CAPSTONE3 - Application of Deep Learning for the Stability Prediction of the Smart Grids

1 Introduction

1.1 Problem statement

Decentral Smart Grid Control (**DSGC**) is a new system implementing demand response without significant changes of the power grid infrastructure. It does so by binding the electricity price to the grid frequency, which is one of the most important factors of the power grid stability[1][2]. The theoretical model of the power grid system made some simplifications to monitor and control the operation of the power grids. With the development of data collection and data mining, it is possible to use deep learning methods to predict the stability of a four-node-star electrical grid with a centralized production system. This project will develop a deep learning model for this four-node-star electrical grid with a centralized production that the precision is more than 90% to predict the stability of this power grid.

1.2 Dataset

The dataset, created by KIT, contains results from simulations of grid stability for a reference 4-node star power grid. This power grid system is the most common power grid system in the world. The original dataset contains 10,000 observations and includes 12 primary predictive features and two dependent variables. The parameters and status were recorded for about 2 months with a ten-minute interval, which include stable and unstable scenarios.

The original dataset contains 10,000 observations. As the reference grid is symetric, the dataset can be augmented in 3! (3 factorial) times, or 6 times, representing a permutation of the three consumers occupying three consumer nodes. The augmented version has then 60,000 observations. It also contains 12 primary predictive features and two dependent variables.

Predictive features:
'*tau1*' to '*tau4*': the reaction time of each network participant;
'*p1*' to '*p4*': nominal power produced (positive) or consumed (negative) by each network participant;
'*g1*' to '*g4*': price elasticity coefficient for each network participant;

Dependent variables:
'*stab*': the maximum real part of the characteristic differentia equation root (if positive, the system is linearly unstable; if negative, linearly stable);
'*stabf*': a categorical (binary) label ('stable' or 'unstable').

1.3 Study object and goal

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:

- the total **power balance** (nominal power produced or consumed at each grid node);
- the response time of participants to adjust consumption and/or production in response to price changes (referred to as "**reaction time**");
- energy **price elasticity**.

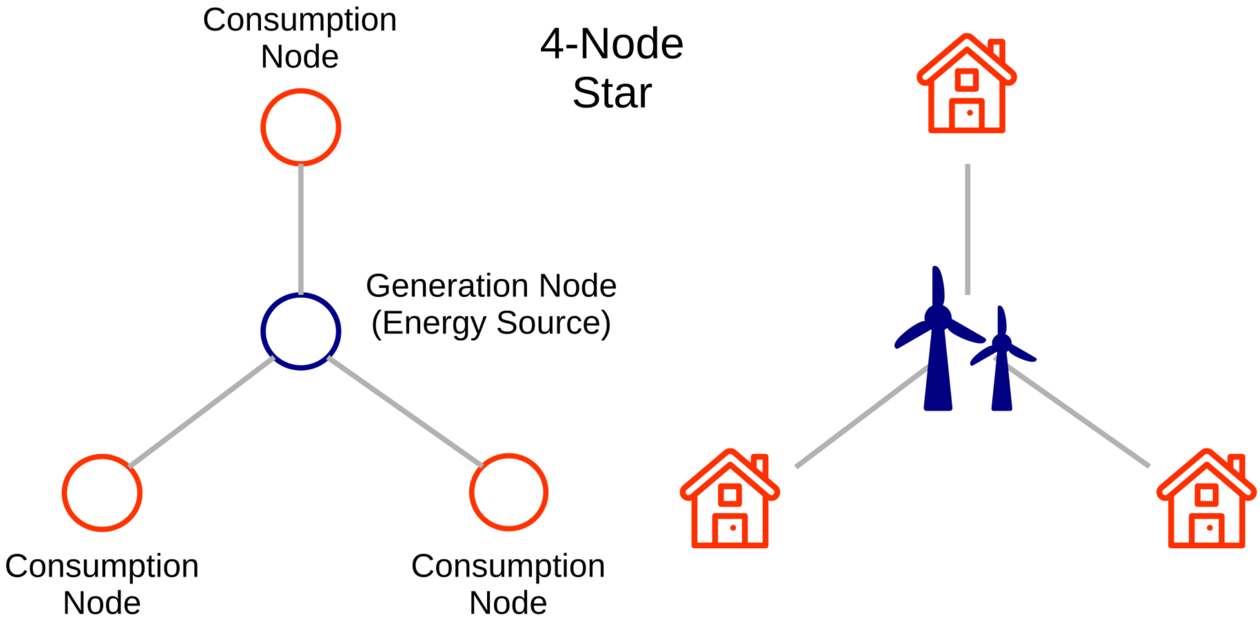


Figure 1: 4-node star power grid model

2 Data Wrangling

2.1 Data collection

Goal: Read the raw data and investigate the data structure of data frame.

