
# Rinse Over Run

Final report - Guillaume Thomas - 2019-03-18

# Problem understanding

- Physic understanding
  - Turbidity sensors measure how dirty a flow of water is
  - Turbidity multiplied by flow during the final rinsing indicates the residual “dirtiness” of an equipment before the final rising
  - There’s a clear relationship between this volume of turbidity and the required amount of cleaning water to clean it
  - Predict the turbidity volume ~ Decide the required volume of cleaning water
  - → **Final goal is “Save useless rinsing water”**
- Business perspectives (with figures taken from my field knowledge)
  - Consumption: 5 $m^3$/cip
  - Estimated Cost: 10 €/$m^3$
  - # CIP/day: 86
  - Hypothesis of savings due to prediction: 10%
  - **Estimated savings: 150k€/year/factory → it is worth the try!**

- Bibliography
  - [1] states that turbidity is explained by:
    - Titer
    - Turbulence
    - Temperature
    - Time

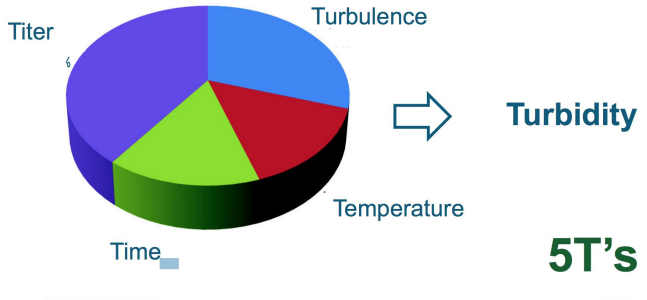



*Fig 1: Influence of physical values on turbidity*

- Terminology
  - MA(P)E: Performance metric defined as $\frac{1}{N}\sum_{i=1}^{N}\frac{|\hat{y}_i - y_i|}{max(|y_i|, 290000)}$
    - Parenthesis expresses that it’s a modified version of MAPE which uses percentage error only for high values.

# Data preparation

- **Goal: adapt data to a prediction at the end of each phase with appropriate weights**
- Several phases recipes in the dataset
- I used an **integer encoding**
  - 12345: All the phases are performed
  - 125: Only pre_rinse & caustic & final_rinse
  - Encoding of input and observed recipes
- Subprocess generation
  - I generated one subprocess for each performed phase with data up until this phase
  - I associate weights according to test data distribution (See Fig 2)
  - Ex: process_id 27001 generates 4 subprocess:
    - 27001-1 (only pre_rinse)
    - 27001-12 (pre_rinse + caustic)
    - 27001-123 (pre_rinse + caustic + intermediate)
    - 27001-1234 (all)
  - **From ~5k process, we generate ~17k subprocess**

