Establishing Infrastructure of Democratizing In-database Machine Learning Systems

Prapti Maitra

4th Year Project Report
Computer Science and Mathematics
School of Informatics
University of Edinburgh

2025

Abstract

The exponential growth in data volume and complexity presents significant challenges for traditional machine learning workflows for relational databases. The OpenDBML project addresses these challenges by enhancing the capabilities of in-database machine learning, making it more accessible and efficient for users. This paper details the advancements made to the OpenDBML framework, focusing on streamlining data transformation processes, integrating feature engineering functionalities, and evaluating the performance of in-database systems like libFM, LMFAO, and Morpheus across various datasets. Our contributions include advanced data transformation support for Morpheus, an improved data transformation process for LMFAO, and the incorporation of essential feature engineering and data exploration tools within OpenDBML. Furthermore, we critically assess the framework's performance on popular datasets, demonstrating its effectiveness in reducing training times and facilitating more accurate machine learning models. By offering a unified system that simplifies the machine learning pipeline for relational databases, OpenDBML significantly enhances the efficiency of in-database machine learning workflows. This project not only democratizes access to sophisticated machine learning tools but also sets the foundation for future developments in the field.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Prapti Maitra*)

Acknowledgements

I would like to thank my project supervisor Amir Shaikhha and his PhD student Mahdi Ghorbani, for their constant guide and support through the duration of this project. I would also like to thank my second-marker, Paolo Guagliardo for his valuable feedback in the first half of the project. Finally, I would like to thank my friends and family for their constant support.

Table of Contents

Chapter 1

Introduction

This project aims to structure the OpenDBML framework, a Python API that aims to democratise in-database machine learning, as a complete data transformation tool and a platform for comparing the performance of existing state-of-the-art in-database machine learning systems.

1.1 Motivation

Machine Learning has become an integral part of our lives, from the recommendations we receive on our favorite retail websites to critical applications like fraud detection and computer vision. Most of the data used for these applications is stored in a normalized form in relational databases that offer data security, scalability, and efficient data management. However, with the exponential increase in data, there is a need for an alternative approach to apply machine learning to relational data. The traditional approach involves exporting the data from database systems, denormalising them over key joins, and then training a machine learning model on this data. This leads to huge, highly sparse tables with redundant data and exceptionally high training times. In-database machine learning systems aim to adapt traditional machine learning algorithms to train on individual relations in a database while respecting the key relationships it shares with other relations. This significantly reduces the training time and can be of tremendous use, especially for time-sensitive applications. Despite this significant advantage, in-database machine learning systems such as libFM, LMFAO, and Morpheus face challenges in widespread applications due to their ad-hoc API and their requirement for relational data to be in unconventional data formats such as a block structure or sparse matrix. OpenDBML aims to mitigate this limitation by introducing a framework that integrates all in-database systems into a unified system and does the data transformation for you, making in-database systems more accessible. While OpenDBML fully supports the conversion for relational datasets into block structure for LibFM, it has offers parital support for the system LMFAO. Additionally, OpenDBML offers limited functionalities for data exploration and feature engineering techniques, that are crucial to optimise machine learning models and go hand in hand with data transformation.

1.2 Objectives

The project aims to achieve the following objectives:

1. Integrating data transformation support for Morpheus system.
2. Identifying and rectifying drawbacks in the data transformation process for LMFAO.
3. Integrating fundamental feature engineering and data exploration functionalities in the framework.
4. Critically evaluating the performance of all three in-database systems on 8 popular datasets used for training recommendation systems
5. Demonstrating the use of OpenDBML as an end-to-end tool for democratising in-database systems.

1.3 Structure

The rest of the report is organized into four chapters, structured as follows:

- **Chapter 2 (Background)** covers some of the established current in-database machine learning systems, along with an overview of the functionalities and architecture of the existing OpenDBML implementation. It also presents the most commonly used data exploration and feature engineering techniques used in machine learning.
- **Chapter 3 (Design and Implementation)** presents an overview of the design plan for the new implementation and the implementation details of the newly added functionalities to the OpenDBML framework.
- **Chapter 4 (Experimental Results)** presents the experimental setup, evaluation metrics, and results enabled by the enhancements to OpenDBML.
- **Chapter 5 (Conclusion)** summarizes the findings and contributions of this project, concluding with final remarks on the applications of the OpenDBML framework and scope for future work.

1.4 Contributions

Our main contributions to enhancing the OpenDBML framework are summarized as follows:

- Data transformation support for Morpheus, allowing users to convert their datasets into a normalised matrix format.
- A rectified data transformation process for LMFAO, allowing users to use the system on a wider range of datasets.

- A set of fundamental feature engineering and data exploration functionalities within OpenDBML, allowing users to manipulate features to improve model performance.
- A comprehensive evaluation of libFM, LMFAO, and Morpheus across eight popular datasets, demonstrating the capabilities of OpenDBML and providing insights into the strengths and weaknesses of each in-database system.
- The establishment of OpenDBML as an end-to-end tool for democratising in-database machine learning showcases its ability to support the data preparation for any dataset.

Chapter 2

Background

To understand the contributions made by this project, we need to understand the topics covered in this chapter. The following sections in this chapter provide a brief summary of the state-of-the-art in-database machine learning systems. This is followed by an overview of the OpenDBML framework and its implementation architecture, and finally, it covers some of the most fundamental feature engineering and data exploration techniques.

2.1 Introduction to In-Database Machine Learning

In-database machine learning (In-DB ML) is an approach to machine learning that involves performing the entire machine learning process directly within databases. This technique is particularly useful when dealing with large and complex datasets since it eliminates the need to export data from databases, which can be a time-consuming and error-prone process. In addition, it reduces the risk of privacy and security breaches that can occur when exporting data outside of the database.

By conducting machine learning tasks within the database, In-DB ML saves time and allows for more efficient use of resources. It also makes it possible to perform machine learning on large datasets in terabytes or petabytes, which was previously not feasible due to the limitations of traditional ELT procedures. This approach streamlines the entire machine learning workflow, from data preparation to model deployment, within a single environment.

Furthermore, In-DB ML enables real-time analytics, which means that machine learning models can be applied to data as it is updated in real time. This feature allows for more dynamic data analysis and provides greater accuracy in predictive modeling.

In-database Machine Learning System is a broad term used to recognise architectures and solutions that allow you to integrate machine learning inside a database. These solutions are extended machine learning systems and factorised machine learning-based systems.

1. **Extended ML Systems**: The core idea is to extend traditional database man-

agement systems (e.g., PostgreSQL) with additional functionalities to support machine learning operations. This is typically done by extending machine learning operations through UDFs which are custom functions in query languages that users can define to perform operations not natively supported by the DBMS. UDFs can execute complex computations or algorithms directly on the database, allowing users to analyse and train data using SQL commands. For example, MADlib, IBM Db2 Warehouse on Cloud, Amazon Redshift ML etc.

2. **Factorised ML systems** : These solutions focus on eliminating time inefficiency and redundancy arising from denormalising data over primary-foreign key relations from the typically normalised version stored in databases. They pose the question: Why not push the ML operators over the joins instead of joining relations and performing ML operations? This is done in several ways: one, transforming the operations within an ML algorithm into a series of aggregations and pushing it through joins, for example, LMFAO and a second class of solutions leverages the fact that most operations in a machine learning algorithm can be written as a linear algebra matrix operations to function over normalised data, example LibFM, Morpheus.

2.1.1 Library for Factorisation Machines (LibFM)

LibFM, or the Library for Factorization Machines, is a software implementation of factorisation machines (FMs) [1]. Factorisation machines(FM), is a supervised algorithm that can be used for classification, regression, and ranking tasks. It is a very efficient algorithm for high-dimensional sparse datasets. This is because factorisation machines model interactions between variables using factorised parameters. Instead of directly learning a parameter for each interaction (the approach of a linear regression algorithm), FMs learn latent factors for each variable and model their interactions through the dot product of these latent factors. This approach significantly reduces the number of learning parameters, making the model scalable and efficient for high-dimensional sparse data. This is particularly useful for prediction problems in recommendation systems.LibFM supports three data formats: text, binary, and block structure. The block structure format is the most efficient for prediction tasks among these. This format is efficient because relational data often exhibits repeated feature patterns[2]. Factorization Machines can exploit these patterns to improve predictive performance. The dataset is divided into blocks, preserving this pattern and representing a group of features or variables that share a common characteristic or domain. For example, in a movie recommendation system, one block could contain features related to user information, while another could focus on movie genres. LibFM offers three learning algorithms for optimising its factorisation machine models: Stochastic Gradient Descent (SGD), Alternating Least Squares (ALS), and Markov Chain Monte Carlo (MCMC). However, only MCMC and ALS are compatible with the block structure format.

2.1.2 Morpheus

Morpheus is a Python framework of algebraic rewrite rules that is commonly found in ML algorithms[3]. The normalised matrix is a logical abstraction introduced by this

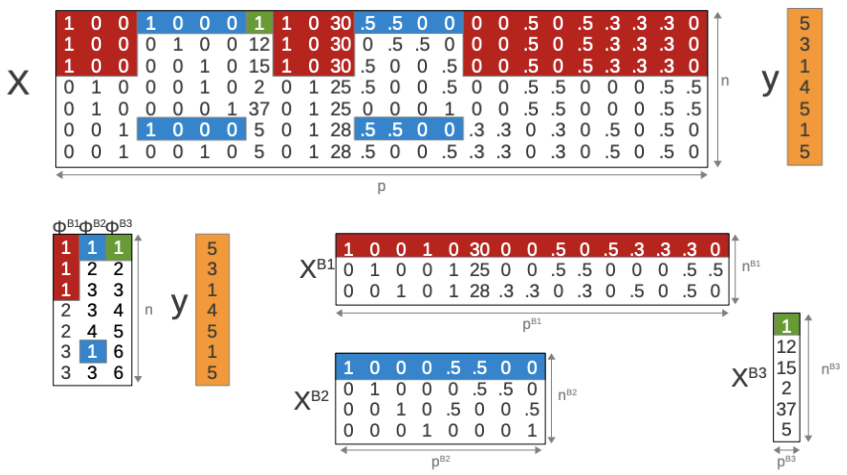

Figure 2.1: Block Structure Format

framework that is used instead of traditional single, denormalized tables that result from joining multiple relations.It represents the relational data in a way that maintains the relationships between different normalised relations without physically merging them. The normalized matrix consists of:

- S: Your entity table representing the primary dataset.
- R: Attribute tables representing normalised datasets.
- K: An indicator matrix that maps the relationships between entities in S and attributes in R through foreign keys and primary keys.