2.1.1 Import the libraries

```
# Import pandas, matplotlib.pyplot, and seaborn, datetime below
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os

from scipy import stats
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.model_selection import KFold

from keras.models import Sequential
from keras.layers import Dense

from datetime import datetime
```

2.1.2 Load the raw data

Read two datasets: one is the original dataset with 10,000 observations and the other is the augmented dataset with 60,000 observations that were permutated (3!=6) by the original dataset.

```
df = pd.read_csv('../data/smart_grid_stability_augmented.csv')
df1 = pd.read_csv('../data/Data_for_UCI_named.csv')
```

2.1.3 Check the raw data

There are 12 predictive features:

'*tau1*' to '*tau4*': the reaction time of each network participant. It is a real value within the range 0.5 to 10 ('*tau1*' corresponds to the supplier node, '*tau2*' to '*tau4*' to the consumer nodes) from the results of df.describe() above.

'*p1*' to '*p4*': nominal power produced (positive) or consumed (negative) by each network participant, a real value within the range -2.0 to -0.5 for consumers ('*p2*' to '*p4*'). As the total power consumed equals the total power generated, p1 (supplier node) = - (p2 + p3 + p4).

'*g1*' to '*g4*': price elasticity coefficient for each network participant. It is a real value within the range 0.05 to 1.00 ('*g1*' corresponds to the supplier node, '*g2*' to '*g4*' to the consumer nodes; '*g*' stands for '*gamma*') from the results of df.describe() above. As defined in [1], gamma = c_1 dot c_2

$$p_i(\frac{d\theta_i}{dt}) = p_\omega - c_1\frac{d\theta_i}{dt} \qquad \mathrm{i} = 1,\dots,N. \qquad (1)$$

$$\hat{P}_i(t) \approx P_i + c_2(p_i - P_\Omega) \qquad \mathrm{i} = 1,\dots,N. \qquad (2)$$

$$\hat{P}_i(t) = P_i - \gamma_i\frac{d\theta_i}{dt}(t) \qquad \mathrm{i} = 1,\dots,N. \qquad (3)$$

where γi is proportional to the price elasticity of each node i, i.e., measures how much a producer or consumer is willing to adapt their consumption or production. In general, such an adaptation will not be instantaneous but will be delayed by a certain time τ by a measurement and the following reaction.

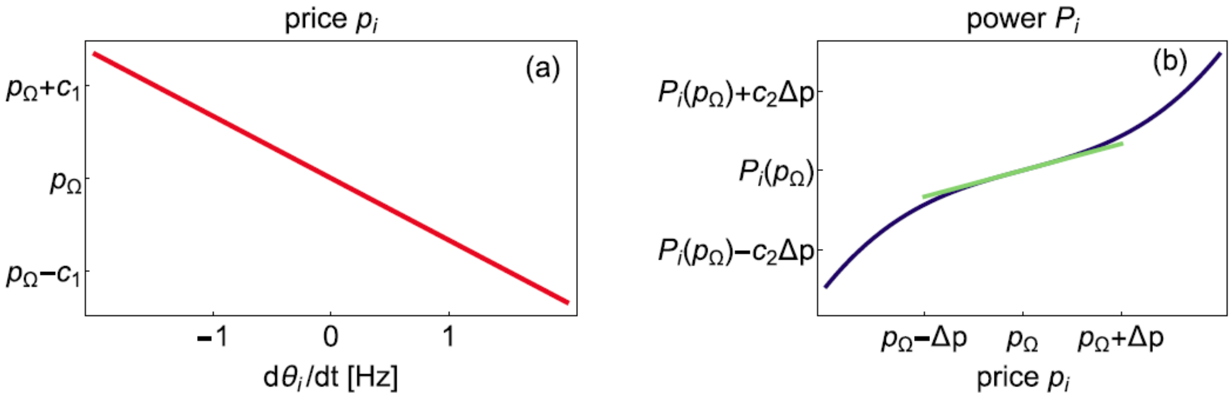


Figure 2: Using linear relations the power becomes a linear function of the frequency deviation dθi/dt. (a) We assume a linear price-frequency relation to motivate consumers to stabilize the grid. For example, if the production is larger than consumption, the power grid frequency increases. Hence, decreasing prices should motivate additional consumption. (b) Although consumers might react non-linearly towards price-changes (dark blue), we assume a linear relationship (light green) close to the operational frequency Ω which corresponds to dθi/dt = 0.

There are two dependent variables:

'*stab*': the maximum real part of the characteristic differentia equation root (if positive, the system is linearly unstable; if negative, linearly stable);

'*stabf*': a categorical (binary) label ('stable' or 'unstable').

As there is a direct relationship between 'stab' and 'stabf' ('stabf' = 'stable' if 'stab' <= 0, 'unstable' otherwise), 'stab' will be dropped and 'stabf' will remain as the sole dependent variable.

2.2 Data definition

Goal: Gain an understanding of your data features to inform the next steps of EDA.

