BIL 401 / BIL 501 - Big Data and Distributed Data Processing
Project Status Report

Date: June 20, 2025
Project Title: Predicting Stack Overflow Question Quality using Distributed Machine Learning on Spark: CPU and GPU Implementations
Team Members: Beren Ünveren

1. Executive Summary

This project aims to predict the quality of Stack Overflow questions (High Quality, Low Quality - Edit, Low Quality - Close) using machine learning techniques on a large dataset. The primary focus is to take advantage of the Apache Spark framework for distributed data processing and model training. The project involves developing a CPU-based machine learning pipeline as a baseline and exploring the potential for GPU acceleration using tools like the RAPIDS Accelerator for Spark later to compare performance. This report summarizes the project's status, detailing the platform setup progress, initial findings, challenges encountered, and providing a draft of the initial sections for the final project paper.

2. State of Data Collection

The dataset specified in the proposal, '60k Stack Overflow Questions with Quality Rating', was successfully downloaded from Kaggle. The primary data file, train.csv, is approximately 50.36 MB in size. The data has been stored locally on our development machine within the project directory. No additional data sources are required for the initial phases of the project.

- **Status:** Completed.
- **Detail:** The dataset is readily available on the local file system, prepared for ingestion into Spark DataFrames for further processing and analysis.

3. Status of Platforms and Systems

Below is the current status of the platforms, frameworks, and tools planned for the project:

- **Apache Spark (PySpark):** Installed and basic functionalities (SparkSession initialization, data loading, schema/data inspection) tested successfully. Initial development & testing are being performed in local mode.
- **Machine Learning Library (Spark MLLib):** Included with the PySpark installation, ready for use in the machine learning pipeline development.

- **Machine Learning Library (RAPIDS Accelerator for Spark):** Setup initiated. Research and initial configuration efforts for integrating RAPIDS with Spark and the GPU environment are underway.
- **Programming Language (Python):** Installed. Necessary core libraries (pyspark, pandas, numpy, matplotlib) have been successfully installed and configured within the development environment.
- **Hardware (GPU):** Initial configuration efforts for driver and CUDA setup have begun.
- **GPU Software Stack (CUDA, cuDF, cuML):** Setup initiated, dependent on successful GPU hardware configuration. Efforts are focused on meeting RAPIDS/Spark compatibility requirements.
- **Data Storage (Local file system):** Configured as the initial storage location for the dataset and intermediate outputs. Ready for read/write operations by Spark.
- **Development Environment (Local machine):** Configured and ready for coding, testing, and execution of Spark jobs. Initial development & testing are being performed in local mode.
- **Visualization Tools (Matplotlib, Seaborn):** Installed, ready for basic data visualization and result plotting.
- **Version Control (GitHub):** A project repository has been created and is being used for managing code versions.

4. Installation Problems, Know-How, Demo Runs

Installation Problems:

- Initially encountered challenges setting up SPARK_HOME and HADOOP_HOME environment variables and adding %SPARK_HOME%\bin to the system PATH on Windows. This was resolved by carefully following the manual installation steps for a pre-built Spark distribution and ensuring new command prompts were opened for changes to take effect.
- The most significant challenge encountered post-installation is the parsing of the train.csv file using spark.read.csv. Initial attempts with inferSchema=True resulted in incorrect schema detection (most columns inferred as StringType) and, crucially, a corrupted target variable (Y) column containing mostly nulls or seemingly random text snippets instead of the expected quality labels (HQ, LQ_EDIT, LQ_CLOSE). Attempts to resolve this using manual schema definitions (StringType or IntegerType for Y) confirmed that the core issue lies in how Spark handles the specific formatting (potentially internal quotes, escape characters, or malformed rows) within the large CSV file itself, leading to data corruption during loading. While basic file integrity (permissions, line endings) was verified, the complex parsing issue persists.
- The initial steps to set up the GPU environment and RAPIDS Accelerator introduced potential complexities regarding version compatibility between CUDA drivers, RAPIDS components, PySpark and the operating system, after which the appropriate version was found and waiting to be developed.

