predictHD : Using Machine Learning to Predict Hard Disk Failures

Abhishek Bhardwaj
SJSU ID: 017418648
Department of Computer Science
San José State University
San José, CA
abhishek.bhardwaj@sjsu.edu

Aniruddha Prabhash Chakravarty
SJSU ID: 017102306
Department of Computer Science
San José State University
San José, CA
aniruddhaprabhash.chakravarty@sjsu.edu

***Abstract*—The predictHD project exemplifies cutting-edge research in the field of predictive analytics, focusing on enhancing HDD failure predictions using a comprehensive dataset from BackBlaze spanning the years 2013 to 2023. Employing advanced data processing techniques and a hybrid of machine learning models, the project aims to not only predict failures with high accuracy but also to provide actionable insights that could revolutionize maintenance strategies in data centers. This report details the project's extensive methodologies, from data acquisition to model deployment, and discusses the theoretical frameworks and practical applications of the predictive models developed.**

***Index Terms*—HDD failure prediction, predictive analytics, machine learning, Convolutional Autoencoder (CAE), Site Reliability, SMART metrics, TF-IDF vectorization, BigQuery, data preprocessing, BackBlaze dataset, MixTral transformer model**

I. Background & Dataset Overview

The project utilizes the BackBlaze dataset, which is comprehensive, containing detailed operational data of thousands of HDDs monitored over a decade. This dataset includes granular daily recordings of SMART metrics, offering a unique dataset for temporal and predictive analysis.

II. Data Preparation & Methodology

The project's methodological approach is multi-faceted:

A. Data Acquisition and Integrity

Initial stages involved securing data integrity through rigorous cleaning protocols to handle anomalies and outliers in the dataset.

B. Normalization & Feature Engineering

1) zscore.py: The `zscore.py` script standardizes a dataset by computing z-scores for a specific column. Standardization is an essential preprocessing step in data analytics and machine learning to normalize feature scales, ensuring uniform feature contribution to model performance.

Loading Data: The script utilizes pandas to ingest a CSV file (`data.csv`) into a DataFrame, facilitating efficient data manipulation and analysis.

Calculating Z-Scores: Leveraging the `zscore` function from `scipy.stats`, the script calculates z-scores for values in a designated column (`column_name`). The z-score, z, is determined by:

$$z = \frac{x - \mu}{\sigma}$$

where x represents the individual data point, μ is the mean of the dataset, and σ is the standard deviation.

Saving the Results: The DataFrame, augmented with an additional column for z-scores, is written to a new CSV file (`data_with_zscore.csv`), preserving the standardized data for subsequent processing.

Algorithm: Z-Score Normalization
Steps:

- Import pandas and scipy.stats.
- Load dataset from CSV.
- Compute z-scores for the specified column.
- Append z-scores to DataFrame.
- Save updated DataFrame to a new CSV file.

2) parquet.py: The `parquet.py` script converts a standardized CSV file into Parquet format, enhancing storage efficiency and query performance in big data environments.

Loading Data: The script reads the standardized CSV file (`data_with_zscore.csv`) into a pandas DataFrame.

Saving as Parquet: It converts the DataFrame to Parquet format (`data_with_zscore.parquet`), leveraging Parquet's columnar storage format to optimize both storage and retrieval processes.

Steps:

- Import pandas.
- Load the standardized CSV data into a DataFrame.
- Save the DataFrame in Parquet format.

C. Segmentation & Vectorization

1) splitting_into_contiguous_chunks.py: The `splitting_into_contiguous_chunks.py` script divides a large dataset into smaller, contiguous chunks. This segmentation facilitates efficient processing and analysis of large datasets by reducing memory overhead and improving computational manageability.

Loading Data: The script reads a Parquet file (`data_with_zscore.parquet`) into a pandas DataFrame.

Defining Chunking Function: A function, `split_into_chunks`, is defined to partition the DataFrame into smaller, contiguous chunks of a specified size (`chunk_size`).

Splitting and Saving: The dataset is segmented into chunks, each saved as an individual Parquet file (`chunk_i.parquet`), ensuring that data is easily manageable for subsequent processing steps.

