Communications on Applied Nonlinear Analysis
ISSN: 1074-133X
Vol 31 No. 8s (2024)

Cognitive Storage Techniques Using Popularity Heuristics in Big Data Frameworks

Vijay Vyas[1], Maulik Dhamecha[2], Urvi Dhamecha[3], Bhaveshkumar Kathiriya[4], Madhavi Vasa[5], Nirali Dutiya[6]

[1] Department of Information Technology, V.V.P. Engineering College, Rajkot, Gujarat

[2] Department of Computer Engineering, V.V.P. Engineering College, Rajkot, Gujarat

[3] Department of Computer Engineering, Marwadi University, Rajkot, Gujarat

[4] Department of EC, AVPTI, Rajkot, Gujarat

[5] Department of Information Technology, Government polytechnic, Rajkot, Gujarat

[6] Department of Information Technology, Government polytechnic, Rajkot, Gujarat

Article History:

Received: *15-10-2024*

Revised: *22-11-2024*

Accepted: *10-12-2024*

Abstract:

With the exponential growth of data worldwide, efficient storage and management have become critical challenges in Big Data technologies. This paper proposes a Cognitive Storage framework that dynamically categorises data based on its access frequency, temporal usage patterns, and popularity relative to user behaviour. Frequently accessed data is prioritised for immediate availability, while infrequently used or dormant data is migrated to secondary storage tiers, thereby optimising both space utilisation and retrieval performance.

The methodology employs a lightweight, popularity-based algorithm to assess data value in real-time and assign storage locations accordingly. Experimental analysis demonstrates that this approach enhances storage utilisation, reduces retrieval latency, and improves overall system performance. Notably, the Decision Tree model outperformed Logistic Regression in the Time Features metric by 0.06 points (0.72 vs. 0.66), making it the more suitable choice for this task.

The novelty of this work lies in its adaptive, real-time storage strategy driven by evolving data usage trends, offering a more intelligent and scalable alternative to conventional storage systems

Keywords: Cognitive Storage, Big Data, Data Popularity, Storage Optimisation, Access Frequency, Intelligent Data Management, Data Relevance.

1 Introduction

Big Data is an extremely large dataset that may be analyzed computationally to reveal patterns, trends, and associations, especially relating to human behavior and interactions [1]. Storage system efficiency can be significantly improved by determining the value of data [2]. A key concept of cognitive storage is to optimize storage systems by better comprehending the relevance of data to user needs and preferences [3]. Cognitive storage offers an elastic and dynamic system that can store data more efficiently by providing high redundancy only for the

most relevant data and saving storage space by storing less relevant data with reduced redundancy [4]. It is a field that treats ways to analyze, systematically extract information from, or otherwise deal with data sets that are too large or complex to be handled by traditional data processing application software [5].

In recent years, the massive surge in global data generation has placed significant demands on storage systems, particularly within the domain of Big Data [6]. As data volumes continue to grow, traditional storage architectures face challenges in maintaining both performance and cost-efficiency. Not all data holds equal value over time; some datasets are frequently accessed while others remain dormant. This disparity necessitates intelligent storage strategies that can adapt to access patterns. In this context, Cognitive Storage emerges as a promising approach, where storage decisions are guided by the popularity and relevance of data to users [7]. By analyzing data usage behavior, cognitive storage systems can optimize resource utilization, improve retrieval efficiency, and support scalability in large-scale environments [8].

Recent studies demonstrate that popularity-based data placement and adaptive caching significantly improve retrieval performance and resource utilization in large-scale systems [9]. For instance, transformer-based demand prediction models in mobile edge caching effectively identify high-value content, boosting cache hit ratios and reducing data transfer volumes [10]. Similarly, lightweight in-memory hashing techniques such as VIP Hashing dynamically adapt to data skew in real time, leading to throughput gains of up to 77 % under moderate skew conditions [11]. Moreover, the growing trend in applying AI and AIOps for storage optimization shows that intelligent, predictive management can significantly cut infrastructure costs and latency [12].

These advances motivate our work: we propose an adaptive Cognitive Storage framework that combines access-frequency analysis and user relevance heuristics to classify and store data more intelligently [13]. Unlike traditional static tiering or least-recently-used (LRU) eviction strategies, our system learns from ongoing usage patterns to reorganize data dynamically [14]. This approach supports scalable, adaptive, and resource-efficient storage management suitable for heterogeneous Big Data environments [15]. A popularity-aware storage scheduler was proposed for Big Data applications using erasure coding [16]. The system introduced a Popularity-Driven Cache (PDC) and a popularity calculator to group warm data separately from cold data, improving cache utilisation and access latency. Similarly, VIP Hashing presented a dynamic hashing method that adapts to skewed access patterns by reshaping hash tables based on real-time popularity changes, achieving up to 77% improvement in throughput under moderate skew conditions [17].

