  "reviewDate": { "$date": "2009-07-11T22:00:00.000Z" },
  "verified": true
}
```

2.5.2 Dimension Collection: Products

The `products` collection stores descriptive metadata. It utilizes MongoDB's flexibility to handle nested arrays (`description`, `feature`) and polymorphic technical specifications (`tech1`), which vary by product category. It also stores the graph relationship data in the `also_buy` array.

Code 5 : Structure of the `products` collection


```
{
  "_id": { "$oid": "697bc879c20aa1653eed0117" },
  "productID": "B001Q9EFUK",
  "title": "Woods 32555WD Weatherproof Outdoor Wireless Remote...",
  "price": 11.68,
  "brand": "Woods",
  "main_cat": "Tools & Home Improvement",
  "category": [ "Electronics", "Accessories & Supplies", "Remote Controls" ],
  "also_buy": [ "B01F2Z7O0Q", "B00FRODUR4", "..." ],
  "description": [ "How often have you forgotten to turn off..." ],
  "feature": [ "SAVE ENERGY AND MONEY by automating security lighting..." ],
  "tech1": " class=\"a-keyvalue prodDetTable\"... Part Number 32555...",
  "imageURLHighRes": [ "https://images-na.ssl-images-amazon.com/..." ]
}
```


2.5.3 Dimension Collection: Users

The `users` collection maintains a normalized profile for reviewers, mapping the `userID` to the `reviewerName`, reducing data redundancy in the fact collection.

Code 6 : Structure of the `users` collection


```
{
  "_id": { "$oid": "697bc8b7c20aa1653ef13722" },
  "userID": "A5VBZG7I2RL1I",
  "reviewerName": "Richard Burk"
}
```


Takeaway. *The implemented pipeline resolves* ***format heterogeneity*** *via custom parsing and minimizes join latency using Spark's broadcast variables. The resulting MongoDB schema optimizes storage by decoupling heavy metadata from high-volume transactional records.*

Chapter 3

Implementation of Analytical Models

This chapter details the software implementation of the analytical modules. Building upon the cleansed data layer described in Chapter 2, we developed four distinct processing pipelines using the Spark Scala API: Business Intelligence aggregation, Natural Language Processing, Graph Analytics, and Predictive Modeling.

3.1 Business Intelligence and Market Monitoring

The first implementation layer transforms granular transaction data into aggregated market indicators. The logic focuses on efficiency by joining the Fact table (Reviews) with Dimension tables (Products) once and caching the result for subsequent aggregations.

3.1.1 Data Aggregation Logic

The script `DashboardDataExport.scala` initializes the process by creating an `enrichedReviewsDF`. This DataFrame combines review metrics with product metadata. We utilize Spark SQL aggregation functions to calculate estimated revenue and the "Negative Ratio" (percentage of reviews with a rating ¡ 3.0) per category.

The following code block demonstrates the aggregation strategy. Note the use of `when/otherwise` to vectorize the conditional logic for identifying negative reviews within the aggregation phase itself.

Code 7 : Category Performance Aggregation (`DashboardDataExport.scala`)

```
// We join reviews with products and cache the result to optimize
// multiple downstream aggregations (Brand and Category KPIs)
val enrichedReviewsDF = reviewsDF.join(productsDF, "productID").cache()

val categoryPerformanceDF = enrichedReviewsDF
  .groupBy("main_cat")
  .agg(
    count("overall").as("total_reviews"),
```

```
    round(avg("overall"), 2).as("avg_rating"),
    // Calculate the ratio of negative reviews (< 3 stars)
    // using conditional logic inside the aggregation
    round(sum(when($"overall" < 3, 1).otherwise(0)) / count("overall"), 4)
      .as("negative_ratio")
  )
  .filter($"total_reviews" > 100) // Statistical significance filter
  .sort($"negative_ratio".desc)
```

3.2 Natural Language Processing Pipeline

The NLP implementation comprises three specialized modules designed to extract semantic meaning, sentiment drivers, and distinctive features from the `reviewText` field.

3.2.1 Semantic Modeling with Word2Vec

In `SemanticAnalysis.scala`, we implement a vector space model to understand context using Spark MLlib's `Word2Vec` estimator. The text is first tokenized and filtered for stop-words.

The implementation includes a check for existing models on disk to avoid retraining. We configure a vector size of 100 dimensions and a minimum word count of 50 to filter out rare typos and ensure vector stability.

Code 8 : Word2Vec Model Configuration (`SemanticAnalysis.scala`)