Know-How:

- Gained practical experience in initializing and managing SparkSession objects in PySpark.
- Learned how to load CSV data into Spark DataFrames using spark.read.csv and understood the importance of parameters like header, inferSchema, and explicitly defined schema.
- Became proficient in using df.printSchema(), df.show(), df.count(), df.describe(), and df.groupBy().count() for initial data inspection and understanding the structure and basic statistics of a Spark DataFrame. Solidified understanding of the necessity and correct configuration of Spark-related environment variables (SPARK_HOME, HADOOP_HOME) for seamless PySpark execution on Windows, including setting up for local mode.
- Learned that simply reading a CSV is not always straightforward with large, potentially complex files, and that incorrect parsing can critically impact data integrity and downstream tasks. Developed initial problem-solving approaches for CSV parsing issues.

Demo Runs: A minimal PySpark script was developed and executed to validate the basic Spark environment setup and data loading capability. The script performs the following steps:

1. Initializes a SparkSession.
2. Attempts to read the train.csv file into a DataFrame.
3. Prints the DataFrame schema (printSchema()).
4. Prints the total row count (count()).
5. Displays basic statistics (describe()).
6. Attempts to show the distribution of the Y column (groupBy("Y").count()).

Result: The script successfully initialized Spark and read the CSV file without throwing a file-not-found error. However, the output of printSchema(), show(), and groupBy("Y").count() clearly showed the data parsing issue, particularly the corruption or absence of valid data in the Y column. The total row count (869081) was correctly identified, indicating the entire file was loaded, but the content was not correctly structured into columns. This run served as a critical demonstration of both successful basic environment setup and the identification of the primary data ingestion challenge. Outputs of these runs have been captured for documentation.

5. Paper Draft: Abstract, Related Work, Proposed Implementation

5.1. Abstract:

This project will investigate the application of distributed machine learning techniques using Apache Spark to predict the quality of questions posted on Stack Overflow. Leveraging a publicly available dataset of Stack Overflow questions and their corresponding quality labels, the research aims to develop a machine learning pipeline capable of classifying questions into categories such as High Quality (HQ), Low Quality - Edit (LQ_EDIT), and Low Quality - Close (LQ_CLOSE). The project will involve standard natural language processing and feature engineering steps (tokenization, TF-IDF) implemented on Spark DataFrames. A baseline classification model will be trained and evaluated using Spark MLLib on a CPU-based

environment. Furthermore, the study will explore the feasibility and performance benefits of accelerating computationally intensive stages of the pipeline (particularly model training) by integrating GPU resources using tools like the RAPIDS Accelerator for Spark, providing a comparative analysis of CPU versus GPU performance for this large-scale text classification task.

5.2. Related Work:

- **E. Skenderi, J. Huhtamaki, S.-M. Laaksonen, and K. Stefanidis, "Assessing Text Representation Methods on Tag Prediction Task for StackOverflow," Proceedings of the 56th Hawaii International Conference on System Sciences | 2023, Hawaii, USA, 2023, pp. 585-594.** — compared six text representation methods: two classical baselines (Bag of Words and TF-IDF) and four methods based on pre-trained models (Fasttext, Universal Sentence Encoder - USE, Sentence-BERT, and Sentence-RoBERTa), resulting feature vectors from these methods were used as input to a ML-kNN classifier. Sentence-RoBERT yielded the best performance, and classical methods often outperformed Fasttext and Sentence-BERT.
- **P. K. Roy, J. P. Singh, and S. Banerjee, "Is this question going to be closed?:Answering question closibility on Stack Exchange," Journal of Information Science, pp. 1291-1307, 2024. doi: 10.1177/01655515221118665**. — developed a supervised learning system to predict question closibility for the manual closure of low-quality problems. They used a data dump that covers Jan 2009 to Mar 2017 that classifies questions as closed or open, extracted 17 features, recognized the data imbalance and applied SMOTE before training, tested several traditional ML methods (Gradient Boosting, Logistic Regression, Naive Bayes, Random Forest) and found Gradient Boosting to be the best performer. They also briefly compared against deep learning models which did not perform as well with their chosen features.Feature analysis indicated that non-textual features and the textual feature "number of interrogative words" were among the most important predictors.
- **M. B. Atiku, N. Ismail, and M. Alhadji, "Classifying Stack Overflow Questions Quality using SVM," B. Sc. Eng. thesis, Dept. Comput. Sci. Eng., Islamic Univ. Technol., Dhaka, Bangladesh, 2021.** — they used the data dump that mentioned above (Jan 2009 - Mar 2017), preprocessing steps included unifying the title and body, removing non-text characters, and cleaning HTML tags, TF-IDF was used to represent the textual data as numerical feature vectors, used SVM classifier based on the TF-IDF features and used Optuna framework was used for hyperparameter tuning. Experiments with common deep learning models did not outperform the traditional machine learning approach with their selected features.
- **I. Annamoradnejad, J. Habibi, and M. Fazli, "Multi-view approach to suggest moderation actions in community question answering sites," Information Sciences,**