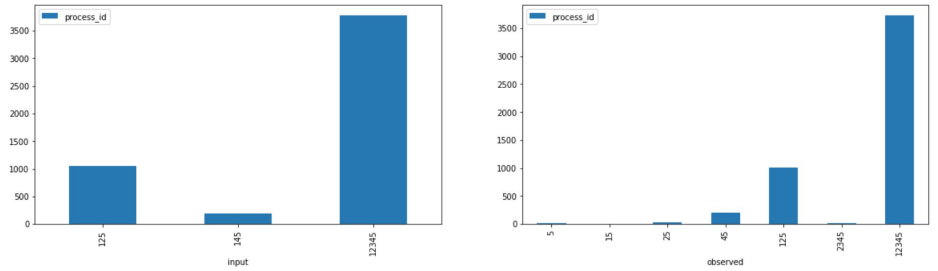



*Fig 1: Recipes distributions*

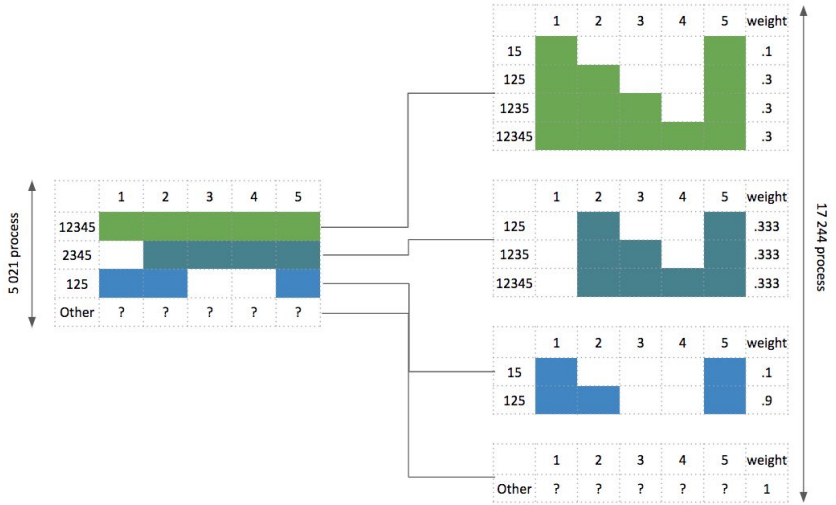



*Fig 2: Subprocess generation scheme*

# Feature engineering

- Based on the input columns type, I computed several features
  - Categorical
    - pipeline
    - input_recipe: integer which represents the cleaning recipe (see slide 3)
    - observed_recipe: integer which represents the observed recipe as it might be different from the cleaning recipe
    - object_id
    - **No encoding since tested algorithms support categorical features**
  - Numericals: For each measure, I compute several **time-domain** aggregations
    - Basic: *min*, *max*, *avg*, *std*, *sum*
    - Percentiles: *percentile_XX* (with XX in (10..90))
    - Derivative: *diff_avg* (the mean over the differences between subsequent time series values), *diff_abs_avg* (the mean over the absolute differences between subsequent time series values)
    - **Frequency-domain aggregations have been tested without interesting results**
  - Booleans. other aggregations:
    - Basic: *mean*, *sum*
    - *changes_count* (count of value change over subsequence time series values)
  - This produces a 17244x271 matrix (Table 1 shows the 5 first lines)

| index | object_id | pipeline | input_recipe | observed_recipe | ... | supply_flow\|sum | supply_flow\|mean | supply_flow\|max | supply_flow\|diff_avg |
|---|---|---|---|---|---|---|---|---|---|
| **20001-1** | 405 | 4 | 12345 | 1 | ... | 1.224125e+07 | 46193.399008 | 55674.910 | 177.611606 |
| **20001-12** | 405 | 4 | 12345 | 12 | ... | 4.896349e+07 | 49308.647817 | 56886.570 | 45.951376 |
| **20001-123** | 405 | 4 | 12345 | 123 | ... | 5.136290e+07 | 48547.169139 | 58145.258 | 45.360073 |
| **20001-1234** | 405 | 4 | 12345 | 1234 | ... | 7.053979e+07 | 49501.604051 | 59396.703 | 11.091989 |
| **20002-1** | 301 | 3 | 125 | 1 | ... | 5.572700e+05 | 22290.798688 | 31933.594 | 896.538558 |

*Table 1: 5 first lines of final features matrix*

# Model & parameters selection

- Models tested
  - Only ensemble methods tried: random forest, xgboost & lightgbm
  - **Focus on lightgbm** which showed quickly better results
- Using an 5-fold adapted cross validation (named later: 5-fold aCV)
  - Since there are redundancy between rows associated to the same parent process, I made sure that all the subprocess (ex: 27001-1 & 27001-12 & 27001-123 & 27001-1234) of one process belong either to train or test data.
- For the parameters optimization, I've used the following method
  - I fixed **booster=dart** & **objective=MAE** since they quickly showed substantially better results.
  - I used lightgbm native support for categorical features (eg: *pipeline*, *input_recipe*, *observed_recipe* and *object_id*).
  - I started with a *learning_rate=0.1* and repeated manually the following steps
    - Use bayesian optimization [1] for
      - lightgbm parameters (*num_leaves*, *max_depth*, *num_leaves*, *lambda_l1*, *cat_smooth*)
      - custom parameters (*thr1, thr2, tau1, tau2)* (See slide 6 for their meaning)
      - Usage or not of a feature
      - Fig 1 shows parameters usage histograms from which I deduced optimal parameters (ex: thr1=4e5 seems to be optimal for a 0.04 learning_rate)
    - Decrease learning_rate, narrow hyper parameters domains, remove useless features & start over

