## PREDICTING SMART GRID STABILITY **USING VARIOUS ML ALGORITHMS**

(**AID-505: Course Project)**

Muskan Gupta

muskan\_g@mfs.iitr.ac.in Enrollment:23565007

**Abstract**

This project assesses [Electrical Grid Stability data](https://archive.ics.uci.edu/ml/datasets/Electrical+Grid+Stability+Simulated+Data+) to predict if a given combination of power system conditions would result in an unstable grid - and therefore risk causing blackouts or damaging equipment.  After brief data exploration and processing, I ran baseline models, confirmed my feature selection, then optimised the models. The best classifier scored 98.3% accuracy on held out data, while the best regressor gave an R2 score of 95.9%.

* 1. **Main Objectives**

**-**Predicting Grid stability using different ML algorithms

**-** Comparing the results obtained by these models and deciding the best models for such a dataset

**2. Introduction**

Stable operation of complex flow and transportation networks requires balanced supply and demand. However, intermittency of weather -dependent renewable sources makes it harder and more expensive to maintain grid stability (a balance of electricity production and consumption).

The [Decentral Smart Grid Control](https://iopscience.iop.org/article/10.1088/1367-2630/17/1/015002#njp505903s5) (DSGC) concept was proposed as a way to adjust price based on supply and demand in a decentralised way - giving consumers an incentive to adjust their usage and help stabilise the grid without needing to centrally collect their usage data.

We will use various machine learning to determine stability of Smart Grid. However, many algorithms are available to us, like SVM, KNN, Logistic Regression, Random Forests, etc, So, in this project, I am trying to find out the best algorithm for this task by comparing them using metrics like accuracy on test set, f1 score, area under ROC curve, etc.

For this project, the data set I’m using was originally simulated to explore if grid stability can be maintained under DSGC, assuming a 4 node architechture: one producer providing electricity to three customers. I have split this data into training, and test set (80% training, 20% test).

**3 Dataset and Pre-processing**

The data set, available on Kaggle, has **10,000 instances and 12 attributes**:

* p[x] (p1 to p4): power produced or consumed; p1 = abs(p2 + p3 + p4)
* g[x] (g1 to g4): willingness of each node to adapt their consumption or production per second (gamma, proportional to price elasticity)
* tau[x] (tau1 to tau4): how long it takes for each node to adapt their production or consumption in seconds

where p1, g1 and tau1 are related to the electricity producer; the rest are related to the electricity consumers.

There are also two target variables:

* stab: a number representing grid stability (positive if unstable)
* stabf: a categorical version of stab

# Basic Quality checks

Since the data was simulated and very clean, I only briefly explored and processed it, e.g. basic quality checks, look at class balance, encode labels and rename columns for clarity.

# Balanced Data

As expected, no missing data and `stab` values of less than 0 are labelled 'stable' in `stabf`.

**A chart with a bar and a bar chart with a bar and a bar chart with text

Description automatically generated with medium confidence**

**Figure 1:** Visualisation of balance in data

# Visualising the correlation between two variables

# Correlation is a statistical measure that quantifies the extent to which two variables change together.

# The correlation between stability and delay[x] or adapt[x] columns were weak, whereas there was no obvious relationship between power[x] columns and stability.

# power generated was correlated with the power consumed, as expected, but there was no obvious correlation within delay[x] or adapt[x] columns.

