Sentiment Analysis on Amazon Book Reviews using PySpark

Team Members: Nitin Sunil Chaube, Dharani Guntupalli, Akshaya Botla
Course: Big Data

Abstract

With the rapid growth of online reviews, automatic sentiment analysis has become crucial for businesses to understand customer opinions at scale. This project builds a scalable sentiment analysis system using PySpark on the Amazon Book Reviews dataset (approximately 10.3 million reviews). We perform extensive data preprocessing (cleaning text, handling missing values and duplicates, and feature engineering) and convert star ratings into sentiment classes (Positive, Neutral, Negative). Class imbalance is addressed by undersampling the dominant class to improve model training. Two machine learning models – Logistic Regression and Random Forest – are developed using PySpark MLlib pipelines, with text features extracted via TF-IDF and Word2Vec techniques. Hyperparameter tuning is applied to optimize the models. The evaluation on a large test set shows promising accuracy, with Logistic Regression slightly outperforming Random Forest in classifying review sentiments. We demonstrate that scaling up (using powerful hardware) and scaling out (distributed computing across nodes) were essential to process the massive dataset efficiently. Overall, the project highlights how big data tools and machine learning can derive actionable insights from vast collections of user-generated reviews, and we provide recommendations for further improvements.

Introduction

Online consumer reviews have become a rich source of information for both customers and businesses. **Sentiment analysis** – the automated detection of opinions or feelings (positive, negative, neutral) in text – is important because it helps businesses gauge customer satisfaction and product perception at scale. With the rapid increase in online reviews, analyzing customer sentiment has become crucial for companies to monitor brand reputation, improve products, and enhance customer service. However, the volume of data (often millions of reviews) poses a significant challenge for traditional data processing tools.

In this project, we address the problem of large-scale sentiment analysis by leveraging big data technologies. We focus on a case study of Amazon book reviews, using PySpark (the Apache Spark Python API) to handle data processing and model training on a dataset of about 10.3 million book reviews. Our goal is to build a Big Data sentiment analysis system that can classify each review's sentiment (positive, neutral, or negative) based on the review text and star rating. By using PySpark, we can take advantage of distributed computing to process the data and train machine learning models in parallel, demonstrating the ability to **scale up and scale out** the analysis.

The remainder of this report is organized as follows. We first outline the objectives of the project. We then describe the dataset and its characteristics. The methodology section details our data preprocessing steps, exploratory data analysis, sentiment labeling approach, and how we handled class imbalance. Next, we discuss the machine learning models used, the feature extraction techniques (TF-IDF), and the PySpark ML pipeline with hyperparameter tuning. We then present the model evaluation results and discuss the performance, including considerations for scaling and runtime. Finally, we conclude with key findings and provide recommendations for future improvements or extensions of this work.

Objectives

The project's objectives were defined as follows:

- **Understand the dataset structure and characteristics:** Gain familiarity with the Amazon book reviews data (its fields, size, and basic statistics).
- **Preprocess large-scale textual data using PySpark:** Clean and prepare the review data for modeling (handling missing values, removing duplicates, and text cleaning) in a distributed manner for efficiency.
- **Perform Exploratory Data Analysis (EDA):** Analyze the data to uncover patterns or insights (such as distribution of ratings and review lengths) that could inform our modeling strategy.
- **Develop machine learning models for sentiment classification:** Build predictive models that classify review sentiment (positive, negative) from the review text (and other features), using scalable algorithms in Spark.
- **Evaluate model performance on big data:** Assess the accuracy and effectiveness of the sentiment classifiers on a large test dataset, ensuring the models generalize well.
- **Demonstrate scale-up and scale-out capabilities:** Show that the solution can handle big data by utilizing PySpark's ability to run on variable data (scale-up) and across multiple machines (scale-out) if needed.
- **Draw conclusions and provide recommendations:** Summarize what was learned from the analysis and modeling, and suggest improvements or future work based on the results.

Data Description

Dataset: We used the **Amazon Book Reviews** dataset, which contains approximately **10.3 million** customer reviews for products in the Books category on Amazon. Each record in the dataset corresponds to a single review by a user.

Data Fields: Key features (columns) in the dataset include:

- **review_id:** A unique identifier for the review.
- **user_id:** The identifier of the user (reviewer) who wrote the review.
- **rating:** The star rating given by the user, ranging from 1 to 5 stars (in the raw data some entries were 0, but valid ratings are 1–5).
- **text:** The content of the review text, written by the customer.

Methodology

Our methodology consisted of several stages: data preprocessing, exploratory analysis, feature engineering (including sentiment labeling), handling class imbalance, model building, and evaluation. We used PySpark for all these steps, which enabled parallel processing of the data. Below, we detail each component of the methodology.