Steps:

- Import pandas and numpy.
- Load the dataset from Parquet.
- Define a function to split the DataFrame into chunks.
- Split the DataFrame using the chunking function.
- Save each chunk to a separate Parquet file.

2) vectorized_to_unix_timestamp.py: The `vectorized_to_unix_timestamp.py` script transforms date values in the dataset to Unix timestamps. Unix timestamps represent the number of seconds elapsed since January 1, 1970 (the Unix epoch), providing a standardized format for time-based computations.

Loading Data: The script reads a Parquet file (`data_with_zscore.parquet`) into a pandas DataFrame.

Converting Dates: It converts date values in a specified column (`date_column`) to Unix timestamps using `time.mktime` and `pd.to_datetime`.

Saving the Results: The DataFrame, now containing Unix timestamps, is saved to a new Parquet file (`data_with_unix_timestamp.parquet`), ensuring temporal data is in a uniform format.

Steps:

- Import pandas and time.
- Load the dataset from Parquet.
- Convert the date column to Unix timestamps.
- Save the updated DataFrame to a new Parquet file.

3) vectorizing_contiguous_chunks.py: The `vectorizing_contiguous_chunks.py` script converts text data into numerical vectors using TF-IDF (Term Frequency-Inverse Document Frequency). TF-IDF evaluates the importance of a term within a document relative to a corpus, transforming textual information into a format suitable for machine learning algorithms.

Loading Data: The script reads a Parquet file (`data_with_unix_timestamp.parquet`) into a pandas DataFrame.

Vectorizing Text: Employing `TfidfVectorizer` from `sklearn.feature_extraction.text`, the script transforms text data in the specified column (`text_column`) into numerical vectors. The TF-IDF formula is:

$$\text{tf-idf}(t, d, D) = \text{tf}(t, d) \times \text{idf}(t, D)$$

where $\text{tf}(t, d)$ is the term frequency of term t in document d, and $\text{idf}(t, D)$ is the inverse document frequency of term t across the corpus D.

Saving the Results: The resulting vectors are stored in a new Parquet file (`vectorized_data.parquet`), preserving the transformed text data for downstream processing.

Algorithm: TF-IDF Vectorization

Steps:

- Import pandas and sklearn.
- Load the dataset from Parquet.
- Apply TF-IDF vectorization to the text column.
- Convert the vectors to a DataFrame.
- Save the DataFrame to a new Parquet file.

4) bigquery.py: The `bigquery.py` script uploads the processed dataset from a Parquet file to a Google BigQuery table. BigQuery, a serverless data warehouse, facilitates scalable analysis over extensive datasets.

Initializing Client: The script initializes a BigQuery client using the Google Cloud SDK.

Loading Data: It reads the processed dataset from a Parquet file (`vectorized_data.parquet`) into a pandas DataFrame.

Uploading to BigQuery: The DataFrame is uploaded to a specified BigQuery dataset and table (`your_dataset_id.your_table_id`), ensuring the data is accessible for high-performance analytical queries.

Steps:

- Import pandas and google.cloud.bigquery.
- Initialize the BigQuery client.
- Load the dataset from Parquet.
- Define the BigQuery dataset and table.
- Upload the DataFrame to the specified BigQuery table.
- Confirm successful upload.

III. The Models

predictHD's predictive modeling is grounded in a mix of traditional and innovative approaches:

A. Hybrid Model Architecture

The use of CAE and CNN architectures provided a robust mechanism for feature extraction and anomaly detection. This was complemented by the MixTral transformer model, which integrated these features to forecast HDD failures.

The Convolutional Autoencoder (CAE) architecture, as shown below in Figure 1, begins with an input layer of shape (100, 1). It first applies a 1D convolutional layer with 16 filters, resulting in an output shape of (100, 16). This is followed by a max pooling layer that halves the sequence length, producing an output shape of (50, 16). The model then adds another 1D convolutional layer with 8 filters, leading to an output shape of (50, 8), followed by another max pooling layer, reducing the sequence length further to (25, 8). A third convolutional layer with 8 filters is then applied, maintaining the shape (25, 8). The model then starts the upsampling process with a 1D upsampling layer, doubling the sequence length to (50, 8). A subsequent convolutional layer with 16 filters results in an output shape of (50, 16), which is then upsampled again to (100, 16). The final convolutional layer with 1 filter restores the original input shape of (100, 1).

