II. Code submission and result reproducibility

All our code can be found on GitHub: https://github.com/GillesVandewiele/Rinse-Over-Run

Moreover, we will try to transfer a zip file with all code and the output of the intermediate steps.

III. Model documentation and write-up

You can respond to these questions either in an e-mail or as an attached file (any common document format is acceptable such as plain text, PDF, DOCX, etc.) Please number your responses.

1. Who are you (mini-bio) and what do you do professionally?

Gillesvdw	Gilles Vandewiele is, just like all other team members, a computer science engineer. He's 25 years old and lives in Ghent (Belgium), but is originally from the much more beautiful Bruges, a few kilometers West. Gilles is currently working as a PhD Student for the Internet and Data Science Lab (IDLab) at the University of Ghent, where he is conducting research in the domain of (white-box) machine learning and Semantic Web technologies. Gilles is passionate about sport-related data science, video games (and writing AIs for them), Python, and loves playing football.
tfmortie	With master's degrees in Computer Science Engineering and Statistical Data Analysis, Thomas Mortier is currently working as a teaching assistant and PhD student for the research unit KERMIT (Knowledge-based Systems) at the Faculty of Bioscience Engineering, University of Ghent. His current research interests include Bayes-optimal multilabel and multi-class classification algorithms. If you can't find Thomas behind his desk, you'll probably find him cycling, playing football, cooking in the kitchen or spending time with his girlfriend or friends.

2. High level summary of your approach: what did you do and why?

The main property we wanted our approach to have, is that the foundation of our pipeline was as explainable as possible, which is, of course, beneficial for the report that needs to be written during the second phase. Therefore, we went for a classical, feature-based approach which was largely influenced by the DrivenData benchmark.

From each of the timeseries provided for each process, we extracted aggregates and descriptive statistics. These features were then fed to a plethora of different sci-kit learn regressors (roughly 25) to create out-of-sample predictions for training and test set. These out-of-sample predictions were then fed, along with the original features to three gradient boosting algorithms: CatBoost, eXtreme Gradient Boosting (XGB) and Light Gradient Boosting Machines (LGBM). Finally, the average from the predictions generated by the gradient boosters is calculated and used as submission. The final percentage was gained by taking the average of previous submissions.

In total, 11 different models were trained using this pipeline: one for each unique (recipe, present phases)-combination of the processes in the test set. If we assign a bit to each phase, we can represent both the recipe and the present phases of a process in the data as a bitstring of length 4 or a number between 0 and 15 (e.g. the recipe with all phases gets bitstring 1111 or 15; if we have all phases except for the "acid" phase, we can represent this as 0111 or 7; all processes having these properties can be categorized under the (15, 7) combination). The exhaustive list of combinations is: (3, 1), (3, 2), (3, 3), (9, 8), (15, 1), (15, 2), (15, 3), (15, 6), (15, 7), (15, 14), (15, 15). It should be noted that the underlined combinations are "outliers" and only account for roughly 1% of the data.

3. Copy and paste the 3 most impactful parts of your code and explain what each does and how it helped your model.

A) This is a rather straight-forward piece of code, but in our experience, gradient boosters will always achieve SOTA performance on tabular data, without much hyper-parameter tuning. Especially when you use an objective, that closely matches the evaluation metric on the leaderboard. We also implemented our own custom MAPE function, that takes into account the extra max-operator (with 290000) in the denominator, but CatBoost does not support custom metrics/objectives on GPU… Another small detail that had a huge impact, was to fit models on a logarithmic transform of the targets, instead of the original ones (making the targets more normally distributed, and somewhat mitigating outlier target values by squeezing together the range).

```
cat = CatBoostRegressor(iterations=100000, od_type='Iter', od_wait=100,
learning_rate=0.33, loss_function='MAPE', eval_metric='MAPE', task_type='GPU')
cat.fit(X_train, np.log(y_train), eval_set=(X_val, np.log(y_val)), verbose=50)
```

B) Stratifying the models based on the recipe of a process, and what phases are present for that specific process, gave a significant increase in performance. As such, we created 11 different models, based on every possible combination of recipe and present phases of processes in the test data. For each model, we constructed the test set, by just taking the corresponding processes from the test data. The training set was created by using as much processes as possible, by masking other phases and so on. By stratifying the models with this strategy, we mimic the masking done for some processes in the test data.

```
def get_processes(data, phases, train=True):
        filtered_processes = []
        phases = set(phases)
        processes = set(data['process_id'])
        for process in processes:
        process_phases = set(data[data['process_id'] == process]['phase'])
        if train:
        if phases.issubset(process_phases):
                filtered_processes.append(process)
        else:
        if len(phases) == len(process_phases) == len(phases.intersection(process_phases)):
                filtered_processes.append(process)
        return filtered_processes

# Example: get all train processes for recipe 15 (so planned to do all phases) and
# present phases 7 (so all phases except the acid one are present)
recipe_processes = recipe_df[recipe_df['recipe'] == 15].index
recipe_train_data = train_df[train_df['process_id'].isin(recipe_processes)]
train_processes = get_processes(recipe_train_data, 7)
```

