DOCUMENTATION

Machine Learning applied to Argo floats temperature and salinity Delayed-Mode Quality Control (Core-Argo DMQC)

Robin Le Guen, Ifremer

https://doi.org/10.13155/74394

Fiche documentaire

Titre du rapport : Machine Learning applied to DMQC	
Référence interne : IRSI/ISI-DTI/RAP-19-071 **Diffusion :** ☒ libre (internet) ☐ restreinte (intranet) – date de levée d'embargo : AAA/MM/JJ ☐ interdite (confidentielle) – date de levée de confidentialité : AAA/MM/JJ	**Date de publication :** 2019/11/21 **Version :** 1.0.0 **Référence de l'illustration de couverture** **Langue(s) :** Anglais
Résumé/ Abstract :	
Mots-clés/ Key words :	
Comment citer ce document :	
Disponibilité des données de la recherche :	
DOI : https://doi.org/10.13155/74394	

Commanditaire du rapport :	
Nom / référence du contrat : ☐ Rapport intermédiaire (réf. bibliographique : XXX) ☐ Rapport définitif (réf. interne **du rapport intermédiaire** : R.DEP/UNIT/LABO AN-NUM/ID ARCHIMER)	
Projets dans lesquels ce rapport s'inscrit (programme européen, campagne, etc.) :	
Auteur(s) / adresse mail	**Affiliation / Direction / Service, laboratoire**
LE GUEN Robin	ISI
Encadrement(s) : Thierry Carval	
Destinataire :	
Validé par :	

Introduction

This document is the synthesis of a study to apply Machine Learning to Argo floats temperature and salinity Delayed Mode Quality Control (Core-Argo-DMQC). There already exist numerous DMQC tests. Most of the times they are specific to a particular type of problem with an Argo profile or an Argo measure. For example, there are tests to detect drifts, other tests to detect spikes, others for thermal lags and so on … To get a clean database, all the alerts generated by those tests sum up and analysts need to study the corresponding profiles.

The aim of our study is to try to use Machine Learning to detect any kind of problem with Argo profiles and reduce the amount of time and work for the analysts.

The model can be upgraded in many ways but it already gets better performances than the existing solution we used to benchmark our model. For the same detection rate of BAD profiles, the model generate about 25% of alerts less tha the benchmark solution. This document reference the process we developed to get those results.

Table of content

1 Machine learning applied to DMQC

1.1 Target of the algorithm

1.1.1 Definition of the classes

Our aim when applying machine learning to DMQC is to **predict if a profile should be visualized by an analyst or not**. It seems more appropriate than working at observation-level because :

- If one observation needs to be visualized, then the analyst will visualize the whole profile.
- There are more information at the profile level than at the observation level.

A profile should be visualized if there is at least 1 BAD observation in it.

So the 2 classes to predict with the algorithm are:

- GOOD profile : No BAD observations
- BAD profile : At least 1 BAD observation

1.1.2 PSAL as target

In order to make a proof of concept, we decided to first search for BAD profiles in **PSAL**. And more specifically when PSAL is considered as BAD because of **conductivity sensor** or for **problem of transmission**. It means that if PSAL is BAD because of temperature sensor, we don't focus on it for the moment.

So when QC TEMP is BAD for an observation and QC PSAL is GOOD, we delete this observation from dataset. We make this choice because TEMP QCs needs to be predicted too so it can be considered as 2 different studies and it is probably a little ambitious to try to solve them both at once for a first try.

If this approach is validated, it will be possible to try to apply it to DMQC on TEMP.

1.2 Benefits of machine learning

Rather than generate alerts for every kind of problems (drifts, spikes, thermal lags…), here we try to have everything done in one place.

So if this approach succeeds, it means:

- There will be no **duplicates** in alerts. For example, if a profile has a spike and a drift, there will be only one alert.
- The algorithm can find **correlations between some features**.

2 Reliable history

We use supervised machine learning algorithms and this kind of algorithms needs a reliable history. Hence, we decided to request 2 kind of sources :

- Profiles visualized by analysts
- Profiles in ARGO DELAYED mode

2.1 Profiles visualized by analysts

The safest way to get reliable profiles is to use profiles visualized by at least one analyst.

Here we define the different sources of profiles visualized by analysts that compose our dataset.

2.1.1 ISAS source

Notebook : build_raw_dataset_isas.ipynb

ISAS is a solution that raises alerts. It is based on a climatology test. Each time an alert is raised on an observation, an analyst visualize the whole profile.

2.1.1.1 Profiles with TEMP and PSAL ISAS alerts

ISAS raises alerts for PSAL measures and for TEMP measures. But we assume that when a type of measure raises an alert, the analyst check every types of measures. So even if alerts are raised on TEMP, we keep the profile when we try to make predictions on PSAL.

2.1.1.2 Profiles with ISAS QC 6

ISAS use QC 6 to qualify BAD datas. It is an equivalent of ARGO QC 4 but it permits to identify that the BAD QC come from ISAS analysis.

Sometimes ISAS analysts visualize profiles that didn't raised alerts. Thanks to QC 6, we know that they checked it anyway. So we also use those profiles in the dataset.

2.1.2 SCOOP source

SCOOP is a tool that permits to visualize and validate profiles.

When a profile is visualized, it is tracked in metadatas of the profile. We use profiles with those tracks in the dataset.