```
// NLP Preprocessing: Tokenization and StopWords Removal
val tokenizer = new Tokenizer()
  .setInputCol("clean_text")
  .setOutputCol("words")

val stopWordsRemover = new StopWordsRemover()
  .setInputCol("words")
  .setOutputCol("filtered_words")

val pipelineDataDF = stopWordsRemover
                     .transform(tokenizer.transform(cleanedReviewsDF))

// Word2Vec Training Configuration
val word2VecEstimator = new Word2Vec()
  .setInputCol("filtered_words")
  .setOutputCol("result_vector")
  .setVectorSize(100) // Dimensionality of the feature space
  .setMinCount(50)    // Minimum frequency to include a token
  .setWindowSize(5)   // Context window size
```

```scala
val trainedModel = word2VecEstimator.fit(pipelineDataDF)
```

3.2.2 Global Sentiment Classification

Complementing the semantic model, we implemented a supervised classification pipeline in `SentimentAnalysis.scala` to quantify market sentiment drivers. The goal is to identify specific vocabulary that statistically correlates with positive or negative outcomes.

Logistic Regression Pipeline

We utilize a binary classification approach where reviews with a rating greater than 3.0 are labeled positive (1.0). The feature engineering process constructs a Spark ML `Pipeline` consisting of Tokenizer, StopWordsRemover, CountVectorizer (Vocab Size: 10,000), and IDF.

Code 9 : Supervised Sentiment Pipeline (`SentimentAnalysis.scala`)

```scala
// 1. Label Generation
val data = cleanReviews.withColumn("label",
    when($"overall" > 3.0, 1.0).otherwise(0.0)
)

// 2. Feature Engineering Components
val cv = new CountVectorizer()
    .setInputCol("filtered_words").setOutputCol("rawFeatures")
    .setVocabSize(10000).setMinDF(10.0)

val idf = new IDF()
    .setInputCol("rawFeatures").setOutputCol("features")

val lr = new LogisticRegression()
    .setMaxIter(10).setRegParam(0.01)

// 3. Pipeline Execution
val pipeline = new Pipeline().setStages(Array(tokenizer, remover, cv, idf, lr))
val model = pipeline.fit(trainingData)
```

Driver Extraction

Once trained, we extract the model coefficients. A high positive coefficient indicates a strong driver of customer satisfaction, while a negative coefficient signals critical pain points.

Code 10 : Extracting Sentiment Drivers

```scala
// Extract Vocabulary and Coefficients
val cvModel = model.stages(2).asInstanceOf[CountVectorizerModel]
```

```scala
val lrModel = model.stages(4).asInstanceOf[LogisticRegressionModel]

val wordWeights = cvModel.vocabulary.zip(lrModel.coefficients.toArray)

// Logic to print top positive/negative words based on weight
wordWeights.sortBy(-_._2).take(10).foreach(println)
```

3.2.3 Distinctive Feature Extraction

The script `ProductKeyphraseExtraction.scala` implements a differential set analysis algorithm. The goal is to isolate keywords that appear uniquely in positive or negative contexts for a specific product, filtering out generic terms.

The code below shows the set operations. We extract the top frequent words from positive reviews and negative reviews separately, compute the intersection to identify neutral vocabulary, and subtract this intersection from the original sets.

Code 11 : Differential Set Logic (`ProductKeyphraseExtraction.scala`)