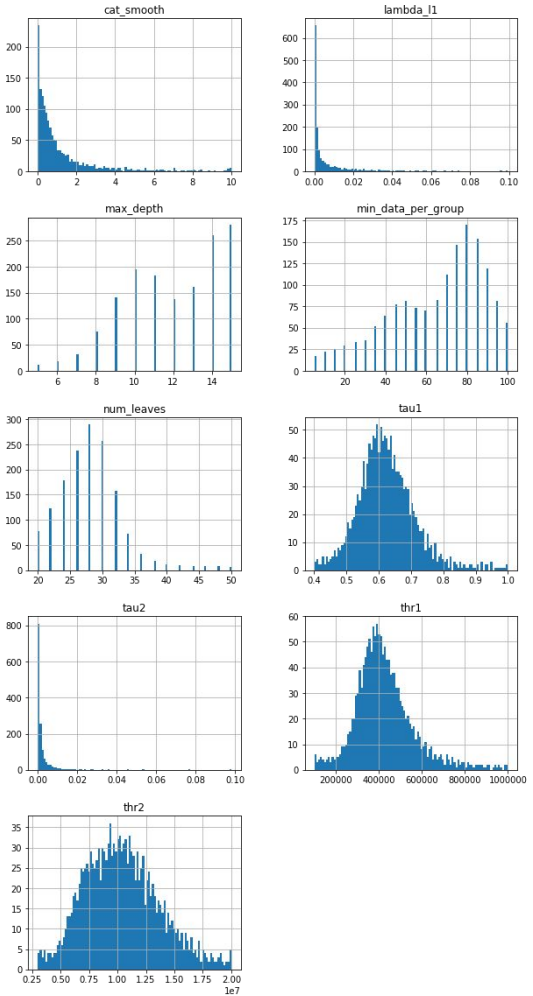
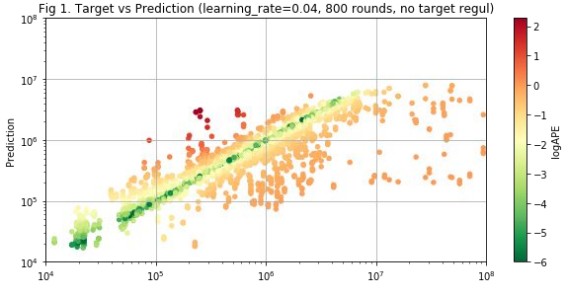


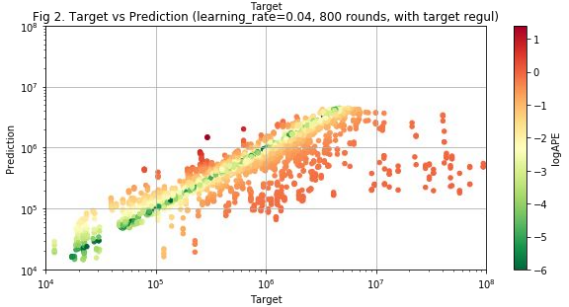

*Fig 1: Parameters usage histogram for learning_rate=0.04 with bayesian optimization*

[1] BERGSTRA, James S., BARDENET, Rémi, BENGIO, Yoshua, *et al.* Algorithms for hyper-parameter optimization. In : *Advances in neural information processing systems*. 2011. p. 2546-2554.

