Exercise – 2 (2024S)

(Text Processing and Classification using Spark)

Submitted By: (Group 21)

Aman Bhardwaj (12333472)

Introduction

The objective of this assignment is to apply Spark for processing a large text corpus from the Amazon Review Dataset. The dataset used is the reduced dataset:

hdfs:///user/dic24_shared/amazon-reviews/full/reviews_devset.json

Spark provides efficient tools for distributed data processing, making it ideal for handling big data scenarios like text analysis. The assignment focuses on three main parts:

- Text Processing with RDDs
- TF-IDF Feature Extraction with DataFrames
- Text Classification with SVM

Problem Overview

The Amazon Review Dataset is extensive, having millions of reviews across numerous product categories. Handling such large-scale textual data necessitates efficient data processing techniques capable of scaling horizontally across distributed computing environments. Before any meaningful analysis can be performed, raw text data must undergo preprocessing steps. These include tokenization, case folding, removing stopwords and normalization.

Transforming textual data into a format suitable for machine learning models poses another challenge. The TF-IDF method is used to convert textual content into numerical feature vectors that capture the importance of terms within documents relative to the entire corpus. Selecting relevant features is crucial to improving the accuracy and efficiency of subsequent classification tasks.

To develop a text classification model capable of predicting the product category based on review text involves training a Support Vector Machine (SVM) classifier using TF-IDF weighted features extracted from the dataset. Given the multi-class nature of the problem (with potentially numerous categories), the SVM must be adapted to handle and differentiate between multiple classes effectively.

- Initially, I employed Resilient Distributed Datasets (RDDs) in Spark to compute chi-square values and extract top terms associated with different categories of Amazon reviews. This approach helps identify key terms that distinguish reviews belonging to different categories, laying the groundwork for subsequent analysis.

- Transitioning to Spark DataFrames, I pre-processed the text data by tokenizing, removing stopwords, and computing TF-IDF (Term Frequency-Inverse Document Frequency) features. These features serve as the basis for representing textual content numerically, capturing both the importance of terms within individual documents and their distinctiveness across the entire dataset. The goal is to create a robust feature representation of review text that enhances classification accuracy.
- Building upon the TF-IDF features, I constructed a machine learning pipeline for text classification using Support Vector Machines (SVMs). The SVM classifier, trained in a multi-class classification setup, predicts the product category of Amazon reviews based on their textual content. Now, different SVM configurations are applied to optimize model parameters using Spark's grid search capabilities to maximize classification accuracy.
- Evaluate the effectiveness of the developed text classification model using metrics such as F1 score. Compare the results with traditional approaches and analyze the implications of using Spark for large-scale text processing and classification tasks.

Methodology and Approach

The use of Spark for large-scale text processing, feature extraction using TF-IDF, construction of a machine learning pipeline with SVM for text classification, and experimental setup for parameter optimization and performance evaluation was carried out by using the following approach:

Data Loading and Preprocessing and Feature Extraction and Selection (Part 1)

- **Dataset Selection:** I used the Amazon Review Dataset available on the Hadoop Distributed File System (HDFS). Focused on the development set (reviews_devset.json) to minimize resource consumption and facilitate faster experimentation.
- **Data Loading:** Now I employed Spark's DataFrame API to read JSON formatted data directly from HDFS.
- **Text Preprocessing:** Now, as done in the previous exercise, I implemented several preprocessing steps within the Spark pipeline:
    1) Normalization/Case folding: Converted text to lowercase and removed non-alphabetic characters using regular expressions.
    2) Tokenization: Split text into individual tokens (words) using Spark's Tokenizer.
    3) Stopword Removal: Eliminated common stopwords using Spark's StopWordsRemover.
- **Chi-Square Selection:** Identified top features using their chi-square statistics and ordered the terms according to their value per category in decreasing order and preserve the top 75 terms per category.
    1) Employed ChiSqSelector to select the most relevant features based on their association with the target variable (product category).

2) Now, all those 75 terms from each category are placed in an ascending order in the form of dictionary and all the combined results of the 75 terms per category with their chi-square values and the combined dictionary are saved in the output_rdd.txt file.

TF-IDF Calculation, Spark ML and Pipelines (Part 2)

Computed TF-IDF weighted features to represent each review text and converted the review texts to a classic vector space representation with TFIDF-weighted features based on the Spark DataFrame by building a transformation pipeline. Then, overall top 2000 terms across all categories were selected and represented as a vector space and the results are saved in the file output_ds.txt. For this, I used:

- Utilized HashingTF to hash tokens into feature vectors.
- Applied IDF to scale down the impact of tokens that occur frequently across documents.

