

HYPERPARAMETER DATABASE PROJECT

GROUP DB16

Presented by

Mansi Nagraj
Dhawal Priyadarshi
Mayur Vyas

Abstract:

"A good choice of hyperparameters can make an algorithm shine".

The project aims to develop a relational Database that serves as a backend to store the details about various experiments performed on diverse datasets from different industrial domains such as retail, healthcare, finance etc. It provides useful insights and recommends best-hyperparameter values for a given set of conditions by means of use-cases written in SQL.

Introduction:

Model optimization is one of the greatest challenges in the implementation of deep learning principles[2]. Hyperparameters are settings that can be tuned to control the behavior of a machine learning algorithm.

Hyperparameters are important because they directly control the behavior of the training algorithm and have a significant impact on the performance of the model is being trained. Conceptually, they can be considered orthogonal to the learning model itself as they live outside the model, yet there is a direct relationship between them.

Hyperparameter- optimization has two benefits:

- Efficiently search the space of possible hyperparameters
- Easy to manage a large set of experiments for hyperparameter tuning [1]

The project operates in three broad sections to develop a recommender for best values of hyperparameters to be used while developing a ML model for any dataset.

Experiments are performed on a various datasets for different set of run-times using H2O software to generate models with varying hyperparameter values governing their output.

The JSON files generated by the software are then analyzed and modelled into a MySQL Database to store the data of various runs. It involves writing useful views, functions and stored procedures to enhance the ease of use of the Database. Indexing is done to improve Db-performance. The third section involves developing a front-end interface that could be interfaced with the Database to recommend different insights to the users.

This report focuses on the conceptual schema design, modelling and developing SQL scripts to deal with the Database and targets to develop meaning use-cases to serve the end goal of recommending ideal hyperparameter values.

Data Gathering and JSON Parsing:

The dataset in consideration (on which the ML models were trained) was obtained from Kaggle NYS Salary Information for the Public Sector [5], which is provided by the State of New York under Creative Commons public license. This data was run on H2O for various run-times for the hyperparameter search. Therefore, the input came in two forms:

- *Dataset related meta-data* – Pulled from Kaggle describing the dataset and variables (see *socrata_metadata_salary-information-for-local-authorities.json*)
- *Leaderboard Metadata and Hyperparameters* – These are files generated for each run containing information regarding leading models, their performance metrics, and extensive list of model-wise metadata and hyperparameters (see files in folders named *DB<num>* inside the *data/* folder of *source/* folder in the parent directory)

Data munging was done using Python to retrieve DB relevant information from the metadata files and to re-formulate them into columns as per the ER model. As most of the input files were json format, the “json” package was used to read and parse the data from these files into json objects in Python. The relevant information was retrieved and loaded into “pandas” data frames designed in the ER model format with each data frame representing each table in the ER. Finally, the data frames were exported into CSV files for import into MySQL database.

Conceptual Diagram:

The hyperparameter search revolves primarily around the leaderboard, which considers the performance of various ML models on the test dataset. The database was conceptualized keeping this point in mind. The Leaderboard and Leaderboard Metadata are two of the most important tables in the schema. While the Leaderboard table stores the specific model related information like performance metrics, name, etc., the Leaderboard Metadata uniquely identifies various search runs with relevant information on type of run (regression, classification, time-series analysis).

As the information on dataset used for the runs is equally important, the dataset metadata was modeled with as much attention, making sure that all information about the dataset that might be useful to the ML modeler is readily apparent like size of data, rows, columns, etc. Another

important feature included here is to be able to add tags to the dataset as the user sees fit and be able to search datasets for modeling based on these tags. The tags could describe the dataset domain like Finance or Healthcare besides any relevant information that the modeler thinks aptly describes the use-case for the data.

It would be impossible to derive insights on the hyperparameters without properly cataloging the hyperparameter values themselves. Therefore, five entities were developed to achieve highly search efficient yet flexible model to store the hyperparameters for each model in every run. The entity-wise details are as follows –