2.2.1 Clean the dataframe

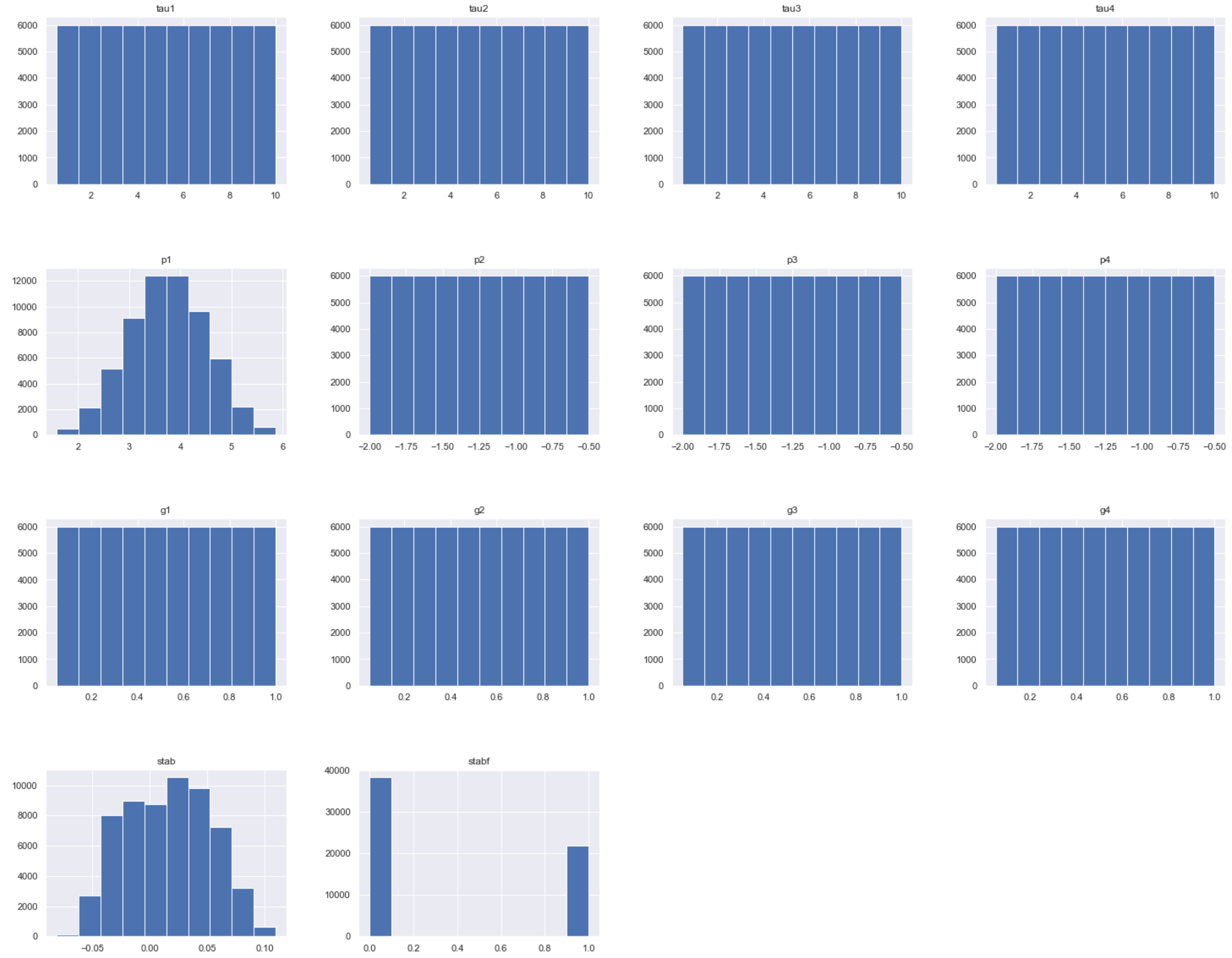
As the dataset content comes from simulation exercises, there are no missing values. Also, all features are originally numerical, no feature coding is required. Such dataset properties allow for a direct jump to machine modeling without the need of data preprocessing or feature engineering.

```
# replace the 'unstable' with '0' and 'stable' with '1' for column stabf
map1 = {'unstable': 0, 'stable': 1}
df['stabf'] = df['stabf'].replace(map1)
df1['stabf'] = df1['stabf'].replace(map1)

# shuffle the data
df = df.sample(frac=1)
df1 = df1.sample(frac=1)
```

2.2.2 Variable distribution

Through the codes below, we can see the distributions of the variables that *p1* is a normal distribution between 1 and 6. Other dependent variables are almost even distribution. The predictive variable *stabf* has the ratio of unstable and stable about 2.

```
# plot the histgrams for each variable (12 predictive features and 2 dependent variables) to
look at the distribution
df.hist(figsize=(25,20))
plt.subplots_adjust(hspace=0.5);
```



Let us briefly look at the distribution of these variables.

tau1, *tau2*, *tau3* and *tau4*: the reaction time of each network participant. They are even distribution from 0.5 to 10 and each data bin (0-1, 1-2, etc.) has about 6000 samples for each variables.