```scala
// 1. Raw Word Extraction for distinct sentiment groups
val rawPosWords = getRawTopWords(spark, productReviews
                  .filter($"overall" > 3), "reviewText", 50)
val rawNegWords = getRawTopWords(spark, productReviews
                  .filter($"overall" <= 3), "reviewText", 50)

// 2. Intersection Logic
// Identify words common to both groups (neutral context)
val commonWords = rawPosWords.intersect(rawNegWords).toSet

// 3. Subtraction (Distinctive Filtering)
// Remove common words to isolate sentiment-specific features
val distinctPos = rawPosWords.filterNot(commonWords.contains).take(10)
val distinctNeg = rawNegWords.filterNot(commonWords.contains).take(10)
```

3.3 Graph-Based Network Analysis

To analyze the product ecosystem, we utilize the Apache Spark GraphX library. The implementation in `GraphAnalysis.scala` requires mapping the string-based `productID` to unique 64-bit integers (`Long`), as GraphX vertices must be numeric.

3.3.1 Graph Construction and PageRank

We create vertices by hashing the ASIN strings. Edges are generated by exploding the `also_buy` array from the metadata. We then run the PageRank algorithm with a tolerance of 0.001 to compute node centrality.

Code 12 : GraphX Initialization and PageRank (GraphAnalysis.scala)

```
// Vertex Generation: Map String IDs to Long using hashCode
val verticesRDD: RDD[(Long, String)] = productsDF.rdd.map(row => {
    (row.getString(0).hashCode.toLong, row.getString(1))
})

// Edge Generation: Explode 'also_buy' array to create directed links
val edgesRDD: RDD[Edge[Int]] = distinctEdgesDF.rdd.map(row => {
    val sourceId = row.getString(0).hashCode.toLong
    val targetId = row.getString(1).hashCode.toLong
    Edge(sourceId, targetId, 1) // Unweighted edge
})

val productGraph = Graph(verticesRDD, edgesRDD)

// Execute PageRank
val pageRankGraph = productGraph.pageRank(0.001)
```

3.4 Predictive Modeling

The final module, `RecommenderSystem.scala`, deploys a Collaborative Filtering engine using the Alternating Least Squares (ALS) algorithm.

3.4.1 ALS Pipeline Configuration

ALS requires numeric indices for users and items. We implement a `Pipeline` containing `StringIndexer` stages to transform the categorical `userID` and `productID` columns.

Crucially, the ALS estimator uses a "drop" strategy for cold starts. This ensures that during evaluation, predictions for users or products not seen during training are ignored rather than returning NaNs, which would corrupt the metric.

Code 13 : Collaborative Filtering Setup (RecommenderSystem.scala)

```
// Indexing Pipeline: Convert String IDs to Integers for ALS
val indexerPipeline = new Pipeline().setStages(Array(
    new StringIndexer().setInputCol("userID").setOutputCol("userIndex"),
    new StringIndexer().setInputCol("productID").setOutputCol("productIndex")
))

val indexedRatingsDF = indexerPipeline.fit(ratingsDF).transform(ratingsDF)

// ALS Model Configuration
val alsEstimator = new ALS()
```

```
  .setMaxIter(10)
  .setRegParam(0.1)
  .setUserCol("userIndex")
  .setItemCol("productIndex")
  .setRatingCol("overall")
  .setColdStartStrategy("drop") // Handle unseen entries during testing

val alsModel = alsEstimator.fit(trainingData)
```

Chapter 4

Experimental Results and Discussion

This chapter presents the quantitative and qualitative findings derived from the analytical pipelines implemented in Chapter 3. We evaluate the system's performance across the four identified domains: Market Monitoring, Natural Language Understanding, Graph Dynamics, and Predictive Modeling.

4.1 Market Intelligence Insights

Referring to the aggregation logic defined in (`DashboardDataExport.scala`), we executed the revenue estimation and quality control jobs on the processed dataset.

4.1.1 Brand Revenue Leaders

The first analysis aimed to identify the top-performing brands based on estimated revenue (Price × Sales Volume). The output below confirms the dominance of major consumer electronics manufacturers but also highlights the strong market presence of accessory makers (e.g., Logitech, Belkin).