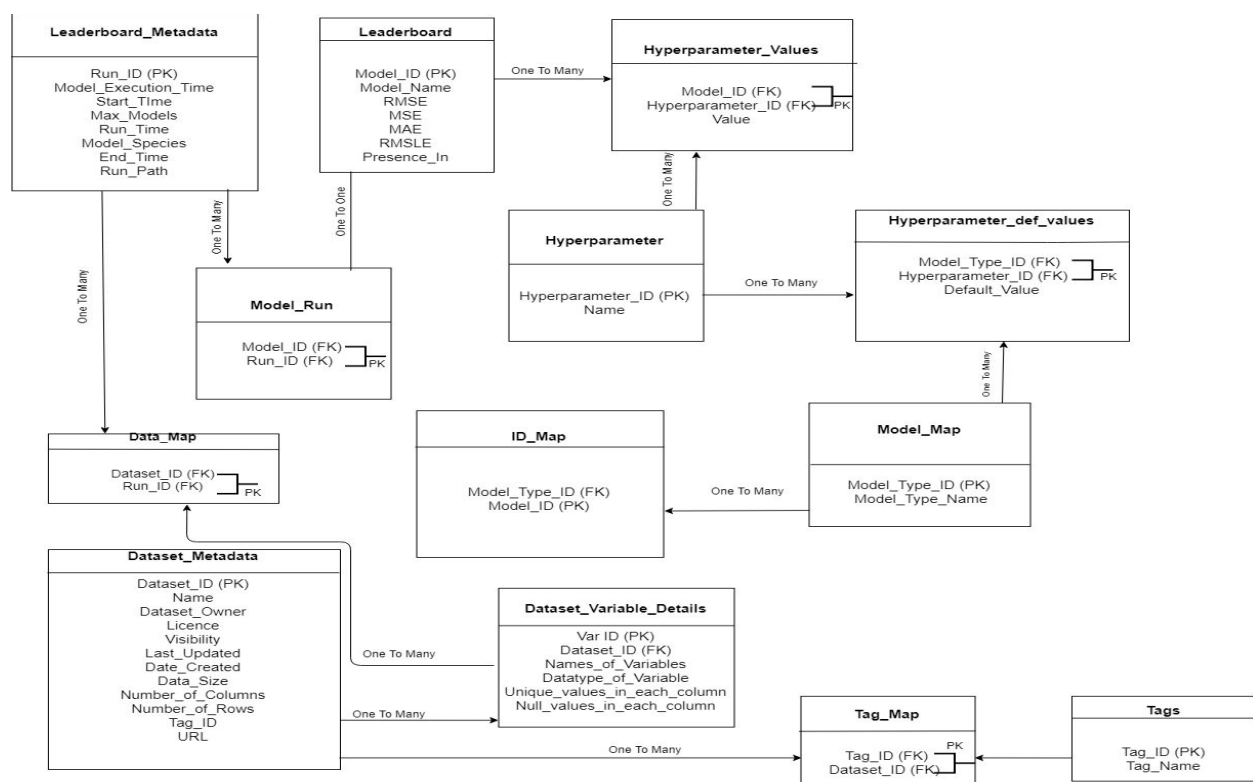
Hyperparameter table – this table stores the unique hyperparameter names and assigns it a unique value.

Hyperparameter Values table – this table stores the hyperparameter values for each model of each run (except default values).

Model Map table – this table describes the general model algorithms used across the runs like GLM, GBM, Neural Net, etc. This does not give the specific model name, which could be found in the leaderboard table.

Hyperparameter Default Values – this table stores only the algorithm related default hyperparameter values as a particular algorithm has a fixed set of default values that does not change across runs.

ID_Map – this table maps a specific model to its model algorithm in the Model Map table.



Final ER Diagram:

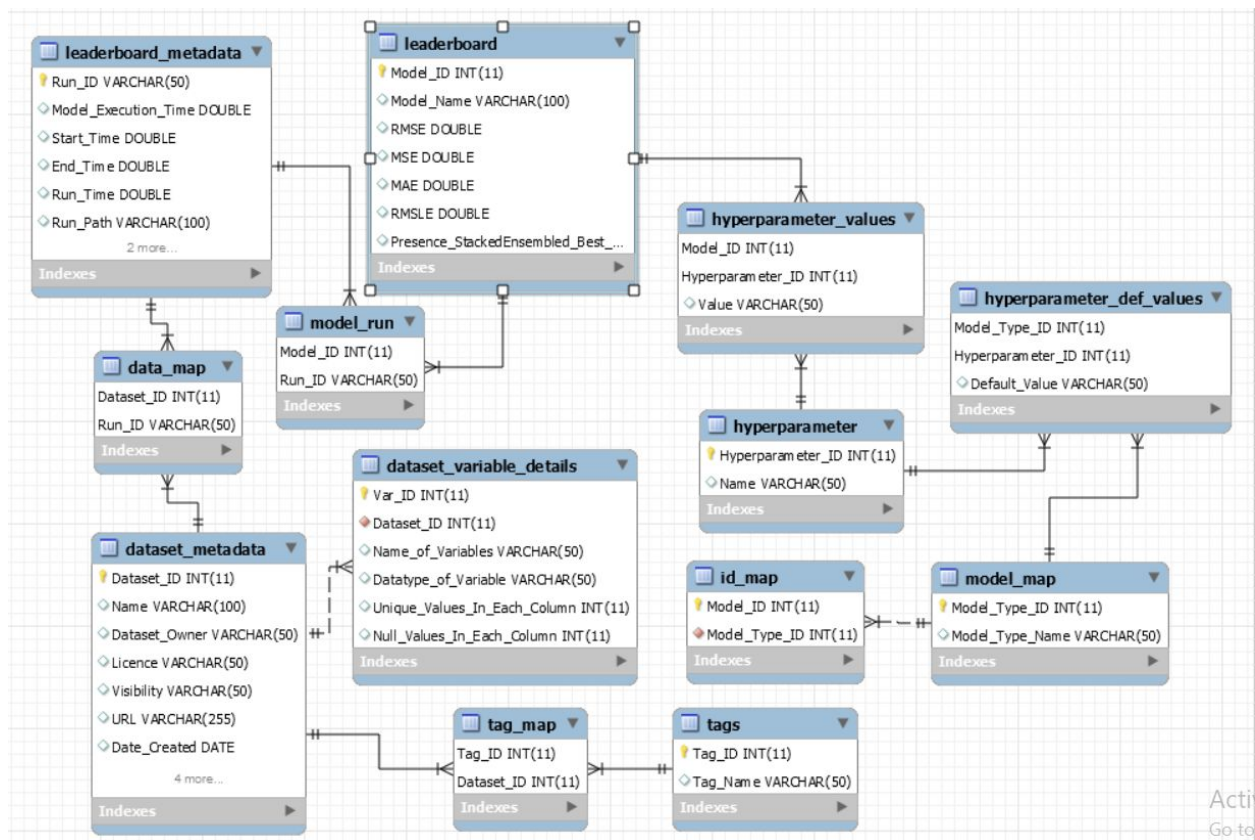


Figure: Final ER Diagram for the DB

Normalization:

Check for the tables if they are in First Normal Form (1NF)

- Each table in our database contains a single value and the records are unique, we have primary key in all the tables for few tables where Primary key is not present we have generated a composite key hence the tables are in the First Normal Form.

Check for the tables if they are in Second Normal Form (2NF)

- Requirements for 1NF are satisfied and data doesn't have partial dependencies helping us achieve Second Normal Form.

Check for the tables if they are in Third Normal Form (3NF)

- Requirements for 2NF are satisfied and it does not have transitive dependencies helping us to achieve Third Normal Form.

Some Use-Cases Explained:

Case 01: Suggest latest 10 datasets used by other users based on an industry domain.

```
33 # Case 01: Suggest latest 10 datasets used by other user's based on an industry domain (Tag given to a dataset)
34 • select D_Mdata.Name, D_Mdata.URL, D_Mdata.Data_Size, D_Mdata.Number_of_Rows, D_Mdata.Number_of_columns
35 from dataset_metadata as D_Mdata
36 inner join Tag_map as TM on TM.Dataset_ID = D_Mdata.Dataset_ID
37 where TM.Tag_ID = (select Tag_ID from tags where tag_name = "employee")
38 order by D_Mdata.Last_updated desc
39 limit 10;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

Name	URL	Data_Size	Number_of_Rows	Number_of_columns
Salary Information for Local Authorities	https://www.kaggle.com/new-york-state/hys-s...	59570291	347601	20

Explanation: The case asks the user to input the name of the industry such as finance, products etc. and recommends the most recent 10 Datasets present in the Database that were used for study for this domain (identified by the Tag assigned to a dataset). It displays the Name, Data-size, Source URL, Number of Rows and Columns present in the dataset.

Case 02: For the datasets from a particular industry (tag), size (observations) and type of analysis to be done (Regression /Classification/Time-series forecasting), which is the best model?

```
52 # Case 06: For the datasets from a particular industry (tag), size (observations) and type of analysis to be done
53 # (Regression/Classification/Time-series forecasting), which is the best model?
54 • select MR.Model_ID, (RMSE/(Dt_Mdata.Number_of_Rows*Dt_Mdata.Number_of_columns)) as metric
55 from dataset_metadata as Dt_Mdata
56 inner join Data_Map on Data_Map.Dataset_ID = Dt_Mdata.Dataset_ID
57 inner join leaderboard_metadata as LDR_Mdata on LDR_Mdata.Run_ID = Data_Map.Run_ID
58 inner join model_run as MR on MR.Run_ID = LDR_Mdata.Run_ID
59 inner join leaderboard as LDR on LDR.Model_ID = MR.Model_ID
60 inner join tag_map on tag_map.Dataset_ID = Dt_Mdata.Dataset_ID
61 where LDR_Mdata.Run_time = 300
62 and Dt_Mdata.Dataset_ID in (select Dataset_ID from Tag_Map where Tag_ID in(select Tag_ID from tags where tag_name like "employee%"))
63 and Dt_Mdata.Number_of_Rows between 500 and 350000
64 and Dt_Mdata.Number_of_columns between 5 and 50
65 and tag_map.tag_ID = (select Tag_ID from Tags where tag_name like "employee%")
66 and LDR_Mdata.Model_Species = "regression"
67 order by metric desc
68 limit 1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows: |

Model_ID	metric
4	0.005320196407004022

Explanation: The case accepts tag-name, No. of rows and columns in user's dataset and the type of analysis to be performed as the input and displays the model-id of the best-model

based on a metric that takes into account the RMSE for regression models and AUC for classification. It also considers matching the observations present in user's dataset and the ones used as a basis for recommending.

