Decentralized Smart Grid Stability Prediction

Mohammad Alikhani Najafabadi
Pattern Recognition and Machine Learning
Universitat Politecnica de Catalunya
Barcelona, Spain
Mohammad.najafabadi@estudiantat.upc.edu

***Abstract*—This paper presents an approach to predict stability in decentralized smart grids using machine learning and deep learning algorithms, focusing on the Decentral Smart Grid Control (DSGC) system. The study leverages a publicly available dataset, augmented from 10,000 to 60,000 samples, derived from controlled simulations of a 4-node star architecture. Key features include reaction times, power balances, and price elasticities, with grid frequency serving as the primary indicator of stability. The dataset is preprocessed and split into training (54,000) and testing (6,000) sets, followed by feature scaling using StandardScaler. Various classifiers—Support Vector Machine (SVM), Gradient Boosting Machine (GBM), and Artificial Neural Networks (ANN)—are evaluated, with hyperparameter tuning via randomized search and cross-validation. Results demonstrate high accuracy, with SVM achieving up to 94% with an RBF kernel, GBM reaching 93.6% with optimized tree depth and learning rate, and ANN attaining 94.4% with a deep architecture. The findings highlight the potential of these models to enhance smart grid stability prediction, addressing challenges posed by bidirectional energy flows and prosumer dynamics.**

***Index Terms*—Decentralized Smart Grid Control (DSGC), grid stability prediction, machine learning, deep learning, Support Vector Machine (SVM), Gradient Boosting Machine (GBM), Artificial Neural Networks (ANN)**

I. Introduction

A. Renewable Energy Sources and Smart Grids

The ascent of renewable energy sources provides the global community with a much demanded alternative to traditional, finite and climate-unfriendly fossil fuels. However, their adoption poses a set of new paradigms, out of which two interrelated aspects deserve particular attention:

1. Prior to the rise of renewable energy sources, the traditional operating ecosystem involved few production entities (sources) supplying energy to consumers over unidirectional flows. With the advent of renewable options, end users (households and enterprises) now not only consume energy but have the ability to produce and supply it, hence a new term has arised to designate them, 'prosumers'. As a result, energy flow within distribution grids 'smart grids' has become bidirectional;

2. Despite the increased flexibility brought in by the introduction of renewable sources and the aforementioned emergence of 'prosumers', the management of supply and demand in a more complex generation / distribution / consumption environment and the related economic implications (particularly the decision to buy energy at a given price or not) have become even more challenging.

Relevant contributions on how to tackle the requirements of such new scenario have been offered by academy and industry over the past years. Special attention has been devoted to the study of smart grid stability.

B. Modeling grid stability

In a smart grid, consumer demand information is collected, centrally evaluated against current supply conditions and the resulting proposed price information is sent back to customers for them to decide about usage. As the whole process is time-dependent, dinamically estimating grid stability becomes not only a concern but a major requirement.

Put simply, the objective is to understand and plan for both energy production and/or consumption disturbances and fluctuations introduced by system participants in a dynamic way, taking into consideration not only technical aspects but also how participants respond to changes in the associated economic aspects (energy price).

The work of researchers cited in foreword focuses on Decentral Smart Grid Control (DSGC) systems, a methodology strictly tied to monitoring one particular property of the grid - its frequency. The term 'frequency' refers to the alternate current (AC) frequency, measured in cycles per second or its equivalent unit, Hertz (Hz). Around the world AC frequencies of either 50 or 60 Hz are utilized in electric power generation-distribution systems.

It is known that the electrical signal frequency "increases in times of excess generation, while it decreases in times of underproduction" [1]. Therefore, measuring the grid frequency at the premise of each customer would suffice to provide the network administrator with all required information about the current network power balance, so that it can price its energy offering - and inform consumers - accordingly.

The DSGC differential equation-based mathematical model described in [1] and assessed in [2] aims at identifying grid instability for a reference 4-node star architecture, comprising one power source (a centralized generation node) supplying energy to three consumption nodes. The model takes into consideration inputs (features) related to:
1. The total power balance at each grid node, considering both production and consumption.
2. Participant reaction times to adjust their consumption and/or production in response to price changes.("reaction time")
3. Energy price elasticity, which describes the relationship between consumer demand and energy prices

In this way, the DSGC framework supports precise predictions of grid instability, enabling network administrators to implement prompt corrective measures to prevent power outages. This mathematical approach is adaptable for analyzing intricate power systems, promoting efficient and dependable smart grid operations [3].

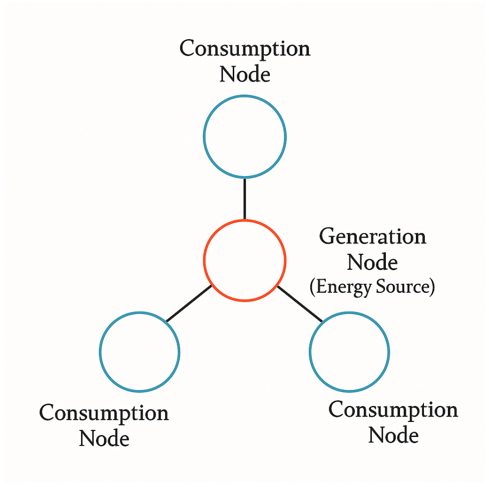

Fig. 1. Four-node star architecture.

C. DSGC mathematical model

The DSGC mathematical model is designed to predict grid stability and manage power dynamics in smart grids, modeling producers and consumers as a network of synchronous machines represented by oscillators [4]. In the following, the key components of the model are described, divided into simulation of energy production and consumption, and modeling changes in electrical energy price relative to the frequency of the grid [4].