2.1.3 Min-Max alerts source

Min-Max is a method that raises alerts when a measure is out of an interval parametred by a minimum bound and maximum bound. Each time an alert is raised the whole profile is visualized. So we use profiles that raised a Min-Max alerts in the dataset

2.2 Profiles in ARGO DELAYED mode

At first, we used only profiles visualized by analysts because we assumed that analysts were more focused when analyzing a profile that raised an alert than when analyzing all the profiles of a station to pass it in DELAYED mode.

But we realized that **profiles that raise alerts are either BAD profiles or GOOD profiles that seem BAD**. Because if a profile raises an alert, it is due to the fact that something doesn't seem

normal in it. And if we train only on this kind of profiles, the algorithm doesn't know how to treat the great majority of profiles which are GOOD profiles that don't seem BAD. That is the reason why we add profiles in ARGO DELAYED mode to the dataset.

3 Performance metrics

In order to evaluate the performances of the algorithms, we need to define metrics.

3.1 Metric 1 : Approximative number of alerts per year given a fixed detection rate on BAD profiles

The aim of the algorithm is to reduce the workload of analysts that treat alerts and get the best quality of dataset anyway. So we **fix a detection rate of detection of BAD profiles** and we **determine the number of alerts** it would generate yearly. **The objective is to reduce this number of alerts**.

We start by trying to detect **70% of the BAD profiles**. This detection rate can be changed but we need to consider that the **definition of a BAD profiles can be a bit vague**. Indeed, we lead a study (**notebook : study_class_of_duplicates.ipynb**) on profiles seen by different analysts and **15% of the 1099 profiles were of different classes**.

3.2 Metric 2 : Comparison with ISAS15 alerts only on GOOD profiles of the snapshot

The aim of this metric is to be able to **compare our results to an existing solution**. We choose to compare to the solution **ISAS15**.

ISAS15 raises alerts from a climatology test.

3.2.1 Conversion of ISAS15 alerts at profile level

ISAS15 is trying to detect BAD observations when we are trying to detect BAD profiles. So the first step is to convert ISAS alerts from observations to profiles (X alerts for X observations of a same profile is considered as one single alert).

3.2.2 Only the GOOD profiles of the SNAPSHOT

ISAS15 is applied on a snapshot of ARGO database and try to **detect BAD observations that are not already flagged as BAD in the snapshot**. It means that if a profile has 11 BAD observations and 10 are already flagged as BAD, then ISAS15 has to find the single last one. We can not compare our results with ISAS15 results on this kind of profiles because our algorithm will use all the observations of the profile to say if it is BAD one or not. The task is much harder for ISAS15 which can't use the 10 observations that are already flagged BAD. (We can presume that this 10 observations are easier to detect if already flagged).

Hence, we decide to base this metric **only on profiles that are full of GOOD measures in the snapshot. It means that we try to detect new BAD profiles**.

3.2.3 Detection rate fixed to get corresponding number of alerts

In the **notebook 'get_conf_mat_of_ISAS_on_good_ARGO_prf.ipynb'**, we identify the ISAS alerts on PSAL that corresponds to some requirements:

- The profile is full of GOOD QCs in the snapshot
- The profile date is in or after 2014. We will see in the section on validation strategy that we train before 2014 and validate since 2014.

- There are BAD QCs in the first 120 observations. (Here we talk of QCs after ISAS analysis and not in the initial snapshot). We will see in the section on preprocessing that for the moment our algorithm is applied only on the 120 first observations of each profile. So if the BAD QCs of a BAD profile are after the 120th observations, we can not detect it.

Thus, we can see that :

- Out of 15691 alerts, 5114 corresponds to our requirements.
- Among those 5114 alerts, 755 are alerts on BAD profiles. The others can be considered as false alerts.
- The number of profiles in the snapshot corresponding to the first 2 requirements ('full of GOOD QCs in the snapshot' and 'since 2014') is 294 012.
- 1600 of those 294 012 profiles are BAD profiles.

We can deduce the **detection rate on BAD profiles of the 5114 ISAS alerts which is 47%** (755/1600).

The metric we use to compare the results of our algorithm to ISAS15 is **the number of alerts we generate to get this same detection rate divided by the number of ISAS alerts**. So we get a pourcentage of the number of alerts generated to detect as much BAD profiles. This way we can know if we reduce or grow the workload of analysts.

4 Validation strategy

One of the most important step in machine learning process is the definition of the validation strategy. It means the definition of the datasets used to train the model and the datasets used to validate the model. If it is not well defined performances can't be reliable.

4.1 Split on dates

We can split the datasets of the validation strategy in multiple ways. It can be random or fit certain conditions.

In our case, we **split the datasets on dates** because it permits to **avoid overfiting**. If it is not splitted on dates, it would mean that we can only predict the class of a profile if we know the situation around this profile at the same time. And we could wonder what would happen when there are no profiles corresponding to this criteria ?

Futhermore, if the split is done on dates, **the model can also be applied in Real-Time mode**. Indeed, the model is validated on datas it never saw before so we can assume that it generalizes well in the future too. It is **the split that get us the closest to a deployment in production**.

4.2 Training, validation and test sets

When there are enough datas, it is pretty common in machine learning to **split the datas in three datasets: the training set, the validation set and the test set**.

First, the model is trained on the training set and use to make prediction on the validation set. **All the work of optimization is done this way and never looking at performances on the test set**.

Then when optimization is finished, we can use the test set to verify that the performances are the same that on the validation set. It can be done with 2 different ways :

- Keep the same model and predict on test set.
- **Re-train the model on training set concatenated with validation set and predict with this new model on test set**. In our case, we prefer this approach because the split is done on time. If we keep the same model and it is less performant on the test set than validation set, it can be due to the fact that it was trained on datas until a certain date and the validation set corresponds to (this date + 1 year) while the test set corresponds to (this date + 2 years). This would implies that knowledge about recent datas is valuable to the model and we presume that it is true in our case. (It would be interesting to verify this hypothesis.)