Case 03: Calculate difference in hyperparameter values as compared to the default for a particular Model Type (GBM/GLM/Linear/Logistic) and run-time.

```

111 # Case 13: Calculate change in hyperparameter values for a particular Model_Type and run-time
112 • select hyp_val.hyperparameter_ID , hyp_val.value, hyp_def_val.default_value, (hyp_def_val.default_value-hyp_val.value) as diff
113 from hyperparameter_values as hyp_val
114 inner join hyperparameter_def_values as hyp_def_val on hyp_def_val.hyperparameter_ID = hyp_val.hyperparameter_ID
115 where hyp_val.Model_ID in
116 (select model_ID from Model_Run
117 where Run_ID in
118 (select Run_ID
119 from leaderboard_metadata
120 where Run_time = 300 and Model_species = "Regression"));
121

```

hyperparameter_ID	value	default_value	diff
0	4288133765923973916	-1	-4.288133765923974e18
0	4288133765923973916	-1	-4.288133765923974e18
1	0.1	0.1	0

Explanation: The case asks the user for a run-time such as 300 /500/1000 sec for H2O and the type of model that the user wants to study. It then displays the default, actual and difference in the two values for all the models of that type and matching run-time.

Case 04: For regression model, enlist the no. of linear, logistic, GBM and GLM models generated for a particular runtime.

```

76
77 # Case 08: For regression model, enlist the no. of linear, logistic, GBM and GLM models generated for a particular runtime
78 • select count(IM.model_ID), MM.Model_Type_Name
79 from ID_Map as IM
80 inner join model_run as MR on MR.Model_ID = IM.Model_ID
81 inner join model_map as MM on MM.Model_Type_ID = IM.Model_Type_ID
82 where MR.Run_ID in (select Run_ID from leaderboard_Metadata where run_time = 300 and model_species = "Regression")
83 group by IM.model_type_ID;
84

```

count(IM.model_ID)	Model_Type_Name
1	GBM
1	GLM
2	Ensemble

Explanation: The case enlists the count of each type of model generated during a particular run for a given dataset.

Views:

They are basically used for security and simplicity, below we have shown few of the view queries and their output.

View 01: To view the basic details about the given Dataset – Named “dataset_basicdetails”

Query to generate the view

```
CREATE
  ALGORITHM = UNDEFINED
  DEFINER = `root`@`localhost`
  SQL SECURITY DEFINER
VIEW `hyperparameter_db`.`dataset_basicdetails` AS
  SELECT
    `d_mdta`.`Name` AS `Name`,
    `d_mdta`.`Date_Created` AS `Date_Created`,
    `d_mdta`.`Number_of_Columns` AS `Number_of_Columns`,
    `d_mdta`.`Number_of_Rows` AS `Number_of_Rows`,
    `d_vardt`.`Name_of_Variables` AS `Name_of_Variables`,
    `d_vardt`.`Datatype_of_Variable` AS `Datatype_of_Variable`
  FROM
    (
      (`hyperparameter_db`.`dataset_metadata` `d_mdta`
      JOIN `hyperparameter_db`.`dataset_variable_details` `d_vardt` ON
      ((`d_mdta`.`Dataset_ID` = `d_vardt`.`Dataset_ID`)))
    )
  WHERE
    (`d_mdta`.`Dataset_ID` = 1);
```


Output

```
1 • SELECT * FROM hyperparameter_db.dataset_basicdetails;
```

Name	Date_Created	Number_of_Columns	Number_of_Rows	Name_of_Variables	Datatype
Salary Information for Local Authorities	2018-12-11	20	347601	Authority Name	text
Salary Information for Local Authorities	2018-12-11	20	347601	Has Employees	text
Salary Information for Local Authorities	2018-12-11	20	347601	Last Name	text
Salary Information for Local Authorities	2018-12-11	20	347601	Middle Initial	text
Salary Information for Local Authorities	2018-12-11	20	347601	First Name	text
Salary Information for Local Authorities	2018-12-11	20	347601	Title	text
Salary Information for Local Authorities	2018-12-11	20	347601	Group	text
Salary Information for Local Authorities	2018-12-11	20	347601	Department	text
Salary Information for Local Authorities	2018-12-11	20	347601	Pay Type	text
Salary Information for Local Authorities	2018-12-11	20	347601	Exempt Indicator	text
Salary Information for Local Authorities	2018-12-11	20	347601	Base Annualized S...	number
Salary Information for Local Authorities	2018-12-11	20	347601	Actual Salary Paid	number