# Target reweighting

- Objective function for lightgbm was a crucial choice
  - MA(P)E is a mix of MAE for low values (~24%) and MAPE.
  - MAPE objective is known to suffer several issues [1]
  - All others parameters being equals, MAE shows significant better results.
- MAE major issue is that it overweights errors on high target values
  - ⟶ Positive bias (ie prediction > target) for small target values
  - ⟶ Negative bias for high target values
  - Fig 1 illustrates this by showing prediction vs target colored by the log of APE
- To overcome this, I added a “target reweighting”:
  - If thr1 <= target < thr2, then weight := weight * tau1
  - If thr2 <= target, then weight := weight * tau2
  - Fig 2 shows that with this reweighting, prediction are “shifted on the right”.
  - Table 3 shows that MA(P)E is significantly improved with this reweighting
  - APE is also impacted
    - **Without, the overall estimation has a global negative bias of ~4%**
    - **With reweighting, this negative bias increases to ~13%**




| Target regularization | MPE | MA(P)E |
|---|---|---|
| No | -0.0452 | 0.2881 |
| Yes | -0.1249 | 0.2637 |

*Table 3: Results with or without target regularization using 5-fold aCV*

- **This important bias is a side effect of the inadequacy between algorithm inner objective and global performance metric. If this bias is an issue, two options: challenge the performance metric or build a custom objective function adapted to this performance metric.**

[1] Makridakis, Spyros. "Accuracy measures: theoretical and practical concerns." International Journal of Forecasting 9.4 (1993): 527-529

# Results

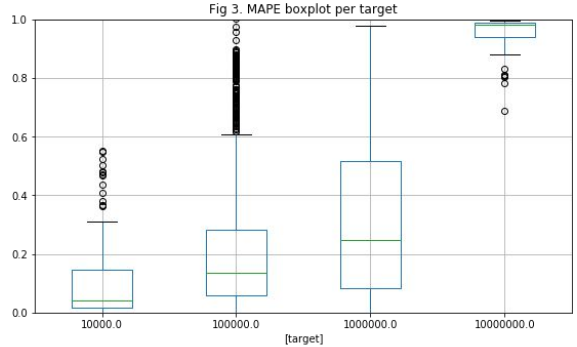
| Train MA(P)E | Test MA(P)E | Private LB MA(P)E |
|---|---|---|
| 0.2233 | 0.2665 | 0.2725 |

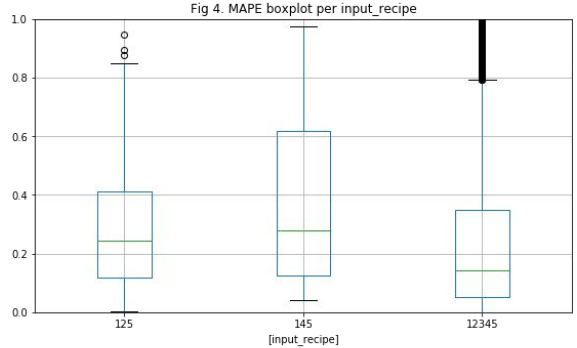
*Table 1: Training results (using 5-fold aCV) associated to the best private leaderboard results with thr1 = 1e6, thr2 = 8e6, tau1 = 0.5, tau2 = 1e-5*

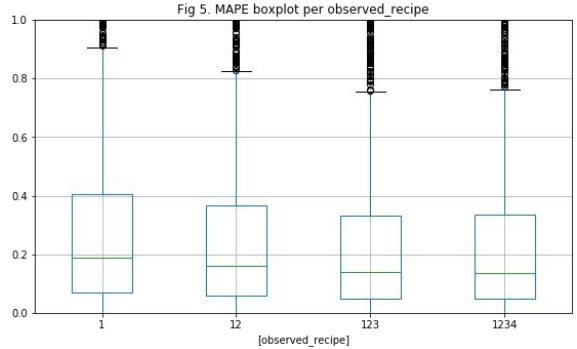
- Table 1 shows training results corresponding to my best private leaderboard score (**0.2725**).
- Table 2 shows statistics of APE distribution
  - APE deviation is important (~0.29) and the distribution tail shows important errors. For instance, for 5% of the process, the prediction has an error greater than 0.87.
  - **Given prediction precision & confidence, it's hard to consider its integration in a industrial process without strong safeguards.**
- What does this distribution depend on?
  - Fig 3. shows that **the greater the target value, the worse the prediction**, both in term of absolute value and in term of confidence interval.
  - **Prediction is much better for a full recipe** (eg all the phases). (See Fig 4.)
  - Fig 5. shows boxplots for the MA(P)E per observed recipe. We can observe a trend which means that **the more data we have, the better the prediction**. Ending at first phase has **0.2821** MA(P)E where ending after acid phase has **0.2488** MA(P)E. One can see that the impact of the prediction moment is relatively limited compared to other feature such as the object_id or the MA(P)E itself.

