Abstract

Appliance Classification using load monitoring is an approach that is being utilized to increase energy efficiency and reduce energy consumption. In a non-intrusive set-up, machine learning techniques need to be applied to classify appliances. In order to achieve this classification, with the help of data exploration and feature engineering, we incorporated XGBoost and Random Forest models whose performances within different hyperparameter configurations were then compared. After going through a series of optimization steps, we ended up with a mixed optimal configuration using Random Forest for some appliances and XGBoost for others.

Table of Contents

1 Introduction

In this project, we deal with Residential Energy Appliance classification where load monitoring is used to identify the appliances being used. The approach used here is that of Non-Intrusive Load Monitoring which is the process of identifying individual appliances by monitoring the electric loads (Pereira & Nunes, 2018). Since the method is non-intrusive, it is extremely cost effective and convenient to set-up and analyze load data (Hurtig & Olsson, 2019). Hence, being able to correctly classify the appliances is important as it helps us to understand the electricity consumption and as a result increase energy efficiency and reduce greenhouse gas emissions.

With present energy generation in most places unable to meet the ever increasing energy demands combined with the issue of energy production being one of the leading causes of greenhouse gas emissions, it has become necessary to find a solution to the issue (Khan, Latif & Sohaib, 2018). Setting up new energy generation plants is neither cost-effective nor does it solve the issue of emissions. In this scenario, a more acceptable solution is to have more efficient energy consumption. This is where Non-Intrusive Load Monitoring(NILM), an idea proposed in the 1980s by George Hart comes in (Hart, 1992). NILM uses a limited number of sensors to measure the electric loads. This can then be analyzed and by utilizing machine learning, we can identify the appliances in use. This information can be passed on to consumers to prompt them to utilize energy efficiently.

The goal of this project is to utilize different machine learning/classification techniques to identify the appliances in use - these can be air conditioner, electric vehicle charger, oven, cloth washer, or dryer. We intend to find the best performing model for each appliance and infer why the selected features and model delivers the best results in each case.

In order to achieve this, we have completed the following steps:

- Exploring the data: this allowed us to make findings about the variables distributions and their relationships that helped us at the time of building the different models.
- Modelling: taking into account what we found during exploration, we built 2 different model types for each appliance: XGBoost and Random Forest classifiers. We went through a sequence of steps in order to improve our models' performances, finally keeping the combination of models that yielded the best overall performance.
- Making predictions on the test set using the models selected on the previous step.

All 3 students participated actively in each of these steps.

2 Exploration, pre-processing and feature engineering

2.1 Exploration

We first tried to understand how our targets were distributed, and how each of them was related to the predictors. This part was a little tricky because our targets are binary variables. To be able to understand their distributions, we had to convert them into continuous values in the following way:

- Calculating proportion of time the specific target was switched on. We aggregated the data, by hour and day of the week for instance, and calculated the mean of each target. In this way we were able to compare hourly proportion of usage between targets, understand their relationship with load and with the temporal features as well. This procedure also allowed us to determine that the hours of the day had a lot of influence on the proportion of usage of each appliance (considerably more than the days of the week for instance), hence, also on the variation of the value of load. That is why we decided to introduce a new feature representing the part of the day (Early Morning, Morning, Afternoon, Evening, Night) which we will discuss in the following section.
- Calculating average time the target stayed on once switched on. This particular calculation helped us define the values for the windows taken for each of the classifiers.
- We measured the variation of load with respect to all the possible actions in the targets (switching on, switching off, keeping on, keeping off) to better understand the relationship between predictors and target. Adding this to our previous calculations, we concluded that the ranking of relevance of the appliances regarding the load variation is as follows: ac, ev, oven, dryer and wash. Moreover, we

expected the performance of the different appliance models to also follow that order because of the strength and predictability of the relationship between each appliance and the main predictor load.