A screenshot of a graph

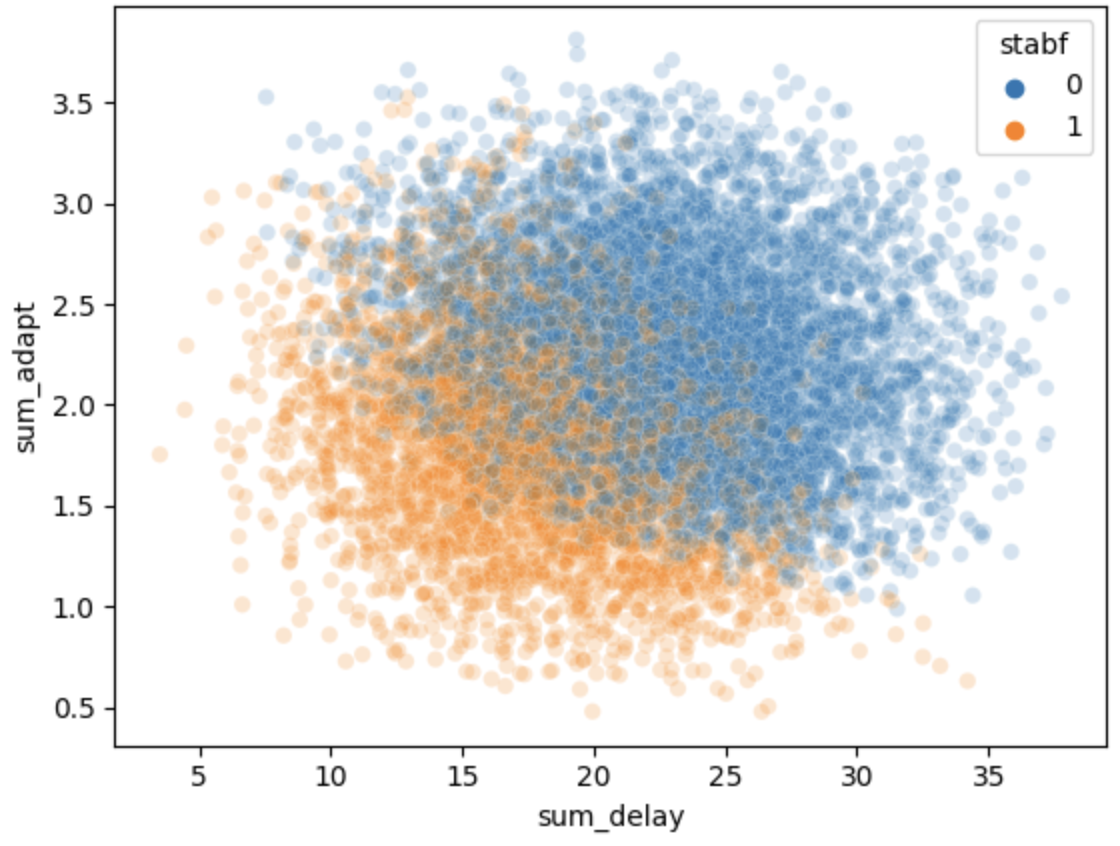
Description automatically generated

**Figure 2**: Visualizing the correlation between two variables

Share of unstable events:

* increased with reaction delay (of both producers and consumers) until roughly 5 seconds, after which the share of unstable events was relatively unaffected by further increases in delay times
* increased linearly with willingness to adapt (of both producers and consumers)
* still seemed uncorrelated with amount of power produced or consumed

Together, very high or very low sums of delay[x] and adapt[x] values should be indicators of (in)stability. Without further processing, these summarised values would highly correlate with existing features, so I'd rather keep the individual features. In contrast, it might be worth removing all power[x] columns if they're unhelpful.



**Figure 3**: Visualizing the correlation between sum of delay and adapt columns and stabf column

### **Build baseline models**

Fit baseline models (include feature scaling in pipeline to avoid data leakage but allow for easy adjustments when optimising), use feature/permutation importance and coefficients to confirm feature selection, check for overfitting

I first split the data, holding out 20% as the test set and using stratification to keep a consistent class share for the classification task.

# K Nearest Neighbours (KNN)

k-nearest neighbours makes predictions based on values of the nearest neighbours. For classification, this is summarised by a simple majority vote, whereas averaging is used for regression. This of course means hyperparameters related to the number of neighbours, how distances are calculated and how much weight is put on each neighbour's value affect the performance.