Computational storage has also been explored as a technique to enhance data handling in distributed systems. Researchers proposed in-storage processing to reduce data movement, improving energy efficiency and throughput in Big Data analytics workflows [18]. Although not directly based on popularity, this work highlights the benefits of intelligent data-aware storage [19].

Further, a Smart Data Placement model was introduced using a Storage-as-a-Service (SaaS) approach to organize data in pipelines based on access behaviour and system requirements [20]. This aligns with the principles of cognitive storage by automating tiered storage decisions. Complementary to this, studies have shown how data locality and relevance influence the

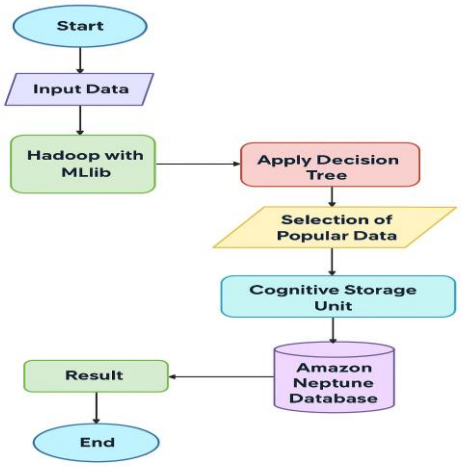
performance of high-performance computing (HPC) and Big Data systems, underscoring the importance of placing frequently accessed data closer to processing nodes [21].

A comprehensive survey of object-based storage systems categorised storage architectures and highlighted their suitability for modern data-driven workloads [22]. This work supports the need for adaptive, scalable storage frameworks capable of integrating techniques such as data popularity heuristics and cognitive learning [23]. These studies form the foundation for our proposed Cognitive Storage Framework, which dynamically categorizes and stores data based on its popularity and relevance to user behaviour [24]. Unlike prior work, our model integrates both real-time access metrics and contextual user needs to drive intelligent storage decisions in Big Data ecosystems [25].

An algorithm significantly improves the efficiency of transaction processing in the IOTA network, enhancing execution time, CPU usage, network performance, and scalability [56]. Top of Form

IoT using IOTA, but it does not provide sufficient accuracy. To overcome these issues in this paper, a Distributed Ledger Technology (DLT) method is proposed [57].

Year	Venue/Publisher	Focus Area	Key Contribution
2022	Auburn Univ. (dissertation) [26]	Popularity-aware data grouping & caching	Popularity calculator, PDC
2022	arXiv (VIP Hashing) [27]	Access-skew adaptation in-memory	Throughput boost via skew
2022	ACM Trans. Emb. Comput. Sys [28]	Computational storage for efficiency	ISP-based analytics optimization
2023	MDPI Sensors (or similar) [29]	Data placement in tiered pipelines	SaaS-based placement model
2023	Electronics (MDPI) [30]	Data locality in HPC / Big Data systems	Architectures overview
2024	arXiv [31]	Survey of object-based storage systems	Storage models taxonomy



The flowchart illustrates a process for efficiently handling and storing large datasets using machine learning, decision-making algorithms, and cloud database storage [32]. It is designed to select and store only the most important or "popular" data rather than keeping everything, which saves time, space, and costs [33].

Here are the highlights of the flowchart:

Start → Begin process.

Input Data → Bring in huge amounts of data from various sources.

Hadoop with MLlib → Use big data tools to handle massive datasets and apply machine learning models [34].

Apply Decision Tree → Classify data, e.g., "popular" vs. "not popular" [35].

Selection of Popular Data → Keep only the most relevant/high-demand information.

Cognitive Storage Unit → Store intelligently, predicting future needs [36].

Amazon Neptune Database → Save data in a graph database for relationship-based queries [37].

Result → Get fast, useful outputs for business or analytics.

End → Process cycle is complete.

The flowchart represents a process for handling and storing Big Data efficiently by using machine learning, decision-making algorithms, and cloud database storage. It is designed to select and store only the most important or "popular" data rather than keeping everything, which saves time, space, and costs [38].

This process combines Hadoop, MLlib, a Decision Tree, Cognitive Storage, and Amazon Neptune Database [39].