vol. 600, pp. 144-154, 2022. doi: 10.1016/j.ins.2022.03.085. — implemented a multi-view machine learning approach to automatically suggest moderation actions and tried the idea on Stack Overflow dataset. The method extracts features from the question title and body using a multi-view approach, generating subjective features (via a BERT regression model), software-related features (via a SoftNER model), and general lexical/statistical features, which are then used to train a Gradient Boosting classifier (XGBoost) to predict one of three moderation actions on a manually classified Stack Overflow dataset. The approach achieved a high overall multiclass accuracy of 95.6% and significantly outperformed state-of-the-art baseline models in terms of F1-score.

- **J. Hu and B. Yang, "Posts Quality Prediction for StackOverflow Website," IEEE Access, vol. 12, pp. 135601-135615, 2024. doi: 10.1109/ACCESS.2024.3440879.** — addresses the challenge of manual Stack Overflow moderation by developing an automated system to predict question quality, using automatic text vectorization combined with machine learning and deep learning models. Utilizing a dataset of 60,000 questions balanced across three quality categories (High-Quality, Low-Quality Edit, Low-Quality Closed), the authors employed Doc2vec as the primary vectorization method, comparing it with TF-IDF and Word2vec using BERT, and evaluated five traditional machine learning models along with two deep learning models (Bi-LSTM and fine-tuned BERT). They showed that leveraging attention mechanisms in BERT significantly improves prediction accuracy for Stack Overflow post quality.
- **H. Alharthi, D. Outioua, and O. Baysal, "Predicting Questions' Scores on Stack Overflow," in Proc. 2016 3rd International Workshop on CrowdSourcing in Software Engineering, Austin, TX, USA, 2016, pp. 1-7. doi: 10.1145/2897659.2897661** — investigates factors influencing the quality of questions on Stack Overflow by predicting their scores, using a dataset of 12,077 filtered questions and examining sixteen numerical factors related to question format, content, and community interactions through nonparametric analysis and a Multiple Linear Regression model. The study found high positive correlations with factors like views and answers, statistically significant negative associations with factors like code length and tags number, and developed a regression model that demonstrated high predictive power with significant predictors.
- **R. Shovon, L. R. Dyken, O. Green, T. Gilray, and S. Kumar, "Accelerating Datalog applications with cuDF," in 2022 IEEE/ACM Workshop on Irregular Applications: Architectures and Algorithms (IA3), 2022, pp. 41–45**. — implements methods for significantly accelerating the execution of Datalog applications by specifically implementing their core underlying relational algebra primitives directly on GPUs. The authors highlight that their approach achieves this acceleration by heavily leveraging the RAPIDS suite of libraries, particularly cuDF for GPU accelerated data manipulation, while explicitly stating that their methodology does not involve the use of Apache Spark. Performance evaluations are conducted by directly comparing the execution times using

traditional CPU based pandas dataframes against the GPU accelerated cuDF dataframes to demonstrate the benefits of the GPU implementation.