The key to representing the entire dataset without joining lies in the K matrix. For each entity in S, its corresponding attributes in R are not duplicated alongside it (as would be in a denormalized table). Instead, K uses sparse encoding to indicate these relationships efficiently. This method drastically reduces the memory space, especially for large datasets with significant redundancy due to one-to-many relationships. By leveraging algebraic rewrite rules, Morpheus transforms linear algebra operations common in ML algorithms—such as matrix multiplication, inversion, and aggregation—into equivalent operations that can be directly applied to the normalized matrix. These rewrite rules are meticulously designed to mimic the computational effects of having performed a join operation, thereby allowing ML algorithms to operate as if on a single table while actually working on multiple, logically connected tables. Morpheus currently implements four algorithms - linear regression, logistic regression, gaussian non-negative matrix factorization and k-means classification algorithm.

2.1.3 Layered Multiple Functional Aggregate Optimization (LMFAO)

LMFAO is a memory optimization and execution engine specifically designed to enhance the efficiency of processing large batches of group-by aggregate operations over joins in relational databases[4].The motivation behind LMFAO comes from the fact that denormalising relational data for standard machine learning tools like Tensorflow

and scikit-learn requires extensive joining and aggregation across tables which causes the computational overhead to become significant, leading to inefficiencies.LMFAO generates a single, optimized query plan for a batch of aggregate queries, based on a join tree structure. This approach allows for strategic planning of data access and computation, significantly reducing redundant operations LMFAO introduces a layered architecture of optimizations specifically designed to address these challenges, enhancing both the efficiency and scalability of processing aggregate queries over joins. These layers consists of :

- **The View Layer :** This layer is responsible for analyzing the incoming batch of queries and the database schema to create an optimized execution plan. This involves understanding the relationships between tables, the nature of the queries, and the data distribution.t generates a ”join tree,” a conceptual structure that represents how tables are interconnected through joins and how data should flow through these connections for efficient query processing.
- **Multi-Output Optimization Layer :** Once the join tree is established, the multi-output optimization layer takes over to manage the actual data processing. This layer focuses on optimizing how the intermediate query results are computed and combined across different queries. It involves grouping related views that can be calculated in a single pass over the data, reducing the need for repetitive data accesses.This layer strategically organizes computations to maximize the reuse of intermediate results and minimizes the computational overhead by sharing as much of the processing workload as possible.
- **Code Generation Layer :** The final layer of LMFAO is where the optimized execution plans are translated into actual executable code. This layer focuses on generating highly efficient, C++ low-level code that is specifically tailored to the queries and the database schema.It involves compiling the multi-output execution plans into C++ code, applying further optimizations such as inlining function calls, optimizing data structures for storage and access (e.g., using tries, sorted arrays, or hashmaps), and leveraging parallel computing techniques to distribute the workload effectively across multiple processors.

LMFAO currently supports optimizations for three different algorithms, linear regression kmeans clustering and decision trees, however, the open-sourced engine only offers regression and kmeans clustering algorithm. LMFAO requires the input datasets to be in csv files along with 2 configuration files, one containing details about the tree decomposition of the relation data and other containing details of all the features in the dataset.

2.2 Introduction to OpenDBML

Despite their numerous benefits, current in-database (in-DB) machine learning systems have not seen widespread adoption due to two main reasons. Firstly, there’s the necessity for data to be in specific, less-common formats, like the block structure required for LibFM and the sparse matrix format for Morpheus. Secondly, the lack of a Python API presents a significant barrier; LibFM and LMFAO are closely integrated with an engine,

and Morpheus runs on the now outdated Python 2, which conflicts with the software environments of most modern operating systems. OpenDBML emerges as a solution to these challenges by providing a unified framework that bridges the gap between these specialized in-DB machine learning systems and the broader data science community's needs[5]. It offers a Python API that simplifies the process of integrating with and utilizing these systems, ensuring compatibility with present-day software practices. OpenDBML also tackles the issue of data format incompatibility by facilitating data transformation into the required formats for each system, thus removing a significant barrier to entry for using advanced in-DB ML functionalities. By addressing these key issues, OpenDBML aims to enhance the accessibility and usability of in-DB machine learning, paving the way for its wider adoption and application in real-world scenarios.

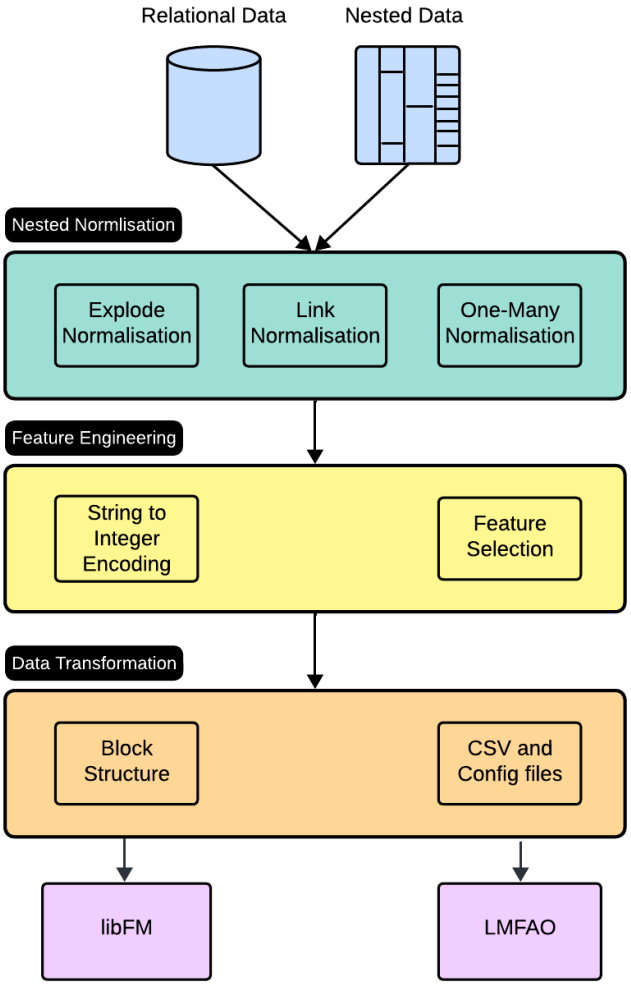

Figure 2.2: OpenDBML workflow

The framework is designed to accept datasets in more popular formats, such as CSV files. OpenDBML provides functionalities for normalizing nested data structures within relational datasets. This process involves expanding columns that contain nested data and storing it as a new relation in the dataset. OpenDBML implements three distinct

normalization strategies:

- **One-Many Normalization**: This approach is used when a column contains dictionary-like data in each cell, expanding nested rows into sub-relations.
- **Explode Normalization**: This method deals with nested columns listing items, converting them into a sub-relation with an n-hot encoded representation.
- **Link Normalization**: It expands a nested column into a new table with a uniquely generated primary key and creates an intermediate table to link the main table's primary keys with those of the new sub-relation.

OpenDBML can also perform elementary feature engineering techniques, such as string to integer encoding, feature selection, and histogram binning methods. Finally, OpenDBML allows users to convert their datasets into data formats supported by LibFM, LMFAO, and Morpheus. The OpenDBML framework demonstrates it's work on 8 popular recommendation system datasets, namely MovieLens [1], Openflights[2], Favorita[3], Yelp[4], Expedia[5], Walmart[6], Lastfm [7]and Bookcrossing datasets. https://www.kaggle.com/datasets/somnambwl/bookcrossing-dataset

2.2.1 Functionalities of Classes in OpenDBML API

In the OpenDBML framework, four primary classes - PolyDataset, DatasetConfig, and RelationalDataFrame - play crucial roles in managing and transforming relational datasets for in-database machine learning systems.the `RelationalDataFrame` class acts as the backbone of the entire implementation and used to represent a single relation in the relational dataset adds on to functionalities offered by a Pandas DataFrame. To use the framework, a user utilises the `FileConfig` class which is responsible for reading individual data files coming from different file formats and assigning it to a `RelationalDataFrame` object. It ensures that a data file is appropriately formatted and ready for analysis. A `FileConfig` object is initialised by providing details of the raw data file, such as name, schema (a dictionary storing the columns and datatypes of the file), the primary key of the file and information about any nested columns that need to normalised. Next, an object of `DatasetConfig` class, which is responsible for assembling all the individual preprocessed data files, must be initialised with their corresponding`FileConfig` objects. This class is responsible for maintaining the overall coherent structure of the dataset while preserving relational integrity through primary and foreign key definitions. Finally, the user must initialise an object of the `PolyDataset` class with this `DatasetConfig` object. The `PolyDataset` class is then used to call data transformation functions on the datasets.

[1] https://www.kaggle.com/datasets/samiravaez/the-movielens-1m-dataset
[2] https://openflights.org/
[3] https://www.kaggle.com/competitions/favorita-grocery-sales-forecasting
[4] https://www.yelp.com/dataset
[5] https://www.kaggle.com/datasets/yasserh/walmart-dataset
[6] https://www.kaggle.com/datasets/yasserh/walmart-dataset
[7] http://millionsongdataset.com/lastfm/