1) Energy Dynamics and Stability:

- The energy dynamics of a synchronous machine are governed by the equation:

$$p^{\text{source}} = p^{\text{accumulated}} + p^{\text{loss}} + p^{\text{consumed}} (eq.1),$$

where p^{source} represents total power, $p^{\text{accumulated}}$ is stored energy, p^{loss} is power lost during transfer, and p^{consumed} is power used by the client.
- The power source is further detailed as:

$$p^{\text{source}} = \frac{1}{2} M_j \frac{d}{dt} (\delta'_j)^2 + \kappa_j \sum_{k=1}^{N} K_{jk}^{\max} \sin(\delta_k - \delta_j) (eq.2),$$

with M_j as the moment of inertia, δ'_j as the rotor angle velocity, κ_j as the friction coefficient, $K_{jk}^{\max}$ as the maximum transmittable power, and δ_j, δ_k as rotor angles of machines j and k.

2) Frequency and Phase Dynamics:

- Grid frequency deviation is defined as $\omega_j = \frac{d}{dt}\delta_j$ from a reference grid rotating at $\Omega = 2\pi \cdot 50\,\text{Hz}$[5], with the phase given by $\delta_j(t) = \omega t + \theta_j(t)$, where ω is the grid frequency and $\theta_j(t)$ is the relative rotor angle.
- The phase dynamics follow the second-order differential equation:

$$\frac{d^2\delta_j}{dt^2} = P_j - \alpha_j \frac{d\delta_j}{dt} \sum_{k=1}^{N} K_{jk} \sin(\delta_k - \delta_j) (eq.3),$$

where P_j is mechanical power (positive for production, negative for consumption), $\alpha_j = \frac{2\kappa_j}{M_j}$ is the damping factor, and K_{jk} is the coupling strength.

3) Time Delay and Price Elasticity:

- Incorporating time delay τ, the equation becomes:

$$\frac{d^2\delta_j}{dt^2} = P_j - \alpha_j \frac{d\delta_j}{dt} \sum_{k=1}^{N} K_{jk} \sin(\theta_k - \theta_j)$$
$$- \gamma_j \frac{d}{dt} (\theta_j(t - \tau_j) - \theta_j(t - \tau_j - T_j)) (eq.4),$$

where γ_j represents price elasticity, τ_j is the reaction time, and T_j is the averaging time.
- Mechanical power is modeled as $P_j = p_0 - c_1 \frac{d\delta_j}{dt}$[5], with p_0 as base power and c_1 as the price elasticity coefficient. Energy use is linearly correlated with price:

$$\hat{P}_j(p_j) \approx p_j + c_j(p_j - p_0) (eq.5),$$

where c_j is a proportionality factor.

4) Input Parameters and Stability Analysis:

- Key parameters include coupling strength K_{jk}, damping factor α_j, reaction time τ_j, price elasticity γ_j, and mechanical power P_j, with chosen values: $K_{jk} = 8\,\text{s}^{-2}$, $\alpha_j = 0.1\,\text{s}^{-1}$, $\tau_j = [0.05, 10]\,\text{s}^{-1}$, $\gamma_j = [0.05, 1]\,\text{s}^{-1}$, and $P_j = [-0.5, -2]$.
- Stability is evaluated using linear stability analysis, where a negative maximum eigenvalue indicates a stable system [5].

D. simplifications problem in the model

So far we have discussed a mathematical model with which grid instability can be predicted, The need of a tool to predict grid instability would have been met, and the binary classification (”stable” versus ”unstable”) problem would be solved. However, the execution of this model relies on significant simplifications. A differential equation-based model can be manipulated in several ways. One traditional approach consists in running simulations with a combination of fixed values for one subset of variables and fixed value distributions for the remaining subset. this strategy leads to two primary issues, referred to as the ”fixed inputs issue” and the ”equality issue”.

II. State of Art

Innovative solutions have been developed to tackle the limitations of the Decentral Smart Grid Control (DSGC) model, focusing on enhancing stability in smart grids. Venayagamoorthy [1] proposed situational awareness (SA) as a key concept for grid stability, describing it as the ability to perceive environmental factors within a defined time and space, essential for secure and efficient smart grid operations.

Merely collecting more data does not resolve uncertainty; the focus must be on the accuracy of sensed information. Intelligent systems rely on sense-making agents, decision-making agents, and adaptation agents, all underpinned by a central knowledge base. In complex, time-sensitive scenarios, intelligent sense-making is critical, with techniques such as neural networks, fuzzy logic, swarm intelligence, and adaptive critic designs showing promise. Fuzzy logic-based approaches are particularly effective for evaluating network stability. Wide Area Monitoring (WAM) is vital, especially with increasing grid interconnections and plug-and-play components functioning as sinks or sources. Advanced monitoring systems, incorporating learning mechanisms like cellular neural networks and stochastic identifiers, are essential for managing these dynamic systems.

Arzamasov et al. [2] devised a novel demand response framework that requires minimal alterations to existing infrastructure and avoids heavy data processing by linking electricity pricing to grid frequency. This approach assumes conditions like the fixed inputs issue and the equality issue. To mitigate these, the system was tested across various design points, with decision trees used to interpret the outcomes. Large datasets were simplified by aggregating inputs into features. Findings indicate that quick adaptation generally bolsters stability, but delays beyond 8 seconds lead to persistent instability. In stable grids, a consumer with a reaction time exceeding 8 seconds can still support stability if paired with a fast-reacting consumer, provided the average reaction time remains moderate. Slow-reacting consumers were found advantageous in such cases. The study reveals a balance between avoiding the rebound effect with short reaction times and expanding the stability range with longer ones. Higher averaging times positively influence stability, while power levels have negligible impact. Nonetheless, the model needs further generalization, achieving 80 % accuracy, which suffices for a general perspective but falls short for precisely defining stability regions. Scaling this analysis to grids with more than 10 users remains complex.

