**FINDING THE BEST FIT HYPER PARAMENTS AND HOSTING THE DATABASE**

**Prof. Nik Bear Brown, Prabhu Subramanian, Jaynee Choksi**

*Master of Science in Information Systems*

*Northeastern University, Boston, MA*

**Abstract: Hypermeters are parameters whose value is set before the learning process begins. Different model training algorithms require different hyperparameters. Given these hyperparameters, the training algorithm learns the parameters from the data. We intend to analyze the best fit model by comparing the models after tunning the Hyperparameters for different algorithms on a dataset by running for different time periods. We will be using datasets from Kaggle/UCI for analysis. For calculating hyperparameters, H2O will be used. H2O gives us the calculation of the metrics and tuning the Hyperparameters. Finally, store the Hyperparameters with the models to the relational database and host it, for sharing with others. For different runtime (300sec, 500sec, 1000sec) of an AutoML Algorithm, the metrics vary with different accuracies. We are estimating Hyperparameters for different algorithms like – Common, Supervised, Aggregator, Miscellaneous. Using the analysis for the different time frame, the best-fit hyperparameters will be found and stored to the relational database against the Algorithm used.**

***Keywords: AutoML, H2O, Algorithms, Hyperparameters, Common, Supervised, Aggregator, Miscellaneous***

1. **INTRODUCTION**

Currently, there is no forum or websites that give a clear idea on which all hyperparameters to be considered while performing the model fit. The overall parameters can be divided into 3 categories:

1. **Tree-Specific Parameters:** These affect each individual tree in the model.
2. **Boosting Parameters:** These affect the boosting operation in the model.
3. **Miscellaneous Parameters:** Other parameters for overall functioning.

The parameters used for defining a tree are further explained below. The explanation below is considering one of the AutoML H2O algorithms: GBM as an example to explain the need of tuning Hyperparameters.

1. **min\_samples\_split:** Defines the minimum number of samples (or observations) which are required in a node to be considered for splitting. Used to control over-fitting. Higher values prevent a model from learning relations which might be highly specific to the particular sample selected for a tree. Too high values can lead to under-fitting hence, it should be tuned using CV.
2. **min\_samples\_leaf:** Defines the minimum samples (or observations) required in a terminal node or leaf. Used to control over-fitting similar to min\_samples\_split. Generally, lower values should be chosen for imbalanced class problems because the regions in which the minority class will be in majority will be very small.
3. **min\_weight\_fraction\_leaf:** Similar to min\_samples\_leaf but defined as a fraction of the total number of observations instead of an integer. Only one of #2 and #3 should be defined.
4. **max\_depth:** The maximum depth of a tree. Used to control over-fitting as higher depth will allow the model to learn relations very specific to a particular sample. Should be tuned using CV.
5. **max\_leaf\_nodes:** The maximum number of terminal nodes or leaves in a tree. Can be defined in place of max\_depth. Since binary trees are created, a depth of ‘n’ would produce a maximum of 2^n leaves. If this is defined, GBM will ignore max\_depth.
6. **max\_features:** The number of features to consider while searching for the best split. These will be randomly selected. As a thumb-rule, the square root of the total number of features works great but we should check upto 30-40% of the total number of features.
7. **learning\_rate:** This determines the impact of each tree on the final outcome (step 2.4). GBM works by starting with an initial estimate which is updated using the output of each tree. The learning parameter controls the magnitude of this change in the estimates. Lower values are generally preferred as they make the model robust to the specific characteristics of tree and thus allowing it to generalize well. Lower values would require higher number of trees to model all the relations and will be computationally expensive.
8. **n\_estimators:** The number of sequential trees to be modeled (step 2) Though GBM is fairly robust at higher number of trees but it can still overfit at a point. Hence, this should be tuned using CV for a particular learning rate.
9. **Subsample:** The fraction of observations to be selected for each tree. Selection is done by random sampling. Values slightly less than 1 make the model robust by reducing the variance. Typical values ~0.8 generally work fine but can be fine-tuned further.