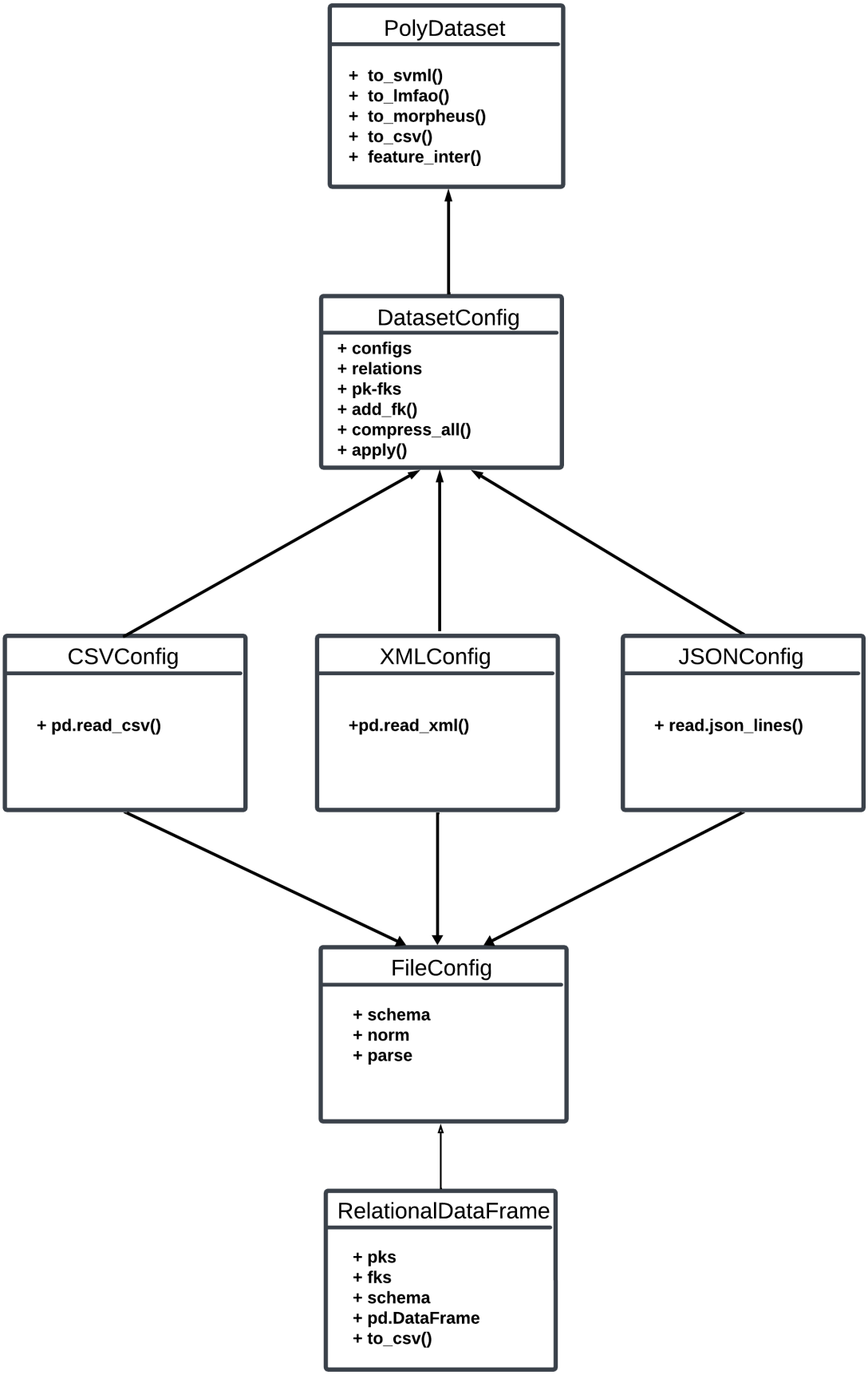

Figure 2.3: Class Diagram for existing OpenDBML implementation

PolyDataset

The PolyDataset class acts as a wrapper class for the entire dataset, managing the application of the data transformation and data modelling functionalities. It acts as the final layer that brings all the indivdual relations together to form an analyzable dataset ready for machine learning model training.

Functionalities

- **Lazy Loading and Application**: Allows for the option of lazy loading, whereby the dataset configurations are not immediately applied upon initialization. This is useful for scenarios where the dataset needs to be dynamically altered or configurations applied just-in-time.
- **Feature Interaction Modeling**: Allows the modeling of interactions between features within the dataset. This helps in uncovering relationships between variables that may enhance the predictive performance of machine learning models.
- **Data Export**: Provides methods like to_csv, for exporting the processed dataset into more popular formats compatible with popular machine learning frameworks and tools like Pandas and Scikit - Learn.
- **Data Transformation**: Responsible for all the data transformations for in-database systems supported by OpenDBML through methods such as to_svml for LibFM and to_lmfao for LMFAO.

DatasetConfig Class

The DatasetConfig class stores the configuration details for the entire dataset. It creates a dataset structure by configuring the details of individual datafiles which are encapsulated in FileConfig instances and assigning RelationalDataFrama with. This class maintains information about each individual relation and well as the entire dataset as a whole. This includes configuration details of individual relations and primary -foreign key relationships across all relations.

Functionalities

- **Aggregation of Data Configurations**: Allows for the integration of data obtained from different formats and configurations, such as CSVConfig, JSONConfig, and XMLConfig, into a unified dataset configuration.
- **Management of Key Relationships**: Enables the definition of foreign-primary key relationships across different relations within the dataset through the add_fk method. This helps preserve the relational integrity of the dataset and facilitates operations through these relationships.
- **Feature Compression**: Provides methods like compress and compress_all for integer encoding any string selected features or the entire dataset, which improves

memory usage and processing speed. This is particularly useful when working with large datasets.

- **Unified Data Configuration**: Ensures the mapping of individual FileConfig objects to a RelationalDataFrame object, allowing for abstraction between the initial data format of relation and its data frame through the `_apply` method. It also ensures the application of normalisation and correct parsing for any nested columns in a File Config object before assigning it to a RelationalDataFrame object. It stores the normalised column as a new RelationalDataFrame object and defines a new mapping of it to the parent relation through its primary key.

FileConfig Class

The FileConfig class is responsible for holding the configuration details of a single relation. This includes the relation's schema, primary key, and foreign key, as well as information about any nested columns and flags on the type of normalisation and parsing to apply to these columns.

Functionalities

- **Primary Key Management**: Enables assignment of one or more columns as primary keys for the relation, which is essential for establishing relationships between other of the dataset.
- **Parsing Rules Specification**: Allows assignment of custom parsing rules for data columns through the parse method, enabling the transformation of raw data into more useful forms.
- **Normalization Methods Assignment**: Supports the assignment of normalization methods such as explode, one-many or link normalisation. to nested data columns through the normalize method.
- **Extendability for Various Data Formats**: Serves as the base class for more specific data format configurations, such as CSVConfig, JSONConfig, and XML-Config, each of which implements format-specific loading and preprocessing logic based on the generic structure provided by FileConfig.

RelationalDataFrame Class

The RelationalDataFrame class acts as a wrapper function for the dataframe of a single relation. It contains the primary key and all foreign keys of a single relation, as well as the schema defined by the user when declaring FileConfig objects. It is designed to extend the functionalities of traditional data frames to better accommodate relational data structures.

Functionalities

- **Dataframe Management**: Responsible for building a Pandas data frame that represents a relation while ensuring that it adheres to specified datatype for each

column specified in the schema of its corresponding FileConfig object.

- **Data Normalization and Transformation**: Responsible for removal of attributes that have been stored a separate relation as a result of normalization as well as integer encoding any string variables in the relation when the compress method of file config is called.
- **Data Export**: Allows users to export individual relations in the dataset into csv files using that to_csv method.

2.3 Feature Engineering

Feature engineering is a crucial step in the machine learning pipeline's data preparation process. It involves converting raw data into a more appropriate and effective format for modeling. By carefully creating features from the data, this process enables models to better comprehend the underlying patterns and relationships, significantly improving their predictive performance. Feature engineering includes a range of techniques designed to prepare and enrich data, from handling missing values and encoding categorical variables to selecting the most relevant features and extracting new information from complex data types. Below are some of the most essential feature engineering techniques.:

1. **Handling Missing Values**: This includes imputation (filling missing values with statistical measures like mean, median, or mode), ignoring rows with null values, or filling them with a specific value.
2. **Encoding Categorical Variables**: Machine learning models typically require numerical input, so categorical variables (e.g., gender, country) must be converted into an integer format. This is done through techniques such as one-hot encoding, label encoding, and binary encoding.
3. **Feature Scaling**: Many machine learning algorithms are sensitive to the scale of data and can greatly depend on it for its performance. Feature scaling methods, such as normalization and standardization, are used to standardize the values of features to fall under a specific range.
4. **Feature Selection**: This method is used in scenarios where not all features contribute to predictions or classifications. Feature selection is employed in such situations to select a subset of the most relevant features to reduce model complexity and improve performance.
5. **Implementing Test/Train Splits:**: This technique involves dividing the dataset post-feature transformation into a training set and a testing set. This division is crucial for assessing the generalizability of the model to new data. The training set is used to build and tune the model, while the testing set evaluates its performance, ensuring that the model can make accurate predictions on unseen data. Properly splitting the data helps in diagnosing issues like overfitting and underfitting, and enhances the model's reliability and effectiveness.

2.4 Data Exploration

In the machine learning pipeline, data exploration is the initial step in the data analysis right after data wrangling, which helps understand the characteristics and structure of the dataset through statistical summaries, visualizations, etc. This step is crucial in identifying any underlying patterns between the features and can also help identify potential issues with data quality, such as missing values, outliers, feature redundancies, etc. Some of the key techniques used for data exploration include:

1. **Statistical Summaries**: This involves computing basic statistics such as mean, median, mode, and standard deviation for each feature in the dataset to understand the scale of different features and identify any missing values.
2. **Data Visualization**: This technique includes generating plots and charts such as histograms, box plots, scatter plots, etc. to visually inspect the data and help in identifying outliers and any hidden patterns or clusters between features.
3. **Correlation Analysis**: This method involves computing Pearson or Spearman correlation coefficients to evaluate the strength and direction of the linear relationship between two variables. This helps in understanding how features are associated with each other and with the target variable.
4. **Principal Component Analysis**: Principal Component Analysis (PCA) is a technique that helps in revealing hidden patterns and influential features in high-dimensional data. It does this by transforming the data into fewer dimensions or principal components while retaining most of the variance. By using PCA, the data becomes easier to visualize and interpret, which makes it a particularly useful analysis tool. The key aspects of PCA include uncovering hidden patterns, identifying influential features, and reducing the dimensionality of high-dimensional data.

Chapter 3

Design and Implementation

To make OpenDBML a complete end-to-end data transformation tool for in-database systems, we started by examining the typical machine learning pipeline. It became clear that simply having data transformation functionalities isn't always sufficient. Tailoring data specifically to the task at hand often requires applying feature engineering techniques. Moreover, understanding which features to engineer is a process that goes hand in hand with data exploration.

Acknowledging this, we identified some key use cases essential for OpenDBML to serve as a reliable tool:

- **Correct and Reliable Data Transformations:**The foundation of any machine learning model is high-quality data. Thus, OpenDBML needed to ensure not just the versatility in handling various data formats but also the correctness and reliability of these transformations. This is achieved by integrating support for Morpheus and ensuring LMFAO works across all eight datasets.
- **Feature Engineering Capabilities:** Recognizing the critical role feature engineering plays in preparing data from Section4.3, we added functionalities such as train-test splits, data shuffling, and feature scaling. These techniques help prevent issues like overfitting and ensure the model can generalize well to unseen data.
- **Data Exploration Techniques:** Before diving into model building, understanding the dataset is crucial. Techniques such as generating correlation matrices, exploratory statistics, plots, and PCA analysis allow users to uncover patterns, identify relationships, and determine which features might be most predictive. Incorporating these capabilities into OpenDBML enables users to perform a thorough exploration within the same environment they transform and model their data, streamlining the workflow.