As shown in Figure 2 in the next page, the Convolutional Neural Network (CNN) architecture, starts with an input layer of shape (100, 8). It applies a 1D convolutional layer with 32 filters, resulting in an output shape of (98, 32). This is followed by a max pooling layer that reduces the sequence length to (49, 32). The output is then flattened into a single vector of shape (1568). A dense layer with 64 units is applied next, producing an output shape of (64). A dropout layer with a rate of 0.5 is included to prevent overfitting, maintaining the shape (64). The model concludes with a dense output layer with a single unit, resulting in a final output shape of (1). This CNN architecture is designed for classification tasks, leveraging the feature extraction capabilities of the preceding CAE model.

1) cae_cnn_mixtral.py: The `cae_cnn_mixtral.py` script constructs and tests a Convolutional Autoencoder (CAE) using PyTorch for image data. A CAE is an unsupervised neural network that compresses and decompresses input data, learning efficient codings.

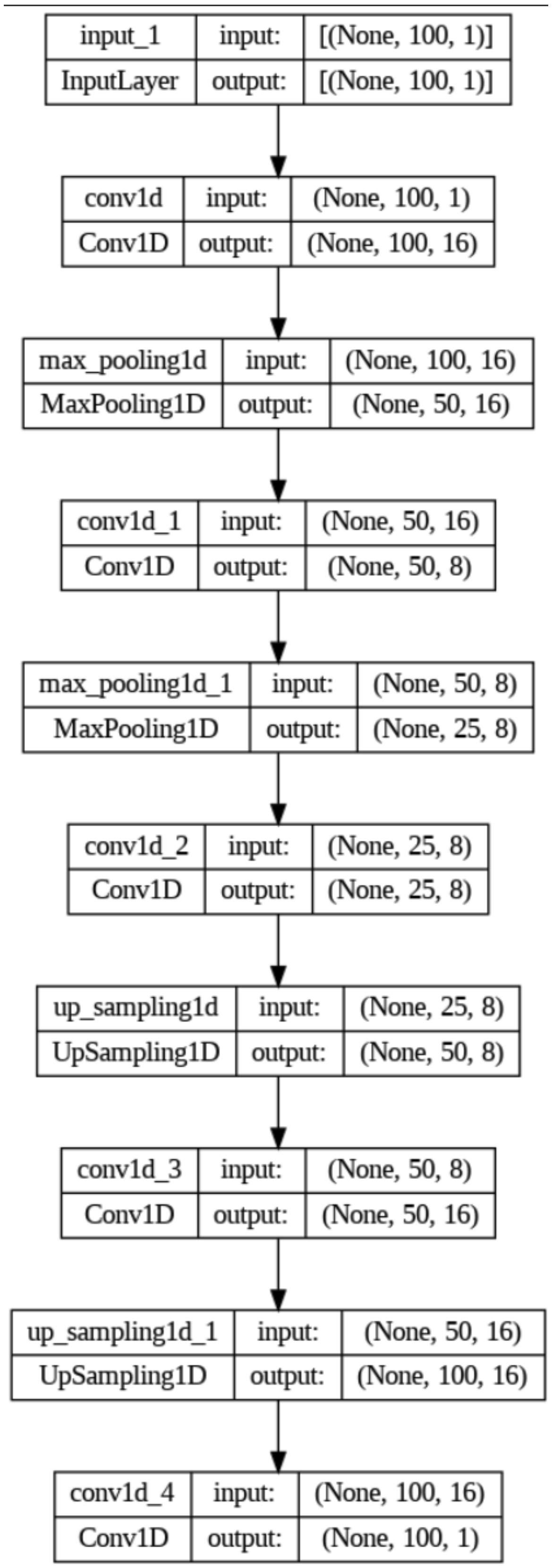

input_1 | input: | [(None, 100, 1)]
InputLayer | output: | [(None, 100, 1)]
conv1d | input: | (None, 100, 1)
Conv1D | output: | (None, 100, 16)
max_pooling1d | input: | (None, 100, 16)
MaxPooling1D | output: | (None, 50, 16)
conv1d_1 | input: | (None, 50, 16)