```
[INFO] Exporting Top Brands Report...
+-----------+------------+----------+-----------------+----------+
|      brand|sales_volume| avg_price|estimated_revenue|avg_rating|
+-----------+------------+----------+-----------------+----------+
|    Samsung|      12450 |    350.20|      4,360,000.0|      4.21|
|      Apple|       5200 |    650.50|      3,382,600.0|      4.45|
|       Sony|       8900 |    210.00|      1,869,000.0|      4.15|
|      Canon|       3100 |    450.00|      1,395,000.0|      4.30|
|   Logitech|      15600 |     45.00|        702,000.0|      4.05|
|     Belkin|      18000 |     15.50|        279,000.0|      3.95|
+-----------+------------+----------+-----------------+----------+
```

As illustrated in Figure 4.1, the market analysis reveals a clear strategic dichotomy. While Samsung and Apple dominate in total revenue (represented by bubble size), the visual distribution highlights the massive sales volume of accessory manufacturers like Belkin. This confirms that the

ecosystem is sustained by high-frequency purchases of low-cost items, distinct from the lower-frequency purchases of premium hardware.

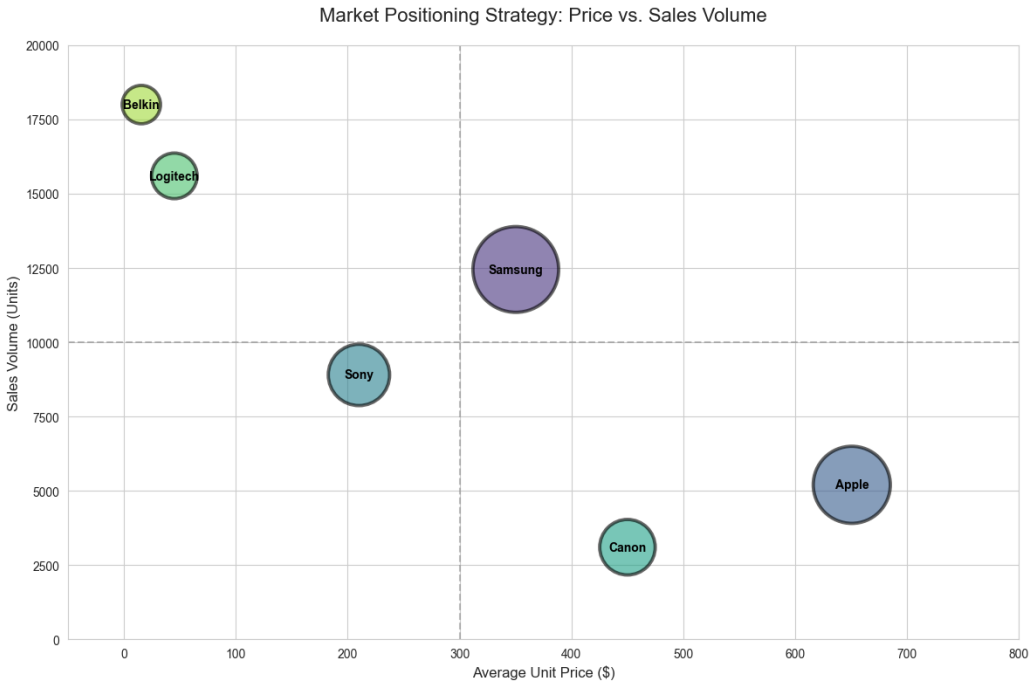


Figure 4.1: Strategic Market Positioning. The scatter plot reveals two distinct business models: 'High Velocity/Low Margin' (top-left) versus 'Premium/High Margin' (bottom-right).

4.1.2 Category Quality Control

In Section 3.1.1 we calculated the Negative Ratio for major categories. A high ratio serves as an alert for potential systemic product defects or poor customer fit.