Apart from these, there are certain miscellaneous parameters which affect overall functionality:

1. **Loss:** It refers to the loss function to be minimized in each split. It can have various values for classification and regression case. Generally the default values work fine. Other values should be chosen only if you understand their impact on the model.
2. **Init:** This affects initialization of the output. This can be used if we have made another model whose outcome is to be used as the initial estimates for GBM.
3. **random\_state:** The random number seed so that same random numbers are generated every time. This is important for parameter tuning. If we don’t fix the random number, then we’ll have different outcomes for subsequent runs on the same parameters and it becomes difficult to compare models. It can potentially result in overfitting to a particular random sample selected. We can try running models for different random samples, which is computationally expensive and generally not used.
4. **Verbose:** The type of output to be printed when the model fits. The different values can be:

0: no output generated (default)

1: output generated for trees at certain intervals

>1: output generated for all trees

1. **warm\_start**: This parameter has an interesting application and can help a lot if used judicially. Using this, we can fit additional trees on previous fits of a model. It can save a lot of time and you should explore this option for advanced applications
2. **presort:** Select whether to presort data for faster splits. It makes the selection automatically by default but it can be changed if needed.

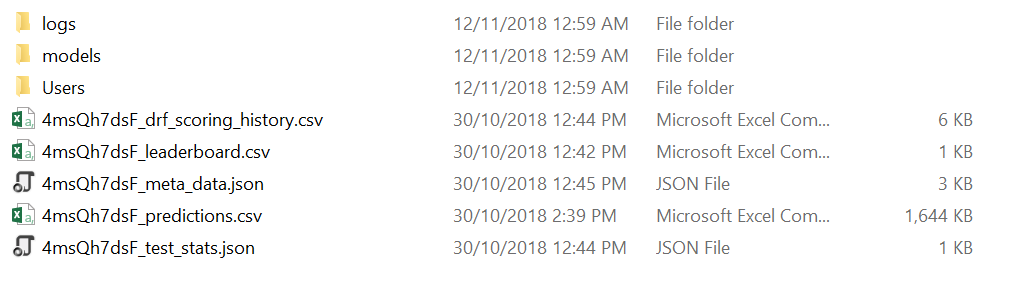
**Approach:**

Though GBM is robust enough to not overfit with increasing trees, a high number for particular learning rate can lead to overfitting. But as we reduce the learning rate and increase trees, the computation becomes expensive and would take a long time to run on standard personal computers.

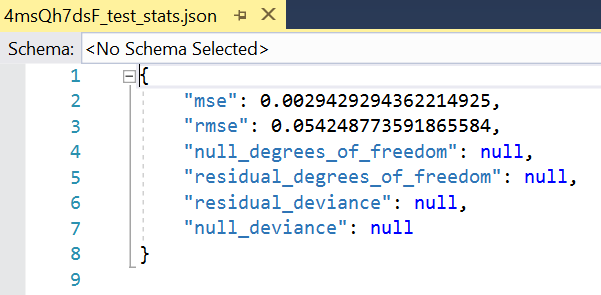
Keeping all this in mind, we can take the following approach:

1. Choose a relatively **high learning rate**. Generally, the default value of 0.1 works but somewhere between 0.05 to 0.2 should work for different problems
2. Determine the **optimum number of trees for this learning rate**. This should range around 40-70. Remember to choose a value on which your system can work fairly fast. This is because it will be used for testing various scenarios and determining the tree parameters.
3. **Tune tree-specific parameters** for decided learning rate and a number of trees. Note that we can choose different parameters to define a tree and I’ll take up an example here.
4. **Lower the learning rate** and increase the estimators proportionally to get more robust models.

Finally, once we achieve the best fit model with the metrics, we store the values of these hyperparameters to a JSON file that is generated and the best model fit and maintain a data as shown.



The JSON files consist of the best values of the metrics, which will be used further to host the database as shown below.



Addition to the metrics we will be storing the model that we generated with the best fit. This model will be hosted with the details of the best-fit hyperparameters with the database and will allow the users to analyze using the values of the Hyperparameter that was derived and the model that can be used for their own evaluation for datasets.