The following sections of the chapter discuss the details of the implementation of the data transformation and feature engineering functionalities added to the OpenDBML framework. Through this chapter, we will be referring to the example of Movielens dataset for a clear understanding of the algorithms covered. The MovieLens dataset

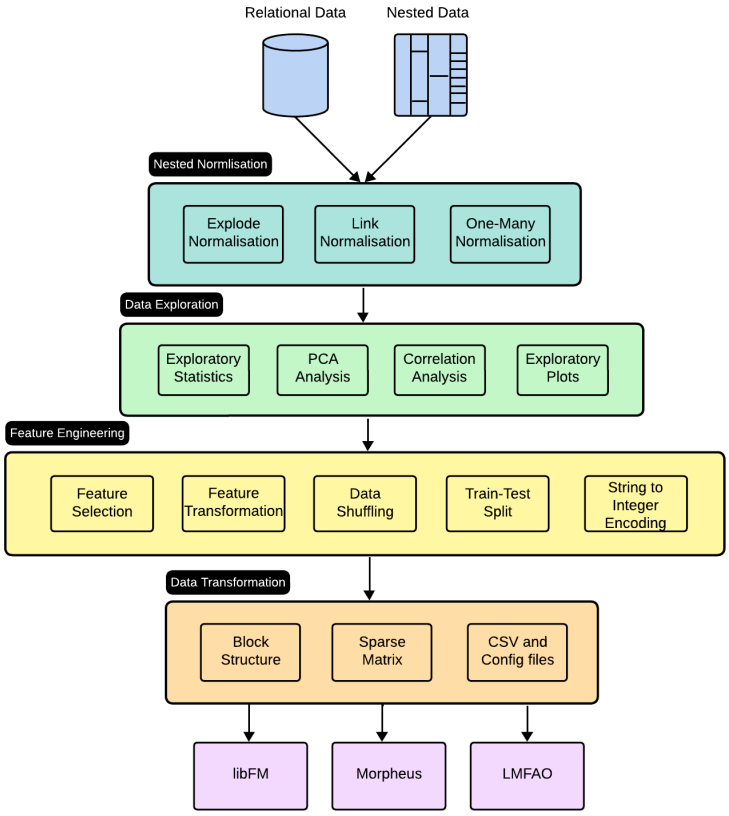

Figure 3.1: Plan for new OpenDBML implementation

[1] consists of 3 relations, ratings, users and movies. The movies relation consists of a nested genres attribute which is represented as a separate relation with the help of OpenDBML's nested normalisation. Furthermore, interactions_user_id models the feature interactions between user_id and movie_id. Additionally, we use certain key terms that are essential to follow the implementation details. They are as follows :

1. **Join Order**: refers to a nested dictionary structure used to represent a join tree of a dataset where each key is a tuple consisting of the name, foreign key and primary key of a relation, and the value is another nested join order or None value indicating a leaf node.
2. **Target Relation/Table**: refers to the outermost relation in the join order or the root node.
3. **Feature/Attribute** : refers to a single column in the relational table.

3.1 Restructuring Transformation for LibFM

The to_svml function in the PolyDataset is the function responsible for converting datasets in a blocked structure format. However, this function acts as a wrapper

[1] https://grouplens.org/datasets/movielens/1m/

```
join_order = {
    ('ratings', None, None): {
        ('users', 'user_id', 'user_id'): None,
        ('movies', 'movie_id', 'movie_id'): {
            ('genres', 'movie_id', 'movie_id'): None
        },
        ('interactions_user_id', 'user_id', 'user_id'): None,
    }
}
```

Figure 3.2: Join Order for Movielens Dataset

function for to_libfm() in the RelationalDataFrame class where the conversion to blocked structure format happens. The to_svml function allows the user to convert a dataset into a blocked structure format or a sparse matrix format. If the 'bs' flag is marked, It takes the join_order of the dataset and a boolean parameter bs standing for blocked structure format which is used as a flag to determine which format to convert the dataset into. The to_svml function then saves the target variable ".txt" format and calls the to_libfm function that converts all the relations into a blocked structure. Within to_libfm, the function takes the join_order of the dataset and then traverses through relations in each level of the join order and recursively calls the to_libfm function for any sub-relations that are present. In the base case, where there are no sub-relations present, it first joins one hot encodes every categorical variable present in the relation and then converts it svmlight format, successfully converting the relation into a blocked structure. This approach is not ideal for two reasons. Firstly, it implements the actual blocked structure conversion for the whole dataset in the RelationalDataFrame, which is supposed to represent a single relation in the dataset, making it monolithic and difficult to adapt. Secondly, it uses a recursive algorithm, which is not very time-efficient, especially for large datasets. Hence, we decided to restructure the implementation of to_svml to make sure the entire data conversion happens in the PolyDataset class, making it modular. We also used an iterative approach, which is known to be more time-efficient using a queue implementation. The restructured to_svml function converts traverses through the join order level by level and adds it to a queue. In the movielens example, the ratings relation is at the top level and hence it gets added to a queue. Following this, the ratings relation is popped from the queue and processed as follows :

1. The function first checks for any sub-relation under ratings.
2. The subrelations movies, users and iteraction_user_id get added to the queue.
3. The ratings relation is then converted into blocked structure format in the same manner as before.
4. Next, users, movies, and interaction_user_id and genres are processed and converted in the same manner and order.

Dataset	Recursion	Iteration
Openflights	2.24	2.126
Walmart	4.74	4.48
Movielens	12.976	12.68
Bookcrossing	14.167	14.1
Favorita	140.252	135.474
LastFM	229.981	233.97
Expedia	517.86	514.45
Yelp	537.123	532.595

Table 3.1: Runtime Comparison :Recursion vs Iteration

3.2 Extending compatibility of LMFAO

The LMFAO system requires its datasets to be in CSV files along with two configuration files. More specifically, it requires two files: features.conf and treedecomposition.conf. The feature.conf file contains information on all the features in the dataset with a flag that indicates if it is categorical or not. 0 indicates it is continuous, and 1 indicates it is categorical. Additionally, it must indicate which relation a feature belongs to. For example, the 'user_id' feature in the Movielens dataset would be stored as "user_id :: 0 :: users". Similarly, the treedecomposition file contains information about the tree breakdown of the relation used to generate queries at the view layer of the LMFAO system, including all the features in each relation and all the edges in the join order of the dataset. While data transformation for the LMFAO system, implemented under to_lmfao in the PolyDataset class, supports 5 out of 8 integrated datasets in OpenDBML used to develop and test the framework, it fails to transform the Openflights, Expedia, and Bookcrossing datasets successfully. Upon investigation, it was found that Openflights has a tuple of attributes as its primary key (composite key), and the existing implementation only supported single-key relationships. To generate CSV files, the function iterates through the join order of the dataset with the help of the primary-foreign key relationship. To accommodate the Openflights dataset and datasets with composite key relationships, the to_lmfao function now checks if a key is a tuple (a set of keys) and retrieves the next relation in the join order accordingly.While the Bookcrossing and Expedia datasets did not run into any errors during their data transformation, the LMFAO system could not train on them as the features in their dataset contained hyphens. The LMFAO system, written in a C++ engine, interprets hyphens as special characters and, hence, cannot read through the configuration files of these two datasets with hyphen-containing features. To fix this, the to_lmfao function now reads through the configuration files to convert any feature names with hyphens in them and change them into an underscore. Thus, "user-id" would become "user_id". These changes allowed for seamless integration of the three datasets.

3.3 Implementation of Morpheus

Morpheus utilizes the normalized matrix format to perform LA operations efficiently over relational data [3]. A normalized matrix is defined as a matrix triple, $TN \equiv (S, K, R)$, where:

- S and R represent the attribute tables derived from the source relational tables in a sparse matrix format.
- K is an indicator matrix formed from horizontally stacking the foreign keys of the target table.

For this reason, Morpheus requires all of its relations to be in sparse matrix format with the naming convention "MLS.txt" for the target table or "MLR{number}.txt" for attribute tables, whereas they require the target and foreign key features to be stored as csv files under the names "MLY.csv" or "MFK{number}.csv" where the number represents the relation number in the join order.

The data transformation for Morpheus is implemented in the PolyDataset class of the framework under the alias to_morpheus. To convert the MovieLens dataset into sprase matrices, to_morpheus takes the the target attribute 'rating' and join order of the dataset as its arguments. It iterates through the join order of the target relation "ratings". The function then checks if the target attribute 'rating' is present in the relation 'ratings' and, if true, adds it to the list of column names to drop from the target relation. It then continues to traverse through the join order, the next relation in the order is users, it checks if the public key user_id of users is present in ratings, and if true, it adds the to the list of column names to drop from the target relation. It then saves this column as "MLFK1.csv" since user is the first relation in the join order of target relation ratings. A key can either be a single attribute or a combination of attributes represented as a tuple and hence the implementation checks for both cases. In the case of a combination of attributes, to_morpheus checks if the attributes in the tuple are present in the target table and reassigns a new index starting from 0 for every new combination of keys. These indices are then stored in a file 'MLK{number}.csv'. Next, to_morpheus checks the the join_order of the current relation to check if it has any subrelations under it and merges all the sub-relations with the current relation using the `merge()` function offered by python's pandas package.[2] In the Movielens dataset, this condition is met when the transforming the movies relation. As seen in Figure 3.1, movies, has a sub-relation genres, hence, `to_morpheus` merges movies and genres into one relation on the key "movies_id". The current relation is then converted into a sparse matrix using one-hot encoding and the coordinates of the non-zero are stored under "MLRnumber.txt" `scipy.sparse.coo_matrix` offered by python's scipy package.[3]

[2] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.merge.html
[3] https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.coo_matrix.html

3.4 Train-Test Split

The Train-Test Split functionality of OpenDBML is implemented at the transformation level for each in-database system for increased efficiency(Refer to Appendix for algorithm). However, it follows a general approach behind how the relations in a dataset are split into test and train relations. We again employ an iterative approach rather than a recursive one for time efficiency. For a clearer understanding of how the train test split is done we conduct a dry run of the algorithm on the MovieLens dataset. The splitting process begins by splitting the target relation into train and test relations and saving their indices in the dictionary. Next, we intialise an empty stack and iterate through the join_order of the target relation and slice the data frame of each relation using these the train-test indexes stored in split. So for the first relation in the join order, users, the algorithm extracts the 'user_id' foreign key from target relation ratings and stores the unique user_id values that are in train and test split of 'ratings' separately. It then slices the users relation into train and test split according to these unique values using the `Pandas.isin()` function [4]. For example, `user[user[user_id].isin(unique_user_id_values[train])]`. The indices of the train and test split of users in a new dictionary. Finally, users, user_id, user's join_order and it's relation_train_test split are appended into the stack. This process repeats for movies, and interactions_user_id..Following this, the algorithm proceeds to process each of the relations in the stack one by one and split the relation according to the indexes in its train_test_split dictionary. Then it proceeds to check if the relations have any sub-relations under it and if true, it proceeds to create a sub_relation_split the same way as before. So when the stack is processing the relation 'movies' it, it creates a split for it's sub-relation genres using the key movies_id. This approach respects the hierarchical and relational dependencies within the dataset and splitting it based on foreign keys ensures that all related attribute values are allocated to the same split dataset.

