Assignment 2

Text Processing and Classification Using Apache Spark

Ellamey Mazen
Erum Naz
Ibrahim Ahmad
Topic Filip

`Group 16`

Data Intensive Computing
May 2025

1 Introduction

This report presents the implementation and analysis of a Spark-based solution for processing and classifying Amazon product reviews. The objective is to efficiently process large-scale review data using Apache Spark's distributed capabilities and to build a machine learning pipeline that can classify reviews into their respective product categories.

The work is divided into three major parts: (1) processing using Spark RDDs, (2) pipeline creation using DataFrames and MLlib, and (3) supervised learning for text classification using an SVM model. The Amazon Review Dataset from Assignment 1 is reused, and the Spark implementation is compared against the prior approach to assess performance and output differences.

2 Problem Overview

The assignment tasks include:

1. Reimplement chi-square term selection using Spark RDDs and compare results with Assignment 1.
2. Use Spark DataFrame and ML Pipelines to tokenize, clean, and vectorize text data using TF-IDF and chi-square selection.
3. Extend the pipeline with a Support Vector Machine (SVM) classifier, normalize feature vectors, tune hyperparameters, and evaluate performance using F1 score.

The key challenges include working with distributed data structures, building efficient transformation pipelines, and designing reproducible machine learning experiments on a constrained cluster environment.

3 Methodology and Approach

This section outlines the technical details and pipeline components used to process the Amazon Review Dataset and perform multi-class text classification with Apache Spark.

3.1 Strategy

- **Data Loading & Preprocessing:** Load Amazon review data from HDFS, tokenize the text using regular expressions, and remove stopwords.
- **Feature Engineering:** Convert tokens into term frequency vectors using `HashingTF`, then apply `IDF` to compute TF-IDF feature weights.
- **Feature Selection & Label Encoding:** Use `ChiSqSelector` to retain the top 2000 most relevant terms. Encode product categories into numeric labels using `StringIndexer`.

- **Normalization & Classification:** Normalize feature vectors with L2 norm and apply a `LinearSVC` classifier wrapped in a `OneVsRest` strategy for multi-class classification.
- **Model Tuning & Evaluation:** Perform grid search over regularization, standardization, and iteration settings using `TrainValidationSplit`, and evaluate the model using F1 score with `MulticlassClassificationEvaluator`.

3.2 Pipeline

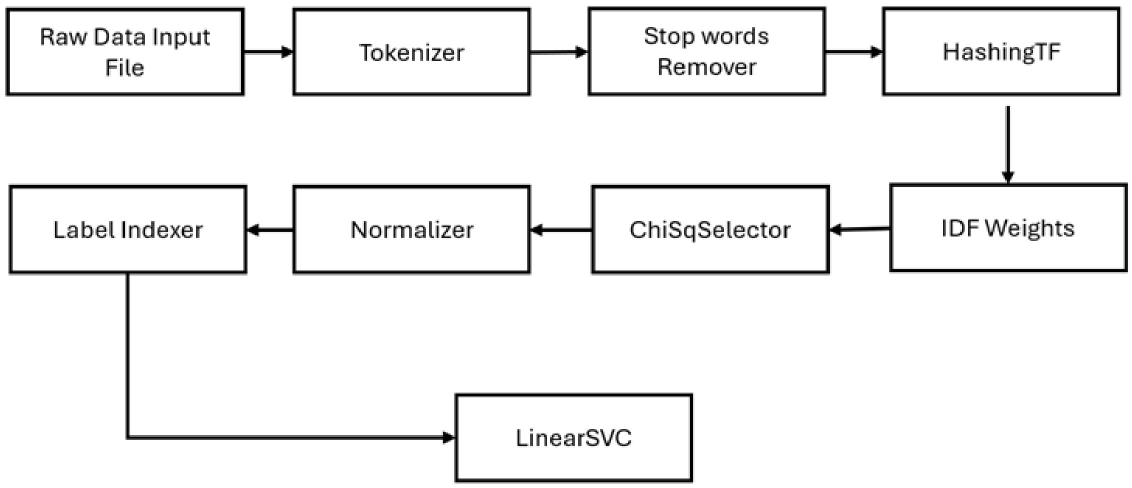

Figure 1: Spark-based strategy pipeline for text preprocessing and classification

3.3 Spark Environment Setup

- `SparkSession` is created with dynamic configuration depending on execution mode (local or cluster).
- Configuration parameters are centrally defined for dataset paths, output locations, model hyperparameters, and pipeline settings.
- Resources such as shuffle partitions and executor cores are dynamically tuned to match the execution environment.

3.4 Task 1 – Chi-Square Feature Selection with RDDs

- **Data Loading:** The JSON dataset is loaded from HDFS and parsed using RDD transformations.
- **Preprocessing:** Text is lowercased, punctuation is removed, and a regex tokenizer is applied. Custom stopword filtering is used.

- **Category Pairing:** Each review is transformed into (`word`, `category`) pairs.
- **Chi-Square Computation:** Term frequencies and expected frequencies are used to compute chi-square values for each word.
- **Output Generation:** The top terms per category and a global dictionary are saved to `output_rdd.txt`.