| | MA(P)E |
|---|---|
| **mean** | 0.2638 |
| **std** | 0.2916 |
| **min** | 0.0001 |
| **25%** | 0.0572 |
| **50%** | 0.1586 |
| **75%** | 0.3713 |
| **90%** | 0.7282 |
| **95%** | 0.8764 |
| **99%** | 0.9933 |
| **99.9%** | 2.0529 |
| **max** | 4.0580 |

*Table 2: MAPE statistics (using 5-fold aCV)*



# Feature selection: Which are the important features?

| feature | decrease |
|---|---|
| object_id | 0.8131 |
| return_turbidity | 0.0824 |
| supply_flow | 0.0452 |
| input_recipe | 0.0325 |
| supply_pre_rinse | 0.0252 |
| supply_pressure | 0.0159 |
| tank_level_pre_rinse | 0.0100 |
| return_flow | 0.0088 |
| supply_caustic | 0.0075 |
| return_drain | 0.0052 |
| tank_temperature_pre_rinse | 0.0047 |
| return_conductivity | 0.0041 |
| return_temperature | 0.0025 |
| tank_level_clean_water | 0.0024 |
| tank_level_caustic | 0.0023 |
| tank_level_acid | 0.0021 |
| tank_temperature_acid | 0.0016 |
| tank_concentration_caustic | 0.0015 |
| supply_pump | 0.0013 |
| tank_concentration_acid | 0.0012 |
| tank_temperature_caustic | 0.0012 |
| pipeline | 0.0003 |
| return_caustic | 0.0002 |
| supply_clean_water | 0.0001 |
| return_acid | 0.0000 |
| supply_acid | 0.0000 |
| return_recovery_water | 0.0000 |
| observed_recipe | 0.0000 |

*Table 1: Feature permutation importance*

| Used features | MA(P)E |
|---|---|
| none | 0.751 |
| input_recipe + object_id | 0.286 |
| input_recipe + object_id + return_turbidity + supply_flow + supply_pre_rinse + supply_pressure + tank_level_pre_rinse | 0.268 |

*Table 2: MA(P)E obtained by incrementally adding features*

- Feature importance computed using the **permutation importance** method [1]
  - For each measure, I shuffle all the aggregated features associated to this measure (X_min, X_max …) and compute the increase of the MA(P)E.
- *Table 1* shows the results ordered by decreasing importance
  - **object_id** is by far the most important feature
  - **input_recipe**, **return_turbidity**, **supply_flow, supply_pre_rinse, supply_pressure & tank_level_pre_rinse** have an intermediate importance (increase ≥ 1e-2)
  - 5T model states that turbulence, titer & temperature influence on turbidity is equivalent. Considering that flow & pressure estimate turbulence, conductivity estimates titer, **turbulence seems to have a much more important influence on turbidity than titer & temperature.**
  - **pre_rinse measures seems to be important which might indicate that what happens during this phase has an impact on final turbidity.**
  - **Curiously, observed_recipe is not used by the model even if MA(P)E strongly varies with it.**
- *Table 2* shows the MA(P)E obtained by incrementally adding features.
  - Using only input features (**input_recipe** & **object_id**) provides already a good MA(P)E which means that most of the signal is in those features.
  - Adding process measurements has a limited impact on MA(P)E.

[1] BREIMAN, Leo. Random forests. *Machine learning*, 2001, vol. 45, no 1, p. 5-32.