Data Preprocessing

Before building any models, we performed extensive data preprocessing to clean and prepare the dataset. The following tasks were carried out in PySpark on the full dataset:

- **Handling Missing Values:** We first checked for any missing or null values in important fields (such as the review text or rating. There were no missing value found in data.
- **Removing Duplicates:** Given the large number of entries, duplicate records could exist (for example, the same review recorded twice). We used PySpark's `dropDuplicates()` function to eliminate any exact duplicate rows by considering rows having both value of title and review same as duplicate row, ensuring each review appears only once.
- **Text Cleaning:** The review text was processed to make it suitable for analysis. This involved:
    - Converting all text to lowercase (to avoid treating "Good" and "good" as different words).
    - Removing stopwords – common words like "the", "and", "is", etc., which carry little semantic meaning – using a predefined stopword list. PySpark's `StopWordsRemover` was utilized after tokenizing the text into words.
- **Feature Engineering – Review Length:** In addition to the raw text, we engineered an extra feature for each review: its length. We calculated the **review length**, for example as the number of words in the `text` after cleaning (or number of characters). We used TfIDF to convert words into numbers so that my model can learn on these vectors.

(The conversion of ratings into sentiment labels and the handling of imbalanced classes are also crucial preprocessing steps, but we discuss them separately in the next subsections for clarity.)

Overall, the data preprocessing stage ensured that our dataset was clean, structured, and enriched. By performing these steps in PySpark, we were able to scale the cleaning operations across the large dataset efficiently. For example, removing duplicates and filtering was done in parallel on data partitions, and text cleaning (tokenizing and stopword removal) was distributed across the cluster, significantly speeding up what would be very slow on a single machine.

Exploratory Data Analysis (EDA)

After cleaning the data, we conducted exploratory data analysis to understand the distribution and characteristics of the reviews and to inform our modeling decisions. Key EDA findings include:

- **Distribution of Ratings**
- **Review Length Analysis**
- **Textual Patterns**.

These EDA findings were valuable. They confirmed the presence of **class imbalance** and justified our approach to handle it (through resampling, described later. Overall, EDA helped shape our preprocessing and modeling strategy.

Sentiment Labeling

In order to train a sentiment analysis model, we needed to assign each review a sentiment category label (Positive and Negative) rather than the original numeric rating. We defined a **sentiment labeling scheme** as follows:

- **Positive sentiment:** Reviews with rating **4 or 5 stars**. These represent satisfied or happy customer opinions.
- **Negative sentiment:** Reviews with rating **0,1 ,2, or 3 stars**. These ratings indicate dissatisfaction, so they are labeled as negative sentiment.

Using this scheme, we created a new column (e.g., `sentiment`) in the DataFrame with categorical values `{Positive,Negative}` for each review.

Handling Class Imbalance (Undersampling)

Given the skewed class distribution (with *Positive* reviews dominating), training a model on the raw data could lead to a biased classifier that always predicts the majority class. To ensure the model learns to recognize *Negative*, we employed a **class balancing strategy**. We chose to use **undersampling** of the majority class in the training data:

- We significantly reduced the number of positive reviews in the training set by random undersampling 40.
- We performed this downsampling using PySpark transformations. For example, we could filter the DataFrame for each sentiment and use `.sample()` or limit the count of positive examples. Alternatively, we could use a stratified sampling approach to get a desired ratio of classes. The resulting balanced dataset was then cached for model training.

Machine Learning Models

With the data prepared and labeled, we proceeded to build and train machine learning models to classify review sentiment. We used Apache Spark's MLlib library to create a scalable machine learning pipeline. Our approach involved two types of models and two types of text feature representations, as described below.

Models Used

We experimented with two **traditional machine learning classification models** that are well-supported in PySpark MLlib:

- **Logistic Regression:** This is a linear model (generalized linear classifier) that is commonly used for binary and multiclass classification. Logistic regression was chosen because it is fast to train, can handle high-dimensional sparse input (like text TF-IDF features) efficiently, and often provides a strong baseline performance for text classification.
- **Random Forest:** This is an ensemble model that consists of many decision trees and makes predictions by aggregating (majority vote of) the trees' outputs. Random Forest classifiers can capture non-linear relationships and interactions between features. We included Random Forest to see if a non-linear model could improve performance over logistic regression for our task.

Both models were trained and tested in a distributed manner using PySpark. By comparing these two, we aimed to see which is more suitable for large-scale sentiment analysis of text. Logistic Regression often does well with text data because sentiment might be mostly a linear separable problem in a good feature space (e.g., certain words add up "positivity" or "negativity" linearly). Random Forest could potentially capture interactions (like the presence of certain negation phrases or multi-word expressions) that a basic logistic model might not.

Feature Extraction Techniques

The raw text of reviews must be transformed into numeric features that machine learning models can understand. We implemented feature extraction techniques for text, and evaluated models using:

- **TF-IDF (Term Frequency–Inverse Document Frequency):** TF-IDF is a classic technique to convert a collection of texts into a matrix of feature vectors. The idea is to represent each review by the words it contains. First, we tokenize each review into individual words (terms). Then, for each review, we compute term frequencies (how many times each word appears in that review). We also compute inverse document frequency values for each word across the entire corpus (to downweight very common words). The product of these gives TF-IDF scores for each word in each review. The result is a high-dimensional feature vector for each review, where each dimension corresponds to a specific word (or token) and the value is the TF-IDF score of that word in the review. In practice, we used PySpark's `hashingTF` and `IDF` transformers in our pipeline: HashingTF converts tokens to hashed term frequency vectors (to keep the dimensionality manageable without a massive dictionary lookup), and then IDF scales those by inverse document frequency. TF-IDF provides a sparse representation emphasizing important words in each review.