It is important to note that **the performance on validation and test sets must be similar**. If it is best by far on the test set, it is not necessaraly a good news. We can conclude from this that the model is unstable.

4.3 Custom split on validation and test sets

Earlier in the section 3.1 (Metric 1), we exposed that when 2 analysts visualize the same profile, the class of the profile is sometimes GOOD for an analyst and BAD for the other. So we know that we can not be 100% sure of the definition of a BAD profiles.

And we presume that if no analyst saw a profile, the incertainty in the quality of the QCs is greater.

That is the reason why we use a split which can be a bit strange at first sight but it is the solution that seems the more reliable to our point of view. This solution consist of using 2 datasets for validation set and 2 datasets for test set :

- **First dataset** : A dataset composed of **the most reliable datas in our possession : only profiles visualized by analysts** (see Section 2.1 : Profiles visualized by analysts). We make the predictions on this dataset and **find the threshold of prediction corresponding to the desired detection on BAD profiles** (see section 3.1 (Metric 1)). **Because this dataset is composed of reliable datas, we assume that if the algorithm can have a certain detection rate on this dataset, it is the detection rate of the algorithm**.
- **Second dataset** : A full year of profiles. Here we are **not concerned about the quality of the profiles**. This dataset is here to **estimate the number of alerts the algorithm generates yearly for the desired detection rate**. So we take the threshold of prediction determined on the first dataset and we apply it to the second one to get this number. The profiles with a prediction larger than the threshold are considered as alerts.

The first and the second dataset are **both representing the same year**. For validation set, there are only profiles of 2014 and for test set, only profiles of 2015.

The 2 datasets overlap because the first dataset is included in the second one but much more smaller : for example on year 2014, there are about 4000 profiles in the first dataset and 150 000 in the second one. We don't see it as a problem because they have 2 different purposes and we replicate exactly the same strategy on validation set and test set.

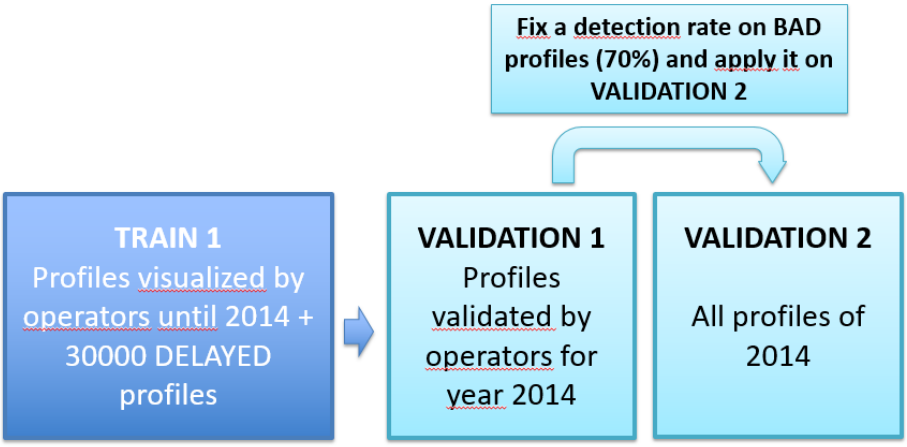


Figure 1 : Validation of the model

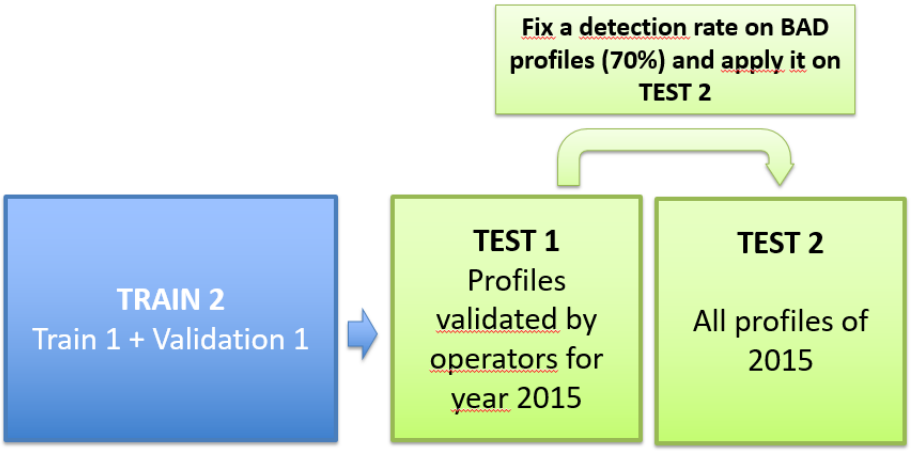


Figure 2 : Test of the model

5 Preprocessing

Notebook : preprocessing.ipynb

Here we preprocess the datas in order to give it the right shape and right format to be compatible with ML algorithms.

The notebook can generate datasets to make **classification of GOOD and BAD profiles** and **regression on the number of BAD observations per profiles** for both targets: **'PSAL' and 'TEMP'**.

5.1 Preprocessing in one place

We explain in section 2 (Reliable history) that multiple sources compose the dataset used in this study. There is a different notebook to request the profiles of each source.