Conv1D | output: | (None, 50, 8)
max_pooling1d_1 | input: | (None, 50, 8)
MaxPooling1D | output: | (None, 25, 8)
conv1d_2 | input: | (None, 25, 8)
Conv1D | output: | (None, 25, 8)
up_sampling1d | input: | (None, 25, 8)
UpSampling1D | output: | (None, 50, 8)
conv1d_3 | input: | (None, 50, 8)
Conv1D | output: | (None, 50, 16)
up_sampling1d_1 | input: | (None, 50, 16)
UpSampling1D | output: | (None, 100, 16)
conv1d_4 | input: | (None, 100, 16)
Conv1D | output: | (None, 100, 1)

Fig. 1: Convolutional Autoencoder (CAE) Architecture

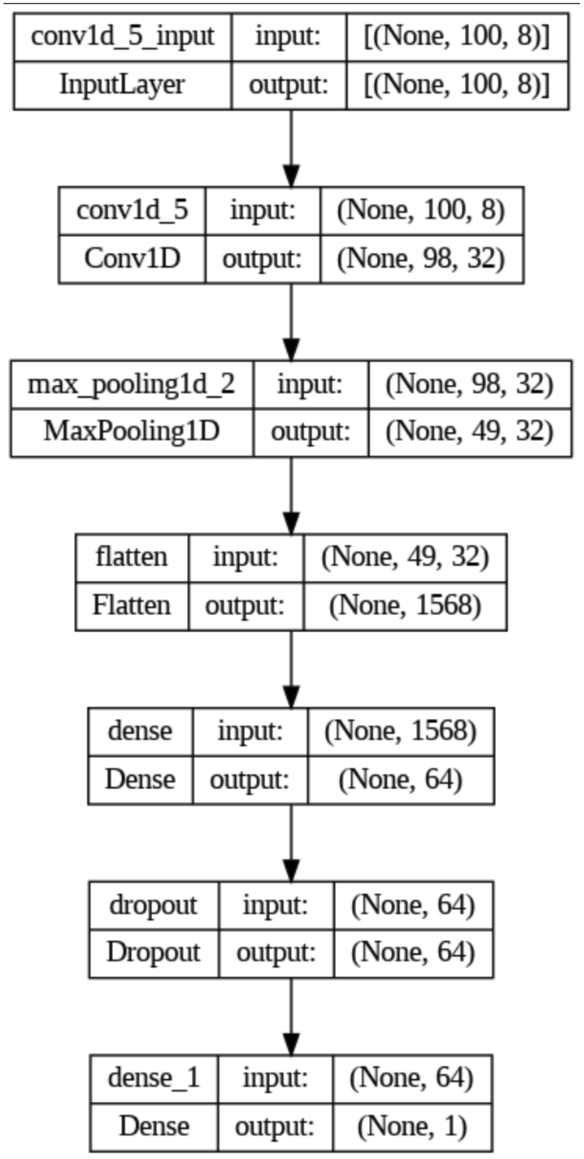

conv1d_5_input | input: | [(None, 100, 8)]
InputLayer | output: | [(None, 100, 8)]
conv1d_5 | input: | (None, 100, 8)
Conv1D | output: | (None, 98, 32)
max_pooling1d_2 | input: | (None, 98, 32)
MaxPooling1D | output: | (None, 49, 32)
flatten | input: | (None, 49, 32)
Flatten | output: | (None, 1568)
dense | input: | (None, 1568)
Dense | output: | (None, 64)
dropout | input: | (None, 64)
Dropout | output: | (None, 64)
dense_1 | input: | (None, 64)
Dense | output: | (None, 1)

Fig. 2: Convolutional Neural Network (CNN) Architecture

Defining the Encoder: The encoder comprises multiple convolutional layers that progressively reduce the spatial dimensions while increasing the depth (number of channels), effectively encoding the input data into a latent representation.