3.5 Feature Transformation

The feature transformation in OpenDBML is implemented in the PolyDataset class under the alias modify_attributes, a wrapper function that allows users to apply appropriate functions to any attribute in the dataset. The function takes as input a list of tuples, where each tuple contains the relation_name, attribute, transformation function, and optionally a new name for the transformed attribute. A user also needs to provide the join_order of the dataset and optionally a boolean value update_schema indicating if they want to update the schema of the relations. Next, the feature transformation happens in three steps:

1. **Locating the relation:** The modify_attributes function conducts a breadth-first search on the nested dictionary join order of the dataset to locate the relation of the attribute the user wants to modify and extract it. Let's say we want to transform the movies_id attribute in the relation genres. The function traverses through the relations ratings, users, movies, and finally genres. We find breadth-first search to

[4] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.isin.html

be the ideal approach for two reasons, first, in case a user wanted to modify the attribute of a higher-level relation, the function wouldn't have to unnecessarily traverse through the depth of other relations and secondly, most datasets don't have a very high level of nesting.

2. **Feature Transformation:** Next, to transform the attribute of the relation, we call the function apply_function() implemented in the RelationalDataFrame class. This function is responsible for the actual feature transformation. It takes in the attribute that needs to be transformed, the function that needs to be applied, and optionally a new name for the modified attribute. It then checks if the attribute is present in the of the relation and then applies the function to the column of the dataframe [5]. If a new name for the modified attribute is provided, it creates a new column with this alias; otherwise, it transforms the original column. The function returns the modified function as a pandas.series.[6]

3. **Updating Schema:** Finally, if the update_schema flag is set to True, it calls the update_schema function that is implemented in the DatasetConfig() class. This function takes the relation, the original attribute name, and the new attribute name as input. It then traverses through the nested schema of the relation using a breadth-first search approach again and updates the schema with the name and datatype of the new transformed attribute.

We also created a file in the implementation of OpenDBML that offers popular feature transformation and scaling functions. It offers the following: Min-Max Scaling, Standard Scaling, Robust Scaling, Maximum Absolute Scaling, Log Transform, Polynomial Transform.

3.6 Data Shuffling

The data transformation feature of the new implementation is implemented at the data transformation level. In order to shuffle the dataset, a user simply needs to set the shuffle flag in the transformation functions, to_svml, to_morpheus and to_lmfao to true. Internally, the shuffling is done right before the train-test split of the dataset. If the flag is set to true, each of the transformation functions passes the same flag to the pandas test_train_split function that splits the indices of the target table. This is sufficient as shuffling the target table results in shuffling the foreign keys of the attribute tables in the join_order, successfully shuffling the entire dataset.

3.7 Get Statistics

The get_statistics function, implemented under the PolyDataset class, enables users to obtain descriptive statistics for attributes within their dataset. It requires the dataset's join order and optionally accepts a relations tuple, specifying the relation name and the list of attributes for statistical analysis. Users may also provide a features list directly, targeting

[5] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html
[6] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.html

specific attributes without needing to identify individual relations.When relations are specified, the function conducts a breadth-first traversal of the join_order to access the desired relation. If relations are not provided, the function combines all relations in the dataset. Internally, it utilizes the pandas.DataFrame.describe function [7] to compute the count, mean, standard deviation, minimum, 25th percentile (first quartile), median (50th percentile), 75th percentile (third quartile), and maximum for the specified attributes or the entire relation/dataset. By setting the `additional` flag, users can also receive statistics on skewness, kurtosis, and the counts of missing and unique values for the selected attributes.

```
------------------------------------------------------------
Statistics for dataset: MovieLens
------------------------------------------------------------

Descriptive Statistics:
              age      rating
count  2101815.0  2101815.00
mean         2.49        3.58
std          1.35        1.12
min           0.0        1.00
25%           2.0        3.00
50%           2.0        4.00
75%           3.0        4.00
max           6.0        5.00

Additional Statistics:
         skewness  kurtosis  missing_values  unique_values
age          0.80      0.28               0              7
rating      -0.55     -0.36               0              5
------------------------------------------------------------
```

Figure 3.3: Descriptive Statistics for Movielens Dataset

3.8 Principle Component Analysis

The pca function, implemented in the PolyDataset class, allows users to perform Principal Component Analysis (PCA) on a dataset to reduce its dimensionality while retaining essential variance. The function inputs the join order of the dataset and the number of components they want. The users can also provide a tuple containing a relation name and its feature list if they wish to perform PCA analysis on a specific relation. Alternatively, they can simply input a feature list for dimensionality reduction without specifying a relation. In cases where a relation is specified, the function retrieves the relation by a BFS of the join order, or else it proceeds to join all the relations into a single dataframe. When a feature list is provided, the function trims the relation or the merged data frame to include only these specified features. If no specific features are selected, PCA is conducted across all attributes within the chosen relation or the comprehensive dataframe.

Next, the function proceeds to perform feature scaling using the scalers provided by

[7] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.describe.html

sklearn's preprocessing package[8]. By default, the function applies standard scaling, though users can opt for alternative scaling methods—min-max, max-abs, robust, or standard scaling—by adjusting the scaling parameter accordingly. After scaling the features, the function proceeds to perform the standard procedure for principal component analysis. It first calculates the covariance matrix of the scaled features using numpy's cov [9]method. Next, it calculates the eigenvalues and eigenvectors of the covariance matrix. Following this, the eigenvalues and their corresponding eigenvectors are sorted in descending order of eigenvalues (which determine the magnitude of variance). Finally, the function selects the top n_components eigenvectors that have the highest variance and projects the scaled data onto these eigenvectors. The users can choose to plot the first two components with the highest variance against each other by setting the 'plot' parameter of the function to True. This helps reveal any patterns and clusters as well as any outliers in the data. The function returns a 2-dimensional array of the principal components and the list of features used for dimensionality reduction. 3.4 shows the the plot of the first two components for all the attributes in the MovieLens dataset.

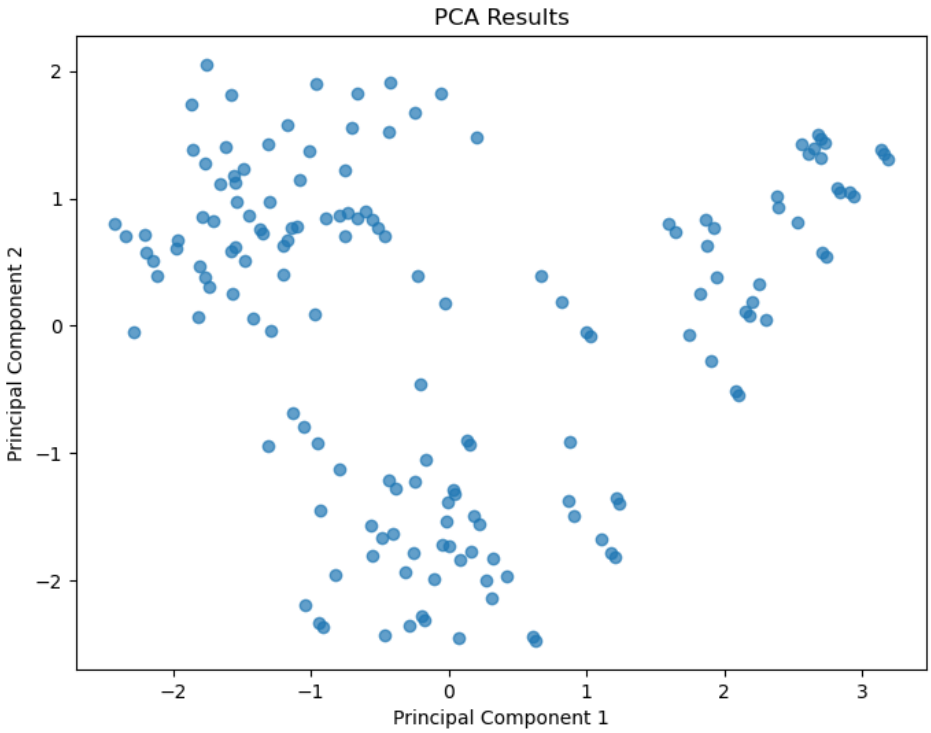

Figure 3.4: A scatter plot PCA1 and PCA2 for MovieLens

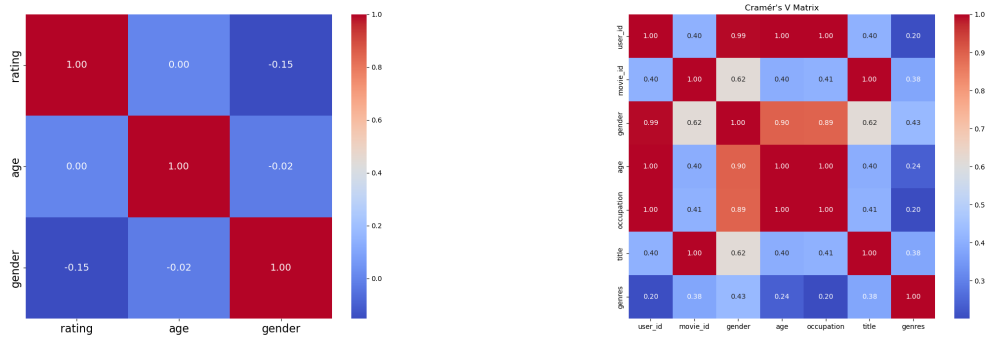

Figure 3.5: Correlation and Cramers' V Matrix for Movielens

3.9 Correlation Analysis

The correlation function, implemented in the PolyDataset class, is used to calculate either the Pearson's correlation for continuous variables (or between a binary categorical variable and a continuous variable) or the Cramér's V coefficient for categorical variables. It takes the join order of the dataset, and optionally, a list of features and a target variable. Users may also provide an optional list of variables under the list "treat_as_numerical" which, as the name suggests, allows the function to treat these variables as numeric. The function first merges all the relations according to the join order of the dataset. Next, if a feature list has been provided, the function filters out the specified features from the dataset. However, in the absence of a feature list, the function proceeds with all the attributes in the dataset. The function then separates numerical and binary categorical features into one matrix, and nominal features into another. It calculates Pearson's correlation for all the numerical/binary features and stores it in one matrix. Next, it computes the Cramér's V coefficient for all nominal features and stores it in a separate matrix. If a user has provided a list of features that need to be treated as numerical, it includes these features in the numerical correlation matrix and calculates their 'numerical' coefficient. No typecasting is needed for this step as OpenDBML by default encodes all the string features into numerical ones during configuration. Finally, a user can choose to set the flag 'plot' to True, in which case the function displays two heatmaps for the correlation matrix and Cramér's V matrix. The function returns the two matrices. Figure 3.5 shows the correlation matrix for age, gender and ratings feature in the MovieLens dataset and all the features for cramers' v matrix.