p2, *p3*, and *p4*: nominal power consumed (negative) by each network participant. They are also even distribution from -2.0 to -0.5.

p1: nominal power produced (positive). *p1=-(p2+p3+p4)*. It is approximately a normal distribution, which is able to be proved by the Central Limit Theorem. The mean is 3.75 and the standard deviation is 0.752.

g1, *g2*, *g3* and *g4*: price elasticity coefficient for each network participant. They are even distribution from 0 to 1 to impact the electricity price as Figure 2(a) and (b).

stab:the maximum real part of the characteristic differentia equation root. If *stab* < 0, the grid is stable and *stabf* = "stable". If *stab* >= 0, the grid is unstable and *stabf* = "unstable". The range of stab is between -0.05 and 0.10.

stabf:a categorical (binary) label ('stable' or 'unstable'). The binary label (*stabf* = 0 means "unstable" and *stabf* = 1 means "stable". From the plots above, there are 40,000 unstable cases (*stabf* = 0), almost twice than the stable cases.

```
# save the data for future use
df.to_csv('../data/df_clean.csv')
df1.to_csv('../data/df1_clean.csv')
```

3 Exploratory Data Analysis

3.1 Inferential statistics

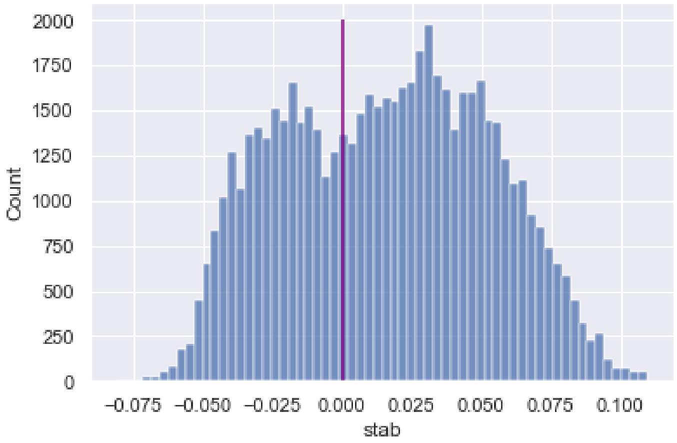
3.1.1 Overview of the dataset

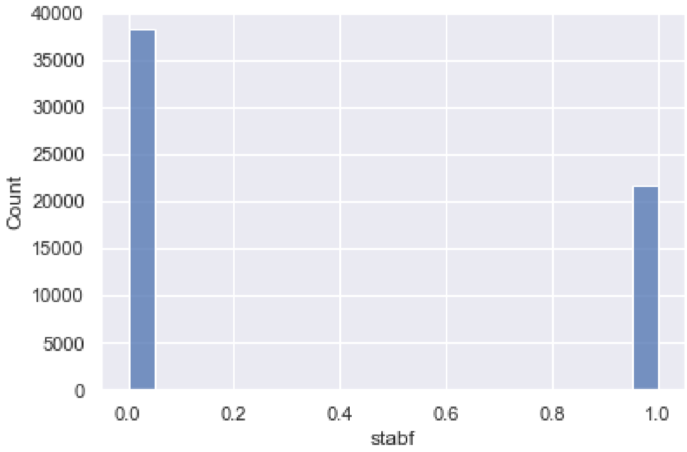
```
# strip the column names to get rid of any pre and post white spaces
stripped_name = [col.strip() for col in list(df.columns)]
df.rename(columns=dict(zip(list(df.columns), stripped_name)), inplace=True)

# verify the data type and missing values
df.info()
```

Column "*stabf*" has been changed to int64 type with 0 and 1 to indicate "unstable" and "stable" cases. Other variables are float64 type data. There is no missing values in the dataset and all data are numeric type of data.

3.1.2 Cleaning, transforming and visualizing

```
# Data are clean and concise. Plot the counts of "stab" cases
sns.histplot(data=df['stab']);
plt.vlines(x = 0, ymin = 0, ymax = 2000,
           colors = 'purple',
           label = 'Border of stable/unstable')
plt.show()
```