In summary, TF-IDF gives us a high-dimensional but sparse representation emphasizing word importance.

Pipeline in PySpark MLlib

We implemented our modeling approach as a **Pipeline** in PySpark MLlib. This allowed us to chain the data transformations and the model training into a single workflow, which can be cross-validated and reused for prediction easily. The pipeline consisted of the following stages in order:

1. **Tokenizer:** Split each review's text into tokens (words). We used Spark's `RegexTokenizer` to break the text by non-alphanumeric characters (after our initial cleaning, mostly splitting on whitespace and punctuation).
2. **StopWordsRemover:** Remove common stopwords from the list of tokens, to reduce noise.
3. **Feature Transformer:** This stage was either TF-IDF . We constructed separate pipeline– where the text tokens go through a HashingTF + IDF to produce TF-IDF features.
4. **Assembler (if needed):** If we wanted to include the additional numeric features (like review length or possibly `verified_purchase` as a binary feature), we would use a `VectorAssembler` to combine the TF-IDF/Word2Vec feature vector with other features into one final feature vector for the model.
5. **Classifier Model:** Finally, the last stage of the pipeline is the classifier (either Logistic Regression or Random Forest). This stage takes the assembled feature vector and learns to predict the sentiment class label.

Using this pipeline structure, we can fit the entire pipeline on the training data in one go – Spark will handle each stage in sequence on the data. We also used the pipeline to **avoid data leakage**: all transformations (like IDF weighting or Word2Vec training) are fitted only on the training data portion and then applied to the test data, to mimic how the model would operate in a real scenario on new data.

Model Evaluation

Evaluation Metric: The primary metric we focused on was **Accuracy**, which is the proportion of reviews in the test set that were correctly classified into the positive, neutral, or negative sentiment category. We also used F1 score as another evaluation method.

Scaling and Performance

One of the critical aspects of this project was ensuring that our analysis could **scale** to the size of the data. We discuss here how we handled scaling in terms of both data and hardware:

- **Scaling Up :** For scaling up we first used 50% of total data and then used 100% of total data which gave us clear understanding of how score fluctuate when using less or more data.

- **Scaling Out (Horizontal Scaling):** Scaling out involves distributing the computation across multiple machines (a cluster), adding more nodes to handle the workload. This is where Apache Spark truly excels. Here we used 2 workers and 1 worker combination, discovering how much time does a single worker takes compared to 2 workers.
- Another approach we used was cross combination of scaling up and scaling out method. Here we trained our Random forest model on multiple combination such as:
    - 2 Executors with 100% of data
    - 2 Executors with 50% of data
    - 1 Executor with 100 % of data
    - 1 Executor with 50% of data.

In conclusion, the use of PySpark ensured that our sentiment analysis pipeline was not only effective but also scalable and performant. We successfully processed and analyzed millions of reviews, which validates the big data approach. The combination of scaling techniques allowed us to manage memory usage and reduce runtime, achieving what would be impossible on a standard single-machine setup.

Results and Discussion

After training and evaluating the models, we obtained the following key results:

- **Model Performance:** The best performing model in our experiments was the **Logistic Regression with TF-IDF features**. It achieved an accuracy of approximately **80%** on the test dataset (balanced among classes). The **Random Forest** model also performed well but had slightly lower accuracy (around 77-78% on the test data). Logistic Regression outperforming Random Forest might be due to the high-dimensional sparse nature of the TF-IDF features, where a linear model can find a good separating hyperplane, whereas Random Forest could overfit on some spurious patterns or struggle with the sparsity. Additionally, logistic regression is easier to optimize in this context and required less computation, which might have allowed more effective cross-validation tuning. The difference in accuracy, while a few percentage points, is meaningful at this scale (given millions of examples). It indicates that for this task, the simpler model generalized a bit better than the more complex ensemble.

Model	Accuracy
1. Linear Regression	81.40
2. Random Forest	67.05

Scale up and scale out combinations result on Random Forest:

Model	% of Data	Number of Executors	Time Taken to train (s)	Accuracy
Random Forest	100%	2	7982	67.05
	100%	1	7432	67.02

	50%	2	3859	67.02
	50%	1	4416	67.04

Conclusion:

In conclusion, the project achieved its primary goals of building a scalable sentiment analysis solution and obtaining meaningful results from a large review dataset. By implementing the above recommendations, one could further enhance the accuracy, robustness, and applicability of the system. This project serves as a strong foundation for big data analytics applied to NLP, and it illustrates how combining domain knowledge (sentiment in text) with big data technology (PySpark) can yield powerful outcomes for analyzing user-generated content at scale.