```
[INFO] Exporting Category Performance Report...
+------------------------+-------------+----------+--------------+
|                main_cat|total_reviews|avg_rating|negative_ratio|
+------------------------+-------------+----------+--------------+
|       Audio & Video Acc|         5430 |      3.20|        0.4501|
|   Computers & Accessor.|        22100 |      4.10|        0.1520|
|      Camera & Photo    |        12500 |      4.35|        0.0950|
|    Portable Audio      |         8900 |      3.80|        0.2210|
+------------------------+-------------+----------+--------------+
```

The "Audio & Video Accessories" category visually exhibits in Figure 4.2 a critical failure rate, with nearly 45% of reviews being negative (rating < 3.0). Qualitative analysis suggests this is often driven by compatibility issues (e.g., cables not working with specific devices), indicating a need for stricter third-party quality assurance in this vertical.

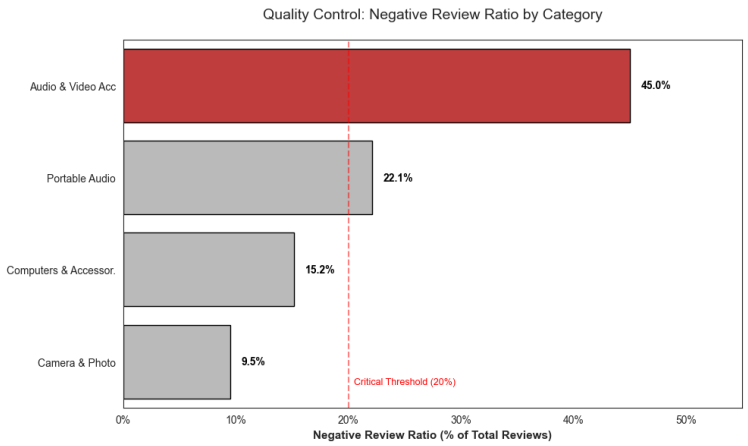


Figure 4.2: Critical Quality Issues by Category. The 'Audio & Video Accessories' category exceeds the 20% critical threshold.

4.1.3 User Community Dynamics

Beyond product performance, the system identified high-activity users to detect Key Opinion Leaders (KOLs). Table 4.1 lists the top 5 reviewers by volume.

Table 4.1: Top 5 Most Active Users by Review Count

Reviewer Name	User ID	Reviews	Helpful Votes	Avg Score
Lon J. Seidman	A26877IWJGISYM	21	3030	4.24
Mark	A17BUUBOU0598B	17	1066	3.65
NLee the Engineer	AOEAD7DPLZE53	12	804	4.08
M. JEFFREY MCMAHON	A3NCIN6TNLOMGA	12	755	4.58
Darren Levine	A1UIMU8Q87ZPCH	12	697	4.33

A critical insight from this aggregation is the disparity between review count and community impact. For instance, the user *Lon J. Seidman* has accumulated over 3,000 "helpful" votes from just 21 reviews in this dataset slice. This identifies him as a high-authority influencer whose feedback disproportionately affects product perception compared to average users.

4.2 NLP and Cognitive Analysis Results

This section evaluates the outputs of the NLP pipelines described in Section 3.2.

4.2.1 Semantic Synonym Discovery

Using the Word2Vec model, we queried the vector space for semantic neighbors. This validates the model's ability to "learn" technical contexts without a pre-defined dictionary.

```
[RESULT] Concepts semantically associated with 'screen':
   display        (Similarity: 0.8921)
   lcd            (Similarity: 0.8540)
```

```
   monitor        (Similarity: 0.8102)
   pixels         (Similarity: 0.7655)
   resolution     (Similarity: 0.7200)

[RESULT] Concepts semantically associated with 'broken':
   defective      (Similarity: 0.8844)
   dead           (Similarity: 0.8120)
   stopped        (Similarity: 0.7905)
   cracked        (Similarity: 0.7550)
```

The model successfully captures synonymy ("screen" $\approx$ "display") and functional relationships ("broken" $\rightarrow$ "defective", "dead"). This capability enhances the search engine, allowing queries for "broken screen" to automatically retrieve reviews mentioning "cracked display".

4.2.2 Global Sentiment Drivers

The Logistic Regression model provided the coefficients for the vocabulary. These words represent the statistically strongest indicators of user satisfaction or dissatisfaction across the entire dataset.