2.2 Pre-processing and feature engineering

As stated above, we decided to introduce a new feature referring to the part of the day as we were able to detect patterns in the different target proportions of usage when grouping the data by hour of the day. Moreover, as the data is sequential, groups of subsequent hours behaved really similarly, regardless of the day of the week.
We also decided to consider a neighbourhood of each data point, also known as window, which is practically traduced by including the load values of n points before and n points after as new features for each row. As we are going to see later in the Model Development section, we treated this as a hyperparameter that we tuned for each appliance. The optimal value turned out to be congruent with the average time each target stayed on, as pointed out previously.
After building a basic version of each of the XGBoost models, we used the feature importance tool to assess the relevance of each feature with respect to the target and found out that there were a few of them always ranked at the bottom in all the classifiers (and day of the week was one of them, as suspected from Exploring the data). We ran different configurations removing each of these variables and ended up with different sets of features for the different appliances models, which we will describe in more detail later on.

3 Models

3.1 Random Forest Classifier

Random Forest Classifier is an ensemble tree-based method. Ensemble refers to combining the predictive power of multiple trees rather than building a single tree which will likely suffer from high variance. This can be overcome by taking multiple bootstrapping samples used for training multiple decision trees and it is called Bagging (Bootstrap Aggregation).
Bagging is applied by the Random Forest algorithm with the particularity that, within each bootstrap sample used to train a new model, also a subset of features is selected every time. This helps overcome the possible correlation between the features and explore combinations between unrepresented variables.
As any bagging algorithm, Random Forest is looking for a reduction in the variance and that is why usually deep trees with large max_depth are required.
The summary of the steps followed by a Random Forest Classifier are then as follows:

1) Take multiple samples with replacement (bootstrap samples) to train multiple trees.
2) Randomly select a subset of features to work with each bootstrap sample.
3) In each tree, split nodes iteratively, gradually segregating the data, choosing at each split the feature that minimizes node impurity (using measures such as gini or entropy which both measure how mixed is the data in each node).
4) Grow each individual tree until data is splitted correctly (or other termination condition, generally handled by a hyperparameter)
5) Make predictions as the majority vote between all the different trees.

The good thing about this Algorithm is that data does not have to meet any assumptions nor feature interaction has to be explored. Apart from this, we also chose to try this model because it usually performs well with non-linearly related data, such as the given one, as we were able to explore.

3.2 XGBoost Classifier

Boosting is also a tree-based ensemble technique but in this case the new models are added sequentially to correct the errors made by existing models. Gradient boosting particularly is an approach where errors are minimized by the application of the gradient descent algorithm.
Unlike Bagging models, Boosting models will try to minimize bias instead of variance. This is the reason why they generally make use of simpler, more shallow trees.
As the variance is not being contemplated, this type of approach can be very prone to overfitting with noisy data. This is where XGBoost comes to the rescue as it is an optimized parallelized implementation of Gradient Boosting with enhancements such as regularization to prevent overfitting. Computational performance by

using the depth-first approach that allows the tree to be pruned backwards is also considerably increased in this model.
We picked XGBoost for this particular task because, as just stated, it is fast and efficient (and our data set is considerably large) and it is proven to have good performance while working with sequential data.

3.3 Other models

In addition to the 2 models detailed above, we tried other models such as Long short-term memory (LSTM) recurrent neural network, MLP(Multi Layer Perceptron) Classifier and Logistic Regression. Nevertheless, as we were unable to obtain good performances with these other models, we decided to move forward with XGBoost and Random Forest.

3.4 Comparison between Random Forest and XGBoost

Theoretically speaking, summing up a few of the differences we stated while explaining both models:

- Random Forest, being a Bagging algorithm, works with multiple trees in parallel while XGBoost, being a Boosting algorithm, works with different models in a sequential manner.
- Random Forest will require bigger and deeper trees than XGBoost.
- Random Forest includes randomness into training and testing data because of the random nature of the feature selection while XGBoost does not.
- Random Forest will be less prone to overfitting because it naturally aims to reduce variance. Nevertheless, XGBoost contains a regularization parameter to help overcome this usual issue that Boosting algorithms suffer from.