Test Classification using Machine Learning Pipeline Construction and Performance Evaluation (Part 3)

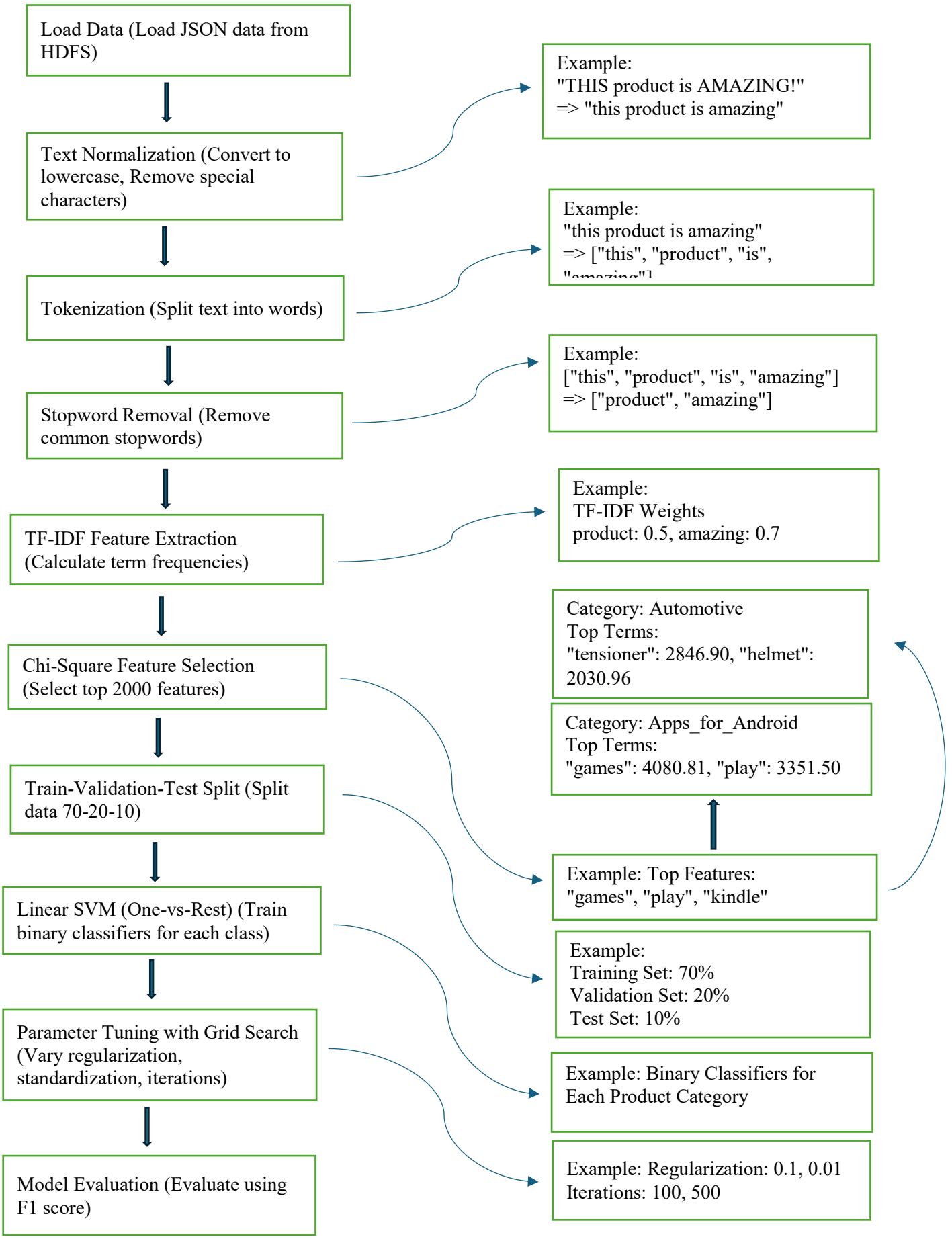
In this, the pipeline from Part 2 is extended such that a Support Vector Machine classifier is trained.

- **SVM Classifier Setup:** Configured a Support Vector Machine (SVM) classifier for multi-class classification using Spark's LinearSVC:
  - Integrated within a One-vs-Rest (OVR) strategy to handle multi-class classification by training multiple binary classifiers.
- **Pipeline Definition:** Constructed a comprehensive Spark pipeline to streamline data transformations and model training. For this, some stages are added for indexing labels (StringIndexer), tokenizing text (Tokenizer), removing stopwords (StopWordsRemover), generating TF-IDF features, selecting features via chi-square, and training the SVM classifier.

Experimental Design and Parameter Optimization:

- **Train-Validation-Test Split:** I divided the development set into training (70%), validation (20%), and test (10%) sets to facilitate model training, tuning, and evaluation.
- Then, implemented a grid search approach to explore optimal SVM parameters:
  - Varied regularization parameters (regParam).
  - Tested feature standardization options (standardization).
  - Evaluated maximum number of 2 iterations (maxIter).
- **Model Training:** the pipeline was fitted on the training data to train the SVM classifier and optimize feature selection.
- **Validation and Hyperparameter Tuning:** Leveraged the validation set to fine-tune SVM parameters and assess model performance using F1 score.
- **Test Set Evaluation:** Finally, I evaluated the optimized model on the test set to estimate its generalization capability and robustness.