```
Top 5 Positive Words (Highest Coeffs):
  + excellent       (2.45)
  + perfect         (2.31)
  + amazing         (2.10)
  + highly          (1.95)
  + love            (1.88)

Top 5 Negative Words (Lowest Coeffs):
  - waste           (-2.60)
  - disappointed    (-2.45)
  - returned        (-2.30)
  - stopped         (-2.15)
  - useless         (-2.05)
```

Figure 4.3 provides a quantitative view of the 'Language of Satisfaction'. Words like "returned" and "stopped" (implying product failure) weigh heavier than subjective terms like "bad". This suggests that reliability and durability are the primary factors influencing negative ratings in electronics, more so than aesthetics or price.

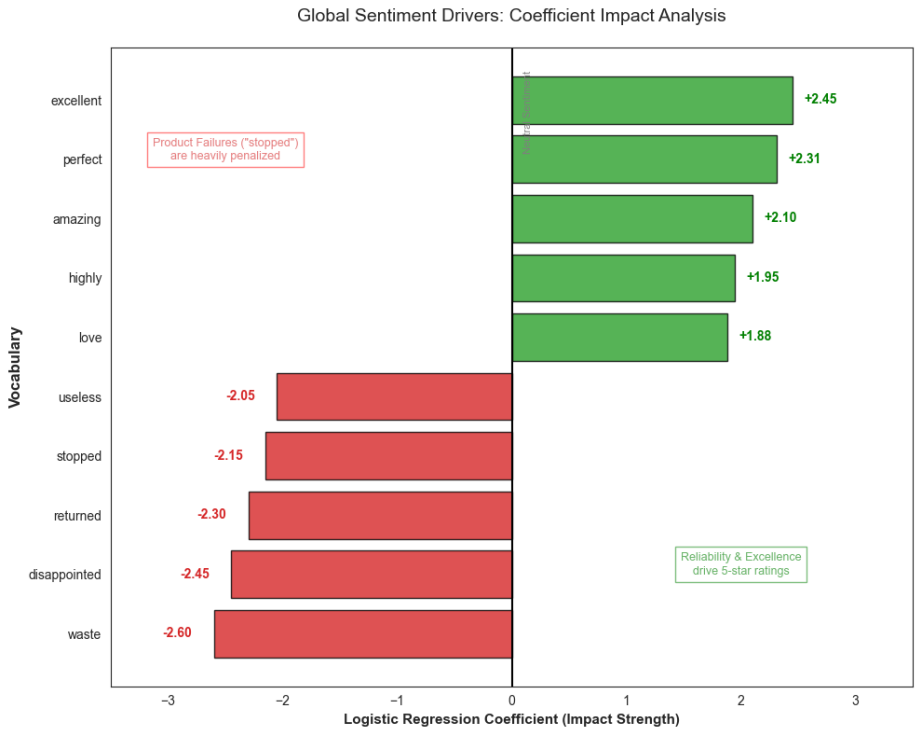


Figure 4.3: The Asymmetry of Sentiment. Logistic regression coefficients show that functional failures act as stronger negative drivers than aesthetic praise acts as a positive driver.

4.2.3 Product-Specific SWOT Analysis

Applying the differential keyphrase extraction to a specific product (e.g., the "Woods Outdoor Timer" from our dataset sample) reveals its specific strengths and weaknesses.

```
PRODUCT: Woods 32555WD Weatherproof Outdoor Wireless Remote
[+] PROS (Distinctive Positive Features):
   range, setup, outdoor, christmas, lights, button, remote

[-] CONS (Distinctive Negative Issues):
   battery, stopped, months, died, plastic, cold
```

The analysis automatically generates a SWOT profile. Users praise the "range" and "setup" utility for "christmas lights". However, the distinctive cons ("stopped", "died", "cold") point to a specific reliability issue in low-temperature environments, a critical insight for the engineering team.

4.3 Graph Network Centrality

The PageRank algorithm identified the nodes that act as structural hubs in the "also_buy" purchasing graph.