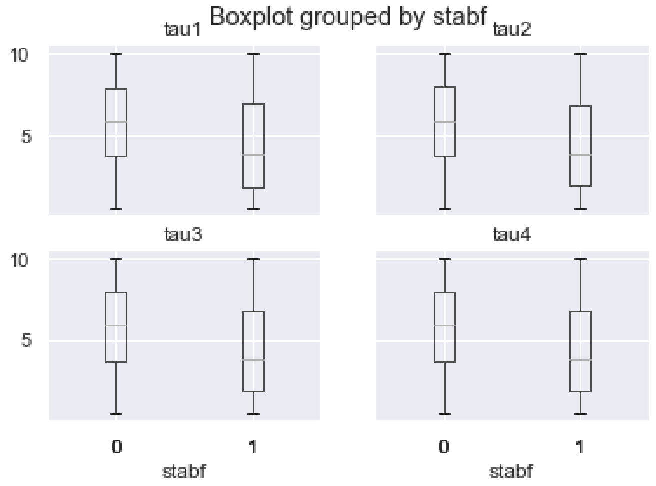


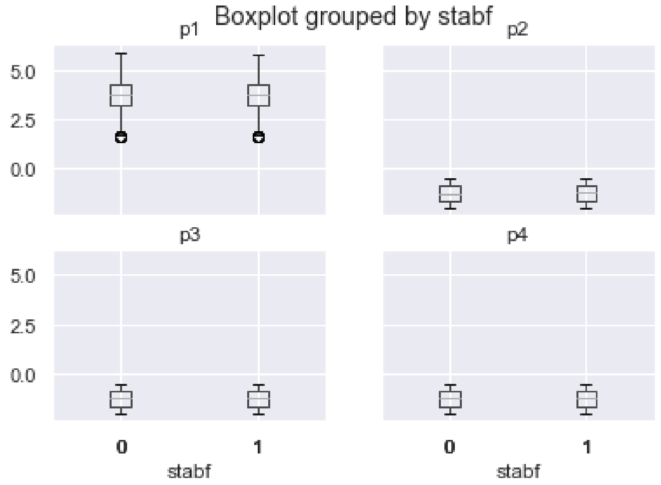
```
# To summarize analytically, let's use the groupby() method on our df.
df.groupby(by = 'stabf').mean()
```

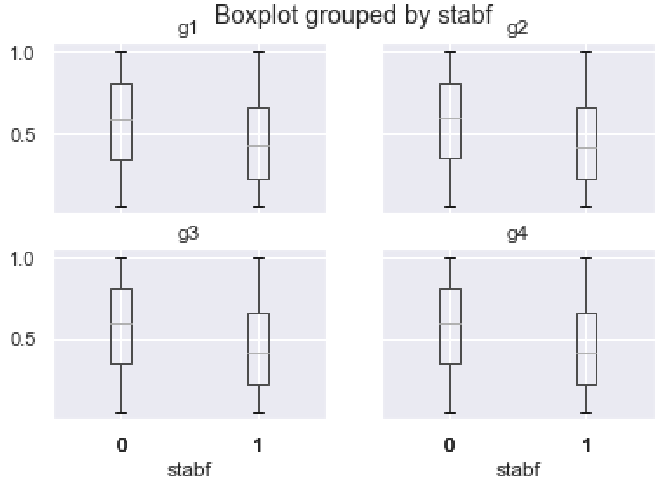
	tau1	tau2	tau3	tau4	p1	p2	p3	p4	g1	g2	g3	g4	stab
stabf													
0	5.735239	5.747946	5.747946	5.747946	3.755630	-1.251877	-1.251877	-1.251877	0.565832	0.570037	0.570037	0.570037	0.038727
1	4.394799	4.372406	4.372406	4.372406	3.740077	-1.246692	-1.246692	-1.246692	0.453035	0.445627	0.445627	0.445627	-0.024798

From the table above, we can see that *tau1*, *tau2*, *tau3*, *tau4* and *g1*, *g2*, *g3*, *g4* are siginificantly different for the unstable (*stabf*=0) and stable (*stabf*=1) cases. We also see slight differences of *p1*, *p2*, *p3*, *p4* in unstable (*stabf*=0) and stable (*stabf*=1) cases.

```
# Call the boxplot() method on our df based on tau.
df.boxplot(by = 'stabf', column = ['tau1', 'tau2', 'tau3', 'tau4']);
df.boxplot(by = 'stabf', column = ['p1', 'p2', 'p3', 'p4']);
df.boxplot(by = 'stabf', column = ['g1', 'g2', 'g3', 'g4']);
```







From the first boxplot grouped by stabf for *tau1, tau2, tau3 and tau4*, the mean and the range (25%, 75%) of tau (reaction time) in unstable cases are significantly higher than those in the stable cases.
There are the same trend for *g1, g2, g3 and g4* about the price impacts.
The normal power produced (*p1*) and the normal power consumed (*p2, p3, p4*) are not significantly different between the unstable and stable cases.
The impacts of these feature variables will be proved by hythesis analysis in the next section.

3.2 Hypothesis tests

3.2.1 Null hypothesis of tau_i

Hypothesis formulation

Our Null hypothesis is:
H_null: the observed difference in the mean of tau_i (i=1,2,3,4) between the unstable cases and the stable cases is due to chance. That is,
$\text{tau}_{i,u} = \text{tau}_{i,s}$

The alternate hypothesis:
H_alternative: there are significant difference (over the significance level 95%) between tau_i (i=1,2,3,4) in the unstable cases and the stable cases. That is,
$\text{tau}_{i,u} \neq \text{tau}_{i,s}$

Calculate the p-value to do the hypothesis test

```
# check how many of the differences are at least as extreme as (95%) our observed differenc
sum([(diff - obs_difference_tau1) > 0 for diff in difference_tau1]) / len(difference_tau1)
```

So actually, zero differences are at least as extreme as our observed difference! So the p-value of our observed data is 0. It doesn't matter which significance level we pick; our observed data is statistically significant, and we reject the **H_{Null}** and accept **$H_{Alternatative}$**.

