**Machine Learning Project Documentation**

**Model Refinement**

**1. Overview**

The model refinement phase is crucial for enhancing a machine learning model's performance. It involves evaluating the **initial model**, **adjusting hyperparameters**, **refining features**, and employing regularization techniques. This iterative process aims to optimize the model's architecture, improve generalization, and address weaknesses. The use of ensemble methods and transfer learning can further enhance accuracy and robustness. The refinement phase ensures that the model meets desired criteria.

**2. Model Evaluation**

In our precedent phase, we have exposed 3 differents models with their performances. Classification Test Accuracy got 97.79% by using **XGBClassifier()**



However we remarks that with our deep learning model we can modify and increase accuracyAccuracy per the confusion matrix: 97.85%

**3. Refinement Techniques**

Describe the techniques used for refining the model. This may include adjusting hyperparameters, trying different algorithms, or incorporating ensemble methods.

We shall try with XGBClassifier() some refinements techniques.

## **Tree-specific hyperparameters control the construction and complexity of the decision trees:**

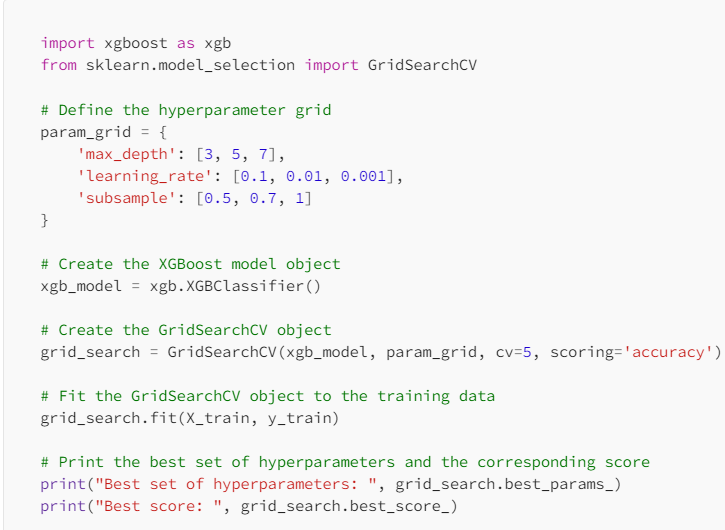
* max\_depth: maximum depth of a tree. Deeper trees can capture more complex patterns in the data, but may also lead to overfitting.
* min\_child\_weight: minimum sum of instance weight (hessian) needed in a child. This can be used to control the complexity of the decision tree by preventing the creation of too small leaves.
* subsample: percentage of rows used for each tree construction. Lowering this value can prevent overfitting by training on a smaller subset of the data.
* colsample\_bytree: percentage of columns used for each tree construction. Lowering this value can prevent overfitting by training on a subset of the features.

## **Learning task-specific hyperparameters control the overall behavior of the model and the learning process:**

* eta (also known as learning rate): step size shrinkage used in updates to prevent overfitting. Lower values make the model more robust by taking smaller steps.
* gamma: minimum loss reduction required to make a further partition on a leaf node of the tree. Higher values increase the regularization.
* lambda: L2 regularization term on weights. Higher values increase the regularization.
* alpha: L1 regularization term on weights. Higher values increase the regularization.

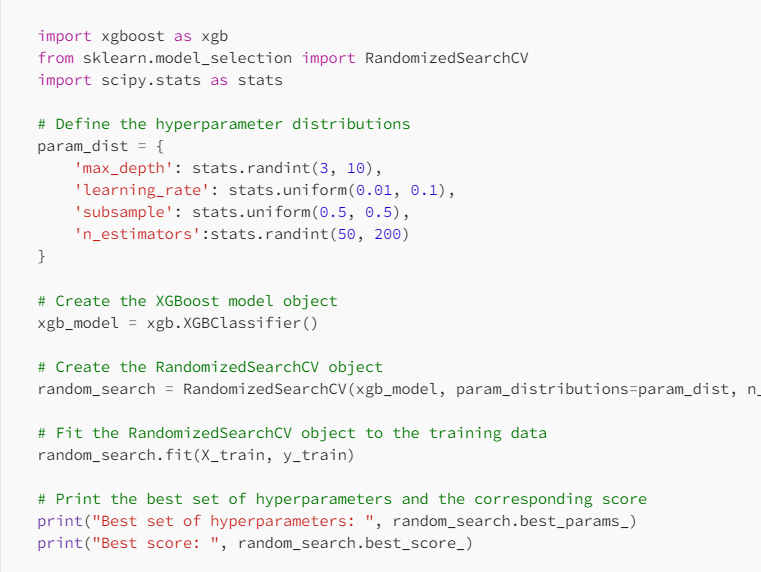
1. *Grid search*

It involves specifying a set of possible values for each hyperparameter, and then training and evaluating the model for each combination of hyperparameter values. This a sample of how to use that method