3.3 Task 2 – Spark ML Pipeline with DataFrames

- **Tokenization:** Used `RegexTokenizer` to split review text based on punctuation, digits, and whitespace.
- **Stopword Removal:** Applied `StopWordsRemover` to remove common uninformative words.
- **TF-IDF Transformation:** `HashingTF` and `IDF` are used to convert text to TF-IDF-weighted feature vectors.
- **Label Encoding:** Categories are indexed numerically with `StringIndexer`.
- **Chi-Square Selection:** Used `ChiSqSelector` to select the top 2000 terms across all classes.
- **Pipeline Construction:** All transformations are assembled into a `Pipeline` and executed on the dataset.
- **Output:** Selected terms are stored in `output_ds.txt`.

3.4 Task 3 – Text Classification with SVM

- **Normalization:** Feature vectors are normalized using `Normalizer` with L2 norm.
- **Classification:** `LinearSVC` with `OneVsRest` is used for multi-class classification.
- **Data Splitting:** The data is split into training (80%), validation (10%), and test (10%) sets.
- **Hyperparameter Tuning:** Used `ParamGridBuilder` and `TrainValidationSplit` to explore:
  - Regularization parameter: `0.01`, `0.1`, `1.0`
  - Standardization: enabled/disabled
  - Max iterations: `50`, `100`
- **Evaluation:** Used `MulticlassClassificationEvaluator` with F1-score as the metric.

4 Code Documentation

This section outlines the major functions, Spark components, and logical flow implemented for each task in the project.

Task 1 – RDD-Based Chi-Square Feature Selection

- `parse_review(line)` – Parses each JSON review line and extracts its category and review text.
- `preprocess_text(text)` – Tokenizes, lowercases, removes stopwords and short tokens.
- `compute_chi(kv)` – Computes chi-square statistics for a given (category, term) pair.
- `topk_per_category(iterator)` – Aggregates top-K scoring terms per category using a local heap.
- `clean_term(term)` – Cleans terms by stripping non-alphanumeric characters.

Task 2 – TF-IDF and Feature Selection with DataFrames

- `RegexTokenizer` – Splits text into tokens based on regular expression patterns.
- `StopWordsRemover` – Filters out stopwords from the tokenized text.
- `CountVectorizer` – Converts filtered tokens into sparse term frequency vectors.
- `IDF` – Applies inverse document frequency weighting to TF vectors.
- `StringIndexer` – Encodes category labels into indexed numeric values.
- `ChiSquareTest.test()` – Evaluates each feature's relevance using chi-square statistics.

Task 3 – Classification Pipeline and Evaluation

- `ChiSqSelector` – Selects top-N features based on chi-square relevance to the target label.
- `StandardScaler` – Standardizes feature vectors to improve classifier stability.
- `Normalizer` – Normalizes vectors using L2 norm to ensure consistent scaling.
- `LinearSVC` – Defines a linear support vector classifier for binary tasks.
- `OneVsRest` – Extends LinearSVC to multi-class classification using one-vs-rest strategy.
- `ParamGridBuilder` – Constructs a grid of hyperparameter combinations for model tuning.

- `TrainValidationSplit` – Performs model selection using train/validation splits and F1 score.
- `MulticlassClassificationEvaluator` – Computes F1 score for multi-class predictions.
- `bestModel.stages[]` – Retrieves the optimal model pipeline and its learned parameters.

5 Results

This section summarizes the runtime performance, outputs, and key findings from each task executed using Apache Spark. All experiments were run on a distributed environment using HDFS and were optimized for performance and reproducibility.

5.1 Task 1 – RDD-Based Chi-Square Feature Selection

- **Runtime:** 30.48 seconds (includes both computation and file writing).
- **Output:** Top 75 chi-square terms were extracted for each product category based on token frequency and category association.
- **Observations:** Discriminative terms such as `games` (Apps), `shampoo` (Beauty), and `diaper` (Baby) were successfully identified, demonstrating the effectiveness of RDD-based frequency aggregation and scoring.

5.2 Task 2 – TF-IDF and Chi-Square with DataFrames

- **Runtime:** 42.22 seconds (including output saving).
- **Output:** Top 2000 global features were selected using Spark ML's `ChiSquareTest` over TF-IDF vectors.
- **Observations:** Results confirmed that the DataFrame pipeline is more scalable and modular. Frequently selected terms such as `iphone`, `dvd`, and `carseat` matched those in Task 1, validating consistency.

5.3 Task 3 – Classification and Evaluation

- **Model Pipeline:** Reused TF-IDF vectors from Task 2 and added: `ChiSqSelector`, `StandardScaler`, `Normalizer`, and `OneVsRest(LinearSVC)`.
- **Tuning Parameters:**
  - `regParam`: {0.01, 0.1, 1.0}
  - `maxIter`: {5, 10}
  - `withStd`: {True, False}
- **Evaluation Metric:** F1 Score on the test set.
- **Test F1 Score:** `0.7510`
- **Best Model Parameters:**
  - `numTopFeatures`: 2000
  - `withStd`: True

  - `regParam`: 1.0
  - `maxIter`: 10

5.4 Summary

All three tasks achieved consistent results in terms of term selection and classification performance. The TF-IDF + Chi-square feature selection aligned well with expected domain keywords, and the classification pipeline achieved a robust F1 score, confirming the pipeline's effectiveness for multi-class review categorization.