OttoML beta API Documentation

Seth Leonard
seth@ottoquant.com

The OttoML API gives programmatic access to OttoQuant's in house machine learning tools. Software to access the API via R or Python is available via GitHub.

Address: 142.93.85.22

Docs: 142.93.85.22/__docs__

1 Overview

OttoQuant's machine learning algorithm is designed to look for relationships or patterns in data that allow us to predict a target series using a set of predictors or right hand side (RHS) variables. We can state this relationship as

$$p(x) = f(y_1, y_2, y_3, \ldots)$$

where $p(x)$ represents the probability of observing some value of x based on our observations $y_1, y_2, y_3, \ldots$.[1] The function $f(\cdot)$ represents our machine learning algorithm. The basic API call returns three objects. `x_point` is the expected value

[1] We use x to describe our target variable and y to describe our observations due to our background in state space modeling; this may cause some initial confusion as it reverses the usual linear model usage of $y_i = \beta x_i + \varepsilon_i$

of x defined as

$$E(x) = \int xp(x)dx$$

The output `x_dist` is the empirical distribution of x, representing the probability of each value in `x_grid`.

Using these algorithms requires two fundamental objects: a set of `training_data` used as a model, and one or more `observations`, that is, right hand side variables used to predict our target variable. These data must meet certain criteria. First, `training_data` must be a data frame like object with each series represented by a column and observations in rows. Moreover, our target series, the series we would like to predict, must be ordered first, that is, in the first column. The order of all other predictors, or RHS variables, does not matter so long as it is consistent with their order in `observations`. This means that `training_data` will have one more column than `observations` Second, all data must be stationary. This means that the data scientist will have to take logs, seasonally adjust, take differences, or preform any other necessary data transformations prior to making an API call. Missing values are accepted in both `training_data` and `observations` as are mixed frequency data, as described below.

1.1 Inputs

training_data A $T \times (k+1)$ data frame like object of training data used to construct a model with the target series in the first column where T is the number of observations and k the number of RHS variables (predictors). The +1 is due to the inclusion of our target series in the first column.

observations A $N \times k$ data frame like object of observations from which we would like to generate predictions of the target series where N is the number of observations we have. For backtesting or bootstraping, N will typically be a single observation.

1.2 Advanced Inputs

MF_weights A $N \times k$ data frame like object of mixed frequency weights. These weights are for situations in which we have partial observations. For example, one of the series in `observations` may be weekly, but if it is Tuesday we will only have two observations for this week. We can still use the average values of these

two observations by including the mixed frequency weight 7/2 in position i, j of the object `MF_weights`. The weight 7/2 indicates to the algorithm that of seven possible observations we have only observed 2. With complete observations, all `MF_weights` will be 1 (the default). In the background, our algorithms will use the entry of 7/2 to scale up the variance of this observation. The underlying assumption is that partial observations will have the same mean but a higher variance than complete observations. For this "same mean" assumption to be correct in this example daily data must be adjusted for weekday effects.

sd A vector of $k + 1$ values corresponding to each column in `training_data` indicating how closely patterns in the data must match to be used for predictions. This corresponds to bandwidth in the technical jargon. A higher value indicates patterns may match loosely, a lower value will require more exact patterns in the data. Thus higher values are typically appropriate to `training_data` with fewer observations.

gp The number of gridpoints to use in the empirical distribution `x_dist`. The default is 11.

shape By default (corresponding to `shape` = 0) our algorithms will try to improve efficiency by using the linear covariance matrix of RHS data in `training_data`. For highly non-linear situations this may be inappropriate. Users can limit the degree to which algorithms use observed covariances by increasing `shape` (for large values covariance will have zero weight).

2 Functions (Endpoints)

2.1 OttoML

`OttoML` is the basic call invoking our ML algorithm and accepts all the above inputs and returning:

x_point Point estimates (the expected value) of the target variable for each of the N observations.

x_dist A $N \times gp$ data frame containing the distribution of the target variable for each row in `observations`. Element i, j of `x_dist` gives the probability of observing the j^{th} value of `x_grid` corresponding to the i^{th} row of `observations`.

x_grid Values corresponding to each column of `x_dist`.

2.2 OttoBoot

`OttoBoot` bootstraps the training data via a standard leave-one-out routine to give a feel for out of sample model performance. It does not accept inputs `observations` or `MF_weights`.

x_point Point estimates (the expected value) of the target variable for each of the N observations.

x_dist A $N \times gp$ data frame containing the distribution of the target variable for each row in `observations`. Element i, j of `x_dist` gives the probability of observing the j^{th} value of `x_grid` corresponding to the i^{th} row of `observations`.

x_grid Values corresponding to each column of `x_dist`.

3 Formatting

Whenever possible we recommend using the R or python code referenced above to access the API. For other methods, please note that .json formatting follows the R conventions of `toJSON()` for data frames such that, for example, `training_data` should look as follows:

```
"training_data": {"V1":[349,298,99,102,209,14,101],
"V2":[83,99,93,87,87,85,81],
"V3":[65,70,69,82,71,77,76]},
```

though column names are not used. Thus a complete call might be:

```
{"keyval":["Abc123"],
"training_data":{"V1":[0.8862,-0.0853,-0.9249,0.8953,0.8229],
"V2":[-0.5563,1.6235,1.2478,0.0271,-0.1052],
"V3":[1.9243,-0.2834,0.2006,-0.2547,2.1879]},
"observations":{"V2":[-0.038,0.8277,-1.2907],"V3":[0.6829,-0.4244,-0.4024]},
"MF_weights":["none"],"sd":[1],"gp":[11],"shape":[0]}
```