Mining Massive Data Sets Endterm Report

1st Luong Canh Phong
Faculty of Information Technology
Ton Duc Thang University
Ho Chi Minh City, Vietnam
522H0036@student.tdtu.edu.com

2nd Cao Nguyen Thai Thuan
Faculty of Information Technology
Ton Duc Thang University
Ho Chi Minh City, Vietnam
522H0092@student.tdtu.edu.com

3rd Tang Minh Thien An
Faculty of Information Technology
Ton Duc Thang University
Ho Chi Minh City, Vietnam
522H0075@student.tdtu.edu.com

4th Truong Tri Phong
Faculty of Information Technology
Ton Duc Thang University
Ho Chi Minh City, Vietnam
522H0167@student.tdtu.edu.com

5th Instructor: Nguyen Thanh An
Faculty of Information Technology
Ton Duc Thang University
Ho Chi Minh City, Vietnam
nguyenthanhan@tdtu.edu.com

***Abstract*—This project implements and evaluates key techniques in mining massive datasets. It covers hierarchical agglomerative clustering of string shingles using Jaccard distance; PySpark-based linear regression for gold price prediction; CUR decomposition for feature dimensionality reduction on gold price data; and PageRank analysis of the `it.tdtu.edu.vn` web graph using PySpark. The work demonstrates practical applications and provides insights into processing large-scale data.**

I. Introduction

The increasing volume of data requires efficient mining techniques. This project implements and analyzes four core algorithms: (1) hierarchical agglomerative clustering for non-Euclidean text data, using 4-shingles and Jaccard distance on alphabetical strings; (2) PySpark-based linear regression to predict Vietnamese gold prices from historical data; (3) CUR decomposition to reduce the dimensionality of gold price features (from 10 to 5) and assess its impact on regression; and (4) PageRank, implemented in PySpark, to identify influential pages within the `it.tdtu.edu.vn` web graph. Python and PySpark are utilized throughout. This report details the methodologies, implementations, and experimental results for each task.

II. First Task: Hierarchical clustering in non-Euclidean spaces

A. Overview of Agglomerative Hierarchical Clustering (AHC)

This task implements the AHC algorithm to group character strings based on similarity in shingles (4-character tokens) and Jaccard distance. The goal is to divide a large dataset (about 10,000 random strings) into a predefined number of clusters.

The main method involves calculating the clustroid (the representative sample with the smallest sum of squared distances to other samples in the cluster) and determining the distance between clusters based on the Jaccard distance between their clustroids. The algorithm gradually merges the closest pairs of clusters until the desired number of clusters is reached, using a heap for optimization.

B. Implementation of the Algorithm

The AHC algorithm partitions a dataset of shingle sets (derived from character sequences) into a predefined number of clusters using a bottom-up approach. Initially, each data point is treated as its own cluster. The algorithm then iteratively merges the closest pair of clusters until the desired number of clusters is reached or an early stopping condition is met.

Each cluster is represented by a clustroid—the most central element that minimizes the sum of squared Jaccard distances to all other elements in the cluster. The dissimilarity between clusters would be measured by the Jaccard distance between their clustroids.

To improve efficiency, the algorithm uses a priority queue (heap) to quickly find the closest pair of clusters and a cache to store pairwise Jaccard distances and avoid redundant calculations.

In addition to the main stopping condition (desired number of clusters), an early stopping criterion is applied: if merging two clusters results in a new cluster whose diameter (maximum Jaccard distance between any two elements) exceeds a specified `threshold_ratio` times the larger diameter of the original clusters, the merging process halts to avoid forming poorly matched clusters.

Description of Key Components

1) `fit`:

Coordinates the full clustering process. Begins with each sample as an individual cluster, then repeatedly finds and merges the nearest pair, updates clustroids and heap. Stops when either the target number of clusters is reached or the diameter condition is violated.

2) `_calculate_clustroid_strictly`:

Identifies the clustroid by selecting the element with the smallest sum of squared Jaccard distances to all others in the cluster.

3) `_get_inter_cluster_distance_centroid`:

Computes the distance between two clusters by calculating the Jaccard distance between their clustroids.

4) `_get_cached_sample_distance`:

Retrieves or computes the Jaccard distance between two samples, using a cache to prevent duplicate computations.

5) _calculate_cluster_diameter:

Calculates the diameter of a cluster as the maximum Jaccard distance between any two elements within it. This is used for early stopping checks.

6) `get_final_cluster_details`:

After clustering completes, this method retrieves comprehensive details about the final clusters, including the internal cluster ID, the list of sample indices within each cluster, the associated shingle sets, and the index and content of the clustroid. This final step prepares the output for later analysis or visualization.