Networked Control Systems (NCS) in smart grids encounter issues like packet dropout, delays, and packet disorder, which impair performance. Conventional power system studies often assume ideal, lossless signal transmission without relays. Singh et al. [6] explored these challenges in Networked Controlled Smart Grids (NCSG), evaluating both UDP and TCP protocols. Although UDP-like protocols provide suboptimal results, they are favored over TCP-like protocols due to the difficulties in analyzing and implementing TCP-based control schemes [7]. Packets are deemed lost if delays exceed the sampling interval. Without packet loss, the control loop remains stable and damped, but packet loss results in missing output measurements, lowering controller accuracy. A case study utilized a 16-machine, 68-bus model, a simplified version of the interconnected New England Test System (NETS) and New York Power System (NYPS), with a 0.1-second sampling period. NCS performance with minimal packet dropout, even at marginal packet delivery probability (MPDP), matches classical control in ideal conditions. In ideal scenarios, NCS surpasses classical systems, damping oscillations more quickly with 22% less controller effort. NCS also performs better under realistic packet delivery conditions. Higher sampling rates enhance the required packet delivery rate, and NCSG maintains stability at a 0.85 delivery probability, highlighting its potential to improve oscillatory stability margins in smart grids.

III. Proposed approach

The main goal of this paper is to determine the ability of Machine Learning and Deep Learning algorithms to develop models for stability prediction with DSGC. For this purpose, various types of algorithms were used, such as support vector machine, Gradient Boosting Machine, and Artificial Neural networks (ANN).

To obtain a result with a high degree of accuracy, a randomized hyperparameter search with cross-validation was used. In the first step, the values of the hyperparameters are chosen at random, and the model is validated using five-fold cross-validation. In the process of cross-validation, the dataset is split into a k-number of subsets (in the case of five-fold cross-validation, k = 5). Then, each subset is used as a testing set, while the remaining subsets are used as a training dataset. In the end, a set of hyperparameters that achieved the highest average score for a certain model is then repeated in the same.

A. Dataset

Our work here relies on a publicly accessible dataset, developed by Arzamasov et al. (2018) [2] conducted 10,000 simulations of the DSGC system, since the grid used in simulations is symmetric we can augment the original dataset 3! (6) times.The final augmented dataset consists of 60.000 samples, 12 primary predictive features and one dependent variable:

- 'tau1' to 'tau4': reaction times for each network participant (supplier node: 'tau1', consumer nodes: 'tau2' through 'tau4'), with ranges from 0.5 to 10.
- 'p1' to 'p4': nominal power production or consumptio values (consumers: negative values, producer: positive value), within the range of -2.0 to -0.5 for consumers and an opposite value for the supplier node ('p1' equals the negative sum of 'p2', 'p3', and 'p4').
- 'g1' to 'g4': price elasticity coefficients, with values ranging between 0.05 and 1.00 for each network participant (supplier node: 'g1', consumers: 'g2' through 'g4')
- 'stabf': a binary categorical label ('stable' or 'unstable') based on the maximum real part of the characteristic differential equation root, which is positive for an unstable system and negative for a stable one.

The dataset originates from carefully controlled simulations, guaranteeing full data integrity with no missing entries and preserving its numerical structure [2]. This characteristic allows for a seamless shift to machine learning modeling, eliminating the need for preprocessing or feature engineering.

TABLE I

TABLE CONTAINING STATISTICAL DESCRIPTION OF THE DATASET USED.

	Mean	Std	Min	Max
Input values				
τ_1	5.25	2.74	0.50	10.00
τ_2	5.25	2.74	0.50	10.00
τ_3	5.25	2.74	0.50	10.00
τ_4	5.25	2.74	0.50	10.00
p_1	3.75	0.75	1.58	5.86
p_2	-1.25	0.34	-2.00	-0.50
p_3	-1.25	0.34	-2.00	-0.50
p_4	-1.25	0.34	-2.00	-0.50
g_1	0.52	0.27	0.05	1.00
g_2	0.53	0.27	0.05	1.00
g_3	0.53	0.27	0.05	1.00
g_4	0.53	0.27	0.05	0.11
Output values				
stab	0.02	0.04	-0.08	0.11
$stab_f$	*stable 36%*		*unstable 64%*	

B. EDA (Exploratory Data Analysis)

A critical initial step for successful dataset preprocessing involves a thorough examination of the dataset, specifically assessing its structure, identifying the various data types it contains, and determining the best strategies for data cleansing to highlight the most relevant information.

1) Feature assessment: The evaluation explored the connections between each of the twelve dataset features and the target variable 'stab'. Most features exhibited uniform distributions, a result of predefined fixed ranges in the simulations. However, the 'p1' feature, calculated as the absolute sum of 'p2', 'p3', and 'p4', showed a normal distribution with a slight negative skew factor (-0.013). This analysis also indicated that approximately 63.8% of the observations were categorized as 'unstable', while about 36.2% were labeled as 'stable', as detailed in Table 1.

2) Correlation Analysis: The examination identified a significant negative correlation (-0.83) between the target variable 'stabf' and the twelve numerical features, suggesting that higher feature values are associated with unstable conditions. Moreover, above-average correlations were noted between 'p1' and its components 'p2', 'p3', and 'p4'. Despite these correlations, the values were not high enough to justify feature removal, as each feature provides unique insights crucial for understanding the dataset's structure.

3) Preparing the Training and Testing Sets: The initial dataset was divided into a training set (comprising the first 54,000 observations) and a testing set (encompassing the last 6,000 observations). The target variable was set as 'stabf', with 12 numerical features acting as predictors, while 'stab' was excluded due to its strong correlation. The split preserved the distribution of stable (1) and unstable (0) observations, with roughly 63.94% and 62.55% in the training and testing sets, respectively. Subsequently, the Pandas dataframes were transformed into NumPy arrays for additional processing

4) Feature Scaling: This stage utilized the StandardScaler from the "scikit-learn" library, fitting the scaler to the training set and applying it to both the training and testing sets using the 'transform' method. This approach ensured that all features contributed equally to the machine learning model development, irrespective of their scales or units.