- **Aguerzame, B. Pelletier, and F. Waeselynck, "GPU Acceleration of PySpark using RAPIDS AI," in Proceedings of the 8th International Conference on Data Science, Technology and Applications (DATA 2019), 2019, pp. 437–442.** — tries to achieve GPU acceleration for Apache Spark applications, specifically targeting the PySpark API, by utilizing the capabilities of RAPIDS AI, the method involves integrating RAPIDS' cuDF library directly within PySpark's pandas UDFs; they primarily evaluated this approach for accelerating general data processing tasks performed within these UDFs, explicitly mentioning that their experiments did not involve the use of Spark MLlib for machine learning workloads.
- **D. Dauletbak, J. Heo, S. Kim, Y. P. Kim and J. Woo, "Scalable Traffic Predictive Analysis using GPU in Big Data," in 2021 International Conference on Electronics, Information, and Communication (ICEIC), 2021, pp. 1-5, doi: 10.1109/ICEIC49085.2021.9338582**. — presents an approach aimed at predicting traffic jams by applying machine learning techniques within a big data platform environment based on Apache Spark, further accelerated through the integration of GPUs utilizing the RAPIDS library, conducted performance comparisons focusing on the computing time required for the analysis and reported a significant reduction in execution time when employing the GPU accelerated Spark platform compared to CPU only configurations.
- **S. Lee and S. Park, "Performance Analysis of Big Data ETL Process over CPU-GPU Heterogeneous Architectures," 2020 IEEE/ACM Workshop on Machine Learning in High Performance Computing Environments (MLHPC) and Workshop on Artificial Intelligence and Machine Learning for Scientific Applications (AI4S).** — conducted a detailed performance analysis of big data Extract, Transform, Load processes, with a primary focus on optimizing workloads executed via Spark SQL through the use of GPUs, found that employing a heterogeneous CPU-GPU architecture can lead to greater overall efficiency compared to relying solely on one type of processor, particularly highlighting scenarios where processing smaller data sizes performs better on the CPU.
- **D. Manzi and D. Tompkins, "Exploring GPU Acceleration of Apache Spark," in 2016 IEEE International Conference on Cloud Engineering (IC2E), 2016, pp. 222-223, doi: 10.1109/IC2E.2016.30.** — explores the potential for accelerating Apache Spark applications by offloading computationally intensive core operations to the GPU, specifically implementing the GPU portions using PyCUDA rather than the RAPIDS library, applies this approach to the KMeans clustering algorithm within Spark MLlib and reported observing a significant speedup compared to the CPU-only execution of the algorithm by leveraging the GPU.
- **K. R. Jayaram, A. Gandhi, H. Xin, and S. Tao, "Adaptively Accelerating Map-Reduce/Spark with GPUs: A Case Study," IEEE, 2018.** — presents a case study

focusing on the adaptive acceleration of iterative machine learning algorithms running on both Spark and MapReduce frameworks by utilizing GPUs, implements four common iterative ML algorithms in Java, integrating GPU execution capabilities via jCUDA within the respective framework's UDF's and explicitly stating that their work did not employ the RAPIDS library for GPU acceleration.

- **S. M. Ghazimirsaeed, Q. Anthony, A. Shafi, H. Subramoni and D. K. Panda, "Accelerating GPU-based Machine Learning in Python using MPI Library: A Case Study with MVAPICH2-GDR," 2020 IEEE/ACM Workshop on Machine Learning in High Performance Computing Environments (MLHPC) and Workshop on Artificial Intelligence and Machine Learning for Scientific Applications (AI4S), Atlanta, GA, USA, 2020, pp. 97-104.** — investigates methods for accelerating GPU-based machine learning workloads within distributed Python environments through the effective use of the MPI library (MVAPICH2-GDR) for interprocess communication; prominently features work utilizing cuML, a key component of the RAPIDS ecosystem positioned as a GPU accelerated counterpart to scikit-learn, and includes a comparison of different communication methods like MPI and NCCL. Spark ML is mentioned within the paper primarily as a representative example of CPU-based machine learning libraries used for comparison contexts.