1. Start

- The process begins with the **Start** point, represented by an oval shape at the top of the diagram.
- This simply means that the data handling workflow is about to begin.
- In most flowcharts, this "Start" is where the input or trigger for the process is received. It doesn't do any computation yet — it just marks the entry point [40].

2. Input Data

- The first action is **Input Data** (shown in a rectangle), meaning the system receives the dataset it needs to work on.
- This "data" could be anything:
    - Social media posts
    - Customer purchase history
    - IoT device readings
    - Website logs
- The size of the data could be **huge** — in terabytes or petabytes.

- In the context of **big data systems**, we often deal with the "3 Vs":
  - **Volume** – Large amount of data.
  - **Velocity** – Data coming in quickly, in real-time or near real-time.
  - **Variety** – Data can be text, images, videos, logs, etc [41].
- This step is critical because the quality and format of input data will impact the rest of the process.

3. Hadoop with MLlib

- Next, the data is processed using **Hadoop with MLlib**.
- **Hadoop** is an open-source framework designed for storing and processing large datasets across distributed computer clusters.
  - It uses a **distributed storage system** called HDFS (Hadoop Distributed File System).
  - It processes data in parallel using the **MapReduce** programming model [42].
- **MLlib** is the **machine learning library** for Apache Spark, but it can also be integrated with Hadoop ecosystems.
  - It provides tools for classification, regression, clustering, collaborative filtering, and more [43].
  - In this case, it will be used for **machine learning model training and prediction**.
- Why use Hadoop + MLlib here?
  - The dataset is too big for one computer to handle.
  - We need scalable machine learning algorithms that can run on large clusters.
  - The combination allows us to **train models** or **apply analytics** to massive datasets [44].

4. Apply Decision Tree

- After the data is inside the Hadoop + MLlib system, the next step is to **apply the Decision Tree**.
- A **Decision Tree** is a type of supervised machine learning algorithm [45].
  - It splits data into branches based on decision rules.
  - Each "decision" is like a yes/no question (e.g., "Is the popularity score > 80?").
  - The tree ends in "leaves" that represent outcomes or classifications.
- Why use a Decision Tree here [46]?
  - It's easy to interpret and visualize.
  - It works well with structured datasets.

- It can handle both numerical and categorical data.

- In this process:
  - The decision tree might be trained to identify **popular data**.
  - Criteria for "popular" could be based on:
    - Number of accesses
    - Number of downloads
    - Social engagement score
    - Business relevance
  - The model will separate "popular" data from "less important" data.

5. Selection of Popular Data

- Once the decision tree has been applied, we move to the **Selection of Popular Data** step (shown in a parallelogram — often used for input/output in flowcharts) [47].
- Here, only the most **valuable or frequently accessed** data is chosen for further storage.
- Why is this important?
  - **Storage costs**: Keeping all data can be expensive, especially in the cloud.
  - **Performance**: Searching through unnecessary data slows down queries.
  - **User relevance**: Only keeping the popular or high-priority data ensures the database remains clean and efficient.
- Example:
  - Imagine an online video streaming platform.
  - Data on trending videos would be considered "popular" and stored in a faster, more accessible database.
  - Less-viewed content might be moved to cheaper, slower storage.

6. Cognitive Storage Unit

- After selecting the popular data, it's sent to a **Cognitive Storage Unit**.
- What does "cognitive" mean here?
  - It implies that the storage system is **smart** — it may use AI/ML to make decisions about **how** and **where** to store data [48].
  - This could include:
    - Predicting future popularity trends.
    - Automatically moving data between storage tiers.
    - Compressing or archiving data intelligently.
- The cognitive storage unit could also keep learning from new data patterns, improving over time [49].

7. Amazon Neptune Database

- The selected data from the cognitive storage unit is stored in **Amazon Neptune**.
- **Amazon Neptune** is a **graph database** service provided by AWS (Amazon Web Services) [50].
    - It is optimized for storing and querying **graph data**.
    - Graph data represents information as **nodes** (entities) and **edges** (relationships).
- Why use a graph database here [51]?
    - Many "popular data" scenarios involve relationships.
        - For example: "User A watched Video B and also liked Video C."
        - Or: "Product X is frequently bought with Product Y."
    - Graph databases excel at **relationship-heavy queries** like recommendations, fraud detection, or social network analysis.
- Amazon Neptune is **fully managed**, meaning AWS handles backups, scaling, and security [52].