In a practical sense, after having performed the experiments, we were able to notice the following differences:

- Random Forest has simpler hyperparameters to understand and less of them. XGBoost has so many and it is harder to maintain the balance of reducing overfitting versus underfitting if these hyperparameters are tuned to extreme values.
- Random Forest is considerably slower than XGBoost. In practical terms, this affected our ability of tuning Random Forest hyperparameters: we just tried less combinations because of computational constraints. As we will be able to see in the following section, the increase in performance because of hyperparameter tuning was larger in the case of XGBoost, in part, because we were able to try a wider range of hyperparameters and their values.
- Relating to Random Forest requiring bigger and deeper trees than XGBoost, we will observe when explaining our experimental results, that, for example, the max_depth optimal value for Random Forest is 100 while it is in the range 6-11 for XGBoost. This obviously translates as well in slower running times.

4 Experiment Setups

4.1 General setup and stages

We were able to build, train and evaluate these 2 models side-by-side, which was very convenient in terms of code saving, code interpretability and potential comparison between them.
The steps we took while building, training and testing these models were the following:

1. We first built the basic Random Forest and XGBoost versions for each target (with all the features and no hyperparameter tuning), to get a general idea of their overall performances. We were going to use Blocking Series Time Split (like a cross validation for sequential data, in which the data is iteratively splitted in training and validation but keeping the order and moving forward) but it was too time consuming so we just used the validation set approach without shuffling, to maintain the logic of the sequence.
2. Secondly, as the data was highly imbalanced towards the negative classes, we calculated the threshold that maximized the F1-score for each target model, using the precision-recall approach. In order to do so, we had to make our models return probabilities as predictions instead of binary outcomes. For each prediction we had a probability of belonging to the positive class, any probability larger than the threshold would produce a positive prediction, and a negative prediction otherwise. With this new calculation of thresholds, we were expecting to obtain thresholds smaller than 0.5 (which is the

default one), at least for the most imbalanced classes. If the threshold is smaller than 0.5 we need a smaller positive class probability for our prediction to be classified as positive, which translates into more positive predictions. We calculated one optimal threshold for every type of model for each target and we used them from this point onwards to train all the subsequent models.

3. We plotted and assessed the feature importance for each model, extracted using the XGBoost library. We were able to come up with a set of commun variables that were being ranked as low-relevant by all the models. We used these sets of low ranked relevance variables to train and evaluate different versions of XGBoost and Random Forest for each target, having extracted one or a combination of them at a time.
4. We introduced the concept of window and treated it as a hyperparameter. The window basically controls how many observations of the variable load, up and down the current instance, are included as new features of the current data point. Finally we chose the value for the window that yielded the highest validation performance for each of the models for further tuning. We believe that it is important to remark here that, to be able to calculate the validation performance, we splitted the data using the validation set approach. We didn't use k-fold cross validation for example because of the data being sequentially ordered. We believe that the performance would have been affected if using k-fold, because of the random nature of the procedure. Nevertheless, we did shuffle the train and validations sets at this particular point, because, having included all the windows as new features (columns) in every data point, then the fact of shuffling or not becomes irrelevant (because the row already has all the information it needs to be trained, making each row independent from each other).
5. Once having defined the window size and the optimal subset of features, we proceeded with the hyper-parameter tuning for both model types.
6. We kept the best-performing model for each target and trained it using the entire training data.
7. To be able to perform predictions on the test set, we also had to pre-process the test data in order to add the optimal window instances to each observation. Once having done so, we were able to make the predictions.
8. We realized that some of the predicted targets, such as ev, which was the most imbalanced, were yielding a positive-negative prediction ratio considerably lower than in the training set. We assumed this could not be a simple coincidence so we decided to lower the threshold even more for this particular variable. We then calculated a new optimal threshold using this time the AUC curve approach that aims to maximize the G-mean which is a metric that measures the balance between classification performances on both the majority and minority classes. We also knew that this approach usually returns smaller thresholds than the precision-recall approach, which was convenient for this particular case, to be able to predict more positively. The good thing was that, for the particularly affected variable (ev), the application of the new lower threshold optimized by AUC curve did not decrease the F1-score (which did happen with all the other targets, but we didn't care because they weren't being as affected as ev) which would have been a problem because it was the metric that we wanted to maximize.
9. Finally, after having calculated the new threshold for ev, we re-trained the models and made our final predictions.