5.3. Proposed Implementation:

The project methodology is structured into several key phases:

- Load the downloaded train.csv dataset into a Spark DataFrame. This step is currently facing challenges due to data parsing issues that need to be resolved.
- Clean and prepare the text data. This involves:
  - Handling the CSV parsing errors to ensure data integrity.
  - Cleaning HTML tags from the 'Body' column using Spark User Defined Functions (UDFs).
  - Handling missing values in relevant columns (e.g., 'Tags', 'Y') by dropping rows or imputation where appropriate.
- Apply standard Natural Language Processing techniques using Spark MLLib's feature transformers:
  - Tokenization: Splitting text (Title, Body, Tags) into individual words or tokens.
  - Stop-word Removal: Eliminating common English words that do not carry significant meaning.
- Transform the preprocessed text data into numerical features suitable for machine learning models. The primary method will be TF-IDF. Additional features derived from metadata or text length may also be considered.
- Train baseline classification models using Spark MLLib on the prepared feature vectors. Algorithms to be explored include Logistic Regression, Naive Bayes, and potentially

Ensemble Methods like Random Forest, focusing on those supported efficiently by MLLib.
- Evaluate the trained models using appropriate metrics for classification (Accuracy, Precision, Recall, F1-score), potentially focusing on metrics relevant to multi-class or imbalanced datasets if necessary. Cross-validation will be used for robust evaluation.
- Identify computationally intensive steps in the CPU pipeline (TF-IDF generation and model training are typically computationally intensive on large text datasets, making them prime candidates for GPU offloading). Configure the Spark environment to utilize the RAPIDS Accelerator. Port or adapt the identified steps to use RAPIDS-compatible operations where available. Train the same model(s) on the GPU-accelerated pipeline. Compare the performance (training time, throughput) and, if applicable, model accuracy between the CPU-only and GPU-accelerated implementations.
- Document the entire process, findings, challenges, and the results of the performance comparison in the final paper.

6. Current Progress

Beyond the successful setup of the core CPU development environment and verification of basic Spark functionalities (as detailed in Section 3 & 4), the following progress has been made on the project's core tasks:

- The dataset (train.csv) has been loaded into a Spark DataFrame. Initial data inspection using printSchema(), count(), describe(), and groupBy("Y").count() has been performed.
- These initial inspection steps led to the critical discovery of significant data parsing issues affecting the integrity of the loaded DataFrame, particularly the target variable (Y). This issue is currently the primary focus before proceeding with further steps.
- Initial planning and design work for the data cleaning (HTML removal, missing values) and text preprocessing steps (tokenization, stop-word removal) using Spark MLLib feature transformers have been initiated. However, implementation is pending resolution of the data parsing problem.
- Minimal progress has been made on feature engineering or model training due to the necessity of resolving the data loading issue first.

7. Challenges & Risks

- The primary technical challenge currently is resolving the persistent data parsing issue when loading the train.csv into Spark. Incorrect data ingestion will block all subsequent preprocessing, feature engineering, and model training steps.

- The delay caused by initial environment setup and the current data parsing issue puts pressure on the timeline for completing the remaining phases of the project, particularly the time-intensive GPU acceleration and comparison phase.
- Processing a large dataset (869,081 rows) for text feature engineering and model training on a local development machine, even with Spark's local mode, may require careful optimization and resource management.

8. Next Steps

- Dedicated effort will be put into identifying and resolving the CSV parsing problem to ensure the dataset is loaded correctly into a Spark DataFrame with accurate column types and data integrity, especially for the Y column. This will involve exploring advanced spark.read.csv options or using external tools for initial cleaning if necessary.
- Once the data loading is stable, implement the planned data cleaning steps (HTML removal, handling missing values) using Spark UDFs and DataFrame operations.
- Implement the text preprocessing pipeline (tokenization, stop-word removal) using Spark MLLib feature transformers.
- Begin implementing the TF-IDF feature extraction step using Spark MLLib.
- Continue efforts towards successfully setting up the GPU environment and the RAPIDS Accelerator for Spark.