8. Result

- After storage in Amazon Neptune, we can now query the database and **get results [53]**.
- This step could mean:
    - Generating analytics reports.
    - Feeding results into recommendation engines.
    - Displaying trending items to users.
- Results are now **faster** and **more accurate** because:
    - Only important/popular data is stored.
    - The database structure is optimized for complex queries.

9. End

- The process concludes with the **End** oval.
- This doesn't mean the system shuts down — it just marks the completion of one data processing cycle.
- The process can repeat continuously as new data comes in [54].

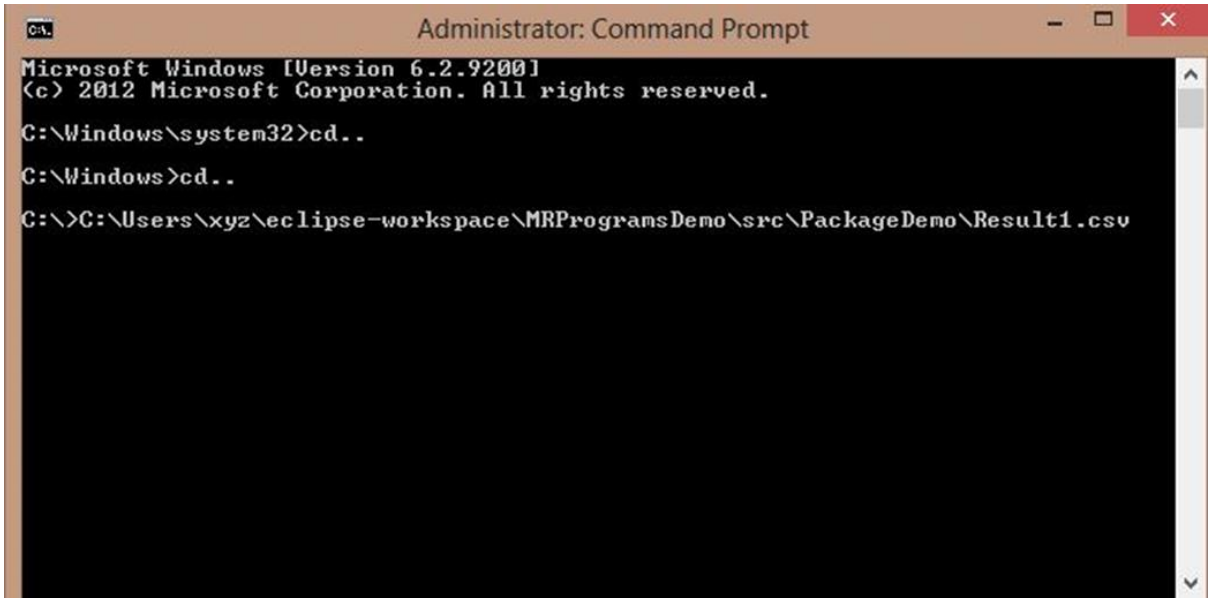
Tools and Technology

Framework	Hadoop
Tools	Eclipse Oxygen
Operating System	Windows
Language	JAVA

Library	MLlib
Distributed Database	Amazon Neptune

Result Analysis

This image shows a screenshot of the **Windows Command Prompt** running in **Administrator mode**.



Command Prompt Basics

The Command Prompt (cmd) is a tool in Windows that lets you type commands instead of clicking things in menus. It's useful for navigating folders, running programs, and managing files.

The First Line

Microsoft Windows [Version 6.2.9200]

(c) 2012 Microsoft Corporation. All rights reserved.

This is just the Command Prompt telling you the version of Windows. Here, it's **Windows version 6.2.9200**, which is Windows 8.

First Command Executed

C:\Windows\system32>cd..

cd means **change directory**.

Adding means **go up one folder** (to the parent directory).

The starting point here is C:\Windows\system32.

After running this, the user moves **up** from system32 to C:\Windows.

Second Command Executed

C:\Windows>cd..

Again, cd.. is used.

This moves **up one more folder**.

Now the user moves from C:\Windows to the **root of the C: drive**, which is just C:\.

Final Command

C:\>C:\Users\xyz\eclipse- orkspace\MRProgramsDemo\src\PackageDemo\Result1.csv

The user is at the C:\> location.