4.2 Hyperparameters

XGBoost Hyperparameters

For the XGBoost models we decided to play around and optimize the following set of hyperparameters:

- max_depth: maximum depth of the tree. Increasing this value makes the tree more complex and more prone to overfitting.
- min_child_weight: hyperparameter to control the growing size of a tree. If the tree partition step results in a leaf node with the sum of instance weight less than min_child_weight, then the algorithm will stop partitioning. Larger values of this hyperparameter are generally associated with reduction of overfitting and reduction of complexity as well.
- learning_rate: size of the step taken in the opposite direction of the gradient while updating the weights of the new features in the gradient descent optimization. Shorter steps make the model more conservative, hence, less likely to overfit.
- lambda_reg: L2 regularisation parameter. As lambda increases, the penalty imposed on the complexity of the model grows, making the model less complex and less likely to overfit.

- n_estimators: how many trees we will build in our ensemble. More trees is generally associated with reducing overfitting.

As pointed out before, and as we can see in the hyperparameters description, the main focus of them all is to address the bias-variance tradeoff. The thing is that there are so many parameters (there are even more and some of the ones we didn't address also can affect this matter) and they all contribute to the final overfitting or underfitting of our model, so that is why it ended up being a little hard to balance them out.

Random Forest Hyperparameters

When going through the documentation, we found it easier to understand these hyperparameters than with the case of XGBoost. The number is more limited and there are not as many of them focused on the Bias-Variance tradeoff, which makes the playing around and balancing easier. Having said that, the computational load was a very big deal here, and we didn't have the chance to explore as many hyperparameters as we would have wanted, such as we did with XGBoost. We just tried the following two:

- max_depth: same concept as in XGBoost, referring to the depth of the tree. As highlighted before, in Random Forest this number is usually considerably larger than in XGBoost.
- n_estimators: number of trees that are being built in parallel to achieve the final prediction.

Our intention at first was to also include a minimum number of features and minimum number of samples left in the final node, which are hyperparameters that can help contain overfitting (we actually used a low value for these 2, as recommended to control complexity and avoid overfitting).

4.3 Performance metric: F1-score

It is also important to remark that, as we were aiming to keep the best-performing configuration of models possible, we were assessing each model individually with the F1-score, comparing them only with models within the same target. This metric is particularly useful in cases such as this one where the data is so imbalanced and metrics such as accuracy or AUC can be misleading.

Nevertheless, as we are going to see right now, we also kept a log of how the overall performances were evolving across all the different steps we just explained. We used in this case the mean of F1-score across all targets, calculating it by model type (RF and XGBoost) and overall. This was always taking into account only the best-performing model of each step.

5 Experiment Results

5.1 Performance by step

As we just pointed out, the following table depicts the evolution of the overall performances over the different steps of our modelling procedure:

	Step	Mean F1-score RF	Mean F1-score XGBoost	Mean F1-score Optimal
0	Basic Models	0.8061	0.8283	0.8310
1	Optimum Thresholds	0.8920	0.8842	0.8996
2	Feature Selection	0.9363	0.8925	0.9392
3	Hyperparameter tuning	0.9375	0.9416	0.9416
4	Window size tuning	0.9397	0.9489	0.9489

Step refers to the steps we explained in the previous section.

Mean F1-score RF at each step corresponds to the mean of the F1-score of the top 1 best-performing Random Forest models for each target at the time of that step. ***Mean F1-score XGBoost*** is the same thing but for XGBoost.