Best classifier pipeline: Pipeline(steps=[('transformer', QuantileTransformer()), ('estimator', KNeighborsClassifier(algorithm='ball\_tree', metric='manhattan', n\_neighbors=15, weights='distance'))])

# KNN on Test Set

Results: Accuracy = 90.9%, precision = 0.91, recall = 0.89, f1-score = 0.90, area under ROC = 0.89

A blue squares with white text

Description automatically generated A line graph with blue and orange lines

Description automatically generated

**Figure 4:** Confusion Matrix for KNN **Figure 5:** ROC Curve for KNN

# 5.2 Critical Points on KNN

* + - KNN takes no training time, so that we can add data to the training set anytime.
    - KNN gives equal weightage to all features, so it works worse in higher dimensions than SVM because some features may be redundant or less useful.
    - KNN is the most straightforward algorithm to implement and does not need many calculations.

# Support Vector Machines (SVM)

SVM generates hyperplanes that separate data points from each class for classification problems, or minimise distance of all points from the plane for regression problems. Some of the most important hyperparameters are probably the kernel that is used to transform the data, and a few parameters that adjust the regularisation.

In the SVM model, I have varied the penalty term (C) value and gamma. kernel is taken as linear.

Best classifier pipeline: Pipeline(steps=[('transformer', RobustScaler()), ('estimator', SVC(C=100.0, gamma='auto'))])

* 1. **SVM on test set**

Results: Accuracy = 98.25%, precision = 0.98, recall = 0.98, f1-score = 0.98, area under ROC = 0.98

**A blue squares with white text

Description automatically generated** A graph of a function

Description automatically generated with medium confidence

**Figure 6:** Confusion Matrix for SVM **Figure 7:** ROC Curve for SVM

# Critical Points on SVM

* SVM works well in a higher dimensional dataset because it can select the features along which there is the highest variance.
* SVM takes more training time than Decision Trees and KNN. Training time will further increase with the number of features and samples.
* SVM does not work well when there is a lot of overlap between classes.

# Random Forests

Random forest models build decision trees from random subsets of samples and features, and also have many hyperparameters to tune. Of course the impact of each parameter may vary depending on the data set. For example, with only eight features in our data set, setting max\_features to "auto", "sqrt" or "log2" should have no significant impact on performance - in all cases, three features should be considered at each split (though max\_features could still be set to a specific number instead).

Best classifier pipeline: Pipeline(steps=[('transformer', RobustScaler()), ('estimator', RandomForestClassifier(max\_depth=50, min\_samples\_split=5, n\_estimators=500, random\_state=1))])

# Random Forest on Test Set

Results: Accuracy = 93%, precision = 0.93, recall = 0.91, f1-score = 0.92, area under ROC = 0.91

A blue squares with white text

Description automatically generatedA graph of a line

Description automatically generated with medium confidence

**Figure 8:** Confusion Matrix for Random Forest **Figure 9:** ROC Curve for Random Forest

# Critical Points on Random Forest

* Random forests take little more time for training than decision trees.
* Random forests reduce overfitting as the samples are randomly selected using bagging for each tree.
* Due to randomness during bagging and each split in each tree, the results vary widely in each model run with the same hyperparameters

# Logistic Regression

Logistic Regression is used in binary classification. I have used a simple logistic regression model trained using the training set and calculated the metrics using the test set.

Best performing pipeline: Pipeline(steps=[('transformer', RobustScaler()), ('estimator', LogisticRegression(C=0.01, random\_state=1, solver='saga'))])

# Logistic Regression on Test Set

Results: Accuracy = 82%, precision = 0.82, recall = 0.79, f1-score = 0.80, area under ROC = 0.79

A blue squares with white text

Description automatically generated A line graph with blue and orange lines

Description automatically generated

**Figure 10:** Confusion Matrix for Logistic Reg. **Figure 11:** ROC Curve for Logistic Reg

# Critical Points on Logistic Regression

* + - Logistic Regression gives weights to features to determine how much a feature affects the output.
    - If there are too many features, Logistic Regression can overfit.
    - Logistic Regression cannot be used in multi-class problems.

# Gaussian Naive Bayes

Naïve Bayes uses the Bayes theorem to determine the probability of each sample being in a specific class. As our dataset has continuous-valued features (like 'Total Charges'), I have used Gaussian Naïve Bayes. The metrics are calculated on the test set after training on the train set.