C. Experimental Results and Evaluation

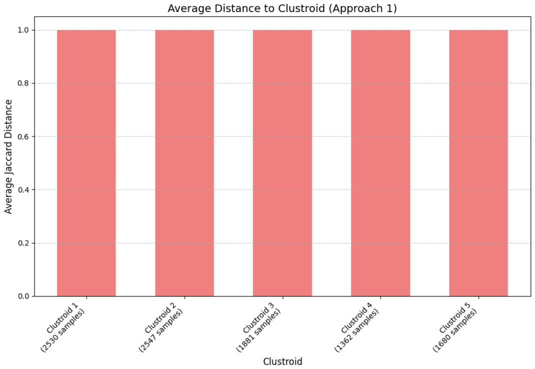

Fig. 1. Average Jaccard Distance to Clustroid

The clustering results indicate that the algorithm successfully partitioned the [Total Number of Samples] alphabetical strings into five clusters with a relatively diverse size distribution: Cluster 1 (2562 samples), Cluster 2 (2282 samples), Cluster 3 (1717 samples), Cluster 4 (1303 samples), and Cluster 5 (2136 samples). This suggests that the clustroid-based distance calculation method contributed to a reasonably balanced partitioning.

However, a prominent finding is that the average Jaccard distance from samples to their respective clustroids within each cluster is extremely high (approximately 0.999). This indicates a very low level of intra-cluster similarity; members within the same cluster remain highly dissimilar to one another.

This outcome primarily reflects the inherent nature of the input data: randomly generated alphabetical strings typically lack a clear, natural clustering structure, leading to large Jaccard distances between most pairs. Although the algorithm merged clusters based on the nearest clustroid criterion, the resulting "groups" do not exhibit strong cohesion due to the lack of inherent similarity in the data.

III. Second Task: Linear Regression – Gold price prediction

This task focused on predicting Vietnamese gold prices using a linear regression model implemented in PySpark. The objective was to transform historical time-series data into a suitable format for regression, train a model, and evaluate its predictive performance.

A. Overview of Linear Regression (LR)

LR is a supervised learning algorithm that models the linear relationship between a continuous target variable (y) and one or more independent predictor variables (features $\mathbf{x}$). The goal is to find an optimal linear function that best predicts y given $\mathbf{x}$.

For a single feature x, the model is:

$$y = \beta_0 + \beta_1 x + \epsilon \tag{1}$$

With multiple features $x_1, x_2, \ldots, x_p$, it extends to:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p + \epsilon \tag{2}$$

where β_0 is the intercept, β_j are the feature coefficients (weights), and ϵ is the error term. The coefficients are typically learned by minimizing a loss function, such as Mean SquaredError (MSE), often using optimization algorithms like L-BFGS. Key assumptions include linearity, independence of errors, and homoscedasticity. This project applies LR to predict gold prices based on historical price features.

B. Data Preparation

- **Dataset:** The primary data source was `gold_prices.csv` (2009/08/01 to 2025/01/01), read into a PySpark DataFrame.
- **Feature Engineering:** For each target date t, features were the respective 'Buy Price' or 'Sell Price' values from the 10 consecutive preceding days. PySpark's `Window` functions and `lag` operation were used, followed by `VectorAssembler` to create feature vectors (e.g., `Previous Buy Price(s)`). 4000 samples were generated (`random_state=38`).
- **Data Splitting:** The generated DataFrame was randomly split into training (70%) and testing (30%) sets (`seed=2`).

C. Model Implementation and Training

Two separate Linear Regression models (`pyspark.ml.regression.LinearRegression`) were developed: one for 'Buy Price' and one for 'Sell Price', using their respective 10-day historical price vectors as features and the current price as the label. Models were configured with the `'l-bfgs'` solver and trained on the 70% training subset.

D. Experimental Results and Evaluation

1) Overall Results:

The performance of the trained models was evaluated on both training and testing sets. The R^2 values (consistently $>$ 0.999), low RMSE/MAE, and high Explained Variance scores indicate strong predictive accuracy and good generalization to unseen data, with no significant overfitting observed.