Lastly, ***Mean F1-score Optimal*** at each step refers to the mean of the F1-scores of the top 1 performing model (no matter which model type) for each target at that particular step. This last column is what would ultimately

reflect the overall performance of our models because we were planning on selecting the mixture of models that yield the best performance for each target.
We believed this table to be a good idea because it shows the contribution to the final performance of each of the stages our modelling went through. It also allows us to see how each step affected the performance of each model type and overall.
If we analyze the results more closely, we can see that XGBoost models yielded a better performance with the basic models. Both models increased considerably their performances when introducing the new optimal thresholds, because this helped them overcome the data imbalance issue.
Random Forest performance particularly increased in the Feature Selection section while XGBoost did in the hyperparameter tuning stage (as we pointed out before, we were able to better explore the XGBoost hyperparameters because of the speed of the algorithm).
In a general sense, we can see that all of the stages contribute to the increase in the particular performance of each model type and on the overall performance of the appliances classification. We can observe that our Mean F1-score Optimal value after the last stage is the same as the Mean F1-score XGboost, meaning that ALL the optimal XGBoost models for each target performed better than their Random Forest counterparts and the final selection is not a mixture, it is only XGBoost models. Nevertheless, we are going to see in the following section that the optimal Random Forest models for some appliances performed really similarly to their XGBoost counterparts so it was worth trying more than one configuration.

5.2 Performance by individual model

In order to decide which models to keep and which ones to discard as we we were going through each of the different stages, we built a table like the following (we did it for each stage):

target	model	model_id	window_size	f1_score	recall	precision	threshold	eliminated_feature	hyperparam
ac	XGBClassifier	XGBClassifier_ac_10_no_['absdif']_{'objective'...	10.0	0.9971	0.9979	0.9963	0.4901	['absdif']	'bir 'us
ac	RandomForestClassifier	RandomForestClassifier_ac_10_no_None_{'bootstr...	10.0	0.9964	0.9968	0.9960	0.5500	None	{'boo 'ccp
dryer	XGBClassifier	XGBClassifier_dryer_20_no_['dif', 'entropy', '...	20.0	0.9774	0.9679	0.9870	0.2793	['dif', 'entropy', 'absdif']	'bir 'us
dryer	RandomForestClassifier	RandomForestClassifier_dryer_0_no_['dif', 'abs...	0.0	0.9718	0.9753	0.9684	0.2800	['dif', 'absdif']	{'max_ 'n_estin
ev	XGBClassifier	XGBClassifier_ev_35_no_None_{'objective': 'bin...	35.0	0.9946	0.9892	1.0000	0.2009	None	'bir 'us
ev	RandomForestClassifier	RandomForestClassifier_ev_40_no_['dif', 'absdi...	40.0	0.9779	0.9667	0.9893	0.2900	['dif', 'absdif']	{'boo 'ccp
oven	XGBClassifier	XGBClassifier_oven_10_no_['dif']_{'objective':...	10.0	0.9439	0.9396	0.9482	0.3672	['dif']	'bir 'us
oven	RandomForestClassifier	RandomForestClassifier_oven_0_no_['dif', 'absd...	0.0	0.9210	0.9117	0.9304	0.3600	['dif', 'absdif']	
wash	XGBClassifier	XGBClassifier_wash_0_no_['dif', 'entropy', 'ab...	0.0	0.8316	0.8146	0.8493	0.2322	['dif', 'entropy', 'absdif']	{'max 'min_chil
wash	RandomForestClassifier	RandomForestClassifier_wash_0_no_['dif', 'absd...	0.0	0.8315	0.9054	0.7688	0.2000	['dif', 'absdif']	{'max_ 'n_estin

This table reflects the best-performing models for each model type and target after all the modelling stages were completed. We can observe how all the information detailed in the Experiment Setup section is present on this table: model type, optimal window, threshold, eliminated features and optimal hyperparameters, each of them carefully selected for each model type and each target.
This data is also showing us that XGBoost ended up outperforming Random Forest in all the appliances, and that is why the Mean F1-score Optimal in the performance evolution table matched the Mean F1-score XGboost. Nevertheless, targets such as wash or dryer have really similar performance in both model types, so that is why we decided to build 2 configurations to make the final prediction.