Before creating the notebook 'preprocessing.ipynb', the preprocessing was executed in the notebooks specific to each sources. But then we realized, it is better to request raw features (measures, QCs, metadatas) of every profiles in those different notebooks and get all the preprocessing executed in only one notebook (preprocessing.ipynb). Here are the reasons that lead to this choice :

- The code is easiest to maintain because the preprocessing is only executed in one place.
- No need to regenerate all datasets when the preprocessing changes. For example, if we want a maximum length of profiles of 150 observations instead of 120, we don't need to regenerate each dataset but just execute the preprocessing once.
- If datasets are not generated at the same time, we are sure the preprocessing is the same.
- While studying predictions, if we want to see what was the profile before preprocessing (for example, is the BAD QC a 3 or a 4 ?), we don't need to find the sources and do a request. And furthermore **sometimes the raw profile can change. For example, profiles that come from GDAC can evolve**.
- If we want to have statistics on numbers of observations deleted by preprocessing we can. Before it was complicated because we had to do a sum of all observations along all datasets. Furthermore it was not precise because some profiles are duplicated in multiple datasets.

5.2 Adapted QCs labels

There are a lot of different QC values but our goal is to determine if a profile should be vizualized by an analyst or not. And a profile has to be visualized by an analyst only if there is at least one measure that deserve a BAD QC in it. So we decide to gather the different values of QCs in only three values :

- LABEL_GOOD (0) corresponds to QCs 1 and 2.
- LABEL_BAD (1) corresponds to QCs 3,4 and 6.
- LABEL_MISSING (2) corresponds to NaNs,7 and QC 9.

5.3 Missing measures.

Replace NaNs by a default value (MISSING_VAL = -999) because most ML algorithm can't take NaNs. Put the corresponding QCs to LABEL_MISSING. This way, we can't have a GOOD or BAD QC corresponding to a missing measure.

5.4 Missing QCs

We can't have a missing QC when there is a measure. So when this happens, we have 2 choices : delete measures when the QC is missing or give them bad QCs.

We differenciate 2 cases :

- If **a profile contains measures and all QCs are MISSING**, it is **probably a sensor with a consequent drift** so ISAS operators stop watching the measures because they already know it is all bad data. In this case, we decide to **fill with bad QCs** because the sensor records wrong datas.
- The other possible case is **a profile with some filled QCs and some missing QCs**. In this case, we can think that a missing QC means BAD but it can also means that they estimated they couldn't judge. So we decide that it is safer to **delete those measures from dataset**.

5.5 Unique profile sizes

Most machine learning algorithms require inputs of unique sizes. In order to satisfy this condition, we decide to create profiles of unique length.

If a profile is longer than the fixed length then it is truncated.

If a profile is smaller than the fixed length then we fill it with VALUE_MISSING (-999) for measures and with LABEL_MISSING (2) for QCs.

The actual version of the dataset is composed of the 120 first observations of each profile.

Note : We made this choice because we started by training the first algorithms with raw measures. But after multiple tests explained later in this documentation, we realized that raw measures are useless when we build specific features. So **the future releases of our project won't carry this limitation anymore**.

5.6 Treat dependecies between types of measures

- If pressure is BAD or MISSING, QC of temperature and salinity can't be determined because we don't know where the measure is taken in the water column.
- If temperature is BAD or MISSING, QC of salinity can not be determined because temperature is used in calculation of salinity.

Those rules are not always respected in available datas so we need to adapt them.

If for a same observation, the QC of a type of measure is BAD and the QC of another, which depends on the first one, is GOOD (for example QC PRES BAD and QC TEMP GOOD), there are multiple solutions:

- Delete the concerned profile.
- Give a MISSING QC to the dependant measure. In this case we need to delete the measure too.
- Give a BAD QC to the dependant measure. If we apply this solution to our case (determine the quality of salinity measure), the algortihm is not trying to only learn the quality of conductivity sensor's measures when sensors for other types of measures (TEMP and PRES) are alright but to understand when whatever sensor is OK or not. This seems to us too ambitious for a first try.

In order to choose if we delete only observations concerned or whole profiles when at least one observation has to be deleted in a profile we counted number of observations and profiles concerned. On the dataset of the most reliable profiles (visualized by analysts) at the date of 20191023, there are 24955 profiles containing BAD QCs on TEMP. It correspond to 667994 BAD TEMP QCs. And there are 6489 profiles full of BAD TEMP QCs. So it seems that **keeping the profiles and delete only the observations with a BAD QC can be a solution to keep more datas in the dataset**.

We choose to apply the following rules :

- If an observation has **PRES MISSING** then we **delete this observation**. TEMP and PSAL can not be reliable if we don't know the pressure.
- If an observation has **TEMP MISSING**, we **delete this observation**. We can not judge TEMP neither PSAL if TEMP is missing because PSAL depends on TEMP measures.
- If an observation has **PRES BAD, TEMP GOOD and target is TEMP** then we **delete this observation**. We can not assign a TEMP GOOD when there is a BAD PRES.
- If an observation has **PRES BAD, PSAL GOOD and target is PSAL** then we **delete this observation**. We can not assign a PSAL GOOD when there is a BAD PRES.
- If an observation has **TEMP BAD, PSAL GOOD and target is PSAL**, we **delete this observation**. We can not assign a PSAL GOOD when there is a TEMP BAD because TEMP is included in the calculation of PSAL.

5.7 Drop duplicated cycles

We explained in section 2 (Reliable history) that we request multiple sources in order to build our dataset. A profile can be represented in more than one of this sources.