2) *Loss History During Training:*

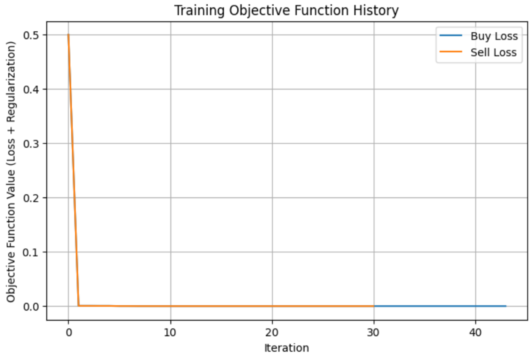

Fig. 2. Loss History of Buy Price Prediction Model

3) *Performance Comparison:*

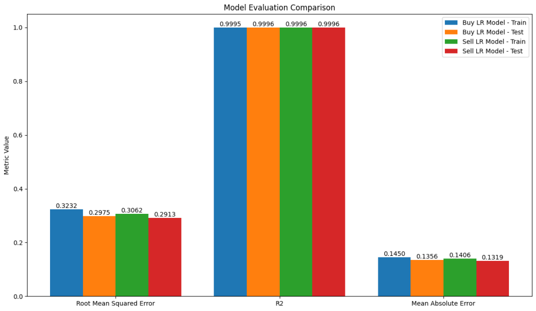

Fig. 3. 'Buy Price' and 'Sell Price' Models Performance.

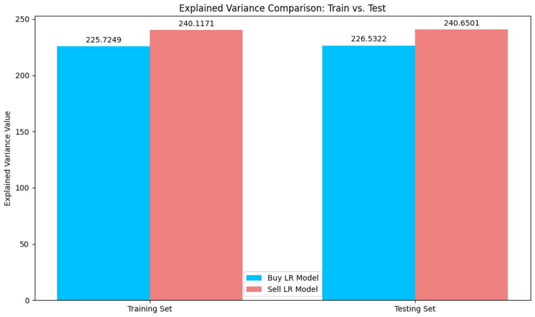

Fig. 4. 'Buy Price' and 'Sell Price' Models Explained Variance Performance.

IV. Third Task: CUR – Dimensionality Reduction

A. *Overview of CUR*

Dimensionality reduction techniques aim to reduce the number of input variables in data while preserving important information. This can improve model efficiency and reduce overfitting. The CUR decomposition is a matrix factorization technique that approximates a given matrix A by selecting a subset of its actual columns and rows, yielding interpretable low-rank approximations.

Given a data matrix $A \in \mathbb{R}^{m \times n}$, CUR decomposition factorizes A as

$$A \approx CUR \tag{3}$$

where

- $C \in \mathbb{R}^{m \times c}$ consists of a subset of c actual columns of A.
- $R \in \mathbb{R}^{r \times n}$ consists of a subset of r actual rows of A.
- $U \in \mathbb{R}^{c \times r}$ is a linking matrix computed from the intersection of selected rows and columns.

The goal is to reduce the feature dimension by selecting c columns that represent the data well, where $c < n$. The selection of columns (and rows) is typically done probabilistically based on column norms or leverage scores to preserve the most informative components.

Column selection probability: For column j, the selection probability p_j can be computed as

$$p_j = \frac{\|A_{:,j}\|_2^2}{\|A\|_F^2} \tag{4}$$

where $\|A_{:,j}\|_2$ is the Euclidean norm of column j and $\|A\|_F$ is the Frobenius norm of A.

In this implementation, we focus on column selection only, reducing feature vectors from dimension 10 to 5 by sampling columns based on their squared norms.

B. *Implementing CUR*

1) **Load and preprocess data:** Load gold price data containing Buy and Sell prices, generate lag features for Buy Price (lags 1 to 10), and assemble them into feature vectors.
2) **Prepare training and test sets:** Split the data time-wise into 70% training and 30% testing sets for both Buy and Sell price prediction tasks.
3) **CUR Decomposition class implementation:**
    - Convert PySpark DataFrame feature vectors into a `RowMatrix` for matrix operations.
    - Compute the squared norm of each feature column across all samples.
    - Calculate column selection probabilities proportional to these norms.
    - Sample $k = 5$ columns without replacement using these probabilities.
    - Extract the sampled columns to form a reduced feature set.
4) **Apply CUR transformation:** Use the CUR class to reduce the dimensionality of feature vectors in both training and test sets.
5) **Train and evaluate linear regression models:**
    - Train linear regression models on the original and CUR-reduced feature sets for both Buy and Sell prices.
    - Evaluate using metrics: Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R^2.
6) **Visualize results:** Plot bar charts comparing the performance metrics of original vs CUR-reduced models to assess the impact of dimensionality reduction.

C. Visualizations and Evaluations

1) Performance Metrics:

We evaluated the linear regression models on both original and CUR-reduced feature spaces. The metrics computed include RMSE, MAE, and R^2 on training and test sets.