5.3 Final Model Configurations

Configuration 1: We kept all the best-performing models detailed in the previous table (being all XGBoost, as we already explained), trained them with the whole data and used them to make predictions on the test set.

Configuration 2: We switched wash and dryer XGBoost best-performing models by their Random Forest counterparts that were performing really similarly (only very slightly lower performances).

Final Configuration: We submitted both configurations to Kaggle and Configuration 2 yielded slightly better results so that was the one we finally kept.
Final configuration: XGBoost for ac, ev and oven; Random Forest for wash and dryer.

6 Conclusion

The final performances of both types of classifiers were very similar. We actually achieved the best overall performance by mixing the Random Forest and XGBoost models that maximized our objective metric in each of the targets. We believe that it is then important to consider several alternatives while modelling mainly when we are dealing with multiple targets in which the best overall performance can be achieved using a mixture.
The new threshold calculation became a key factor when increasing the performance of our models. The data was severely imbalanced, and the calculation of these lower new thresholds that maximized the F1-score for each target allowed our models to predict more positive classes, which is always the difficult part while dealing with imbalance.
Regarding the features used for each classifier we believe that it was really important to assess each target and each model type individually. As we could see from the results, each particular model yielded different features configurations (meaning that the removal of features that improved the F1-score was not the same across models), as well as other parameters such as window size or even the hyperparameters. We individually assessed each configuration looking to maximize the objective metric.
The results also showed us that all the proposed modelling stages were contributing to the increase of the overall performance, with some stages being really relevant for both types of models (such as Threshold Optimization) and others more relevant to one type of model than the other (such as Feature Engineering for Random Forest).
Finally, we learned that it is really important to do previous research and carefully assess which are the types of models that could fit our needs beforehand. If not, you could lose a lot of time by trying models that simply won't work or even have to be dropped in advanced stages of your work because they don't fit a certain requirement of your data/experimental setting.
For instance, in our particular case, we started working with Random Forest and did not realize that the computational load was going to be such a constraint until we tried to tune the hyperparameters. Taking this into account beforehand, knowing the size of our dataset and also knowing that we had 6 targets to classify as independent models, might have led us to the decision of not considering this algorithm, because we ended up not being able to properly explore its hyperparameters. XGBoost on the other hand is really recommendable if running time becomes a constraint, but you will have to know that it will require more knowledge to tune the hyperparameters.
Bottomline: it is recommended that you spend some time to previously understand the task requirements and priorities and select models accordingly.

References

Aprilliant, A., (2021). Optimal Threshold for imbalanced classification. Retrieved from https://towardsdatascience.com/optimal-threshold-for-imbalanced-classification-5884e870c293

Hart, G. (1992). Nonintrusive appliance load monitoring. Proceedings Of The IEEE, 80(12), 1870-1891. doi: 10.1109/5.192069

Khan, S., Latif, A., & Sohaib, S. (2018). Low-cost real-time non-intrusive appliance identification and controlling through machine learning algorithm. 2018 International Symposium On Consumer Technologies (ISCT). doi: 10.1109/isce.2018.8408911

Morder, V., (2019). XGBoost Algorithm: Long May She Reign! Retrieved from https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d#:~:text=XGBoost%20is%20a%20decision%2Dtree,all%20other%20algorithms%20or%20frameworks.

Nagpal, A., (2017). Decision Tree Ensembles - Bagging and Boosting. Retrieved from https://towardsdatascience.com/decision-tree-ensembles-bagging-and-boosting-266a8ba60fd9

Nikulski, J., (2020). The Ultimate Guide to AdaBoost, random forests and XGBoost. Retrieved from https://towardsdatascience.com/the-ultimate-guide-to-adaboost-random-forests-and-xgboost-7f9327061c4f

Olsson, C., & Hurtig, D. (2019). An approach to evaluate machine learning algorithms for appliance classification.

Pereira, L., & Nunes, N. (2018). Performance evaluation in non-intrusive load monitoring: Datasets, metrics, and tools—A review. Wires Data Mining And Knowledge Discovery, 8(6). doi: 10.1002/widm.1265