# Dependency on object_id & target

- Why's the prediction so dependant on object_id & target value?
  - Fig 1 shows a process where return_turbidity is very low (eg < 1 NTU) during target_time_period.
  - Table 2 shows statistics for mean return_turbidity per process. 75% of process have a mean return_turbidity < 0.67 NTU during target_time_period.
  - Since typical turbidity sensors [1] have an accuracy of 0.3 NTU, we can consider that for most process, measured turbidity is mostly noise. Considering this noise as constant in average, predict *sum(return_flow * return_turbidity)* is similar to predict *sum(return_flow)*
  - Fig 3 shows a scatter of target vs prediction of *sum(return_flow)* with a simple lightgbm regressor using only 3 features (pipeline, input_recipe & object_id). Visually, the prediction seems reasonably good.
  - **The intuition here is that the estimated volume of turbidity is in many process, a function of**
    - **A noise which is dependant on the sensor (thus the object_id)**
    - **The sum of return_flow which seems highly dependant on the recipe & the object**
  - **This would explain why the algorithm uses that much object_id and why precision is good on small target values (as predicting *sum(return_flow)* seems to give good results).**

*Fig 1: return turbidity, phase & target_time_period for process_id=23726*

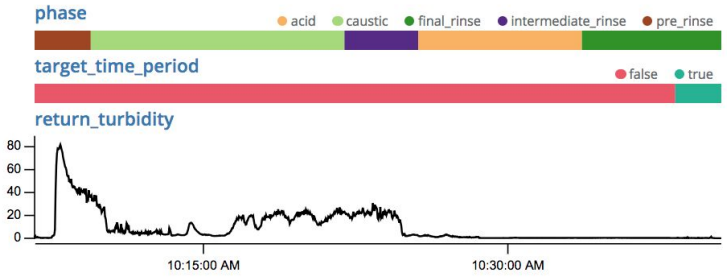
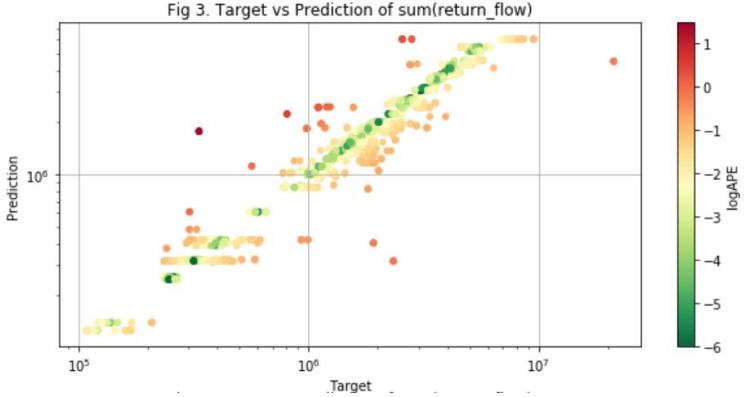



*Table 2: Statistics for mean return_turbidity per process*

| | mean_return_turbidity |
|---|---|
| **50%** | 0.310868 |
| **75%** | 0.674531 |
| **90%** | 1.254024 |
| **95%** | 2.239879 |
| **99%** | 15.277359 |
| **max** | 40.862282 |



# What about high target values?

- Let's train a regressor for process with high target
  - Filtered process with target >= 8e6. This value corresponds roughly to *thr2* (explained in slide 6). Fig 1 shows an example of those process.
  - I used the same methods as described previously without target reweighting.
  - Table 2 shows that MAPE objective give better results than MAE. This is due to the fact that inner objective function is the same as the evaluation function.
  - Nonetheless, Table 3 shows that obtained MAPE is much worse as the one obtained with all the process (~0.26) but better than the score obtained on the same high target process but with an regressor trained on all the data (~0.95).
  - Table 4 shows that important features are very different from slide 8. It's now measurements taken from the process. Pressure & turbidity are still important and conductivity is much more important.
  - **Conclusion: High and small target values are not explained by the same variables: while input features (object_id, input_recipe) are almost "enough" for small values, process measurements have much more importance for predicting high target values.**