View 02: List no. of variables of each data-type present in a given dataset – Named “dataset_datatype_count”

Query to generate the view

```
CREATE
ALGORITHM = UNDEFINED
DEFINER = `root`@`localhost`
SQL SECURITY DEFINER
VIEW `hyperparameter_db`.`dataset_datatype_count` AS
SELECT
    `hyperparameter_db`.`dataset_variable_details`.`Datatype_of_Variable` AS
    `datatype_of_variable`,
    COUNT(`hyperparameter_db`.`dataset_variable_details`.`Datatype_of_Variable`) AS
    `datatype_count`
FROM
    `hyperparameter_db`.`dataset_variable_details`
WHERE
    (`hyperparameter_db`.`dataset_variable_details`.`Dataset_ID` = 1)
GROUP BY `hyperparameter_db`.`dataset_variable_details`.`Datatype_of_Variable`
ORDER BY `datatype_count` DESC;
```


Output

```
1 • SELECT * FROM hyperparameter_db.dataset_datatype_count;
```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
	datatype_of_variable	datatype_count		
▶	text	12		
	number	7		

View 03: Display the max. no. of models generated for a dataset in a particular run – Named “dataset_maxnumberofmodel_run”

Query to generate the view

CREATE

ALGORITHM = UNDEFINED

DEFINER = `root`@`localhost`

SQL SECURITY DEFINER

VIEW `hyperparameter_db`.`dataset_maxnumberofmodel_run` AS

SELECT

`ldr_mdata`.`Run_ID` AS `Run_ID`,

`ldr_mdata`.`Max_Models` AS `Max_Models`,

`dm`.`Dataset_ID` AS `Dataset_ID`

FROM

(`hyperparameter_db`.`data_map` `dm`

JOIN `hyperparameter_db`.`leaderboard_metadata` `ldr_mdata` ON

((`ldr_mdata`.`Run_ID` = `dm`.`Run_ID`)))

WHERE

((`dm`.`Dataset_ID` = 1)

AND (`ldr_mdata`.`Run_ID` = 'XBL7h1dt1'));

Output

```
1 • SELECT * FROM hyperparameter_db.dataset_maxnumberofmodel_run;
```

Result Grid			
Filter Rows:			
Export: Wrap Cell Content:			
	Run_ID	Max_Models	Dataset_ID
▶	XBL7h1dtl	9	1

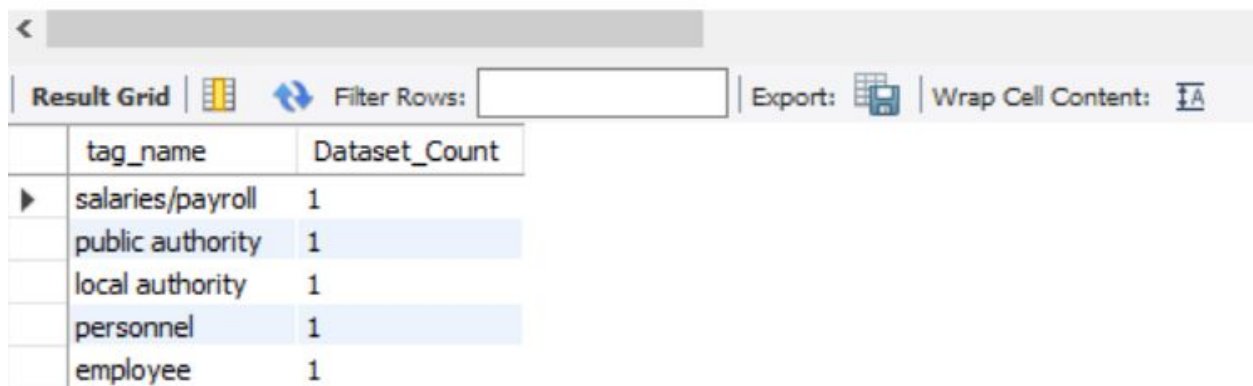
View 04: Display the No. of Datasets present in each domain – Named “dataset_presence_eachdoamin”

Query to generate the view

```
CREATE
  ALGORITHM = UNDEFINED
  DEFINER = `root`@`localhost`
  SQL SECURITY DEFINER
VIEW `hyperparameter_db`.`dataset_presence_eachdoamin` AS
  SELECT
    `hyperparameter_db`.`tags`.`Tag_Name` AS `tag_name`,
    COUNT(`hyperparameter_db`.`tag_map`.`Dataset_ID`) AS `Dataset_Count`
  FROM
    (`hyperparameter_db`.`tag_map`
  JOIN `hyperparameter_db`.`tags` ON
    ((`hyperparameter_db`.`tag_map`.`Tag_ID` = `hyperparameter_db`.`tags`.`Tag_ID`)))
  GROUP BY `hyperparameter_db`.`tag_map`.`Tag_ID`
  ORDER BY `Dataset_Count`;
```

Output

```
1 • SELECT * FROM hyperparameter_db.dataset_presence_eachdoamin;
```



	tag_name	Dataset_Count
▶	salaries/payroll	1
	public authority	1
	local authority	1
	personnel	1
	employee	1

Functions:

Function are defined for reusability, below we have shown few of the function queries and their output.