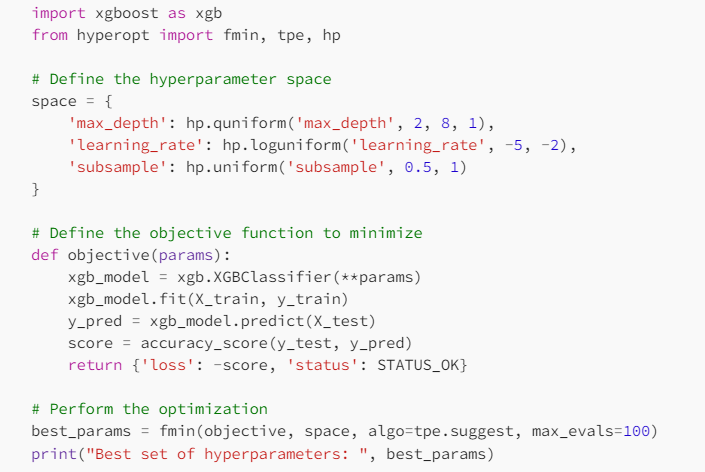
1. *Random search*

*Random search* is a variation of grid search that randomly samples from the set of possible hyperparameter values instead of trying all combinations. This can be more efficient than grid search because it does not need to evaluate all possible combinations. However, it can still be computationally expensive, especially when the number of hyperparameters and possible values is large.



1. *Bayesian optimization*

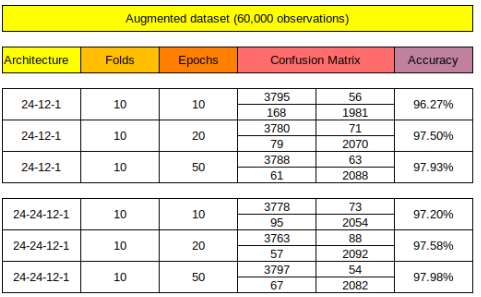
It tries to find the optimal set of hyperparameters by making smart guesses based on the previous results. Bayesian optimization is more efficient than grid or random search because it attempts to balance exploration and exploitation of the search space. It can also deal with the cases of large number of hyperparameters and large search space. However, it can be more difficult to implement than grid search or random search and may require more computational resources.



**4. Hyperparameter Tuning**

From research on Kaggle (https://www.kaggle.com/code/mineshjethva/power-grid-stability-with-deep-learning/notebook), we saw that by changing hyparameter values, the performance of the deep learning model can increase.

Here is the table:



**5. Cross-Validation**

Discuss any changes made to the cross-validation strategy during model refinement and explain the reasoning behind those changes.

1. **Feature Selection**

We kept the same feature selection from our dataset. As expected, more complex ANN architectures **performed better** than simpler ones. An **increased number of epochs** considered during fitting also plays a major role. It is evident that the more the model is exposed to the training set, the better the prediction accuracy. From a machine learning exercise perspective, the use of an **augmented dataset** with 6,000 observations contributed significantly to better results.

**Test Submission**

**1. Overview**

The test submission phase evaluates a pre-trained machine learning model on a diverse test dataset. This involves loading the model, configuring it for inference, and running it on the test data to assess performance. Results are analyzed, and the model may be fine-tuned based on the evaluation. Documentation captures insights, issues, and potential improvements. If successful, the model can be deployed, and ongoing monitoring ensures continued effectiveness in a real-world setting.

**2. Data Preparation for Testing**

We select all rows and the first 12 columns of the DataFrame data and assigns it to the variable X. This typically represents the feature matrix, containing the input features for the machine learning model. Then we select all rows and the 13th column of the DataFrame data and assigns it to the variable y. This typically represents the target variable we will try to predict.