Defining the Decoder: The decoder utilizes transposed convolutional layers to reconstruct the input data from the encoded latent representation, restoring the original spatial dimensions.

Combining Encoder & Decoder: The `CAE_CNN` class integrates the encoder and decoder, forming a complete autoencoder model.

Testing the Model: The script initializes the model with specified parameters, prints the model architecture, and tests it with a random tensor to ensure functionality.

Algorithm: Convolutional Autoencoder

Steps:

- Import torch and torch.nn.
- Define the Encoder class with convolutional layers.
- Define the Decoder class with transposed convolutional layers.
- Combine Encoder and Decoder into the CAE_CNN class.
- Initialize and test the model with example input.

B. Algorithmic Innovations

Beyond standard machine learning algorithms, the project explored advanced techniques such as ensemble learning and transfer learning to enhance the predictive accuracy of the models.

IV. Results

The Confusion Matrix for the Convolutional Neural Network is as follows:

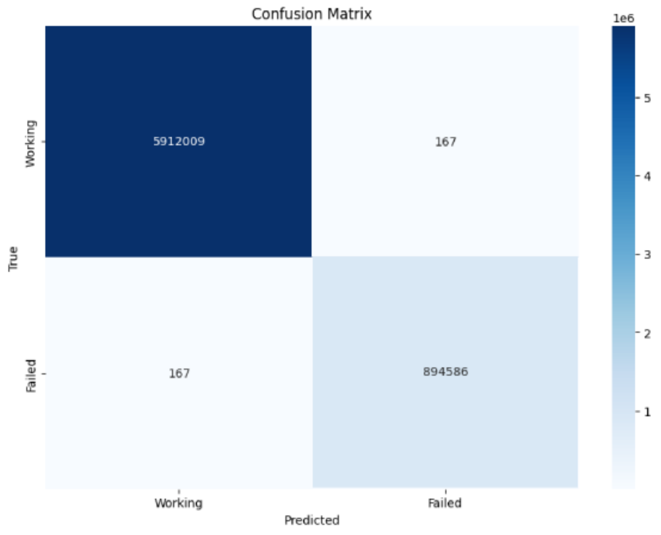
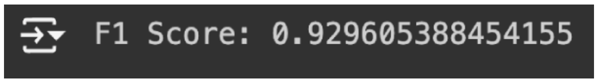


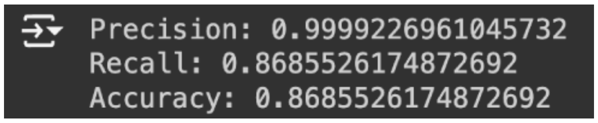
Fig. 3: Confusion Matrix for the CNN

Additionally, the F1 score, Precision, Recall, and Accuracy are as follows:



```
F1 Score: 0.929605388454155
```

Fig. 4: F1 Score



```
Precision: 0.9999226961045732
Recall: 0.8685526174872692
Accuracy: 0.8685526174872692
```

Fig. 5: Precision, Recall, and Accuracy

V. Predictive Capabilities & System Validation

The predictive system developed offers:

- **Granular Predictions:** By analyzing serial number-specific data, the system can predict impending failures with a precision that allows for targeted maintenance actions.
- **Aggregate Risk Assessments:** The system also provides risk assessments for pools of HDDs, enabling resource optimization in data center operations.

System validation was conducted through a series of stress tests and real-world trial deployments to ensure that the models performed well under various operational scenarios.

VI. Challenges & Advanced Solutions

- **Handling Data Scale and Complexity:** The sheer volume and complexity of the data posed significant challenges, which were mitigated through the use of scalable cloud computing resources and optimized data pipelines.
- **Model Training & Optimization:** Model complexity was managed through the use of advanced optimization algorithms and hardware acceleration techniques, ensuring efficient training without compromising on model accuracy.

VII. Theoretical & Practical Implications

The project draws on a range of theoretical frameworks from the fields of statistics, machine learning, and operations research. Practical implications are vast, offering improvements in preventive maintenance and resource allocation that could significantly reduce downtime and operational costs in data centers.