3.10 Exploratory Plots

The plot function implemented in the PolyDataset class allows users to generate plots that help identify any hidden pattern between two features in the dataset. This helps users in getting better insights into any hidden relationship between two features in

[8]https://scikit-learn.org/stable/modules/preprocessing.html
[9]https://numpy.org/doc/stable/reference/generated/numpy.cov.html

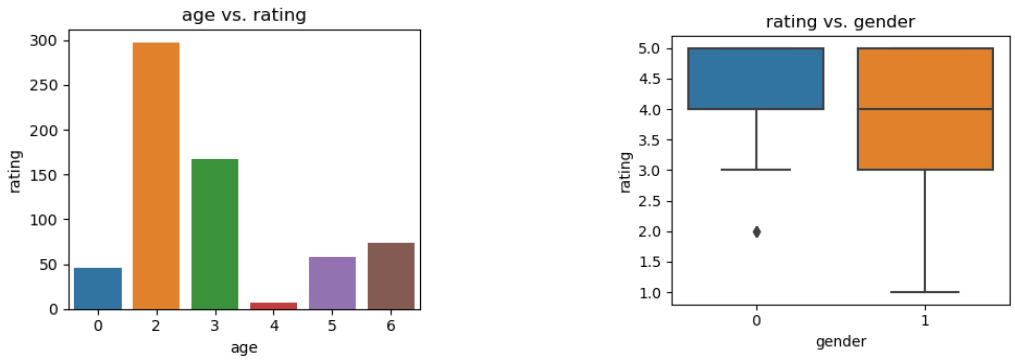

Figure 3.6: Exploratory Plots for Movielens

the dataset that may not come to notice from initial statistical insights. The plot function inputs the join_order of the dataset, and optionally a list of features and a list of categorical attributes that need to be treated as numerical. Internally, the function sorts the schema which is implemented as a dictionary, of the dataset into two lists of numerical and categorical attributes. If the user has provided a list of attributes that needs to be treated as numerical, the function adds these atttibutes to the numerical list. Next, the function proceeds to join all the relations in the dataset according to the join order. If a list of features are provided, only these attributes are selected from the joined table. Next, the function calculates the number of the subplots to be made based on the number of attributes in the joined table. Next, it iterates through all pairs of attributes in the joined table and generates plot. It checks if the same pair of variables has been plotted before and if true, skips to the next pair. This avoids redundancy in the visualisation and prevents confusion. Finally, the function generates different plots depending on the data type of the two variables. For two numerical variables, it generates a scatter plot, for two numerical variables, a box plot for a categorical and numerical variable (and a box plot for vice versa), and a count plot for two categorical variables using pandas seaborn package.[10]

[10]https://seaborn.pydata.org/

Chapter 4

Experimental Results

The following chapter covers the experimental results used to evaluate the impact of our work. Now that we have successfully added the intended functionalities, we use the new features in the OpenDBML API to answer the following research questions :

1. How do the three In-database machine learning systems compare in performance for simple machine learning tasks?
2. How does applying different feature engineering techniques impact the performance of machine learning models?
3. Can the present implementation of OpenDBML be used as a complete end-to-end to democratise In-database machine learning systems?

4.1 Experimental Setup

Our experimental setup is based on a MacBook Pro running Sonoma 14.4, equipped with a 6-Core Intel Core i7 processor at 2.6 GHz and 16 GB of DDR4 RAM, ensuring efficient handling of computational tasks. LibFM and LMFAO were compiled using Apple Clang version 15.0.0. We utilized libFM 1.4.2 for our experiments and generated the necessary binary files for model training via a custom bash script. Morpheus which is originally implemented in Python 2 is not compatible with the operating system used and hence an open-sourced Python3 version of the system was employed to conduct the experiments. [1]Additionally, individual python scripts were written for train all the datasets and the algorithms offered by Morpehus. The original LMFAO engine did not output the RMSE values of its linear regression model, and hence the modified version was used to conduct the experiments. Additionally, it was compiled using CMake version 3.28.3. Two of the datasets used in the experiments, the Expedia and Favorita datasets were cut down to 10% of their original size as they did not run on the Morpheus system otherwise.

[1]https://github.com/baymaks/MorpheusPy.git

4.2 Evaluation Metrics

To evaluate the performance of the machine leearning models offered by the three supported systems in OpenDBML, we relied on two primary evaluation metrics:

- **Root Mean Square Error (RMSE)** for assessing the accuracy of the models. The RMSE quantifies the difference between the predicted and the actual values of the dependent variable. It is calculated using the formula:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{4.1}$$

  where y_i is the actual value, $\hat{y}_i$ is the predicted value by the model, and n is the number of observations. A lower RMSE value indicates higher accuracy of the model.

- **CPU Runtime** to measure the time efficiency of the model training and prediction phases. We utilized the `time` command in bash to obtain the total duration (real time) of these processes. This metric helps in understanding the practicality of using these systems for real-world applications.

4.3 Datasets

For a fair comparison, we employed the same 8 datasets used in the original OpenDBML implementation [5] to analyse the performance of the three systems. The datasets include:

- **MovieLens:** A collection of movie ratings by users, widely used for recommendation systems and collaborative filtering studies. This dataset provides insights into user preferences and movie popularity.[2]
- **OpenFlights:** Contains information on airport locations, airline routes, and flight distances, facilitating analysis on transportation networks and optimization problems. [3]
- **BookCrossing:** Compiled from the BookCrossing community, it comprises user ratings of books, offering a basis for understanding reading habits and preferences for recommendation systems. [4]
- **Yelp:** Features reviews and ratings of businesses by users. It is used in business recommendation systems, and social network analysis. [5]
- **Expedia:** This dataset includes hotel booking information, user clicks, and

[2] https://grouplens.org/datasets/movielens/1m/

[3] https://old.datahub.io/sv/dataset/open-flights

[4] https://github.com/ashwanidv100/Recommendation-System—Book-Crossing-Dataset/blob/master/README.md

[5] https://www.kaggle.com/datasets/yelp-dataset/yelp-dataset

searches, which can be used for predicting hotel booking behaviors and preferences. [6]

- **Walmart:** Comprises sales data across various departments, useful for sales forecasting, market basket analysis, and understanding consumer purchase patterns.[7]
- **LastFM:** Contains user listening data, artist information, and music preferences, suitable for music recommendation and user behavior analysis. [8]
- **Favorita:** A retail dataset from Corporación Favorita, a large Ecuadorian grocery retailer. It includes transactional data for sales forecasting and inventory management studies. [9]

4.4 Performance Comparison

To answer the first research question, we used the linear regression model, a standard machine learning algorithm offered by all three systems, to analyse the runtime and the test RMSE.

Table 4.1: Runtime and RMSE values for Linear Regression for all three systems

	LibFM		LMFAO		Morpheus	
	Runtime	RMSE	Runtime	RMSE	Runtime	RMSE
Movielens	13.323	2.84	2.86	3.44	6.53	2.06
Openflights	0.94	0.01504	15.99	5447	3.76	5.238
Bookcrossing	16.60	4.79	1.81	7.24	7.62	4.62
Walmart	3.63	27827	4.68	334878	6.41	2.768
Yelp	336.65	3.124	2.76	3.629×10^6	64.31	7.42
Lastfm	80.55	644.547	345.78	5952.20	34.47	1.94×10^9
Expedia	4.89	57.7274	220.9	9.709	19.48	58.23
Favorita	25.58	57.57	2.78	33.64	31.75	19.909

[6] https://www.kaggle.com/datasets/vijeetnigam26/expedia-hotel
[7] https://www.kaggle.com/datasets/yasserh/walmart-dataset
[8] https://www.kaggle.com/datasets/hoandan/lastfm
[9] https://www.kaggle.com/competitions/favorita-grocery-sales-forecasting

4.4.1 Quantitative Analysis

Runtime:

- LibFM exhibits variable runtimes across datasets, with a notable increase for larger datasets such as Yelp, suggesting scalability challenges.
- LMFAO generally offers the lowest runtimes, highlighting its efficiency, particularly with smaller datasets. However, its performance on larger datasets like Lastfm indicates potential scalability limits.
- Morpheus demonstrates moderate runtimes, which, while not always the fastest, does not significantly escalate for larger datasets, indicating a degree of scalability.

RMSE:

- LibFM maintains competitive RMSE values across the board, indicating its reliability in delivering accurate predictions across dataset sizes.
- LMFAO shows a wide range of RMSE values, with performance dropping significantly on complex, larger datasets, suggesting accuracy may be compromised under scalability.
- Morpheus consistently achieves lower RMSE values on smaller and medium datasets, showcasing its strength in model accuracy. However, it faces challenges with the Lastfm dataset, indicating potential issues with specific types of large-scale data.

Trade-off Comparison

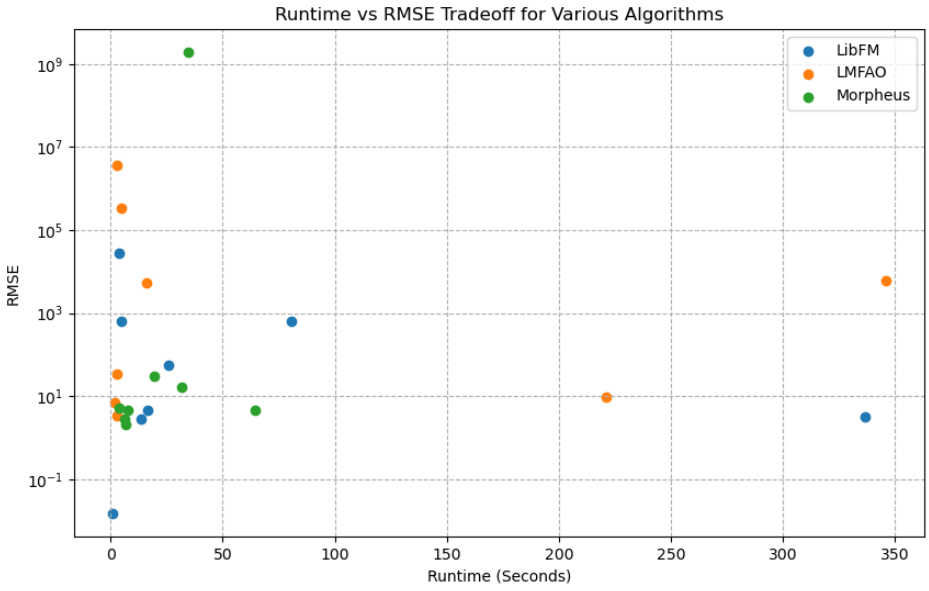

Figure 4.1: Runtime vs Accuracy Plot

The trade-off between runtime and RMSE is visualized in a plot comparing the three

frameworks across all datasets. The plot (Figure 4.1) highlights the efficiency-accuracy balance each framework offers.