Function 01: Number of models in the database for which the values of all the performance metrics are present – Named “All_PerformanceMetric_Models”

Query to generate the function

```
CREATE DEFINER=`root`@`localhost` FUNCTION `All_PerformanceMetric_Models`() RETURNS int(11)
    DETERMINISTIC
BEGIN
    DECLARE Models_AllParam int;
    select count(Model_ID) into Models_AllParam
    from leaderboard
    where RMSE is NOT NULL and MSE IS NOT NULL and MAE IS NOT NULL and RMSLE IS NOT NULL;
    RETURN Models_AllParam;
END $$
```

Output

```
1 • select hyperparameter_db.All_PerformanceMetric_Models();
2 |
```

hyperparameter_db.All_PerformanceMetric_Model	
	4

Function 02: Return the no. of models run on Datasets lying in a particular data range – Named “ModelRun_Count_DataSizeRange”

Query to generate the function

```
CREATE DEFINER='root'@'localhost' FUNCTION `ModelRun_Count_DataSizeRange`() RETURNS int(11)
    DETERMINISTIC
BEGIN
    DECLARE Count_of_ModelID INT;
    select count(MR.Model_ID) into Count_of_ModelID
    from data_map as DM
    inner join model_run as MR on DM.Run_ID = MR.Run_ID
    inner join dataset_metadata as DT_MtData on DM.Dataset_ID = DT_MtData.Dataset_ID
    where DT_MtData.Data_Size between 59570000 and 59570300;
    RETURN Count_of_ModelID;
END$$
```

Output

```
1 • select hyperparameter_db.ModelRun_Count_DataSizeRange();
2 |
```

hyperparameter_db.ModelRun_Count_DataSizeRa	
	4

Function 03: Return a dataset name from a domain having maximum data size

Query to generate the function

```
CREATE DEFINER='root'@'localhost' FUNCTION `Dataset_Name_MaxDataSize`() RETURNS varchar(50)
    DETERMINISTIC
BEGIN
    DECLARE name varchar(50);
    select Dt_MData.Name into name
    from ( tag_map as TM
    inner join dataset_metadata as Dt_MData on Dt_MData.Dataset_ID = TM.Dataset_ID
    inner join Tags on TM.Tag_ID = Tags.Tag_ID)
    where Tags.Tag_Name = "salaries/payroll"
    having max(Dt_MData.Data_Size);
    RETURN name;
END$$
```

Output

The screenshot shows a SQL query execution interface. The query entered is: `select hyperparameter_db.Dataset_Name_MaxDataSize();`. The interface includes a toolbar with options like 'Result Grid', 'Filter Rows', 'Export', and 'Wrap Cell Content'. Below the toolbar, the output is displayed in a table with one row containing the text 'hyperparameter_db.Dataset_Name_MaxDataSize()'. A dropdown menu is open, showing 'Salary Information for Local Authorities' as a suggestion.

hyperparameter_db.Dataset_Name_MaxDataSize()
--

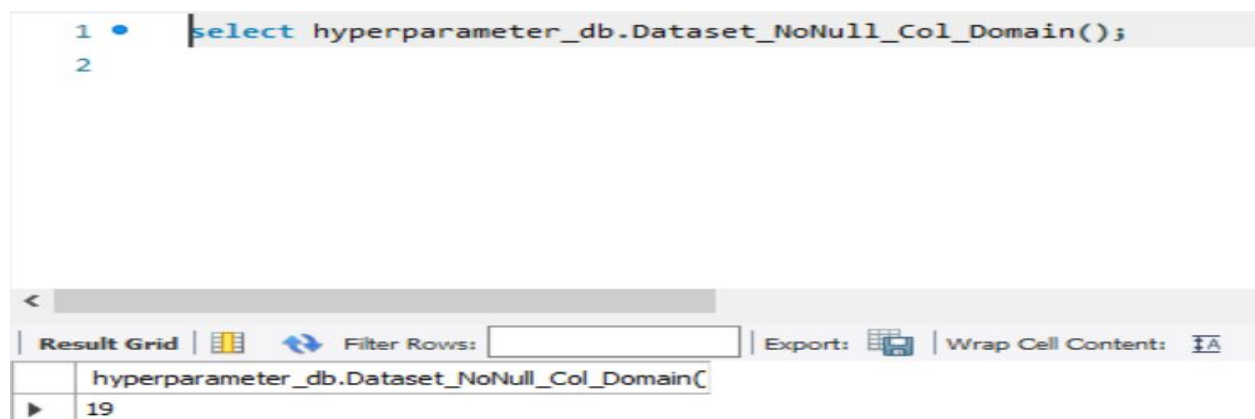
Salary Information for Local Authorities