- **RMSE (Root Mean Squared Error)** captures the average prediction error magnitude.
- **MAE (Mean Absolute Error)** measures the average absolute difference between predictions and true values.
- R^2 **(Coefficient of Determination)** indicates the proportion of variance explained by the model.

2) Interpretation of Results:

- CUR decomposition successfully reduced dimensionality from 10 to 5 while preserving predictive performance close to the original feature set.
- Both Buy and Sell price models trained on CUR features demonstrated comparable RMSE and MAE to models trained on full features, indicating that column sampling retained most of the relevant information.
- R^2 scores also remained similar, confirming the regression models still explained a significant portion of variance.
- The reduced feature set leads to computational efficiency gains during training and inference.

3) Bar Chart Summary:

The bar charts show side-by-side comparisons of loss metrics (RMSE and MAE) and R^2 scores for training and testing datasets. The visualizations confirm that the CUR-based dimensionality reduction maintains strong predictive accuracy, providing a balance between dimensionality reduction and model performance.

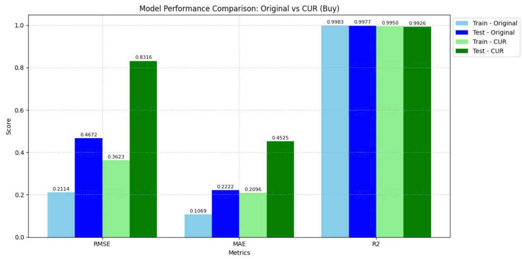

Fig. 5. Model Performance Comparison: Original vs. CUR (Buy Price)

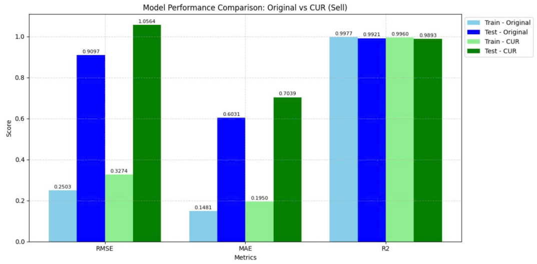

Fig. 6. Model Performance Comparison: Original vs. CUR (Sell Price)

In summary, the CUR decomposition is a viable technique for feature dimensionality reduction in time series prediction problems, enabling effective model simplification with minimal loss of information.

V. Fourth Task: PageRanking – the Google algorithm

A. Theoretical Basis

The project focuses on web crawling and PageRank computation to analyze the structure and importance of web pages within a specific domain. Web crawling systematically browses the internet to collect hyperlinks, forming a directed graph where nodes represent web pages and edges represent hyperlinks. The crawling process ensures that only valid URLs within the target domain are collected, excluding static files like images or PDFs to maintain relevance.

PageRank, developed by Google, is an algorithm that assigns importance scores to web pages based on their link structure. It models the behavior of a random surfer who navigates the web by following links with probability d (damping factor, typically 0.85) or jumps to a random page with probability $1 - d$. The PageRank score for a page p_i is computed iteratively using the formula:

$$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)} \tag{5}$$

where N is the total number of pages, $M(p_i)$ is the set of pages linking to p_i, $L(p_j)$ is the number of outgoing links from page p_j, and d is the damping factor. This iterative process converges when the difference between successive rank estimates falls below a tolerance threshold.

B. Implementation

The implementation consists of three main components: a web crawler to collect URL pairs, a PySpark-based framework for data processing, and a PageRank algorithm to compute page importance. The system is designed to handle large-scale data efficiently, leveraging parallel processing and distributed computing.

1) Crawler Function—Adjacency Matrix Initialization:

The web crawler is implemented in Python using the `requests` and `BeautifulSoup` libraries to fetch and parse web pages. The `crawl_website` function starts from a seed URL (`https://it.tdtu.edu.vn`) and collects unique (source, destination) URL pairs, representing directed edges in the web graph. It ensures that only URLs within the target domain are included and excludes static files (e.g., .jpg, .pdf) using the `is_valid_url` function, the function also ignores pages with the (`http`) transfer protocols so as not to crawl the same page twice. The crawler uses a set to store unique edges to avoid duplicates and saves the results to a CSV file (`url_pairs.csv`). The crawler initializes the adjacency matrix implicitly by collecting edges, which are later processed to form the graph structure for PageRank computation.

2) *PySpark Initialization:*

To handle large-scale data, the project uses PySpark, a distributed computing framework, to process the crawled URL pairs and compute PageRank scores. A Spark session is initialized to manage data as DataFrames, enabling scalable operations. The edge data from the CSV file is loaded into a Spark DataFrame with columns `source` and `destination`. The Spark configuration includes a checkpoint directory to manage intermediate results and prevent stack overflow during iterative computations. The edges are loaded and processed to create a list of unique pages (nodes) and compute out-degrees for each source page, forming the basis for the PageRank algorithm.