3.2.2 Null hypothesis of g_i

Hypothesis formulation

Our Null hypothesis is:
H_null: the observed difference in the mean of g_i (i=1,2,3,4) between the unstable cases and the stable cases is due to chance. That is,
$g_{i,u} = g_{i,s}$

The alternate hypothesis:
H_alternative: there are significant difference (over the significance level 95%) between g_i (i=1,2,3,4) in the unstable cases and the stable cases. That is,
$g_{i,u} \neq g_{i,s}$ Similarly, zero differences are at least as extreme as our observed difference!
So the p-value of our observed data is 0.
It doesn't matter which significance level we pick; our observed data is statistically significant, and we reject the **H_{Null}** and accept **$H_{Alternatative}$**.

3.2.3 Null hypothesis of p_i

Hypothesis formulation

Our Null hypothesis is:
H_null: the observed difference in the mean of p_i (i=1,2,3,4) between the unstable cases and the stable cases is due to chance. That is,
$p_{i,u} = p_{i,s}$

The alternate hypothesis:
H_alternative: there are significant difference (over the significance level 95%) between p_i (i=1,2,3,4) in the unstable cases and the stable cases. That is,
$p_{i,u} \neq p_{i,s}$ Similarly, he difference percentage is 0.008, which are at least as extreme as our observed difference! So the p-value of our observed data is 0.008, which is less than the significance level 0.05. Our observed data is still statistically significant, and we reject the $\mathbf{H_{Null}}$ and accept $\mathbf{H_{Alternatative}}$.

4 Pre-processing and Training Data

4.1 Customized functions

Functions were developed to assist with graphical analysis of specific dataset elements (features or observations) and mapping correlation. Please refer to the respective docstrings below for details. Note that all function variable names, by coding principle, start with the "f_" string, allowing for containerized processing within the function execution environment, not affecting global variables.
def assessment(f_data, f_y_feature, f_x_feature, f_index=-1):
""" Develops and displays a histogram and a scatter plot for a dependent / independent variable pair from a dataframe and, optionally, highlights a specific observation on the plot in a different color (red). Also optionally, if an independent feature is not informed, the scatterplot is not displayed.
Keyword arguments:

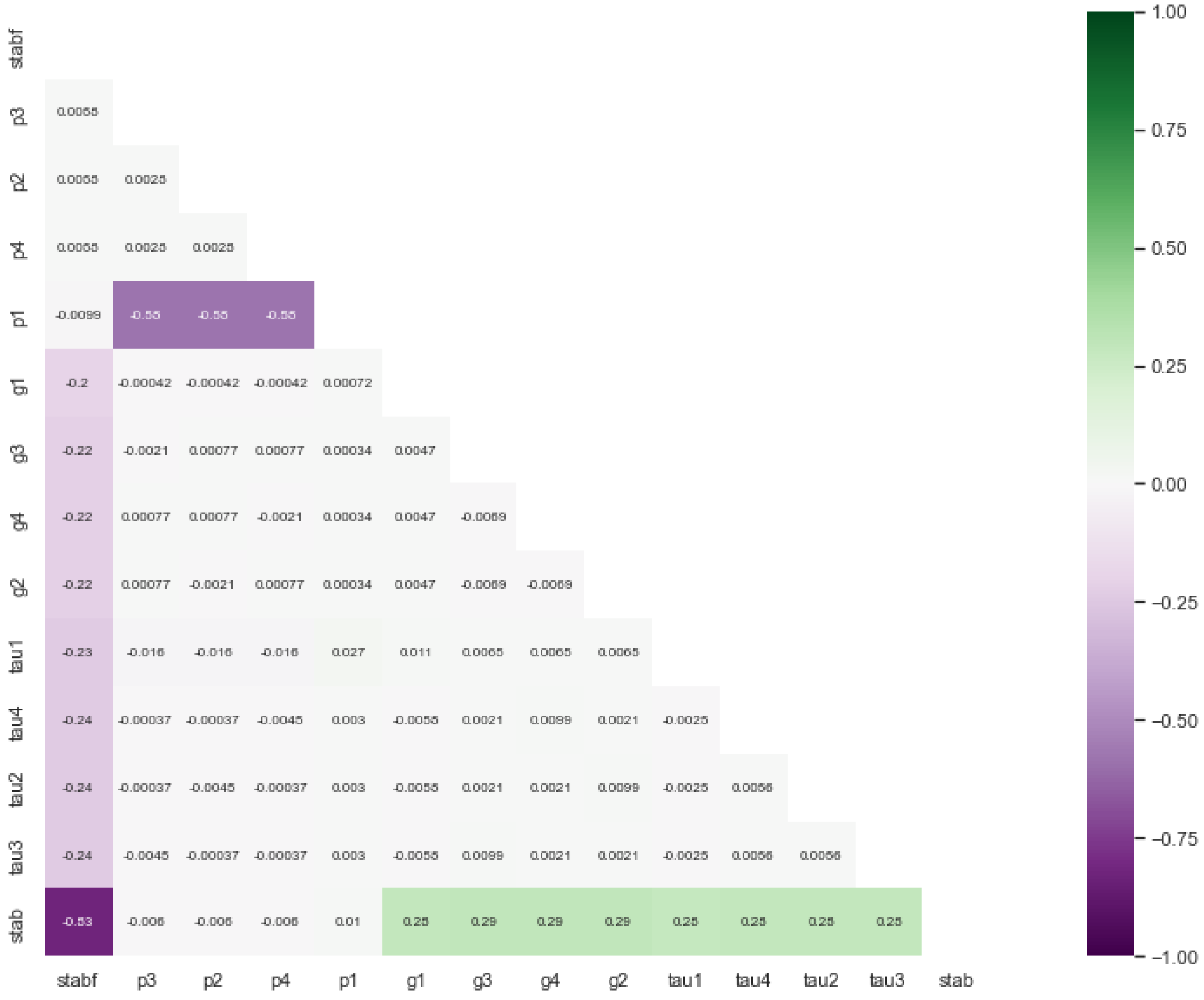
f_data Tensor containing the dependent / independent variable pair. Pandas dataframe f_y_feature Dependent variable designation. String f_x_feature Independent variable designation. String f_index If greater or equal to zero, the observation denoted by f_index will be plotted in red. Integer """