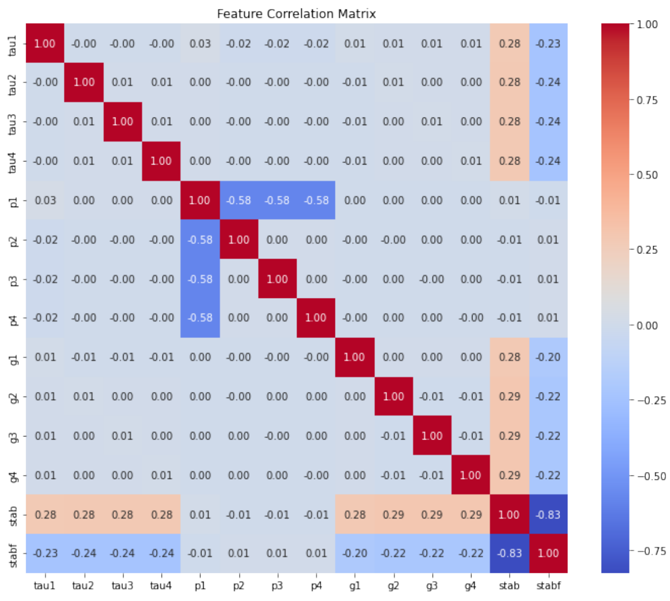

Fig. 2. Correlation matrix fordataset attributes.

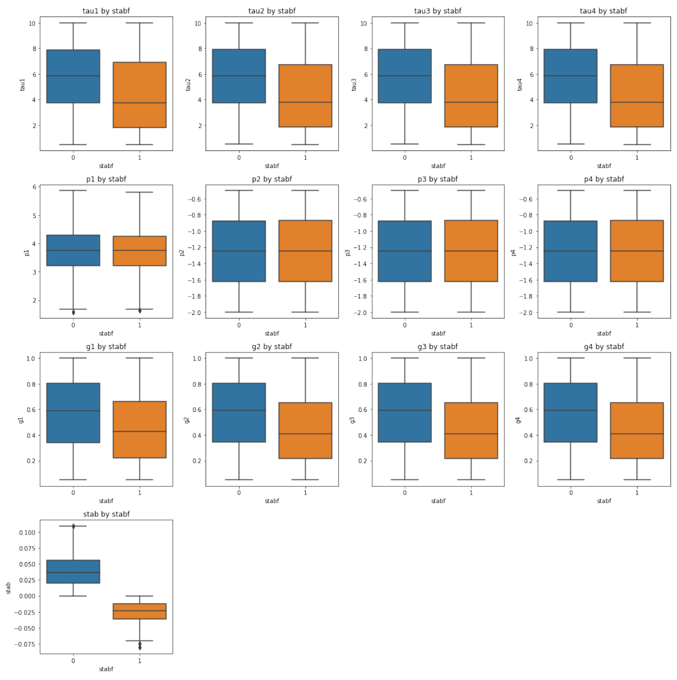

Fig. 3. Box-and-whisker plots of model parameters and the stability index stratified by stability flag (stabf = 0 vs. 1).

IV. EXPERIMENTATION

Following the division of the dataset into training and testing segments, along with the completion of feature preparation and scaling, a variety of classifiers can now be implemented on the prepared dataset.

A. *support vector machine*

The support vector machine (SVM) serves as a supervised learning technique applicable to classification, regression, and outlier detection tasks. It operates by positioning input vectors within a high-dimensional feature space Z, where a decision surface is computed to achieve an effective clustering of these vectors. The optimal hyperplane, identified as a decision function that maximizes the separation margin between two or more groups of vectors, is then determined. To establish this hyperplane, only a select portion of the training data—those points defining the hyperplane's margins, known as support vectors—are considered, as the SVM focuses solely on these critical data points during modeling, disregarding those beyond the margin. Likewise, in support vector regression, a similar subset of data is utilized, excluding training points near their target values [8]. The SVM implementations employed include "SVR" for regression tasks and "SVC" for classification challenges. The hyperparameters evaluated, along with their respective ranges, are presented in Table 2.

TABLE II
USED HYPERPARAMETERS FOR SVC MODELS

Hyperparameter	Range of Values
kernel	'linear', 'poly', 'rbf'
C	[0.1, 1, 10]
degree	[2, 3, 4, 5]
gamma	[0.1, 1, 10]

The hyperparameter 'kernel' specifies the kernel function used to compute the optimal hyperplane; for instance, selecting 'linear' as the kernel results in the hyperplane being derived via a linear function. The 'degree' hyperparameter sets the degree of the polynomial function and is paired with the 'poly' kernel. Meanwhile, 'gamma' dictates the influence of individual training examples, where a higher 'gamma' value indicates that points must be closely positioned to be grouped together. The 'C' hyperparameter acts as a balance between the misclassification of training examples and the simplicity of the decision surface [9].

B. *Gradient Boosting Machine*

The Gradient Boosting Machine (GBM) is an AI-driven technique for addressing regression and classification challenges by integrating multiple decision trees into a unified model. These trees are constructed sequentially, with each subsequent tree correcting the errors of its predecessors, allowing the GBM to learn from the residuals of prior predictions. After a new decision tree is added, the model's updated prediction is calculated as:

$$y_i - f(x_i) \rightarrow \hat{y}_i (eq.6)$$

where y_i represents the target value, $f(x_i)$ is the predicted value, and $\hat{y}_i$ denotes the residual. This residual indicates how far the prediction deviates from the target, providing feedback to refine the model. The updated model prediction is expressed as:

$$f = f_0 + \alpha f_1 (eq.7)$$

where α is the learning rate hyperparameter. The residual is then recalculated based on the new decision tree f_2, which aims to correct the errors made by tree f_1:

$$f = f_0 + \alpha f_1 + \alpha f_2 (eq.8)$$

This iterative process continues until a predefined maximum number of trees is reached. The construction of trees can also be influenced by their depth, determined by the tree's hyperparameters. Greater depth increases training and prediction times but enhances model accuracy [10]. In this study, the GBM implementations used are 'XGBRegressor' for regression tasks and 'XGBClassifier' for classification tasks, with their hyperparameters and ranges detailed in Table 3.