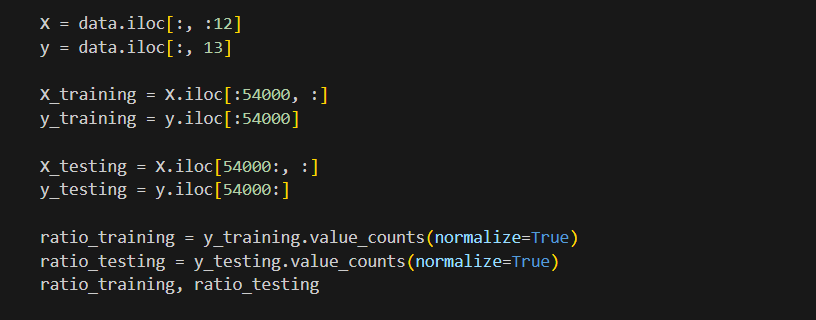
X\_training = X.iloc[:54000, :] and y\_training = y.iloc[:54000]:

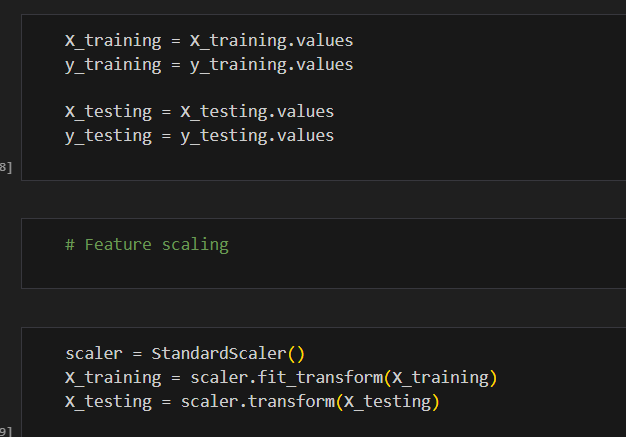
Training sets (X\_training and y\_training) by selecting the first 54,000 rows from the feature matrix X and the corresponding labels y. This is a common way to partition data for training a machine learning model.

X\_testing = X.iloc[54000:, :] and y\_testing = y.iloc[54000:]:

Here we create testing sets (X\_testing and y\_testing) by selecting the rows from index 54,000 to the end from the feature matrix X and the corresponding labels , 6000 values. This is the portion of the data that will be used to evaluate the model's performance.

The resulting ratios can be useful for assessing the class distribution and potential class imbalances in the training and testing data.





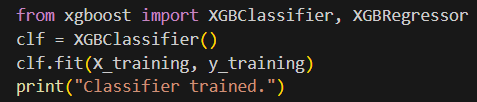
We did not forget to Standar our X\_training and X\_testing before to use in the model.

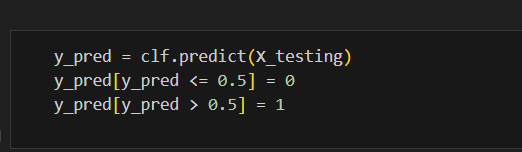
**3. Model Application**

Describe how the trained model was applied to the test dataset. Include code snippets if applicable.

For our project, we mainly used these two models: a classificartion and a deep learning model based on ANN layer model, Sequentiel model.

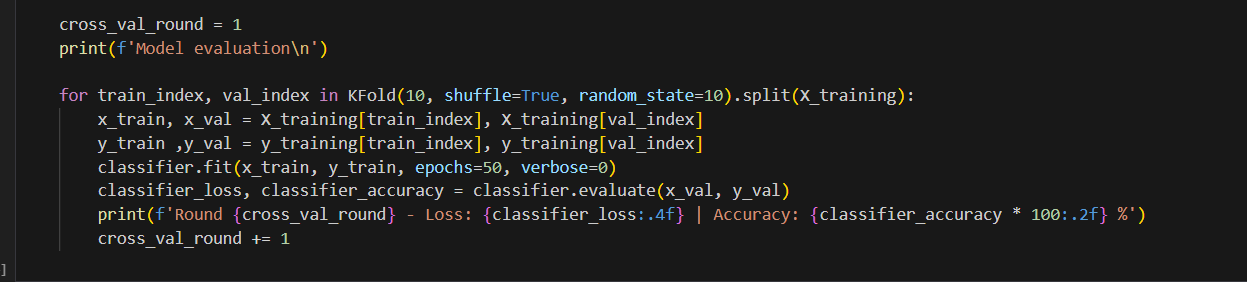
After initialisation of models, we just used it fit() function on X\_trainaing and y\_training for the classification model. Our deep learning model,

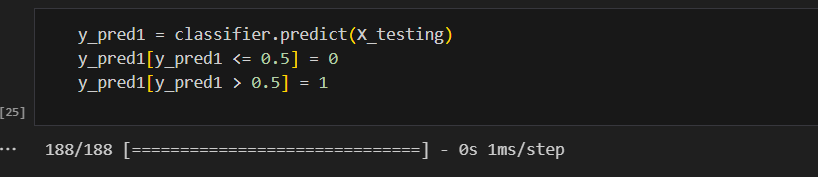




This code implements k-fold cross-validation to assess the performance of a machine learning model. The variable `cross\_val\_round` is initialized to track the cross-validation rounds. The code then iterates through 10 folds, shuffling the data for each iteration. For each fold, the training and validation sets are separated, and the model is trained on the former. The loss and accuracy are evaluated on the validation set, and the results are printed, including the round number, loss, and accuracy. This process is repeated for each fold, providing a more robust evaluation of the model's generalization performance. The use of k-fold cross-validation helps ensure that the model's performance is representative across different subsets of the training data, enhancing its reliability for deployment in real-world scenarios.