*Fig 1: return turbidity, phase & target_time_period for process_id=25408*

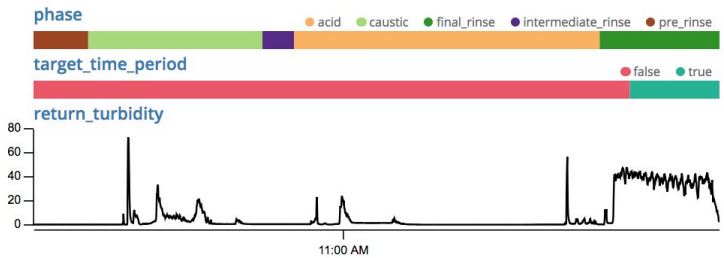



*Table 2: Training results (using 5-fold aCV)*

| Objective | Test MAPE |
|---|---|
| **MAE** | 0.7248 |
| **MAPE** | 0.4771 |

*Table 3: Training results with different supports*

| Training support | Evaluation support | MAPE |
|---|---|---|
| **All the process** | All the process | 0.2637 |
| **All the process** | target > 8e6 | 0.9567 |
| **target > 8e6** | target > 8e6 | 0.4771 |

*Table 4: Feature permutation importance*

| feature | increase |
|---|---|
| supply_pressure | 0.0681 |
| return_turbidity | 0.0421 |
| object_id | 0.0203 |
| return_conductivity | 0.0200 |
| tank_level_caustic | 0.0152 |
| tank_concentration_caustic | 0.0133 |
| tank_temperature_acid | 0.0129 |
| tank_temperature_pre_rinse | 0.0128 |
| tank_level_clean_water | 0.0118 |
| supply_flow | 0.0113 |
| return_flow | 0.0100 |
| return_temperature | 0.0090 |
| tank_level_pre_rinse | 0.0089 |
| tank_concentration_acid | 0.0082 |
| tank_temperature_caustic | 0.0081 |
| tank_level_acid | 0.0056 |
| input_recipe | 0.0026 |
| supply_pre_rinse | 0.0018 |
| return_drain | 0.0017 |
| supply_acid | 0.0016 |
| supply_pump | 0.0011 |
| return_acid | 0.0006 |
| return_caustic | 0.0003 |
| supply_clean_water | 0.0003 |
| supply_caustic | 0.0003 |
| return_recovery_water | 0.0000 |
| pipeline | 0.0000 |
| observed_recipe | 0.0000 |

# Conclusion

- What have we learnt?
  - Model analysis
    - Best found model gives a **0.2665** MA(P)E on train data and **0.2725** on private leaderboard using **lightgbm**.
    - High target values have been underweighted to improve MA(P)E. Ex: Target >= 8e6 have a weight almost equal to 0.
    - *object_id* is by far the most important feature. Using only input features (ie *object_id* & *input_recipe*) gives already a **0.286 MA(P)E**.
    - Most important measurements are return_turbidity, supply_flow, supply_pre_rinse & supply_pressure
  - Error & bias analysis
    - The model is negatively biased as it globally underestimates the target (**~0.12 MPE**). This bias is heavily dependant on the target values (high target → high negative bias, low target → low positive bias). This bias is due to the evaluation metric and accentuated by the target reweighting.
    - The prediction has an important dispersion: **deviation=~0.29** and **5% of the process with A(P)E > 0.87**
    - The impact of the step after which the prediction is made is limited (**0.2821** after pre_rinse vs **0.2488** after acid phase). This is due to the high importance of object_id.
  - Interpretation
    - The choice of the evaluation metric (MA(P)E) has an important impact on the detected signals. Since it gives the same weight to low and high targets, and that “there are more low target”, the best algorithm is likely to focus on what explain low target at the expense of what explain high target values. Thus:
      - I interpret low target values as mostly a measurement of the sensor noise (highly related to object_id) and return_flow (which seems related to object_id and input_recipe)
      - High target values seems much more explained by measurements taken during the process (especially turbidity, pressure & conductivity).
    - **This explains the *object_id* pre-eminence which is a distorsion of what really explains turbidity. Using another evaluation metric (like MAE) would provide a more balanced prediction where process measurements are more important.**
- Going further: how to improve?
  - Try deep learning & implement custom objective function for lightgbm adapted to MA(P)E