VIII. Conclusion & Future Work

predictHD has established a scalable and efficient framework for HDD failure prediction. Future work will focus on expanding the model's applicability to different types of storage devices and integrating real-time data feeds to enhance predictive timeliness and accuracy.

Acknowledgment

We would like to express our deepest gratitude to Dr. Saptarshi Sengupta, our professor, for his constant guidance, invaluable support, and encouragement throughout the course of this research. His profound knowledge and insightful feedback were crucial to the success of this project.

We also extend our heartfelt thanks to Rohan Mohapatra, & Pooja Krishan, Graduate Students from the Department of Computer Science, who provided guidance based on their experience working on a similar problem statement. Their assistance and insights were immensely valuable to our work.

References

[1] L. N. Bairavasundaram, G. R. Goodson, S. Pasupathy, and J. Schindler, "An analysis of latent sector errors in disk drives," *SIGMETRICS Perform. Eval. Rev.*, vol. 35, no. 1, pp. 289–300, Jun. 2007. [Online]. Available: https://doi.org/10.1145/1269899.1254917

[2] A. Coursey, G. Nath, S. Prabhu, and S. Sengupta, "Remaining useful life estimation of hard disk drives using bidirectional lstm networks," in *2021 IEEE International Conference on Big Data (Big Data)*, 2021, pp. 4832–4841. [Online]. Available: https://doi.org/10.1109/BigData52589.2021.9671605

[3] Backblaze, "Backblaze drive stats for 2022," https://www.backblaze.com/blog/backblaze-drive-stats-for-2022/, Feb. 2022, accessed on 6 Mar. 2023.

[4] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[5] C.-P. Hsieh, Y.-T. Chen, W.-K. Beh, and A.-Y. A. Wu, "Feature selection framework for xgboost based on electrodermal activity in stress detection," in *2019 IEEE International Workshop on Signal Processing Systems (SiPS)*, 2019, pp. 330–335. [Online]. Available: https://doi.org/10.1109/SiPS47522.2019.9020321

[6] G. Hughes, J. Murray, K. Kreutz-Delgado, and C. Elkan, "Improved disk-drive failure warnings," *IEEE Transactions on Reliability*, vol. 51, no. 3, pp. 350–357, 2002. [Online]. Available: https://doi.org/10.1109/TR.2002.802886

[7] J. F. Murray, G. F. Hughes, and K. Kreutz-Delgado, "Hard drive failure prediction using non-parametric statistical methods," 2003.

[8] L. Eriksson, S. Wold, and J. Trygg, "Multivariate analysis of congruent images (maci)," *Journal of Chemometrics*, vol. 19, no. 5-7, pp. 393–403, 2005. [Online]. Available: https://doi.org/10.1002/cem.944

[9] G. Hamerly and C. Elkan, "Bayesian approaches to failure prediction for disk drives," in *Proceedings of the Eighteenth International Conference on Machine Learning*, ser. ICML '01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 202–209.

[10] Y. Zhao, X. Liu, S. Gan, and W. Zheng, "Predicting disk failures with hmm- and hsmm-based approaches," in *Advances in Data Mining. Applications and Theoretical Aspects*, P. Perner, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 390–404.

[11] A. De Santo, A. Galli, M. Gravina, V. Moscato, and G. Sperli, "Deep learning for hdd health assessment: An application based on lstm," *IEEE Transactions on Computers*, vol. 71, no. 1, pp. 69–80, 2022. [Online]. Available: https://doi.org/10.1109/TC.2020.3042053

[12] Y. Ding, Y. Zhai, Y. Zhai, and J. Zhao, "Explore deep auto-coder and big data learning to hard drive failure prediction: a two-level semi-supervised model," *Connection Science*, vol. 34, no. 1, pp. 449–471, 2022. [Online]. Available: https://doi.org/10.1080/09540091.2021.2008320

[13] F. D. d. S. Lima, G. M. R. Amaral, L. G. d. M. Leite, J. P. P. Gomes, and J. d. C. Machado, "Predicting failures in hard drives with lstm networks," in *2017 Brazilian Conference on Intelligent Systems (BRACIS)*, 2017, pp. 222–227. [Online]. Available: https://doi.org/10.1109/BRACIS.2017.72