C) Stacking allowed to chip of an extra percentage in the final weeks of the competition.

```python
def fit_stack(clf, name, X_train, y_train, X_test, train_index, test_index, n_splits=10):
  train_predictions = np.zeros((len(X_train),))
  test_predictions = np.zeros((len(X_test), n_splits))
  kf = KFold(n_splits=n_splits, shuffle=True)
  for fold_ix, (train_idx, test_idx) in enumerate(kf.split(X_train, y_train)):
        X_cv_train = X_train[train_idx, :]
        X_cv_test = X_train[test_idx, :]
        y_cv_train = y_train[train_idx]
        y_cv_test = y_train[test_idx]

        clf_clone = clone(clf)
        clf_clone.fit(X_cv_train, y_cv_train)

        train_predictions[test_idx] = clf_clone.predict(X_cv_test)
        test_predictions[:, fold_ix] = clf_clone.predict(X_test)

  train_predictions_df = pd.DataFrame(train_predictions,
                                      index=train_index,
                                      columns=['{}_pred'.format(name)])
  test_predictions_df = pd.DataFrame(np.mean(test_predictions, axis=1),
                                      index=test_index,
                                      columns=['{}_pred'.format(name)])

  return train_predictions_df, test_predictions_df

clfs = [
   (
        'knn_100',
        Pipeline(steps=[
        ('scale', StandardScaler()),
        ('knn_100', KNeighborsRegressor(n_neighbors=100))
        ])
        ),
   (
        'pca_et',
        Pipeline(steps=[
        ('scale', StandardScaler()),
        ('pca', PCA(n_components=25)),
        ('et',  ExtraTreesRegressor(n_estimators=25))
        ])
        ),
        # And many many many more...
]

for name, clf in clfs:
        # Read in the data and store it in X_train, y_train and X_test
        train_pred_df, test_pred_df = fit_stack(clf, name, X_train, y_train,
                                                X_test, train_index, test_index)
        # Write away the out-of-sample predictions
        train_pred_df.to_csv(...)
        test_pred_df.to_csv(...)
```

4. What are some other things you tried that didn't necessarily make it into the final workflow (quick overview)?

As for most data science projects: for each implemented feature/addition in our pipeline, 10 other things were tried that did not work (either reducing the performance, or not giving any significant improvement at a cost of a more computational expensive pipeline). Some things that required quite some effort and that did not end up in the final pipeline:

- Extracting features with other libraries than tsfresh, such as HCTSA
- Data augmentation: creating more samples by masking roughly 20% of the data of each process and repeating this a few times
- Creating a model that predicts the target, based on solely data of a single phase
- Deep learning techniques based on Fourier coefficients of each signal, or based on the raw data
- Extracting meta-features from the previous process in time on the same object (e.g. how much time ago did this machine get cleaned)

→ All of the work during the competition was done in notebooks (but not really in a structured way)... For completeness, we uploaded all notebooks along our code to give an overview of the amount of things done.

5. Did you use any tools for data preparation or exploratory data analysis that aren't listed in your code submission?

No. Python is love, Python is life...

6. How did you evaluate performance of the model other than the provided metric, if at all?

For each trained model, a plot with shapley values (similar to feature importances) was generated, and the residuals were plotted. Allowing us to assess which features were most important and could potentially be extended, and whether the model under- or overestimates.

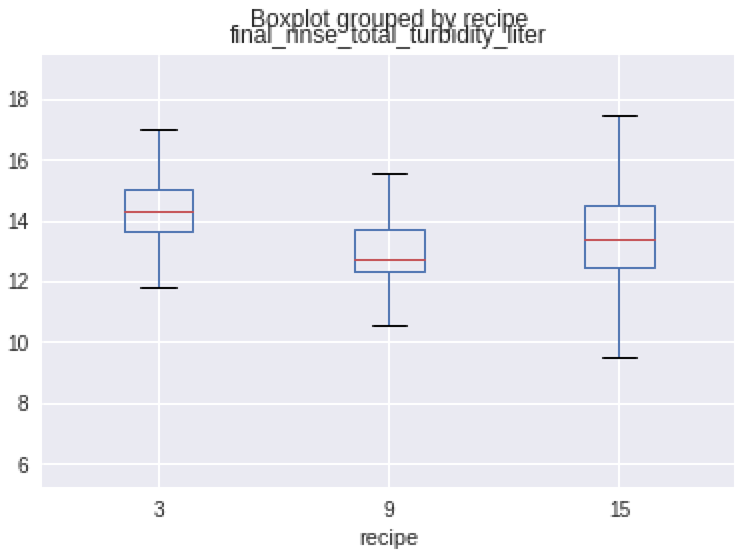
The total MAPE was calculated as a weighted average of the 11 MAPE values of each of the 11 models. The weight for each model was determined as the fraction of processes in the test set corresponding to the according combination. As an example, the model trained on processes of the recipe with all phases, but by masking the "acid" phase, or combination (15 (1111), 7 (0111)), is assigned a weight of 0.2258 as 670 out of 2967 processes in the test set possess these characteristics.

7. Anything we should watch out for or be aware of in using your model (e.g. code quirks, memory requirements, numerical stability issues, etc.)?

Quite a large amount of RAM memory is required (~10 GB). The code could be optimized to reduce the amount of required RAM.

8. Do you have any useful charts, graphs, or visualizations from the process?

You will find many useful visualizations in our report from the second phase. One simple visualization is a boxplot of the target values per recipe, which shows some clear differences in the target distributions of the different recipes:



9. If you were to continue working on this problem for the next year, what methods or techniques might you try in order to build on your work so far? Are there other fields or features you felt would have been very helpful to have?

We did not try very “exotic” approaches, since our experience with those was rather limited and the chance of them not being fruitful can be rather high (high risk – high impact). One cool approach would be to create sequence to sequence models, that can output the entire timeseries at once (a model could be trained for the return_turbidity, return_flow and target_time_period timeseries). Also we did not really look in depth at decomposing the target into multiple targets (e.g. predicting the duration, mean return turbidity, mean flow of the final rinse phase independently could offer extra information or could allow to reconstruct the target variable from the different predicted components).

One extra data source that could potentially be very valuable is the amount and type of usage for each of the machines. The reasoning behind this, is that when a machine has been used very intensively since its last cleaning, the machine will probably be more dirty. Also, you could assume that a machine in which food is stored is harder to clean than a machine in which liquid is store.