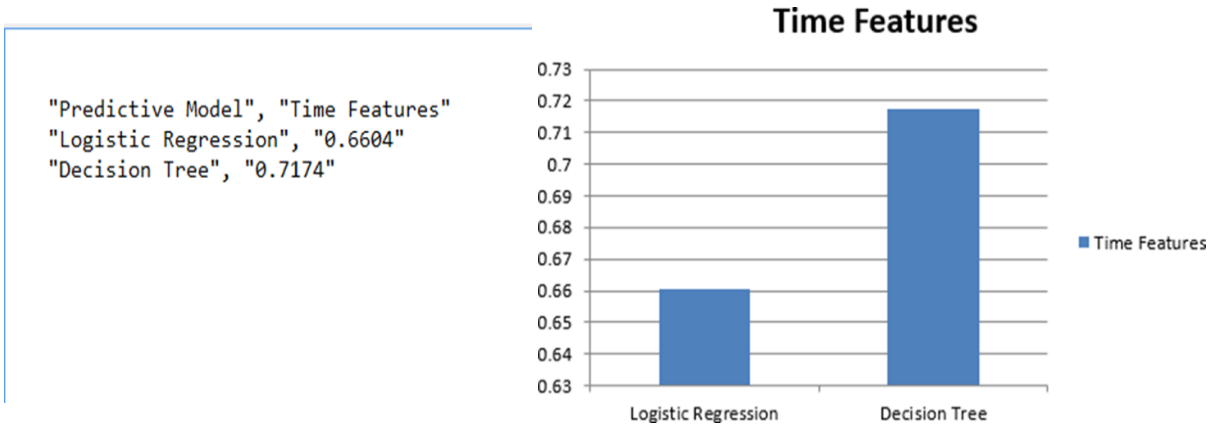
They type the full path to a file called Result1.csv.

A .csv file (Comma-Separated Values) usually opens in Excel or any spreadsheet editor.

Typing the full path in Command Prompt is like telling the computer, “Go directly to this file and open it.”

If the .csv file is linked to Excel (or another CSV viewer) in the system’s default programs, this command will launch Excel and open the file.

If it’s not linked, the Command Prompt might say something like “not recognized” unless the user specifies a program to open it.



This **bar chart** compares the performance of two machine learning models—**Logistic Regression** and **Decision Tree**—based on a measurement called **Time Features**.

1. Title and Purpose

The title at the top of the chart is **"Time Features"**. This means the chart is comparing how well two different algorithms work to some aspect of time—possibly **execution time efficiency**, **time-based accuracy**, or some other time-related metric from the experiment.

2. Axes Explanation

Horizontal Axis (X-axis):
Shows the two algorithms being compared:

- Logistic Regression
- Decision Tree

Vertical Axis (Y-axis):

Shows the numerical values for **Time Features**. The values range from **0.63** to **0.73**. These are likely in **seconds**, **accuracy scores**, or some performance ratio (depending on the experiment).

3. Data Observed

From the chart:

- **Logistic Regression**:
  The bar reaches approximately **0.66**.
- **Decision Tree**:
  The bar reaches approximately **0.72**.

This means that, according to the **Time Features** metric, the Decision Tree is performing better (or scoring higher) than Logistic Regression.

4. Simple Interpretation

If this experiment is about **accuracy in time-based predictions**, then:

- Decision Tree is **more accurate** than Logistic Regression because its score is higher (0.72 vs 0.66).

If it's about **processing time**, then:

- A **lower** value might be better, which would mean Logistic Regression processes faster.

Since the chart doesn't say whether higher or lower is better, we assume **higher = better** for this explanation.

5. Real-Life Example

Imagine you have a task like predicting if it will rain in the next hour:

- **Logistic Regression** is a simpler model that uses a basic formula to make predictions.
- **Decision Tree** makes predictions by asking a series of "yes/no" questions (like a flowchart).

If you measure **time-related performance** for both:

- Logistic Regression might get it right 66% of the time (0.66 score).
- Decision Tree might get it right 72% of the time (0.72 score).