Function 04: Return the count of datasets with no null values in each column for a particular domain – Named “Dataset_NoNull_Col_Domain”

Query to generate the function

```
CREATE DEFINER=`root`@`localhost` FUNCTION `Dataset_NoNull_Col_Domain`() RETURNS int(11)
    DETERMINISTIC
BEGIN
    DECLARE Count_of_Dataset_NoNull INT;
    select count(Dt_MData.Dataset_ID) into Count_of_Dataset_NoNull
    from ( tag_map as TM
    inner join dataset_metadata as Dt_MData on Dt_MData.Dataset_ID = TM.Dataset_ID
    inner join dataset_variable_details as Dt_VarDet on Dt_VarDet.Dataset_ID = Dt_MData.Dataset_ID
    inner join Tags on TM.Tag_ID = Tags.Tag_ID)
    where Tags.Tag_Name = "salaries/payroll" and Dt_VarDet.Null_Values_in_each_column is NOT NULL;
    RETURN Count_of_Dataset_NoNull;
END$$
```

Output



The screenshot shows a SQL query execution interface. At the top, a query is entered: `select hyperparameter_db.Dataset_NoNull_Col_Domain();`. Below the query, the results are displayed in a table. The table has one column and one row, showing the value 19.

	hyperparameter_db.Dataset_NoNull_Col_Domain()
19	


Stored Procedures:


Store procedures have been used in creating not only reusable pre-compiled code but to provide useful external-user interface to the database. These procedures are utility-oriented that help external users (like Python programmers) to add data to the database without writing long and complicated queries. These also make sure that the data entry happens in sequence without violating any relational constraints.






Below are the examples of 3 of the 15 utility procedures that have been created for the DB.

Procedure 1: tag_dataset() - Procedure to tag a modeling dataset.

```
1 • call hyperparameter_db.tag_dataset(1, 'healthcare');
2
```

100%  53:1

Action Output 

		Time	Action	Response	Duration / Fetch Time
	266	17:00:00	SELECT * FROM hyperparameter_db.Hyperparameter LIMIT 0,...	13 row(s) returned	0.125 sec / 0.0012 sec
	267	17:00:09	SELECT * FROM hyperparameter_db.Leaderboard LIMIT 0, 50...	6 row(s) returned	0.0029 sec / 0.00001...
	268	18:55:50	call hyperparameter_db.tag_dataset(2, 'healthcare')	Error Code: 1452. C...	0.069 sec
	269	18:56:07	SELECT * FROM hyperparameter_db.Dataset_Metadata LIMIT...	1 row(s) returned	0.0031 sec / 0.00001...
	270	18:56:28	call hyperparameter_db.tag_dataset(1, 'healthcare')	1 row(s) affected	0.0024 sec

Explanation: Running this procedure with a dataset ID and the tag one wants to add to it, the procedure does a tag search in the Tags table. If the given tag is found, it retrieves the tag ID and appends a tag ID to dataset ID mapping row in the Tag Map table. Else, if the tag is not found, it is added to the Tag table with a new and unique tag ID and then mapped in the Tag Map table.

Procedure 2: put_model_on_leaderboard() - Procedure to add a specific model related information into the DB.

```

1 • set @model_id = 0;
2 • call hyperparameter_db.put_model_on_leaderboard('test123', 'new_GBM_test1', 1.2, 1.34, 0.98, 1.11, 'False', @model_id);
3 • select @model_id;
4

```

	Time	Action	Response	Duration / Fetch Time
272	18:59:29	SELECT * FROM hyperparameter_db.Tags LIMIT 0, 50000	8 row(s) returned	0.00026 sec / 0.000...
273	19:01:48	SELECT * FROM hyperparameter_db.Leaderboard_Metadata LIMIT 0, 50000	3 row(s) returned	0.0010 sec / 0.00000...
274	19:03:21	set @model_id = 0	0 row(s) affected	0.0021 sec
275	19:03:21	call hyperparameter_db.put_model_on_leaderboard('test123', 'new_GBM_test1', 1.2, 1.34, 0.98, 1.11, 'False', @model_id)	1 row(s) affected	0.0078 sec
276	19:03:21	select @model_id LIMIT 0, 50000	1 row(s) returned	0.00022 sec / 0.0000...