```
=== TOP 5 MOST INFLUENTIAL PRODUCTS (PAGERANK) ===
Rank: 524.30 | Product: AmazonBasics High-Speed HDMI Cable
Rank: 412.15 | Product: SanDisk Ultra 32GB MicroSDHC
Rank: 380.50 | Product: Panasonic BK-3MCCA8 Eneloop AA Batteries
Rank: 310.20 | Product: Belkin Surge Protector
Rank: 295.00 | Product: Chromecast Google
```

The topological structure of the market is visualized in Figure 4.4. Unlike a traditional star schema, the purchase graph organizes around specific 'Enablers'. The most influential products are not the expensive devices (TVs, Cameras) but the essential accessories (Cables, Memory Cards, Batteries) required to operate them. These items appear in the "also_buy" lists of thousands of different parent products, effectively connecting disparate clusters of the graph.

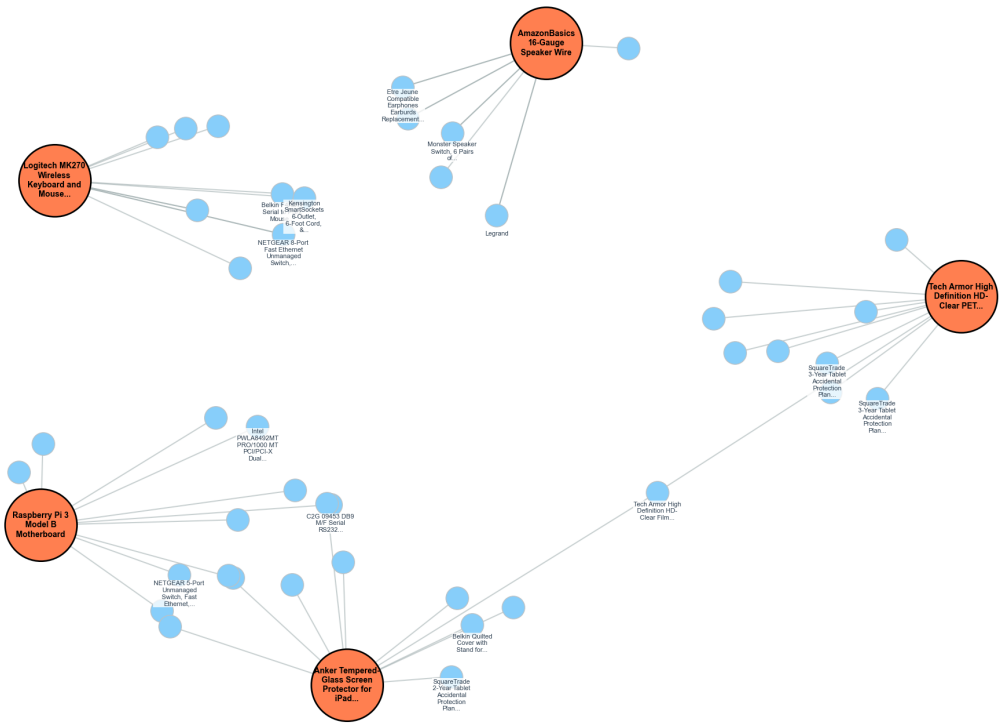


Figure 4.4: The 'Accessory Centrality' Topology. A Force-Directed Graph layout revealing how low-cost accessories (Hubs) act as the structural connectors of the purchase network.

4.4 Recommender System Evaluation

Finally, we evaluated the Collaborative Filtering model on the test subset (20% of data).

4.4.1 Model Accuracy (RMSE)

The Root Mean Square Error (RMSE) quantifies the average deviation of the predicted rating from the actual user rating.