If a profile is duplicated over multiple sources the information can be different between the 2 sources because some sources take profile in the snapshot of 2016 of ARGO database and other sources take profiles from the GDAC (actual state of the ARGO database). And sometimes adjustements are applied to the measures after the snapshot. So adjustements that were not applied in the snapshot of 2016 can exist and change measures in the GDAC.

It can be interesting if we have a duplicate with different measures in the 2 occurrences of the profile. For example, if the second occurence has a new adjustment which change the class of the QC from BAD to GOOD. But it is a problem if they are similar or kind of similar because if a model is able to detect one of the occurence it will easily detect the second occurence and the detection of one profile will count for two in performance metrics (because it is twice the same one). Futhermore if the split between train and validation is random then it can overfit because similar or identical profiles can be in train and validation sets.

In order to solve this problem, **we decide to take only the profile present in the Snapshot 2016 when the profiles is in a source coming from the snapshot and a source coming from GDAC**. We made this choice because the only case when it could happen is when a profile is in an ISAS source (isas_qc6, isas_temp or isas psal) and in a source from GDAC. And the confidence in ISAS results is pretty high.

So to synthetise :

- When there are **duplicates** in lines that contains **only profiles from snapshot**, we take **the first occurence of each unique cycle number**. ('001_A,'001_A','001_B' become '001_A','001_B'). There can be letters (‘_A’, ‘_B’, …) after the cycle number because a same cycle number can have different directions (ascendant and descendant measures) and it can have also different schemes (measures taken every X steps, every Y steps …).
- When there are **duplicates** in lines that contains **only profiles from GDAC**, we take the **first occurence of each unique cycle number**.
- When there are **duplicates** in lines that contains **profiles from both sources** then we take the **first occurence of each unique cycle number only for the profiles that come from snapshot**.

5.8 Delete empty profiles

Profiles with no observations are useless in the dataset so we delete them.

5.9 Get labels for regression and classification

Here, we define the target of machine learning algorithm (the result it tries to predict). For the moment, we only do classification but the notebook permits to generate regression labels too.

Label for classification : class of the profile.

- The profile is GOOD if there are no BAD observations in it.
- The profile is BAD if there is at least one BAD observation in it.

Label for regression : number of BAD observations in the profile. The aim of the regression is to predict this number.

5.10 Drop profiles with BAD QCs for date or position

If the date or the position is not reliable, we can't exploit the profile so every profile with a bad QC on date and/or position is deleted.

5.11 Replace NaNs by medians of the training set

We explained that if some measures were missing, they were replaced by a default value (MISSING_VAL = -999) but for features that are not measures there is still the possibility to get NaNs. **Thus, for features that are not measures, we replace every NaN by the value of the median of the corresponding column in training set**.

6 Features

In this section, we reference all the features available in the dataset. **They aren't all necesseraly used in every model** because **sometimes they can be useless** or **sometimes we wonder if they can lead to overfiting**.

6.1 Metadatas

The dataset contains metadatas :

- wmo
- cycle
- lat
- lon
- direction
- month
- year : With random forest and a validation strategy based on a time split it is not optimised because the validation set contains only values the algorithm has never seen but it has proven to be helpful anyway. It must absolutely be considered as a continuous variable and not a categorical variable to permit to the algorithm know that 2013 is superior to 2014.

6.2 Lenghts of profiles

Some features use mean and standard deviation of information relatives to each observation. It can be useful to the algorithm to know how many observation constitute those means and standard deviations.

And perhaps some sensors are better when they make a lot of measures or really few measures ? This feature can encode this kind of information.

6.3 Negative pressures

Usually there are automatic tests that flag negative pressures but it seems that sometimes it is not the case. So we add features to qualify negative pressures :

- Number of observations with negative pressures in profile
- Minimum of the negative pressures

6.4 Climatology features

For each observation of the profiles, we get the mean of measures, the standard deviation of measures and the number of measures in the corresponding grid square of the desired climatology.

6.4.1 Features on the difference between measures and climatology means

For each observation, we calculate the difference between the measure of the observation and the climatology mean. And from there, we generate the following features for each profile:

- Mean of the differences
- Standard deviation of the differences

- Min of the differences and the corresponding PRES, TEMP and PSAL in order to get more info about the context of this minimum
- Max of the differences and the corresponding PRES, TEMP and PSAL in order to get more info about the context of this maximum

Thanks to this features, the algorithm can know if the majority of the profiles is far from the climatology or not. **Or for example understand that the majority of the profile sticks to the climatology but at least one value gets really far from it** (thanks to features on the maximum of the values).

6.4.2 Features on number of climatology standard deviations between measures and climatology means

For each observation, we calculate the number of climatology standard deviations between measures and climatology means. And from there, we generate the following features for each profile :

- Mean of the values
- Standard deviation of the values
- Min of the values and the corresponding PRES, TEMP and PSAL in order to get more info about the context of this minimum
- Max of the values and the corresponding PRES, TEMP and PSAL in order to get more info about the context of this maximum

6.4.3 Features on number of observations in climatology grid squares

For each observation, we have the number of observations in the corresponding grid square of the climatology. From there, we create the following features :

- Mean of the values
- Standard deviation of the values
- Min of the values
- Max of the values

Thanks to this feature, the algorithm can know if it can have **confidence in the features based on the climatology**. Indeed if the mean of those values i slow then it means that information are less reliable.