- Python codes of this assessment function is not presented.*

4.2 Correlation

It is important to verify the correlation between each numerical feature and the dependent variable, as well as correlation among numerical features leading to potential undesired colinearity. The heatmap below provides an overview of correlation between the dependent variable ('*stabf*') and the 12 numerical features. Note that also the alternative dependent variable ('*stab*') has been included just to give an idea of how correlated it is with 'stabf'. Such correlation is significant (-0.83), as it should be, which reinforces the decision to drop it, anticipated in Section 3. Also, correlation between *'p1'* and its components *'p2', 'p3' and 'p4'* is above average, as expected, but not as high o justify any removal.

```
correlation_map(df, 'stabf', 14)
```



4.3 Segregating train and test sets

As anticipated, the features dataset will contain all 12 original predictive features, while the label dataset will contain only 'stabf' ('stab' is dropped here).

In addition, as the dataset has already been shuffled, the training set will receive the first 54,000 observations, while the testing set will accommodate the last 6,000.

Even considering that the dataset is large enough and well behaved, the percentage of 'stable' and 'unstable' observations is computed for both training and testing sets, just to make sure that the original dataset distribution is maintained after the split - which proved to be the case.

After splitting, Pandas dataframes and series are transformed into Numpy arrays for the remainder of the exercise.

```
# splitting the augmented dataset
X = df.iloc[:, :12]
y = df.iloc[:, 13]

X_training = X.iloc[:54000, :]
y_training = y.iloc[:54000]

X_testing = X.iloc[54000:, :]
y_testing = y.iloc[54000:]

ratio_training = y_training.value_counts(normalize=True)
ratio_testing = y_testing.value_counts(normalize=True)
```

```
ratio_training, ratio_testing

# splitting the augmented dataset
X1 = df1.iloc[:, :12]
y1 = df1.iloc[:, 13]

X1_training = X1.iloc[:9000, :]
y1_training = y1.iloc[:9000]

X1_testing = X1.iloc[9000:, :]
y1_testing = y1.iloc[9000:]

ratio_training_1 = y1_training.value_counts(normalize=True)
ratio_testing_1 = y1_testing.value_counts(normalize=True)
ratio_training_1, ratio_testing_1
```

4.4 Feature assessing and scaling

In preparation for machine learning, scaling is performed based on (fitted to) the training set and applied (with the 'transform' method) to both training and testing sets.

```
# scale the augmented dataset
scaler = StandardScaler()
X_training = scaler.fit_transform(X_training)
X_testing = scaler.transform(X_testing)

# scale the original dataset
scaler = StandardScaler()
X_training = scaler.fit_transform(X_training)
X_testing = scaler.transform(X_testing)
```