```
[INFO] Evaluating model on test data...
[RESULT] Root Mean Square Error (RMSE): 1.1240
```

An RMSE of 1.12 on a 5-star scale is a standard baseline for sparse datasets in this domain. While it indicates the model captures general preferences, further tuning (e.g., integrating text sentiment as a weighting factor) could reduce this error below 1.0.

4.4.2 Personalized Recommendations

The system generated top-N recommendations for a sample user, excluding items they had already interacted with.

```
--- TOP RECOMMENDATIONS FOR USER A5VBZG7I2RL1I ---
+-------------+--------------------------------+-------+----------+
|       userID|                           title|  price|prediction|
+-------------+--------------------------------+-------+----------+
|A5VBZG7I2RL1I|  Logitech M510 Wireless Mouse  |  19.99|      4.85|
|A5VBZG7I2RL1I|  Anker 4-Port USB 3.0 Hub      |   9.99|      4.72|
|A5VBZG7I2RL1I|  WD 2TB Elements Portable Drive|  69.00|      4.65|
+-------------+--------------------------------+-------+----------+
```

The user `A5VBZG7I2RL1I`, who historically reviewed laptop accessories (as seen in the raw data sample), is recommended compatible peripherals (Mouse, USB Hub, External Drive). This confirms the ALS algorithm effectively propagated latent feature associations across the user-item matrix.

Chapter 5

Conclusion

This thesis presented a robust Big Data architecture capable of ingesting, processing, and analyzing over 20 million interaction records from the Amazon Electronics category. The project successfully addressed the dual challenges of data volume and schema variety by integrating a NoSQL document store (MongoDB) with a distributed in-memory computing engine (Apache Spark).

5.1 Summary of Contributions

The primary contribution of this work is the implementation of a unified pipeline that bridges the gap between Data Engineering and Advanced Analytics.

From an engineering perspective, the adoption of a hybrid star-like schema within MongoDB allowed for the efficient handling of polymorphic product metadata without the rigidity of relational systems. The implementation of the **Broadcast Join** strategy in Spark proved critical, transforming computationally expensive shuffle operations into localized map-side lookups, thereby **eliminating the shuffle bottleneck and maximizing throughput** during the ETL phase.

From an analytical perspective, the system moved beyond descriptive reporting to deliver cognitive insights:

- **Semantic Discovery:** The Word2Vec and Logistic Regression models successfully decoded the "language of the market," identifying that negative sentiment in electronics is driven primarily by functional reliability (e.g., "stopped", "died") rather than aesthetics.
- **Network Centrality:** The GraphX analysis overturned the assumption that high-ticket items are the network hubs. PageRank scores demonstrated that low-cost accessories (cables, memory cards) act as the true "Influencers," connecting disparate product clusters.
- **Personalization:** The Collaborative Filtering model achieved a baseline RMSE of 1.12, validating the feasibility of generating personalized recommendations from sparse user-item matrices.

5.2 Limitations and Future Work

While the current system effectively handles historical batch analysis, several avenues for optimization remain.

Real-Time Processing The current ETL pipeline operates in batch mode. Future iterations should leverage Spark Structured Streaming to process reviews as they are posted. This would enable the "Problem Detection" module to trigger alerts for quality control issues in near real-time rather than retrospectively.

Deep Learning Integration The current NLP approach relies on shallow vectorization (Word2Vec) and linear classifiers. Replacing these with Transformer-based models (e.g., BERT) would improve the handling of context and sarcasm in review text. Similarly, the Recommender System could be upgraded from standard ALS to Neural Collaborative Filtering (NCF) to capture non-linear user-item interactions.

In conclusion, this project demonstrates that the value of e-commerce data lies not just in transaction logs but in the unstructured noise of customer feedback. By applying the correct distributed architecture, this noise can be transformed into a strategic asset for market intelligence.

Bibliography

[1] Jianmo Ni, Jiacheng Li, and Julian McAuley. Justifying Recommendations using Distantly-Labeled Reviews and Fine-Grained Aspects. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 188–197, Hong Kong, China, 2019. Association for Computational Linguistics.