This iterative process continues until a predefined maximum number of trees is reached. The construction of trees can also be influenced by their depth, determined by the tree's hyperparameters. Greater depth increases training and prediction times but enhances model accuracy[8]. In this study, the GBM implementations used are ”XGBRegressor” for regression tasks and XGBClassifier for classification tasks, with their hyperparameters and ranges detailed in Table2.

The hyperparameter ”n_estimators” defines the number of boosting rounds or trees added sequentially during training. Increasing this value can lead to better performance but also raises the risk of overfitting. The hyperparameter ”learning_rate” (also known as ”eta”) controls the step size shrinkage after each boosting iteration. Lower values of ”eta” slow down learning by reducing feature weights, thereby making the boosting process more cautious and less prone to overfitting. The ”max_depth” parameter sets the maximum depth allowed for any tree in the model, effectively controlling model complexity; deeper trees can capture more interactions but may overfit. The ”min_child_weight” sets the minimum sum of instance weights (Hessian) needed in a child node—higher values make the model more conservative by preventing the tree from learning from very small partitions.

The hyperparameter ”max_delta_step” specifies the maximum weight change allowed in each boosting step, primarily used for logistic regression to help improve stability on imbalanced datasets. The parameters ”colsample_bytree”, ”colsample_bylevel”, and ”colsample_bynode” determine the fraction of input features (columns) that are randomly sampled for each tree, tree level, and node, respectively. These parameters help introduce feature-level randomness, reduce overfitting, and improve generalization. Lastly, ”num_parallel_tree” indicates how many trees are constructed independently and in parallel for each boosting round; this is typically set to 1 in standard gradient boosting but can be increased for techniques like random forests[10].

TABLE III
USED HYPERPARAMETERS FOR XGBCLASSIFIER MODELS.

Hyperparameter	Values
n_estimators	[50, 100, 200]
max_depth	[3, 5, 7]
learning_rate (eta)	[0.01, 0.1, 0.2]
subsample	[0.6, 0.8, 1.0]
colsample_bytree	[0.6, 0.8, 1.0]
min_child_weight	[1, 3, 5]
max_delta_step	[0, 5, 10]
colsample_bylevel	[0.6, 0.8, 1.0]
colsample_bynode	[0.6, 0.8, 1.0]
num_parallel_tree	[1]

C. *Artificial neural Networks*

In our pipeline, all continuous inputs were standardized by fitting a StandardScaler to the 80% training split and then applying that same transformation to both training and test partitions. This guarantees that the network never "peeks" at the test-set distribution during fitting.

The core classifier is a deep, fully-connected network with one 12-node input layer (for our twelve numeric features), four hidden layers of sizes 288 → 288 → 24 → 12 (each using ReLU activations with a uniform kernel initializer), and a single-unit sigmoid output node. We train against binary cross-entropy loss and monitor accuracy—which, in the binary setting, is by far the most common performance metric in the literature. To optimize weights, we experimented with SGD+momentum, Adam, and Nadam; ultimately, Nadam (the Nesterov-accelerated variant of Adam) delivered the fastest convergence and best held-out accuracy on our data.[11][12]

Because our classes are balanced, we employ stratified 10-fold cross-validation on the training split to guard against overfitting and to tune our choice of optimizer and number of epochs. With 10 folds, we found that training for 50 epochs (batch size 32) was sufficient for the validation accuracy to stabilize—earlier stopping yielded only marginal losses in performance, while longer runs led to slight overfitting. We therefore fixed our final model to 50 epochs under Nadam, retrained on all 10 folds of the training data, and then evaluated once on the held-out 20% test set.

V. RESULTS & CONCLUSION

In, this section we will discuss and compare the obtained results for each of the models.

A. *Support vector machine*

Our cross-validation study of SVM kernels reveals striking differences in how model complexity and parameter choices affect generalization error. For the **linear SVM** (Figure 5), increasing the regularization parameter C from 0.1 to 10 produces only a slight rise in CV error—from 0.1864 at $C = 0.1$ to 0.1869 at $C = 10$. This near-flat trend indicates that even a heavily regularized linear decision boundary (small C) captures most of the exploitable structure in the data, and that granting the classifier more freedom (larger C) yields negligible gains and a mild tendency to overfit.

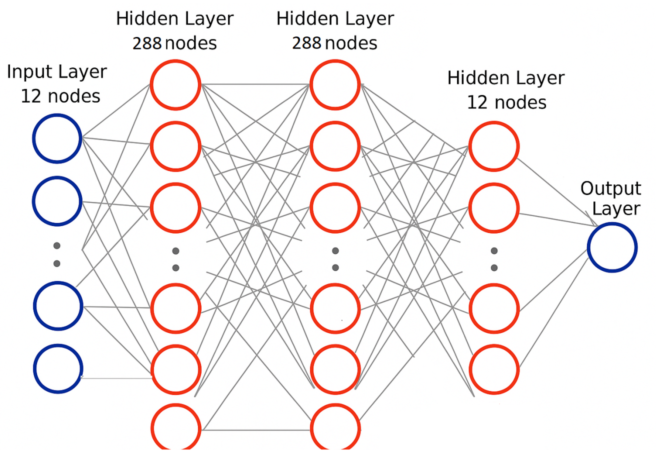

Fig. 4. Neural Network architecture

The **RBF kernel** (Figure 6) shows a much stronger dependence on both C and the kernel width γ. At a moderate $\gamma = 0.1$, increasing C sharply drives CV error down—from 0.19 at $C = 0.1$ to just 0.06 at $C = 10$. This demonstrates that a wider-spread Gaussian (small γ) paired with lower regularization underfits, whereas a narrower kernel (higher γ) coupled with high C can flexibly carve complex decision boundaries. In contrast, larger γ values (1 and 10) remain stuck at error ≈ 0.36 regardless of C, indicating extreme overfitting: the kernel is so narrow that each point acts almost as its own neighbor, and no amount of regularization alleviates this.