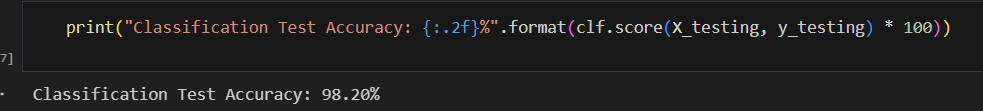


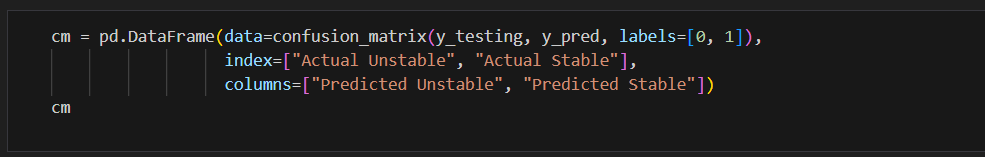


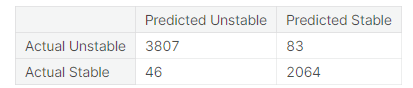


1. **Test Metrics**

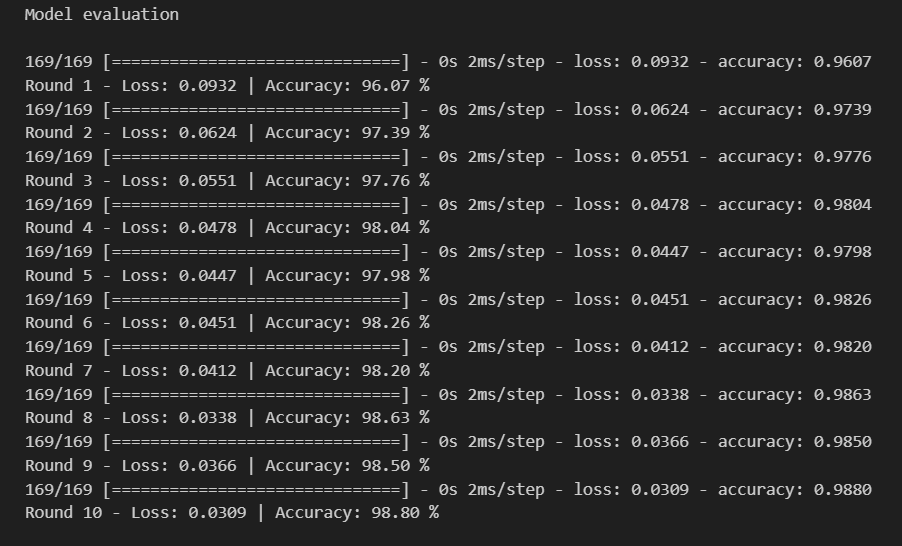
**Classification Model**

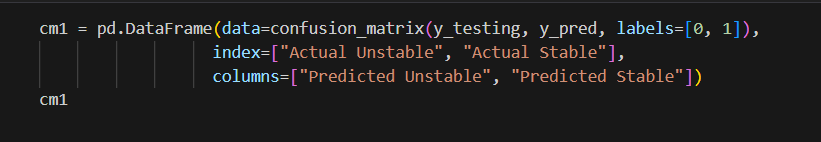


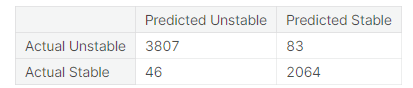




**Deep Learning Model**







**5. Model Deployment**

Deep learning proved to be an outstanding prediction tool for this particular application. Even considering that the dataset is well behaved and needed no significant preprocessing, the **high accuracies** obtained on the testing set confirm that a deep learning model may be safely considered. It would though be up to a smart grid operator to confirm if the accuracy level obtained with deep learning would suffice in practical terms (live network)

1. **Code Implementation**

We implement code and explanation in above sections

**Conclusion**

Summarize the outcomes of the model refinement and test submission phases. Highlight any challenges encountered and the final performance achieved.

**References**

All references have been precised above.