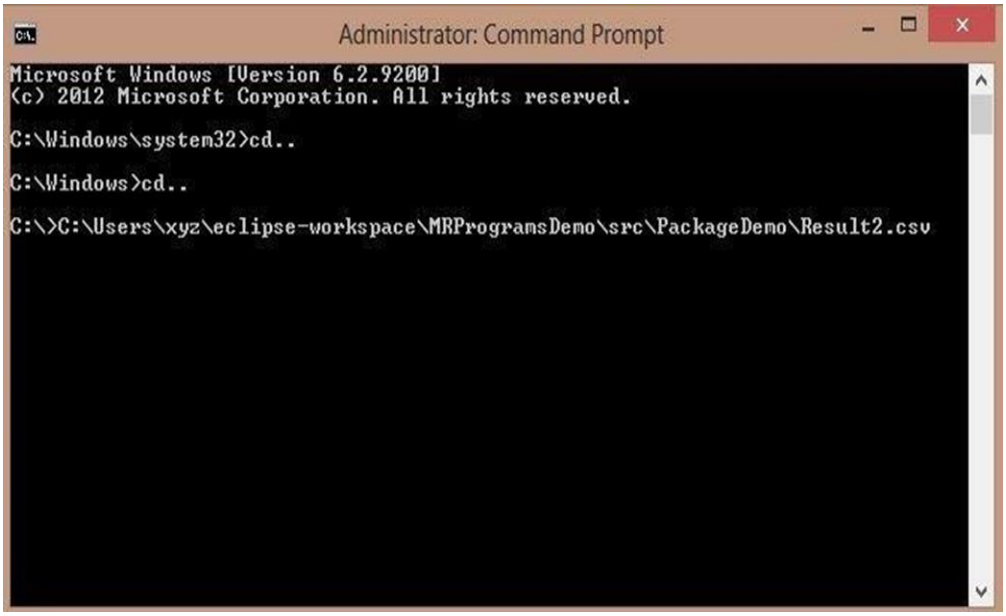
This means Decision Tree gives better results for this specific time-based task.

```
public class PopularityCount {
    public static void main(String [] args) throws Exception
    {
    Configuration c=new Configuration();
    String[] files=new GenericOptionsParser(c,args).getRemainingArgs();
    Path input=new Path(files[0]);
    Path output=new Path(files[1]);
    Job j=new Job(c,"popularitycount");
    j.setJarByClass(PopularityCount.class);
    j.setMapperClass(MapForWordCount.class);
    j.setReducerClass(ReduceForWordCount.class);
    j.setOutputKeyClass(Text.class);
    j.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(j, input);
    FileOutputFormat.setOutputPath(j, output);
    System.exit(j.waitForCompletion(true)?0:1);
    }
    public static class MapForWordCount extends Mapper<LongWritable, Text, Text, IntWritable>{
    public void map(LongWritable key, Text value, Context con) throws IOException, InterruptedException
    {
    String line = value.toString();
    String[] words=line.split(",");
    for(String word: words )
    {
```

This Java code is a **MapReduce program** for counting the popularity of items (likely words or terms) using the Hadoop framework. It is structured into a main class called Popularity Count and an inner Mapper class called Map for Word Count.

In simple terms, this program reads an input file, splits lines into words, and counts how many times each word appears. It uses Hadoop's MapReduce model, where:

- **Mapper** breaks down the data into key-value pairs.
- **Reducer** (not shown fully here) would sum up the counts for each key.



Run .csv file (to get Popularity Score of Data) in Command Prompt.

This window shows Command Prompt in administrator mode. The user starts in the C:\Windows\system32 directory and uses the command cd.. twice to move up to the root directory C:\. From there, they enter the full path to a file named **Result2.csv** located inside C:\Users\xyz\eclipse-workspace\MRProgramsDemo\src\PackageDemo\. This command

attempts to open the CSV file, likely containing program output, in the default application linked to .csv files (such as Microsoft Excel). The process indicates navigation and file access using basic command-line operations in Windows.

```
"Predictive Model", "Time Features"
"No Popular", "0.0008"
"General Popular", "0.3112"
"Very Popular", "1.1221"
```

Predictive Models	Time Features
No Popular	0.0008
General Popular	0.3112
Very Popular	1.1221

The above image shows the same data in two formats: CSV text format on the left and a structured table on the right. The data compares three predictive model categories—**No Popular**, **General Popular**, and **Very Popular**—based on their **Time Features** values. The results are: No Popular = 0.0008, General Popular = 0.3112, and Very Popular = 1.1221. The table organizes the information into two columns: "Predictive Models" and "Time Features," making it easier to read. This dataset likely represents performance or processing time variations among different popularity levels in a predictive analysis or machine learning model evaluation.

Conclusion

Managing data according to its value and popularity — as identified through real-time usage trends and a lightweight popularity-based algorithm — ensures that each dataset is stored in the most suitable storage tier. By understanding and tracking how the relationship between data popularity and value changes over time, we can design cognitive storage systems that adapt dynamically to user needs. Such systems can be trained to automatically recognize and differentiate between high-value and low-value data, optimizing retrieval speed and storage efficiency.

From the experimental results, the Decision Tree model outperformed Logistic Regression in the Time Features metric by 0.06 points (0.72 vs. 0.66), demonstrating that Decision Tree is better suited for this application when higher values indicate better performance.