- LibFM presents a balanced trade-off, with a tendency towards higher accuracy at the cost of longer runtimes, particularly for larger datasets.
- LMFAO prioritizes efficiency, evidenced by its lower runtimes, but this comes at the expense of accuracy, especially as dataset complexity increases.
- Morpheus favors accuracy, often achieving lower RMSE values, though at the cost of increased runtime, suggesting it's a suitable choice for applications where model precision is critical.

This analysis suggests that the selection among LibFM, LMFAO, and Morpheus should consider the specific needs of the application, balancing the importance of runtime against the necessity for accuracy. For tasks where quick iterations are crucial, LMFAO might be preferred, despite potential compromises in model accuracy. Morpheus is the go-to for scenarios where accuracy is paramount, and LibFM offers a reliable middle ground for a wide range of applications.

4.4.2 Qualitative Analysis

During the development of our experimental setup and while running the experiments, we discovered some qualitative capabilities and limitations when dealing with large and sparse datasets. We noticed that Morpheus does not support very large datasets, which may be attributed to its implementation being in NumPy and storing the data in arrays which causes memory-space issues. On the other hand, LMFAO, which generates specific code for each attribute, does not perform well with highly sparse datasets because of the increased code size and longer run times. In contrast, LibFM is designed to efficiently handle both large and sparse datasets, making it a versatile choice.

4.5 Impact of Feature Engineering

To answer the second research question, we compare the RMSE values of the 8 datasets for linear regression and logistic regression (in Morpheus) for data shuffling, before and after applying different feature engineering techniques. This is followed by a comparison of RMSE values after applying different feature transformation functions to the linear regression and logistic regression algorithms offered by Morpheus.

4.5.1 Data Shuffling

Table 4.2: RMSE values for Linear Regression Before and After Data Shuffling

Dataset	LibFM		LMFAO		Morpheus	
	Before	After	Before	After	Before	After
Movielens	2.84	2.05	3.44	0.789	2.06	2.08
Openflights	0.01504	0.01228	5447	1577	5.238	5.14
Bookcrossing	4.79	4.61	7.24	7.05	4.62	4.58
Walmart	27827	27770	334878	334185	2.76	2.80
Yelp	3.124	3.121	3.629×10^6	2.9525×10^6	7.42	4.53
Lastfm	644.547	634.838	5952.20	5456.47	1.94×10^9	1.8×10^9
Expedia	57.7274	57.5708	9.709	4.634	58.23	29.73
Favorita	57.5706	57.571	33.64	30.57	19.909	16.52

Table 4.2 presents the impact of data shuffling on the RMSE values of linear regression models across three frameworks: LibFM, LMFAO, and Morpheus. The analysis reveals the following insights :

- **LibFM** shows an overall improvement in RMSE values after data shuffling for most datasets, possibly indicating that shuffling may help mitigate overfitting or enhance the model's generalization capability.A notable improvement is seen in datasets like Movielens, where RMSE decreases significantly, which suggests the order of data has a considerable impact on model training.
- **LMFAO** benefits significantly from data shuffling, especially in datasets where initial errors were high, such as Openflights and Yelp. This suggests that the initial data ordering might have led to poor model performance, and shuffling contributes to a more robust model fit.
- **Morpheus** presents a varied response to data shuffling. While some datasets show slight improvements or negligible changes in RMSE, others exhibit a slight increase, indicating a complex relationship between data order and model training efficiency.

Table 4.3: Comparison of RMSE before and after Data Shuffling for Logistic Regression (Morpheus)

Dataset	Before	After
Movielens	5.85	4.11
Openflights	7.85	5.781
Bookcrossing	10.404	8.607
Walmart	7.64	6.56
Yelp	6600.931	6290.08
Lastfm	11280.65	11345.467
Expedia	6180.67	926.59
Favorita	290.150	55.168

Table 4.3 presents a detailed analysis of the impact of data shuffling on the performance of logistic regression models, specifically within the Morpheus framework.

Key Observations:

- The application of data shuffling generally leads to a decrease in RMSE for the majority of datasets, indicating an improvement in model accuracy. This is particularly evident for large datasets such as Yelp and Favorita, where the RMSE significantly reduces post-shuffling.
- An exception is noted in the case of the Lastfm dataset, where a slight increase in RMSE is observed after shuffling. This suggests that for certain datasets, the original order may contain beneficial patterns for model training, or that dataset-specific characteristics might influence the effect of shuffling.
- The overall trend clearly highlights the importance of data shuffling as a crucial step in the data preprocessing pipeline, particularly for improving the performance of logistic regression model.

4.5.2 Feature Transformation

Table 4.4: RMSE comparison for three different transformation functions (Linear Regression)

Dataset	Original	Square	Cube	Log
Movielens	2.227	4.41	3.724	4.08
Openflights	5.39	5.56	5.34	5.63
Bookcrossing	4.55	4.62	4.57	4.602
Walmart	4.87	2.98	4.24	3.34
Yelp	5.93	5.50	5.50	5.55
Lastfm	1221.53	1222.58	1221.86	1222.56
Expedia	38.82	56.16	38.84	25.422
Favorita	16.54	15.89	15.96	14.66

Key Observations:

- The results of the transformations applied to different datasets vary significantly, indicating that there is no one-size-fits-all solution that can be applied to all datasets.
- The application of square transformation to Walmart's dataset has significantly improved its RMSE, which suggests that a quadratic relationship might be more suitable for this particular dataset.
- Log transformation has been found to be highly effective in datasets with skewed features, as it has notably reduced the RMSE for Expedia's and Favorita's datasets.
- Openflights, Bookcrossing, and Yelp datasets have shown minimal changes in their RMSE after applying the transformations, which suggests that these datasets may not benefit much from such transformations.
- The RMSE for the Lastfm dataset remains high even after applying the transformations, which indicates that these transformations may not be sufficient to address the complexity of this dataset.

Table 4.5: RMSE comparison for three different transformation functions (Logistic Regression)

Dataset	Original	Square	Cube	Log
Movielens	5.15	4.83	4.11	4.05
Openflights	7.42	6.36	7.19	6.49
Bookcrossing	9.72	9.75	9.07	10.76
Walmart	2.72	3.01	4.79	3.78
Yelp	65.71	62.48	64.98	52.25
Lastfm	1222.03	1223.48	1223.53	1221.86
Expedia	46.36	39.33	40.55	40.51
Favorita	55.128	52.87	50.48	48.76

Observations:

- Across most datasets, feature transformations, particularly the logarithmic transformation, tend to improve RMSE compared to the original features. This suggests a positive impact of transforming features on model accuracy.
- The logarithmic transformation shows a significant improvement in the Yelp dataset, indicating its effectiveness for features with a wide range or skewness.
- Some datasets like Bookcrossing and Walmart exhibit mixed results with transformations, sometimes increasing RMSE, which underscores the importance of dataset-specific analysis before applying transformations.
- The Lastfm dataset shows minimal variation in RMSE across transformations, suggesting that these feature modifications do not significantly impact the model's predictive ability for this particular dataset.

4.6 Case Study - Anime Dataset

To answer the last research question, we use the added functionalities to a new Anime dataset[10], an open-sourced Kaggle dataset that contains information on anime profiles, reviews, and ratings from MyAnimeList.net, and manga database and social networking site. The dataset contains 2 files:

[10]https://www.kaggle.com/datasets/marlesson/myanimelist-dataset-animes-profiles-reviews

- anime.csv contains a list of anime, with their name, genre, type, number of episodes, user rating, and the number of ratings that have added the anime to their watch list and and anime_id of each anime.
- rating.csv contains information about ratings of users x animes, using user_id, anime_id, and rating.

Objective: We are interested in finding out if the attributes that contribute to the rating of an anime and applying feature engineering techniques to these features to reduce the RMSE value of the predictions made by the linear regression model in Morpheus, which, from Section 4.4, is known to be accurate for smaller datasets such as this one.

Step 1 : Data Wrangling

As shown in Figure 4.2, we first begin by defining the schema of the data files and configure them using CSVConfig. The genres features is anime.csv is a nested column that stores a list of genres that a particular anime falls into, and hence is defined as a new relation. Next the dataset is configured and then created using DatasetConfig and PolyDataset respectively.

```
an_anime_schema = {'anime_id' :str,'name':str, 'genre': {'anime_id' : str, 'genre' : str,},
'type': str,
'episodes' :str,
'rating':str,
'members': str
}

an_rating_schema = {'user_id' :str,'anime_id' :str,'ratings':str}

an_rating_config = configs.CSVConfig(f'{an_dir}/rating.csv',name='rating',schema=an_rating_schema,
pk=['user_id'],
engine = 'python',
names=an_rating_schema.keys(),
encoding='latin-1',
nrows=nrows)

anime_parses = {
    'genre': utils.splitString(',')
}
anime_norms = {
    'genre': 'explode'
}

an_anime_config = configs.CSVConfig(f'{an_dir}/anime.csv',name='anime',schema=an_anime_schema,pk=['anime_id'],
parses = anime_parses,
norms = anime_norms,
engine = 'python',
names=an_anime_schema.keys(),
encoding='latin-1',
nrows=nrows)

an_dataset_config = configs.DatasetConfig(
an_rating_config,
an_anime_config)

an_dataset_config.add_fk(an_anime_config, 'anime_id',an_rating_config ,'anime_id')
an_dataset_config.compress_all()
an_dataset = configs.PolyDataset(an_dataset_config, lazy = True, name = 'Anime')
```

Figure 4.2: Code Snippet of Dataset Configuration

Step 2 : Data Exploration

Next, we are interested in finding out the distribution and range of features that may influence the rating of an anime. Hence, we define the join order of the dataset and use it to call the get_statistics function to get information about the features—episodes (number of episodes), genre, type (movie or series) and ratings.

```
target = 'ratings'
join_order = {
    ('rating', None, None) : {
        ('anime', 'anime_id', 'anime_id') : {
            ('genre', 'anime_id', 'anime_id') : None
        }
    }
}
an_dataset.get_statistics(join_order, features = ['episodes', 'genre','type', 'ratings'])
```

Figure 4.3: Code Snippet of Data Exploration

From Figure 4.4, we find that an anime series on average has 47 episodes, and that the number of times an anime has been added to the watch list of an user ranges from 0 to 6706. The ratings of an anime range from -1 (if the anime is not rated) to 11, and hence, the number of unique rating values is shown as 12.

```
------------------------------------------------------------
Statistics for dataset: Anime
------------------------------------------------------------

Descriptive Statistics:
         episodes     ratings       genre     members
count  34872409.0  34872409.0  34872409.0  34872409.0
mean        47.28        6.05       25.82     3425.98
std          45.1        3.76       14.97     1901.21
min           0.0         0.0         0.0         0.0
25%          19.0         2.0        12.0      1842.0
50%          26.0         8.0        27.0      3424.0
75%          87.0         9.0        37.0      5112.0
max         187.0        11.0        82.0      6706.0

Additional Statistics:
          skewness  kurtosis  missing_values  unique_values
episodes      0.73     -0.62               0            185
ratings      -0.63     -1.28               0             12
genre        -0.04     -0.80               0             82
members      -0.05     -1.15               0           6487
------------------------------------------------------------
```

Figure 4.4: Output of get_statistics

After this, we explore the linear relationship between numerical variables and hence we are interested in the pearson's correlation between the features. For this, we use the correlation function.