# Gaussian Naïve Bayes on Test Set

Results: Accuracy = 84%, precision = 0.84, recall = 0.80, f1-score = 0.81, area under ROC = 0.80 and training time = 0 sec.

A blue squares with white text

Description automatically generated A line graph with blue and orange lines

Description automatically generated

**Figure 12:** Confusion Matrix for Naïve Bayes **Figure 13:** ROC Curve for Naïve Bayes

# Critical Points on Naïve Bayes

* + - Recall on the test set is higher than other models, which means it has identified the highest proportion of positive examples.
    - Naïve Bayes can easily be used for multi-class problems, and it takes no training time.
    - Naïve Bayes assumes all features are independent of each, which is rarely the case.

1. **Decision Trees**

A decision tree can go up to a depth equal to the number of features of the dataset (26 in our case); however, we can stop the decision tree from spreading early. This helps reduce overfitting as otherwise, the decision tree will try to fit every sample in the training set, including noise. I trained a decision tree over several values of max\_depth and calculated the accuracy of thevalidation set.

**9.1 Decision Trees on test set**

Results: Accuracy = 85%, precision = 0.84, recall = 0.83, f1-score = 0.83, area under ROC = 0.83 and training time = 0.95 sec.

A blue squares with white text

Description automatically generatedA line graph with blue and orange lines

Description automatically generated

**Figure 14:** Confusion Matrix for Decision Trees **Figure 15:** ROC Curve for Decision Trees

**9.2 Critical Points on Decision Trees**

* Decision tree takes significantly less time to train.
* Decision tree tries to fit every single example in the

training set, including the noise, if it is allowed to

branch out completely.

* Early stopping helps in reducing overfitting.

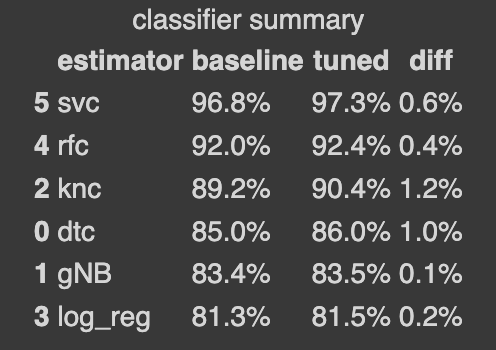
# Results and Conclusions

After simulations, we have been able to achieve the results shown in Table 1. Different metrics suggest different models are the best. It is essential to know which metric should be used to find the best model. Every problem has different goals for which appropriate metrics must be used.

Overall, optimisation maintained or slightly improved accuracy in all classifiers (by 0.1 - 1.5%) with the best performances coming from the SVM classifier and LightGBM regressor.

Testing the final classifier on the held out test data scored 98.3% accuracy. Precision, recall and the resulting F1 scores are high in both classes, and the ROC curve looks great. Of course, a score this high is extremely rare when working with real data, and suggests there isn't a lot of noise in this simulated data set.

Every dataset is different, and these models may give completely different metrics for them. It is essential to analyse all algorithms properly and select the best metrics for the need of the problem.



**Table 1:** Results of ML models on the test set

# References

1. *Electric Grid Stability Simulated data, available on Kaggle: https://www.kaggle.com/datasets/sowlarn/ucis-electrical-grid-stability-simulated-data/*
2. *Learning and Testing Decision Tree, Nader H. Bshouty, Haddad-Zaknoon, arXiv:2108.04587*
3. *Understanding Random Forests, Gilles Louppe, 2014, arXiv:1407.7502*
4. *Support Vector Machine Classifier via Soft-Margin Loss, Huajun Wang, Yuanhai Shao, Shenglong Zhou, Ce Zhang, Naihua Xiu, arXiv:1912.07418*
5. *k-Nearest Neighbour Classifiers: 2nd Edition, Padraig Cunningham, Sarah Jane Delany, arXiv:2004.04523*
6. *Introduction to logistic Regression, Moo K. Chung, arXiv:2008.13567*
7. *Bayes and Naive Bayes Classifier, Vikramkumar, Vijaykumar B, Trilochan, 2014, arXiv:1404.0933*