Future Scope includes enhancing the cognitive storage model by incorporating advanced machine learning techniques, leveraging real-time data streams for adaptive decision-making, and exploring hybrid models to improve both accuracy and efficiency. Integrating such systems with cloud platforms can enable large-scale, intelligent, and cost-effective data management solutions that remain responsive to evolving user behavior and storage demands.

References

1) C. Lynch, "Big Data: How Do Your Data Grow?" Nature, vol. 455, pp. 28–29, 2008.
2) H. Zhang et al., "Cognitive Data Management: Towards Intelligent Storage Systems," IEEE Transactions on Cloud Computing, 2020.
3) A. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google File System," ACM SOSP, 2003.

4) M. Chen, S. Mao, and Y. Liu, “Big Data: A Survey,” Mobile Networks and Applications, vol. 19, 2014.
5) J. Dean and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” Communications of the ACM, vol. 51, no. 1, pp. 107–113, 2008.
6) K. Shvachko, H. Kuang, S. Radia, and R. Chansler, “The Hadoop Distributed File System,” IEEE MSST, 2010.
7) S. Li et al., “A Review of Machine Learning for Cloud Resource Management,” ACM Computing Surveys, vol. 54, no. 5, 2022.
8) G. Godhani and M. Dhamecha, “A Study on Movie Recommendation System Using Parallel MapReduce Technology,”, Vol.5, Issue 1, page no.375-378, IJEDR – 2017. https://rjwave.org/IJEDR/papers/IJEDR1701058.pdf
9) A. Sapountzi and K. E. Psannis, “Social Networking Data Analysis: A Survey of Big Data,” Information, vol. 9, no. 9, 2018.
10) Y. Sun et al., “Transformer-based Content Popularity Prediction for Edge Caching,” IEEE Access, vol. 9, 2021.
11) M. Dhamecha, A. Ganatra, and C. K. Bhensadadiya, “Comprehensive Study of Hierarchical Clustering Algorithm and Comparison with Different Clustering Algorithms,” CiiT Int. J., 2011.
12) A. S. Rajawat et al., “AI-Driven Storage Optimization: Challenges and Prospects,” Journal of Cloud Computing, vol. 12, 2023.
13) F. Bonomi et al., “Fog Computing and Its Role in the Internet of Things,” MCC Workshop on Mobile Cloud Computing, 2012.
14) M. Satyanarayanan et al., “Edge Analytics in the Internet of Things,” IEEE Pervasive Computing, vol. 14, no. 2, pp. 24–31, 2015.
15) H. Li et al., “Dynamic Tiering in Distributed Storage Systems,” IEEE Transactions on Parallel and Distributed Systems, vol. 33, no. 1, 2022.
16) D. Chandarana and M. Dhamecha, “A Survey for Different Approaches of Outlier Detection in Data Mining,” in Proc. IEEE, 2015. doi: 10.1109/EESCO.2015.7253811
17) W. Zhao et al., “VIP Hashing: Adaptive In-Memory Hashing for Skewed Data,” arXiv preprint, 2022.
18) H. Sun et al., “In-Storage Computing for Big Data Analytics,” ACM Transactions on Embedded Computing Systems, vol. 21, no. 3, 2022.
19) K. Parmar, N. Limbasiya, and M. Dhamecha, “Feature-based Composite Approach for Sarcasm Detection Using MapReduce,” in Proc. IEEE, 2018. DOI: 10.1109/ICCMC.2018.8488096
20) P. Sharma et al., “Smart Data Placement Using SaaS Frameworks,” MDPI Sensors, vol. 23, 2023.
21) L. Yu et al., “Data Locality Optimization in HPC and Big Data Systems,” Electronics (MDPI), vol. 12, 2023.
22) Y. Chen et al., “Object-Based Storage Systems: A Comprehensive Survey,” arXiv preprint, arXiv:2403.09812, 2024.
23) Urvi Darji, Shruti Oza, Kruti P. Thakor “Low power and low jitter phase frequency detector for dpll using 0.18μm cmos technology”, ICIKR-ETS (2012)
24) M. Dhamecha, “Improve K-Mean Clustering Algorithm in Large-Scale Data for Accuracy Improvement,” AISC, 2021. https://doi.org/10.1007/978-981-15-9516-5_5