6.4.4 Features on the deepest observation of the profile

In general, the deepest we go in the sea and the more stable are measures over time at a specific location. That is the reason why we decided to dedicate some features to the deepest observation of each profile :

- PRES, TEMP and PSAL
- Difference between the measure of the observation and the climatology mean
- Number of climatology standard deviations between measures and climatology means

6.4.5 Multiple climatologies

Climatologies can be built using different ways. For the moment, we worked only with climatology of WOA18 (World Ocean Atlas by NOAA). And climatology of WOA18 are built among three main criterias :

- Size of the grid squares : 5°, 1°, ¼°

- Period of times : all available measures or a specific decadal
- Type of measures : TEMP or PSAL

We decide to **use multiple climatologies to build multiple times all the features explained above because we think they can be complementary**.

It seemed that size of grid squares can be an important factor. Indeed, bigger grid squares gives a global information about the surrounding of the observation and are built with a lot of observations. And smaller grid squares can give precise information on the surrounding of the observation but can perhaps show lack of observations.

Furthermore, even if our target is PSAL, we wanted to also create features around TEMP because sometimes it happens that TEMP and PSAL doesn't have the same behaviour in the water column and it can relate to a spike on one of the sensor.

Finally, we included a time factor too in the choice of climatologies because WOA18 propose only climatologies with all available measures for the size of grid squares of 5°. And all other climatologies we chosed are based on the last decadal available.

So the combinations of criterias for the climatologies we used are :

- **PSAL; 1/4° ; 2005/2017**
- **PSAL; 1° ; 2005/2017**
- **PSAL; 5° ; Averaged decades from1955 to 2017**
- **TEMP; 1° ; 2005/2017**

7 Tools to analyze models resuts

In this section, we define different tools that permit to get model performances and get some statistics and visualization of the results to understand them better.

7.1 Number of BAD profiles according to a parametred number of alerts

We defined a function that take in input an array of as many number of alerts as we desired. The function returns the numbers of BAD profiles detected for each number of alerts and the corresponding thresholds of prediction.

It is usefull to **easily see how the performance evolves with number of alerts** and **fix this number if a model has to go in production**. Because in production, analysts can treat only a certain number of alerts in a certain time.

7.2 Number of alerts to detect according to detection rates on BAD profiles

We defined a function that take in input an array of as many detection rate on BAD profiles as we desired. The function returns the numbers of alerts generated for each detection rate and the corresponding thresholds of prediction.

It is the same principal as the function above but **having this 2 tools permit to fix objectives** as we want: **do we want to reduce number of alerts generated or do we want to get a better detection rate ?**

7.3 Number of alerts raised for the same detection rate as ISAS15 alerts

This corresponds to the performance metrics 2 (see section 3.2 : Metric 2).

7.4 Confusion matrix with threshold

Get confusion matrix according to a desired prediction threshold.

We can get them in pourcentage and real number of profiles.

	BAD Profiles	GOOD Profiles
Not alerts	A	B
Alerts	C	D

7.5 Confusion matrix on different sources

Get confusion matrix on every sources. It permits to see if the performances are the same on every sources.If they are different, we can try to understand why.

	0	1
0_isas_psal	98	33
1_isas_psal	2	67
0_isas_qc6	100	39
1_isas_qc6	0	61
0_isas_temp	99	11
1_isas_temp	1	89
0_scoop	100	39
1_scoop	0	61

Figure 3 : Confusion matrix by sources (in pourcentage)

7.6 Feature importances

Get the importance of each feature and visualize the evolution of feature importances. This tools is implemented in scikit_learn and permits to understand which features the model use the most to predict classes.

	cols	imp
0	s_lst_obs_dif_clm_decav_s00_5d	0.071830
1	s_lst_obs_nb_sd_decav_s00_5d	0.071754
3	s_lst_obs_dif_clm_A5B7_s00_01	0.044685
2	year	0.037836
8	t_lst_obs_PRES_A5B7_t00_01	0.036201
4	lon	0.034935
5	len_cyc	0.034140
6	s_lst_obs_PRES_A5B7_s00_01	0.030617
9	s_nb_sd_max_decav_s00_5d	0.029202
12	lat	0.027433
13	s_lst_obs_PRES_decav_s00_5d	0.026272

Figure 4: Feature importances

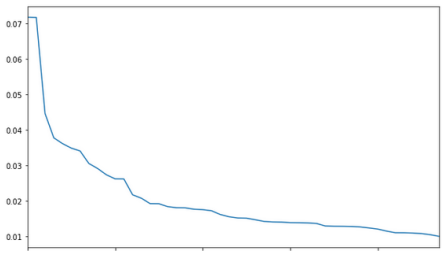


Figure 5: Evolution of feature importances

7.7 Visualize predicted profiles

We defined 4 functions :

- See best GOOD predictions (profiles with highest predictions on class GOOD)
- See worst GOOD predictions
- See best BAD predictions
- See worst BAD predictions

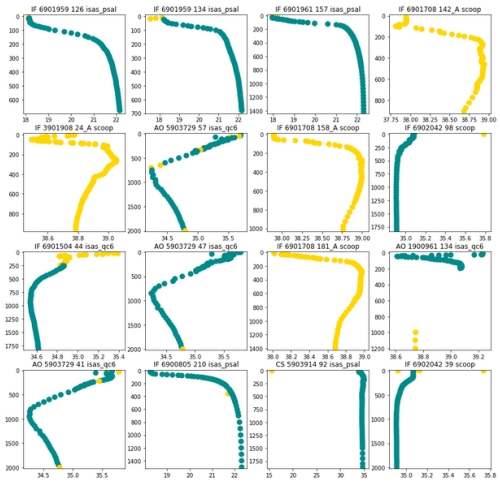


Figure 6 : Best BAD predictions

- GOOD observations are colored in green.
- BAD observations are colored in yellow.