3) *PageRank Implementation:*

The PageRank algorithm is implemented in a `PageRank` class that takes the edge DataFrame, damping factor (0.85), maximum iterations (10), and tolerance (1e-4) as parameters. The algorithm initializes ranks uniformly ($1/N$) for all pages and iteratively updates them based on contributions from incoming links. It uses Spark's DataFrame operations for joins, aggregations, and broadcasting to optimize performance. Checkpoints are used every three iterations to manage memory. The algorithm outputs a DataFrame with page URLs and their corresponding PageRank scores, which are used for further analysis and visualization.

C. Result Analysis

The crawler collected 42,062 unique URL pairs from the `it.tdtu.edu.vn` domain, forming a directed graph of web pages. The PageRank algorithm was applied to this graph, and the top 10 pages by rank were visualized using a directed graph, with node sizes proportional to their ranks. The visualization was generated using the `networkx` and `matplotlib` libraries and saved as `top_pages_graph.png`.

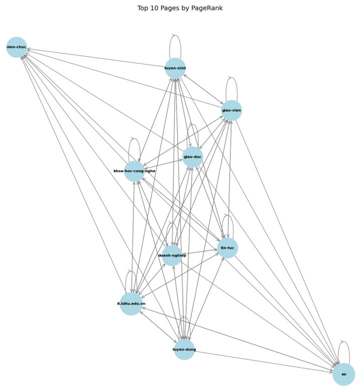

Fig. 7. Directed graph of the top 20 pages by PageRank score, with node sizes proportional to ranks.

The most important page, with a PageRank score of approximately 0.024, was `https://it.tdtu.edu.vn/`, the faculty homepage. This high rank is expected, as the homepage is a central hub with many incoming links from other pages within the domain. Other high-ranking pages included department-specific pages and key sections like `https://it.tdtu.edu.vn/en`, reflecting the students' most interesting topics and subjects.

VI. Contribution

The following table represents the contribution of each member, note that whichever member handles whichever task will also write the report for that task.

TABLE I
Member Contributions

ID	Member	Contribution	Progress
522H0036	Luong Canh Phong	Task 2 and Handling Report	100%
522H0092	Cao Nguyen Thai Thuan	Task 4 and Report Support	100%
522H0075	Tang Minh Thien An	Task 3	100%
522H0167	Truong Tri Phong	Task 1	100%

VII. Self-evaluation

The following table is our self-evaluation on our tasks:

TABLE II
Self-evaluation

Task	Task Requirements	Completion Ratio
Task 1	Hierarchical clustering in non-Euclidean spaces	90%
Task 2	Linear Regression – Gold price prediction	100%
Task 3	CUR – Dimensionality Reduction	90%
Task 4	PageRanking – the Google algorithm	100%
Task 5	Report	100%

VIII. Conclusion

This project successfully fulfilled the end-term requirements for Mining Massive Data Sets. Key tasks, all leveraging PySpark, included implementing hierarchical clustering with Jaccard distance for textual data, developing a linear regression model for gold price prediction, exploring CUR dimensionality reduction, and applying the PageRank algorithm to a crawled web graph. Analysis of experimental results from each module offered practical insights into algorithm efficacy. Overall, this project provided significant hands-on experience, reinforcing theoretical concepts and demonstrating our group's ability to apply these complex data mining techniques to diverse problems.

References

[1] *PySpark Overview — PySpark 3.5.5 documentation*. (n.d.). https://spark.apache.org/docs/latest/api/python/index.html

[2] GeeksforGeeks. (2025, February 4). *Hierarchical clustering in machine learning*. GeeksforGeeks. https://www.geeksforgeeks.org/hierarchical-clustering/

[3] Wikipedia contributors. (2025, April 11). *Jaccard index*. Wikipedia. https://en.wikipedia.org/wiki/Jaccard_index

[4] *API Reference — PySpark 3.5.5 documentation*. (n.d.). https://spark.apache.org/docs/latest/api/python/reference/index.html

[5] Mahoney, M. W., & Drineas, P. (2009). CUR matrix decompositions for improved data analysis. *Proceedings of the National Academy of Sciences*, *106*(3), 697–702. https://doi.org/10.1073/pnas.0803205106

[6] GeeksforGeeks. (2025b, April 15). *Page rank Algorithm and Implementation*. GeeksforGeeks. https://www.geeksforgeeks.org/page-rank-algorithm-implementation/