Explanation: This procedure adds a specific model details to the Leaderboard table and maps it to the corresponding run in the Leaderboard Metadata thus connecting every model to a run. The unique database ID for the model is returned in the output variable.

Procedure 3: store_hyperparameter() - Procedure to add a specific model related hyperparameter values into the DB.

```

1 • set @hyperparameter_ID = 0;
2 • call hyperparameter_db.store_hyperparameter(6, 'ntrees', '12', @hyperparameter_ID);
3 • select @hyperparameter_ID;
4

```

	Time	Action	Response	Duration / Fetch Time
275	19:03:21	call hyperparameter_db.put_model_on_leaderboard('test123', 'new_GBM_test1', 1.2, 1.34, 0.98, 1.11, 'False', @model_id)	1 row(s) affected	0.0078 sec
276	19:03:21	select @model_id LIMIT 0, 50000	1 row(s) returned	0.00022 sec / 0.0000...
277	19:05:53	set @hyperparameter_ID = 0	0 row(s) affected	0.00013 sec
278	19:05:53	call hyperparameter_db.store_hyperparameter(6, 'ntrees', '12', @hyperparameter_ID)	1 row(s) affected	0.0049 sec
279	19:05:53	select @hyperparameter_ID LIMIT 0, 50000	1 row(s) returned	0.00016 sec / 0.0000...

Explanation: This adds hyperparameter values to the database. Scans the DB for the hyperparameter name. If it exists, maps the value for the specific model and hyperparameter. Adds the hyperparameter name if it does not already exist in the table. Returns the unique hyperparameter ID for the given hyperparameter. This ID along with Model ID (as obtained above) helps retrieve the hyperparameter value for a model.

Statistics:

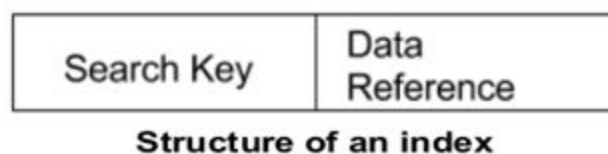
The database provides certain useful statistics and descriptions on the modeling dataset like unique values in each dataset column, count of null values, data types of the columns, dataset dimensions, etc. On the hyperparameter side, the best model RMSE/AUC (minimum RMSE / maximum AUC based on dataset complexity) could be easily found using functions for finding the best model. Hyperparameter ID based search helps find out the range in values for a particular hyperparameter across runs and models. RMSE/AUC value ranges could also be further constrained based on modeling algorithms (GBM, GLM, etc.) and checked.

Indexing:

Indexing is the way to get an unordered table into an order that will maximize the query's efficiency while searching. [3] It optimizes performance of a database by minimizing the number of disk accesses required when a query is processed.

Indexes are created using some database columns:

- The Search key that has a copy of the candidate key of the table. These values are stored in sorted order so that the corresponding data is quickly accessible
- The Data Reference contains a set of pointers that hold the address of the disk block where that particular key value can be found. [4]



There are two kinds of indices:

1. **Ordered indices:** Indices are based on a sorted ordering of the values.
2. **Hash indices:** Indices are based on the values being distributed uniformly across a range of buckets. The buckets to which a value is assigned is determined by function called a hash function. [4]

We have indexed the foreign keys present in the two junction tables to speed-up the query performance. As we had limited data present in the database, we were not able to observe significant change in the query run-time but the index will serve the purpose as the database grows in size.

```
/*
SQL Script to Create Indexes for the Normalised Hyperparameter Database
@Author - Mansi Nagraj (nagraj.m@husky.neu.edu)
@Created On - Apr 2019
*/
CREATE INDEX Index_ModelMap ON id_map(Model_Type_ID);
CREATE INDEX Index_Data_VarDetails ON Dataset_Variable_Details(Dataset_ID);
```

Conclusion:

The hyperparameter database focuses on three primary aspects to hyperparameter search. The dataset, the leaderboard, and the hyperparameter search. The dataset metadata helps easily identify the type, size, and use-case of the dataset through tags. The leaderboard specifies the details of the H2O runs helping connect dataset, run, and the run-specific models. Finally, the hyperparameter values and hyperparameter default values grouped by specific models and specific model algorithm respectively help easy search and analysis of hyperparameters. The search queries are further improved through the use of indexes.

References:

- [1]<https://towardsdatascience.com/understanding-hyperparameters-and-its-optimisation-techniques-f0debba07568>
- [2]<https://medium.com/@jrodthoughts/knowledge-tuning-hyperparameters-in-machine-learning-algorithms-part-i-67a31b1f7c88>
- [3] <https://chartio.com/learn/databases/how-does-indexing-work/>
- [4] <https://www.geeksforgeeks.org/indexing-in-databases-set-1/>
- [5]<https://www.kaggle.com/new-york-state/nys-salary-information-for-the-public-sector#salary-information-for-local-authorities.csv>