For the **polynomial kernel** (Figure 7), error uniformly decreases with both rising polynomial degree and larger C. At $C = 10$, CV error drops from 0.08 at degree 2 to a minimum of 0.055 around degree 4 before a slight uptick at degree 5, suggesting an optimal balance between expressiveness and overfitting. Lower C values follow the same trend but with higher baseline error (e.g., 0.11–0.08 for $C = 1$). Overall, the best polynomial SVM (degree 4, $C = 10$) marginally outperforms the RBF SVM (CV error ≈ 0.06), but at the cost of tuning an additional integer hyperparameter. Taken together, these results recommend an RBF SVM with $\gamma = 0.1$ and $C = 10$ as a strong, parsimonious choice, while a polynomial kernel offers slight further gains if one is willing to optimize degree.

B. *Gradient Boosting Machine*

Our hyperparameter grid search on the XGBoost classifier revealed clear trends in how depth, learning rate, tree count, and subsampling affect cross-validation performance. As shown in the heatmap of **max_depth vs. learning_rate** (Figure 8), shallow trees (depth = 3) struggle at low learning rates ($\approx$78% accuracy), but at $\eta \geqslant 0.1$ all depths converge to 93–94% CV accuracy, with depth = 5–7 slightly outperforming. The **n_estimators vs. subsample** heatmap (Figure 9) further highlights that increasing the number of trees from 50 to 200 steadily boosts CV accuracy by $\sim$5 percentage points,

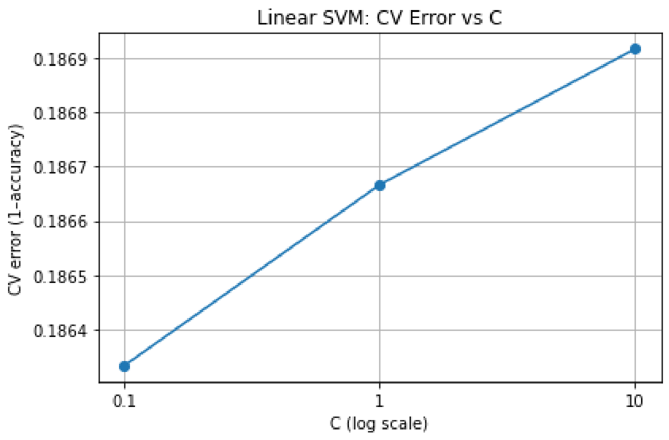

Fig. 5. Linear SVM

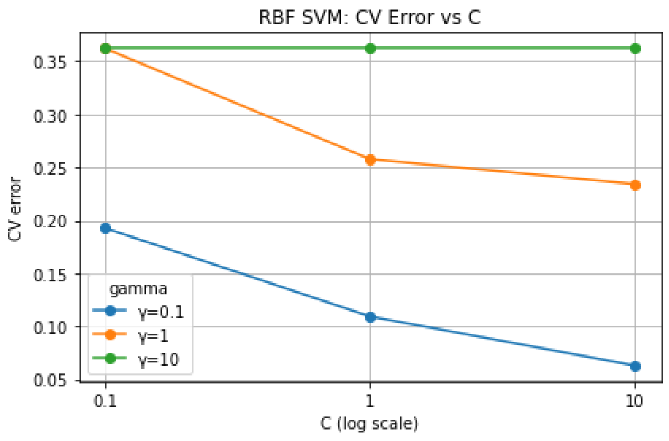

Fig. 6. RBF SVM

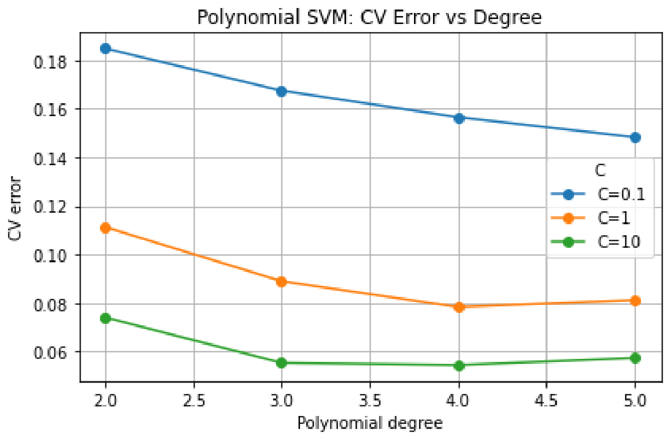

Fig. 7. Polynomial SVM

and that a subsample fraction of 0.6 yields marginally better generalization than using all data (subsample = 1.0).

On the held-out test set, our best-tuned model ($\eta = 0.2$; `max_depth` = 7; `n_estimators` = 200; `subsample` = 0.6) achieved **93.6% accuracy** (Figure 12). The confusion matrix indicates high sensitivity for the **"unstable"** class (96.3% recall) and strong precision for both classes ($> 93\%$), though **"stable"** events were missed at a slightly higher rate (11.2% false negatives). The **ROC** (AUC ≈ 0.96) and **precision–recall curves** corroborate this robust discrimination between stable and unstable grid conditions.

In sum, our visual and quantitative analyses suggest that moderately deep trees, a learning rate around 0.1–0.2, and a modest subsampling strategy strike the optimal bias–variance trade-off for this dataset. The resulting model not only generalizes well ($\approx$93–94% accuracy across CV and test splits) but also provides tunable class-specific performance—allowing practitioners to adjust thresholds or class weights depending on whether false alarms or missed detections carry greater operational cost.