This tool is **useful to understand what kind of profiles the model is good at detected or not**. **From those observations, we can decide which features to create to enhance the model**.

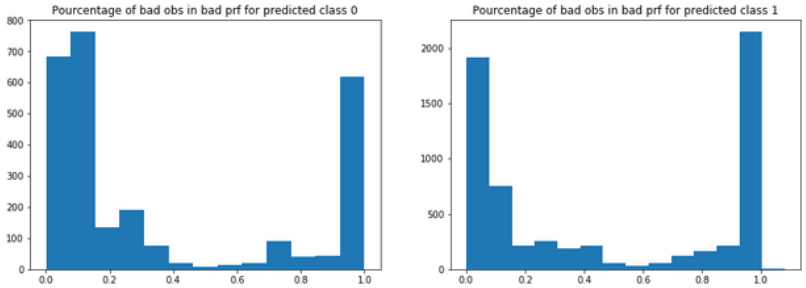
For example, by visualizing worst BAD predictions, we identified on a model that all the profiles that the algorithm couldn't detect were profiles full of BAD QCs. It gave us the idea that perhaps the model needed features to qualify drifts because when a whole profile is wrong it can be due to the fact that the sensor is drifting.

Note : In the future, we'll try to add also the predictions were the model is the less sure of the class of the profile. 50% GOOD /50% BAD.

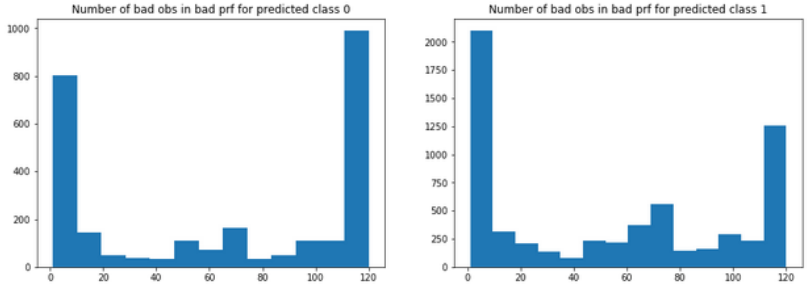
7.8 Plots on BAD observations in profiles

We can plot an **histogram of the percentage of BAD observations in PREDICTED class BAD and PREDICTED class GOOD**. It is done only for BAD profiles because there are not BAD observations in GOOD profiles.

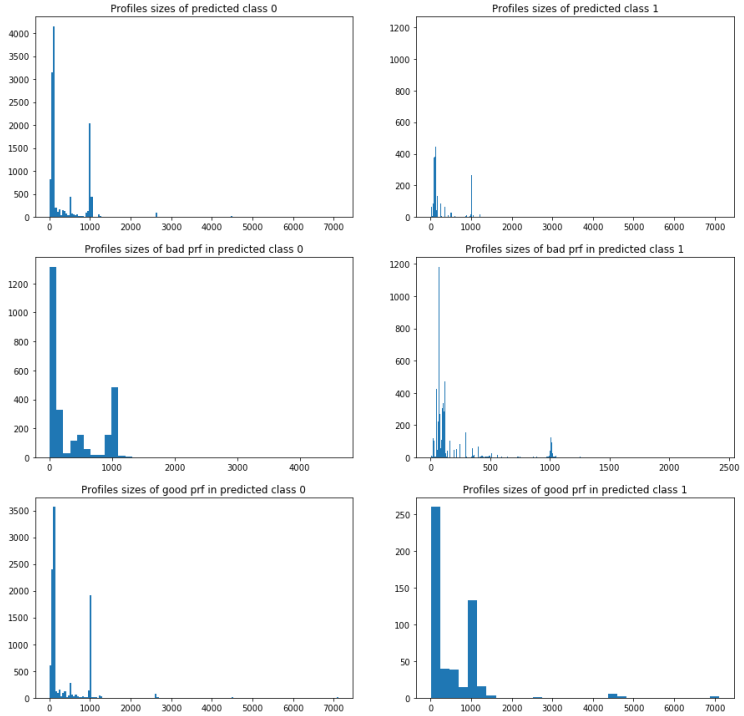
It permits to see if the algorithm is good at predicting profiles full of BAD observations or if it can be good at predicted profiles with few BAD observations too.



We did the same thing with number of BAD observations (rather than pourcentages).



7.9 Plots on profiles sizes



7.10 Maps of predictions

We defined 2 functions :

- Plot GOOD and BAD predictions of GOOD profiles
- Plot GOOD and BAD predictions of BAD profiles