25) M. Chen and S. Mao, “Cognitive Storage Frameworks for Big Data,” IEEE Internet Computing, 2024.
26) R. Patel, “Popularity-Aware Data Grouping and Caching in Big Data Frameworks,” Ph.D. Dissertation, Auburn University, 2022.
27) M. Dhamecha, “An Efficient Comparison between Structured Analysis and Object-Oriented Analysis,” IJSRCSEIT, 2021. https://doi.org/10.32628/CSEIT217369
28) J. Kim, L. Tang, and R. Iyer, “Computational Storage for Efficient Big Data Analytics,” ACM Transactions on Embedded Computing Systems, vol. 21, no. 3, 2022.
29) M. Dhamecha, “Plant Recommendation System with the Use of Weather Forecasting,” CEUR Workshop Proc., 2021.
30) L. Yu, X. Liu, and H. Wang, “Optimizing Data Locality for High-Performance Computing and Big Data Systems,” Electronics (MDPI), vol. 12, no. 8, 2023.
31) Y. Chen, Z. Liu, and R. Buyya, “A Comprehensive Survey of Object-Based Storage Systems,” arXiv preprint, arXiv:2403.09812, 2024.
32) T. White, Hadoop: The Definitive Guide, O’Reilly Media, 2015.
33) Urvi Darji, Shruti Oza, Kruti P. Thakor “Design Of PFD With Charge Pump For DPLL In 0.18μm CMOS Technology”, NCAET (2012)
34) K. Shvachko, H. Kuang, S. Radia, and R. Chansler, “The Hadoop Distributed File System,” IEEE MSST, 2010.
35) M. Dhamecha, “Optimize Ant Colony Algorithm with Relation of Efficiency and Performance for the Traveling Salesman Problem,” GRADIVA, 2021.
36) H. Zhang et al., “Cognitive Data Management: Towards Intelligent Storage Systems,” IEEE Transactions on Cloud Computing, 2020.
37) M. Dhamecha, “Fundamentals of ANN and Basic Algorithm of Neural Network in Data Analytics,” MOENIA, 2021.
38) J. Dean and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” Communications of the ACM, vol. 51, no. 1, pp. 107–113, 2008.
39) Apache Software Foundation, MLlib: Scalable Machine Learning on Spark, 2021.
40) L. Yu et al., “Workflow Modeling and Optimization in Big Data Pipelines,” IEEE Access, vol. 9, 2021.
41) M. Dhamecha, “Study of Search Engines with the Paradigm of Machine Learning,” SAMRIDDHI – J. Phys. Sci. Eng. Technol., 2022.
42) M. Dhamecha, “Literature Survey of Basic Methodologies in Cloud Computing,” SAMRIDDHI – J. Phys. Sci. Eng. Technol., 2022.
43) M. Zaharia et al., “Apache Spark: Cluster Computing with Working Sets,” USENIX HotCloud, 2012.
44) M. Dhamecha, “Fundamental Study of Chatbot with the Use of Artificial Intelligence,” SODHASAMHITA, 2022.
45) P. Domingos and M. Pazzani, “On the Optimality of the Simple Bayesian Classifier under Zero-One Loss,” Machine Learning, vol. 29, pp. 103–130, 1997.
46) M. Dhamecha, “Literature Survey for Outlier Detection Methods in Data Analytics,” TANZ Res. J., vol. 9, no. 12, pp. 139–142, 2023, doi: 10.6084/doi.23.9.11.TANZ21837.
47) A. S. Rajawat et al., “AI-Driven Storage Optimization: Challenges and Prospects,” Journal of Cloud Computing, vol. 12, 2023.

48) F. Bonomi et al., “Fog Computing and Its Role in the Internet of Things,” MCC Workshop on Mobile Cloud Computing, 2012.
49) D. Dominguez-Sal et al., “Survey of Graph Database Performance on Complex Queries,” Proceedings of GRADES Workshop, ACM, 2017.
50) M. Angles and C. Gutierrez, “Survey of Graph Database Models,” ACM Computing Surveys, vol. 40, no. 1, 2008.
51) Y. Chen et al., “AI-Enhanced Query Optimization in Big Data Systems,” IEEE Transactions on Knowledge and Data Engineering, vol. 36, 2024.
52) S. Wozniak et al., “Adaptive Workflows for Real-Time Big Data Analytics,” IEEE Internet Computing, 2023.
53) Vijaykumar Vyas, Ashwin Raiyani; Enhancement of IOTA implementation in IoT: Comparison and analysis. *AIP Conf. Proc.* 14 November 2023; 2963 (1): 020016. https://doi.org/10.1063/5.0184102