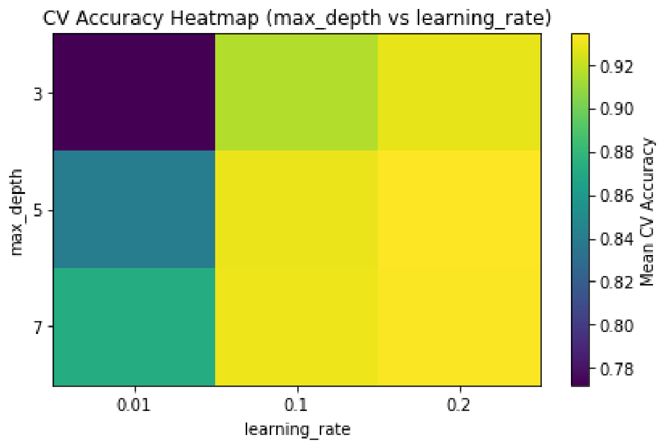

Fig. 8. Accuracy Heat-Map for XGboost Model, Maxdepth vs Learning rate

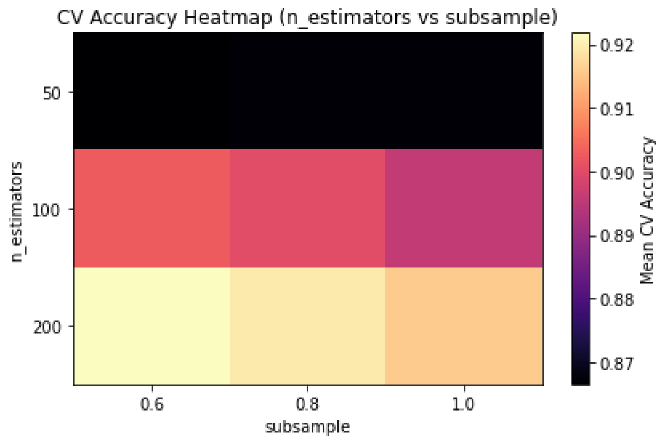

Fig. 9. Accuracy Heat-Map for XGboost Model, N_estimator vs Subsample

C. Artificial neural Networks

Figure 13 shows the evolution of training versus validation accuracy and loss over 50 epochs. The training accuracy climbs rapidly from $\sim$80% to nearly 100%, while the training

