**Lightwood Model Metrics**

Lightwood is the default AI engine used in MindsDB. It deals mainly with classification, regression, and time-series problems in machine learning.

1. **Accuracy metrics**

Lightwood provides ways to score the accuracy of the model using one of the accuracy functions. The accuracy functions include **mean\_absolute\_error, mean\_squared\_error, precision\_score, recall\_score, and f1\_score**.

| CREATE MODEL model\_name FROM data\_source  (SELECT \* FROM table\_name) PREDICT target\_column USING  accuracy\_functions="['accuracy\_function']"; |
| --- |

You can define the accuracy function of choice in the USING clause of the CREATE MODEL statement.

1. **Encoders**

It grants access to configure how each column is encoded. By default, the AutoML engine tries to get the best match for the data.

| USING encoders.column\_name.module = 'value'; |
| --- |

1. **Model arguments**

It allows you to specify the type of machine learning algorithm to learn from the encoder data.

| USING model.args = {"key": value}; |
| --- |

The models options are:

* **BaseMixer**: The foundation for all mixer models in MindsDB, this class defines the basic structure and interfaces that other models inherit. By providing common methods and properties, it allows other models to share standard functionality while adding their unique configurations.
* **LightGBM**: A fast, gradient-boosting decision tree model designed by Microsoft, LightGBM works efficiently on large datasets and is well-suited for tasks with high-dimensional data. It’s popular for classification (like determining labels) and regression (predicting continuous values) because it handles complex interactions and performs well with minimal parameter tuning.
* **LightGBMArray**: A specialized version of LightGBM for time series data, LightGBMArray runs several LightGBM models in parallel. Each instance of the model targets a specific point in the future, so it’s particularly effective for multi-step forecasts where you predict a sequence of future time steps, such as forecasting daily sales for the next week.
* **NHitsMixer**: This model uses the N-HiTS (Neural Hierarchical Interpolation for Time Series) architecture, which is a deep learning approach tailored to capture intricate temporal patterns. It’s particularly useful when handling seasonal data or irregular time series, where traditional methods struggle to identify complex, non-linear relationships.
* **Neural**: This is a fully connected (dense) neural network that transforms and combines feature information to produce predictions. By learning how different features interact through layers of neurons, it’s well-suited to datasets where relationships between features are non-linear or require a high level of feature interaction to predict accurately.
* **NeuralTs**: Based on the Neural model but optimized for time series data, NeuralTs makes predictions by taking past data points as inputs to predict future values. This model is ideal for sequential or temporal datasets where understanding the order of events is critical, like stock price movements or weather predictions.
* **ProphetMixer**: This model wraps around Facebook’s Prophet, a time series model that’s designed for easy forecasting with daily, weekly, or yearly seasonality. It’s especially useful for time series with strong, periodic patterns, making it well-suited to tasks like forecasting sales, energy consumption, or traffic where these seasonal effects are prominent.
* **RandomForest**: An ensemble learning method that builds multiple decision trees and aggregates their predictions. In MindsDB, this mixer can handle both classification and regression by creating “forests” of decision trees, leading to robust predictions that reduce overfitting and provide better generalization on new data.
* **Regression**: This model uses Ridge regression, a form of linear regression with regularization (penalizing large coefficients), which helps prevent overfitting. Ridge regression is useful for problems where the relationship between features and the target is mostly linear and where you want to avoid overly complex models.
* **SkTime**: A wrapper around the SkTime library, which includes various time series forecasting algorithms. SkTime provides a unified interface for different forecasting methods, making it adaptable to many forecasting tasks, such as those involving hierarchical or multivariate time series.
* **Unit**: This model acts as a simple pass-through, relying on the target encoder’s raw predictions. It’s commonly used in cases where advanced encodings, like those from language models for text, need minimal adjustments. The Unit mixer is suitable for straightforward predictive tasks where adding a complex model may not add much value.
* **XGBoostMixer**: A highly versatile, tree-based model using the XGBoost algorithm. Known for its performance on structured data and its ability to handle non-linear relationships, XGBoost uses gradient boosting to improve predictive accuracy. It works well across both classification and regression tasks and is valued for its balance between interpretability, speed, and accuracy.

For more information see:

[Mindsdb Lightwood documentation](https://docs.mindsdb.com/integrations/ai-engines/lightwood#lightwood)

[Mixers](https://mindsdb.github.io/lightwood/mixer.html#)

[All modules for which code is available](https://mindsdb.github.io/lightwood/_modules/index.html)

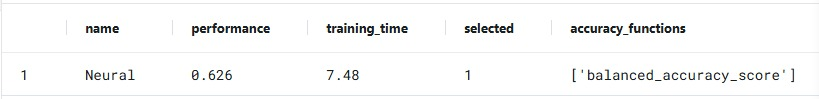
**Example - Changing model to Neural**

For our Use Case, we tried to change the algorithm manually. We used a neural network following the github module instructions: [Source code for neural module (lightwood.mixer.neural)](https://mindsdb.github.io/lightwood/_modules/lightwood/mixer/neural.html#Neural)

The instructions were the following:

| CREATE MODEL mindsdb.risk\_factors FROM mindsdb  (SELECT \* FROM risk\_factors\_view)  PREDICT Risk\_Factors USING   model.args = {"submodels": [{"module": "Neural","args": {"n\_epochs": 50}}]}; |
| --- |

And now we are able to see that the accuracy changed following this new algorithm:



It is also possible to only add the "module", and not add any "args". The model will use the ones by default.

If we select "search\_hyperparameters": True, the model will do an automatic search to find the best hyperparameters during the training process.