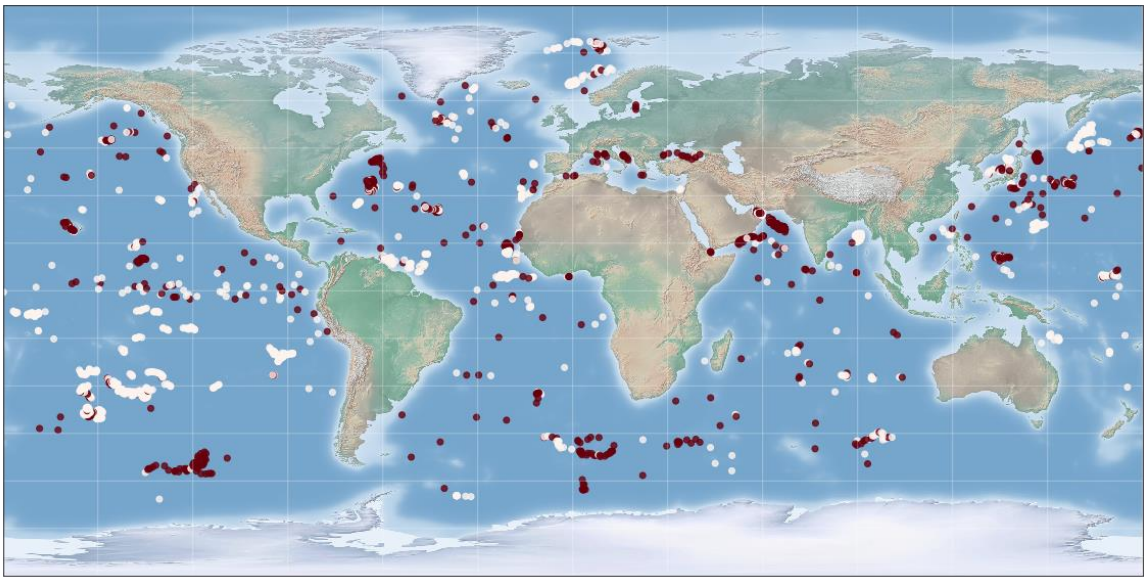


Figure 7: Plot of GOOD and BAD predictions for BAD profiles

8 Optimize models

In this section, we expose the different ways we implemented to enhance the performances of a model. It means that datasets and features are set and from there we try to get the best performances out of it. The models trained for the moment are essentially random forests.

8.1 Add delayed profiles to dataset

We exposed in section 2.2 (Profiles in ARGO DELAYED mode) that we think adding profiles in ARGO DELAYED mode can help the model **learn what is the definition of a GOOD profiles**.

So we tried to add some profiles in ARGO DELAYED mode and here are the corresponding results.

Number of delayed profiles	Number of alerts for 70% detection rate
0	
10000	
20000	
30000	
40000	
50000	
100000	

Note: The random seed is fixed but try to play with this parameter is surely something to test in the future to boost performances by selecting the right set of profiles.

8.2 Recurrent Features Elimination

The features presented in section 6 (Features) represent almost 500 features. We assumed we could **get better performance with less features** so we implemented a RFE (Recurrent Feature Elimination).

The principle of a Recurrent Feature Elimination is based on iteration and at each iteration, the following steps happen:

- Calculate the importance of each remaining feature. It is done by using an implemented function in the library scikit-learn
- Order features by feature importances
- Delete a certain number of the features with the lowest feature importances. This number is set as a parameter.
- Calculate the performance of the model

This way, we keep the set of features with the best performance of the model.

We went from 488 features to 42 features and the perfomances went from X to X.

It permits to get better performances but it permits to get models more stable too. If the model relies on less features it can be less complex.

8.3 Grid search

Machine learning algorithm have hyper-parameters. They are a lot of them. They can do many things like parametrize the architecture of the model or define ways to train the model.

For example, we can set the maximum depth of trees in random forest or the number of trees in the model.

The choice of hyper-parameter can have a considerable influence on the performance of models and the combination of their values is almost infinite.

We need a way to find performant combinations effectively. In order to this we used grid search.

Grid search is implemented in the library scikit-learn but we decided to **implement our own version** because it allows more flexibility. For example, we can implement the custom metrics we defined in section 3 (Perfomance metrics).

8.4 Hyper-parameter after hyper-parameter

Our approach is to test hyper-paremeter one by one. For each hyper-paremeter, we try different values. We keep the best value of the tested hyper-parameter and we go to the next one.

nb_estimators	
criterion	
max_depth	
max_features	
min_samples_leaf	
max_leaf_nodes	
class_weight	

8.5 Random search

We also implemented random search. It permits to set values for different hyper-paremeters, generate all the possible combinations of those values and select a random sample of the available combinations.

This way we can explore combinations around our best combinations using the “hyper-paremeter after hyper-parameter” approach.

It permits to explore completely random combinations too. Sometimes we can have good surprise.

Note: Instead of using completely random search, it is possible to use not totally random method like Bayesian Optimization. We consider using this kind of methods in the future.

8.6 Reduce number of trees

It is a good practice to do the grid search with less trees and add trees when the best combinations of hyper-paremeters is set. It permits to gain a lot of time in training. We **first learn to build performant trees** and **then use a lot of them**.

To fix the number of trees, we train the first models with different numbers of trees. Then we choose a number of trees where the performance starts to be asymptotic. For example, we can get 95% of the best performance by using 50 trees and to get about 100% of the best performance it requires 300 trees so we can go 6 times faster using this method.

Note: Same thing could be done with the number of observations in dataset.

Conclusion

So the actual model already fulfill our goal which was to reduce the amount of work for analysis with actual methods.

We now have the architecture of a complete machine learning workflow. It goes from the constitution of a dataset to the analysis of the performance of a model. We have trust in the validation strategy so the performance are reliable.

All along this process, there are possibilities to enhance the performances of the model. We already developed some of them but we left some others not fully optimized because our first aim was to get this complete workflow. Now we can focus on different sections of the process.

The first task will probably be to treat whole profiles and not the 120 first observations because the RFE proved that raw measures are almost useless features. Thus, we can build the other features with whole profiles and delete the raw measures before training.

Then we can create new features to address every kind of problem with the measures because for the moment the model can't understand every one of them. We can also test other machine learning algorithms like XGBoost or Neural Networks. Then the prediction power of multiple models can be combined to get better performances by stacking them. It is a good track to explore because different type of models find different type of correlations between features so it can be complementary. To conclude, there are many possibilities to enhance performances of our model in the future…