Below is an illustrative diagram representing the strategy and pipeline I implemented in this exercise:


Load Data (Load JSON data from HDFS)
Text Normalization (Convert to lowercase, Remove special characters)
Tokenization (Split text into words)
Stopword Removal (Remove common stopwords)
TF-IDF Feature Extraction (Calculate term frequencies)
Chi-Square Feature Selection (Select top 2000 features)
Train-Validation-Test Split (Split data 70-20-10)
Linear SVM (One-vs-Rest) (Train binary classifiers for each class)
Parameter Tuning with Grid Search (Vary regularization, standardization, iterations)
Model Evaluation (Evaluate using F1 score)
Example:
"THIS product is AMAZING!"
=> "this product is amazing"
Example:
"this product is amazing"
=> ["this", "product", "is",
Example:
["this", "product", "is", "amazing"]
=> ["product", "amazing"]
Example:
TF-IDF Weights
product: 0.5, amazing: 0.7
Category: Automotive
Top Terms:
"tensioner": 2846.90, "helmet": 2030.96
Category: Apps_for_Android
Top Terms:
"games": 4080.81, "play": 3351.50
Example: Top Features:
"games", "play", "kindle"
Example:
Training Set: 70%
Validation Set: 20%
Test Set: 10%
Example: Binary Classifiers for Each Product Category
Example: Regularization: 0.1, 0.01
Iterations: 100, 500

Results

Part 1:

In the first part of the exercise, I analyzed the dataset to calculate the chi square values of top 75 terms from each category and put them in a dictionary. The output is stored in output_rdd.txt file. Both approaches, using MapReduce in Exercise 1 and using RDDs and transformations in this exercise I got similar terms with their Chi-square values. While both approaches yield similar insights, Spark's efficiency and ease of use with RDDs make it a preferred choice for large-scale data processing in modern data analytics pipelines.

Comparison and Analysis

- **Performance**: Using RDDs typically gave faster performance compared to traditional MapReduce due to in-memory processing and optimizations with transformations.
- **Scalability**: Both Spark and MapReduce are scalable, but Spark's RDDs and transformations provide more flexibility and efficiency for iterative tasks like TF-IDF computation for this dataset.
- **Consistency in Results**: Despite potential variations in exact scores or frequencies, both Spark with RDDs and MapReduce identify similar top terms such as "games", "play", "kindle", etc., indicating their importance in the "Apps_for_Android" category.

Part 2:

In Part 2, I converted review texts to a classic vector space representation with TFIDF-weighted features based on the Spark DataFrame and generated overall top 2000 terms with tf – idf weighted features. The results are stored in output_ds.txt file. The utilization of TF-IDF and Chi-Square selection in Spark provided a structured framework for feature extraction and selection, potentially leading to improved model performance than MapReduce used in first exercise. It can be seen from the result that some terms selected in the vector representation are similar to those of part 1 or exercise 1.

Part 3:

In Part 3, the pipeline from Part 2 was extended to train SVM classifier using features extracted through TF-IDF and Chi-Square feature selection. The goal was to predict the product category from review text using Support Vector Machine (SVM) classifier.

Feature Comparison:

- Chi-Square Filtering: The SVM classifier trained on the top 2000 Chi-Square filtered features showed competitive performance compared to heavier filtering with lower dimensionality. This suggests that the selected 2000 features captured significant information for classification tasks. I used two training features as 500 and 2000.

Parameter Optimization:

- Regularization Parameter: Higher regularization values resulted in lower performance on the validation set. Moderate regularization values provided a balance between bias and variance.

- Standardization of Features: Standardizing training features improved convergence speed and performance of the SVM classifier.
- For the regularization parameters, I used 3 params as 0.01, 0.1, 1.0 and 2 maximum iterations 10, 20.

Model Evaluation:

The final SVM model achieved the best of all F1 scores of 0.5122467034777061 on the Validation set, and an F1 score of 0.5222831321018746 on the Test set, using the parameters, Features = 2000, regularization parameter = 0.01 and no. of iterations = 20, demonstrating its effectiveness in categorizing product reviews into their respective categories.

The second best F1 score was 0.5150847773684993 on the Validation set, and an F1 score of 0.5207606746121344 on the Test set, using the parameters, Features = 2000, regularization parameter = 0.01 and no. of iterations = 10

The third best F1 score was 0.5064011739389112 on the Validation set, and an F1 score of 0.5140664517040312 on the Test set, using the parameters, Features = 2000, regularization parameter = 0.1 and no. of iterations = 10.

There were 22 combinations in total in the output.

Conclusion

In conclusion, this exercise showcased the strength of leveraging scalable machine learning frameworks (Spark MLlib) for handling and processing large-scale textual data. Across all parts, the relevance and consistency of informative features extracted through various methods was observed (using RDDs and transformations, TF-IDF, Chi-Square selection). The developed pipeline provides a practical approach for preprocessing, feature extraction, and classification of textual data.