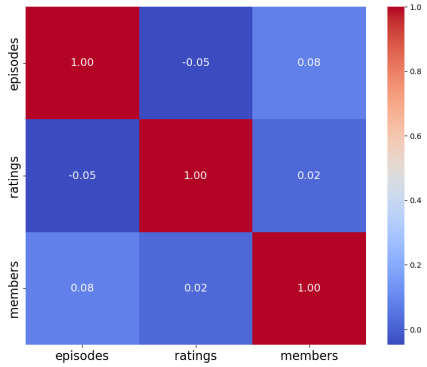

Figure 4.5: Correlation Matrix for Anime dataset

We find that the rating of an anime does not have a significantly strong linear correlation with its popularity and and the number of episodes, however the correlation matrix indicates that there is a negative correlation between the number of episodes and the rating of an anime while there is a positive correlation between the popularity of the anime determined by the number of times its been added to a user's watch list (members) and the rating.

As a final step in data exploration, we want to find out any hidden clusters between the numerical features in the dataset, and hence, we employ the pca function to reduce the dimensions of the features members and episodes and plot the first two components with the highest variance.

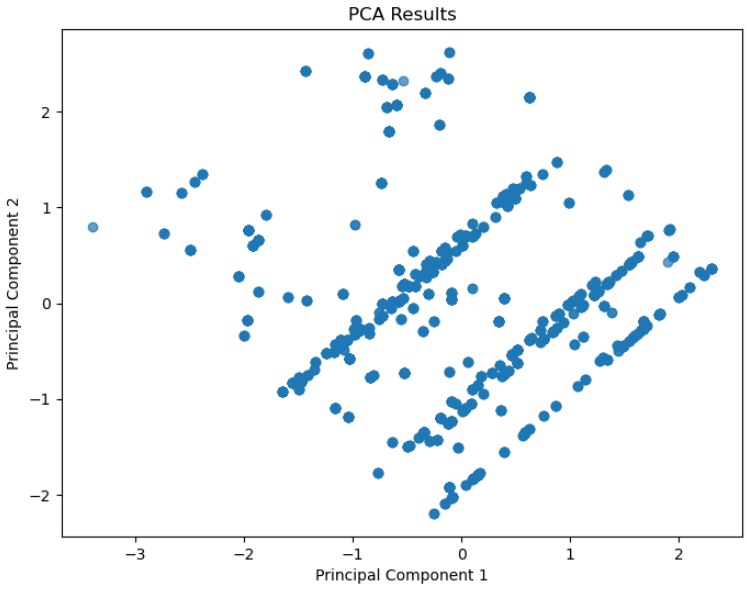

Figure 4.6: Scatter Plot of PCA1 and PCA2 for Anime dataset

Figure 4.6 shows that the transformed features have a linear relationship. This could possibly imply the components may influence the prediction of ratings

Step 3: Feature Engineering and Data Transformation

Finally, we transform the data into sparse matrix format for the Morpheus system by calling the to_morpheus function.To train the model, we define and write a Python script, in the same manner as before, that is used to call the linear regression model API for Morpheus. The Morpheus system has an RMSE of 7.16 after training on the dataset. Applying feature transformation functions on the numerical features does not change the RMSE significantly.

4.7 Qualitative Comparison

The previous implementation of OpenDBML offered data transformation functionalities to use LibFM. However, it had limited support for Morpheus and LMFAO, where OpenFlights, Bookcrossing, and Expedia only worked with LMFAO, and there was no reliable data transformation available for Morpheus. This project has now extended support for all eight datasets for the three systems. This not only increases the adaptability of the framework to a wider range of datasets but also enables users to make an informed decision about which system is best suited for their machine-learning tasks.

Chapter 5

Conclusions

In this project we aimed to present the OpenDBML framework as an end to end for data transformation and consequently democratising the highly efficient in-database machine learning systems. We achieve this by providing users with added functionalities for data tranformation and feature engineering, that allow them to fully harness their data for machine learning tasks. Additionally we also extended data transformation support for Morpheus and extended the support LMFAO for a wider range of datasets. This allowed us to critically evaluate the performance of three state-of-the-art in-database machine learning systems and identifying the strengths and weaknesses of the systems in handling different kinds of data, while realising that libFM to be the most diplomatic system among the three in handling both high volumes and highly sparse datasets. We also evaluate the effect of different feature engineering techniques in the system's performance for prediction tasks. Finally, we use a new dataset to demonstrate all the functionalities added to the framework. The present OpenDBML framework allows users to now plugin their datasets and use either of the three systems, unlike the previous implementation, which only supported libFM.

5.1 Limitation and Future Work

While the user experience with OpenDBML has improved, there are still challenges. The comprehensive process of defining individual schema and configurations for each relation can be tedious for users unfamiliar with database schemas. Therefore, a more streamlined, user-friendly API would be beneficial.To ensure the framework's robustness and applicability to real-world scenarios, it is essential to expand the scope of testing by including a wider array of datasets. This expansion would validate the current enhancements and identify new areas for improvement.Third-party user testing is challenging due to the OpenDBML framework not being open-sourced. Therefore, community feedback is critical to refining and enhancing the framework's features.The implementation of feature engineering techniques has been confined to basic methods due to time constraints. However, there is a need to integrate advanced methods, such as time series manipulation, into the framework for future development.The lack of built-in functionalities for outlier removal and handling of NaN values is a

gap in the framework. Addressing this gap by introducing data cleaning methods would significantly enhance the framework's utility. While OpenDBML supports Principal Component Analysis (PCA) for data exploration, there is still room for exploring its integration as a feature engineering tool. The incorporation of PCA-generated components into the training dataset could unlock new dimensions of model training efficiency and effectiveness.Finally, integrating emerging in-database systems into OpenDBML presents an exciting opportunity. Such additions would enrich the framework's capabilities and adaptability to evolving machine learning and database technologies. By addressing these identified limitations, OpenDBML can continue to evolve as a pivotal tool for democratizing in-database machine learning. This will not only enhance the user experience but also expand the scope of possibilities in the field of in-database machine learning.

Bibliography

[1] S. Rendle, "Factorization machines with libFM," *ACM Transactions on Intelligent Systems and Technology*, vol. 3, no. 3, 2012.

[2] ——, "Scaling Factorization Machines to Relational Data," 2150. [Online]. Available: http://www.kaggle.com/c/WhatDoYouKnow

[3] L. Chen, A. Kumar, J. Naughton, and J. M. Patel, "Towards Linear Algebra over Normalized Data," 2150.

[4] M. Schleich, D. Olteanu, M. Abo, K. Relationalai, H. Q. Ngo, X. Nguyen, and M. A. Khamis, "A Layered Aggregate En-gine for Analytics Workloads," vol. 18, no. 19, 2019. [Online]. Available: https://doi.org/10.1145/3299869.3324961

[5] M. Ghorbani and A. Shaikhha, "Demonstration of OpenDBML, a Framework for Democratizing In-Database Machine Learning." [Online]. Available: https://ace.c9.io/

Appendix A

First appendix

A.1 Algorithms

Algorithm 1 Split data into training and testing sets based on foreign key relationships.

Inputs:
  $train\_ratio$ ▷ Ratio for training set size
  $join\_order$ ▷ Order of table joins or consideration
Outputs:
  $train\_index$, $test\_index$ ▷ Indices for training and testing sets
  $test\_train\_split$ ▷ Mapping of splits to indices
procedure INITIALIZE DATA SPLIT
    Split target column into $train\_index$ and $test\_index$ based on $train\_ratio$
    $test\_train\_split \leftarrow$ {'train' : $train\_index$, 'test' : $test\_index$}
end procedure
procedure INITIALIZE EMPTY STACK
    Create an empty stack for managing relation iterations
end procedure
for each relation in $join\_order$ **do**
    Retrieve unique FK values in train and test splits of the target column
    Determine indices of relation rows in train and test splits
    $relation\_split \leftarrow$ {'train' : $relation\_train$, 'test' : $relation\_test$}
    Push relation details onto the stack
end for
while stack is not empty **do**
    Pop details of the current relation from the stack
    Separate indices for training and testing sets
    Identify relation rows for each set
    if $current\_join\_order \neq$ None **then**
        for each sub-relation in $current\_join\_order$ **do**
            Retrieve and match unique sub-foreign keys
            Determine sub-relation splits
            Append details to the stack for further processing
        end for
    end if
end while

Algorithm 2 Data Preparation for Morpheus

```
Input: out_dir, target_col, join_order, train_ratio, shuffle
if dataset not configured then
    Configure dataset
end if
Initialize counter: relation_counter ← 0
Initialize list: columns_to_drop ← []
for each relation in target table's join_order do
    relation_counter ← relation_counter + 1
    current_relation ← get relation from list of all relations
    df ← get defined data frame of current_relation (no NaN values)
    Add target_col to columns_to_drop
    Save target column in "MLY.csv"
    if foreign key is a string then
        Add foreign key to columns_to_drop
        Save foreign key in "MLFK_{relation_counter}.csv"
    else if foreign key is a tuple then
        Extract and combine keys from tuple
        Assign new index for combinations
        Save in "MLFK_{relation_counter}.csv"
    end if
    if current relation's join order is not None then
        Merge sub_relations with current_relation
        One Hot Encode merged relation
        Convert to COO matrix, indices start at 1
        Save in "ML_R{relation_counter}.txt"
    else
        One Hot Encode current relation
        Save in "ML_R{relation_counter}.txt"
    end if
end for
Drop columns in columns_to_drop from target table
if target table not empty then
    One hot encode target table
    Convert to COO matrix, indices start at 1
    Save in "MLS.txt"
end if
```

Appendix B

Participants' information sheet

If you had human participants, include key information that they were given in an appendix, and point to it from the ethics declaration.

Appendix C

Participants' consent form

If you had human participants, include information about how consent was gathered in an appendix, and point to it from the ethics declaration. This information is often a copy of a consent form.