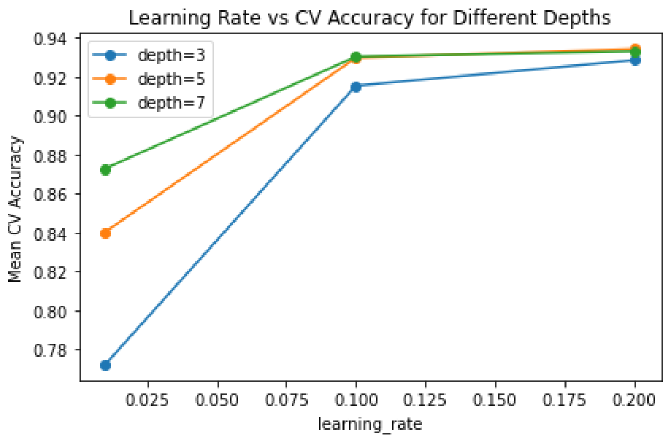

Fig. 10. Learning Rate Vs CV Accuracy for XGboost Model

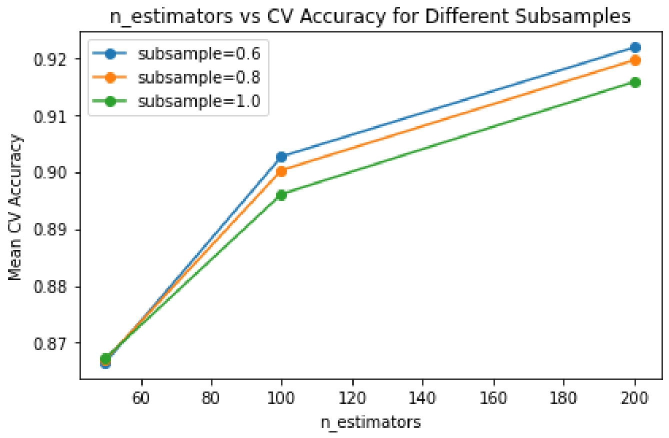

Fig. 11. N_estimator Vs CV Accuracy for XGboost Model

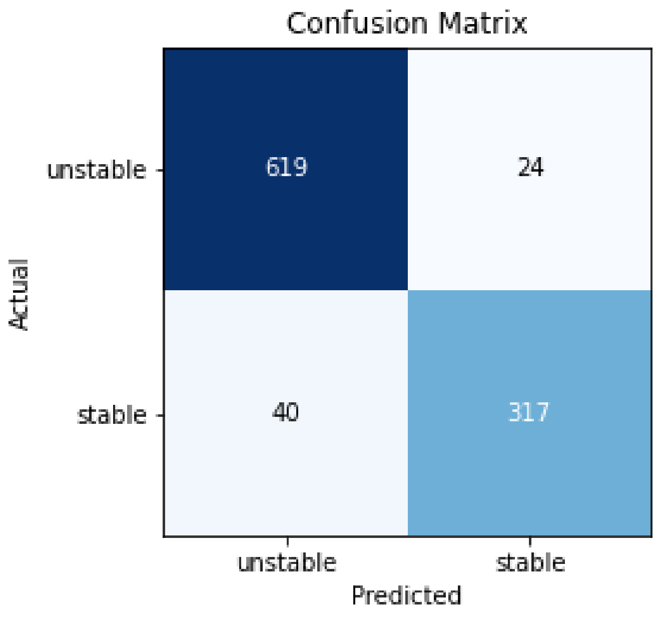

Fig. 12. Confiusion Matrix for XGboost Model

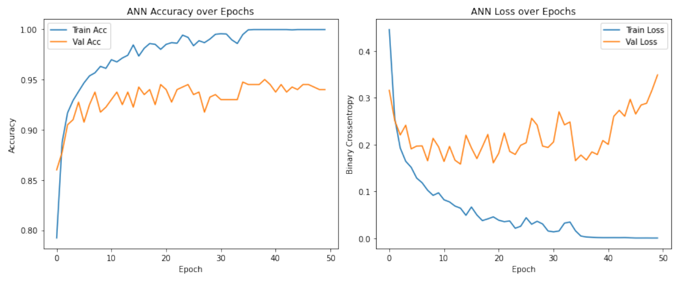

Fig. 13. ANN Accuracy & Loss over epochs

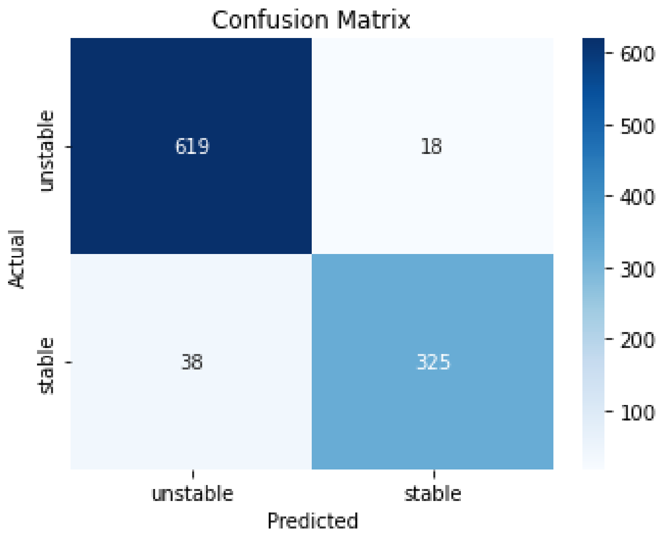

Fig. 14. Confiusion Matrix for ANN Model

loss falls toward zero, indicating that the network has more than enough capacity to memorize the training set. By contrast, validation accuracy rises more slowly, plateauing around 93–95%, and validation loss bottoms out around epoch 15 before drifting upward—evidence of mild overfitting once the model begins to specialize on idiosyncrasies of the training data. Overall, these curves suggest that 50 epochs are sufficient to capture the bulk of the signal, but that earlier stopping (or stronger regularization) might slightly improve generalization.

On the 20% held-out test set (1,000 samples), the ANN achieves 94.4% accuracy, with a precision of 94.8% and recall of 89.5% on the positive (”stable”) class. The detailed classification report (Table 1) shows that the model is somewhat better at recognizing “unstable” conditions (97% recall, 94% precision) than “stable” ones (90% recall, 95% precision), yielding macro- and weighted-average F1-scores of 0.94. The confusion matrix in Figure 14 confirms this balance: of 637 unstable samples, 619 are correctly identified (18 false stable), and of 363 stable samples, 325 are correctly identified (38 false unstable). The relatively low false-positive and false-negative rates demonstrate that the network discriminates effectively between the two classes.

D. *Discussion and Recommendations*

Together, these results indicate that the chosen architecture—four hidden layers of decreasing width with Nadam optimization—learns a robust representation of grid stability, achieving high overall accuracy while maintaining balanced precision and recall. The slight divergence between training and validation loss suggests the benefit of incorporating early-stopping or modest weight decay to curb overfitting. Additionally, it is worth noting that even (false negative) unstable classifications contribute more significantly to the confusion matrix than stable ones, so a boosted stable-class recall at even slight cost may be worth the decision. Tuning dropout or weighted loss to boost recall should be explored in future work.

References

1 Schäfer, B. *et al.*, “Taming instabilities in power grid networks by decentralized control,” *European Physical Journal Special Topics*, vol. 225, no. 3, pp. 569–582, 2016.

2 Arzamasov, V. *et al.*, “Towards concise models of grid stability,” in *2018 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*. IEEE, 2018, pp. 1–6.

3 Omar, M. B., Ibrahim, R., Mantri, R., Chaudhary, J., Selvaraj, K. R., and Bingi, K., “Smart grid stability prediction model using neural networks to handle missing inputs,” *Sensors*, vol. 22, no. 12, p. 4342, June 2022.

4 Schäfer, B., Matthiae, M., Timme, M., and Witthaut, D., “Decentral smart grid control,” *New Journal of Physics*, vol. 17, p. 015002, 2015.

5 Arzamasov, V., Böhm, K., and Jochem, P., “Towards concise models of grid stability,” in *Proceedings of the 2018 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*. Aalborg, Denmark: IEEE, October 2018, pp. 1–6.

6 Weedy, B., Cory, B., Jenkins, N., Ekanayake, J., and Strbac, G., *Electric Power Systems*. Hoboken, NJ, USA: Wiley, 2012.

7 Sinopoli, B. *et al.*, “Optimal linear lqg control over lossy networks without packet acknowledgment,” in *Proceedings of the 45th IEEE Conference on Decision and Control*. IEEE, 2006, pp. 392–397.

8 Suthaharan, S., “Support vector machine,” in *Machine Learning Models and Algorithms for Big Data Classification: Thinking with Examples for Effective Learning*. New York, NY, USA: Springer, 2016, pp. 207–235.

9 Amroune, M., Bouktir, T., and Musirin, I., “Power system voltage stability assessment using a hybrid approach combining dragonfly optimization algorithm and support vector regression,” *Arabian Journal for Science and Engineering*, vol. 43, pp. 3023–3036, 2018.

10 Burkov, A., *The Hundred-Page Machine Learning Book*. Quebec City, QC, Canada: Andriy Burkov, 2019, vol. 1.

11 Agostinelli, F. *et al.*, “Learning activation functions to improve deep neural networks,” arXiv preprint arXiv:1412.6830, 2014.

12 Jakovetić, D. *et al.*, “Fast distributed gradient methods,” *IEEE Transactions on Automatic Control*, vol. 59, no. 5, pp. 1131–1146, 2014.