5 Deep Learning

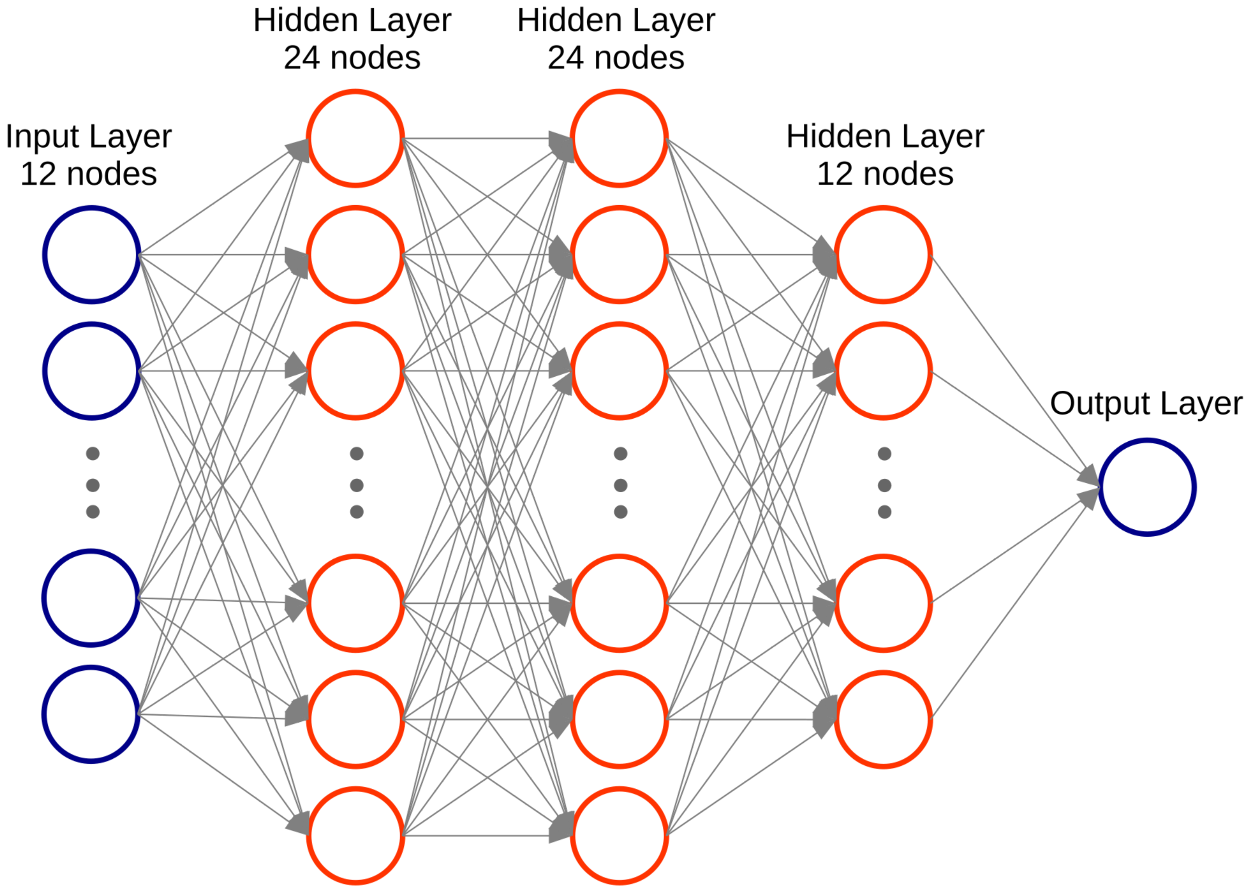
5.1 Model definition

The artificial neural network (ANN) architecture depicted below is the optimal one evaluated in this study. It reflects an sequential structure with:

one input layer (12 input nodes); three hidden layers (24, 24 and 12 nodes, respectively); one single-node output layer. Alternative architectures were evaluated with variations of the code below. Their performance will be discussed in Chapter 6.

As features are numerical real numbers within ranges, the choice of 'relu' as the activation function for hidden layers seems straightforward. Similarly, as this is a logistic classification exercise, where the output is binary ('0' for 'unstable', '1' for 'stable', following the map coding used in Section 2.2.1), the choice of 'sigmoid' as activation for the output layers seems obvious.

Compilation with 'adam' as optimizer and 'binary_crossentropy' as the loss function follow the same logic. The fitting performance will be assessed using 'accuracy' as the metric of choice.



```
# ANN initialization
classifier = Sequential()
# Input layer and first hidden layer
classifier.add(Dense(units = 24, kernel_initializer = 'uniform', activation = 'relu',
input_dim = 12))
# Second hidden layer
classifier.add(Dense(units = 24, kernel_initializer = 'uniform', activation = 'relu'))
# Third hidden layer
classifier.add(Dense(units = 12, kernel_initializer = 'uniform', activation = 'relu'))
# Single-node output layer
classifier.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))
# ANN compilation
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

5.2 Model fitting

Even considering that data is well behaved and in general uniformly distributed, a cross-validation based fitting is proposed. KFold is the cross-validation engine selected, and 10 different validation sets will be utilized.

```
### Case 3: Case 1: 24-24-12-1, 10, 50
# set up the start_time
sns.set()
start_time = datetime.now()

cross_val_round = 1
```

```
print(f'Model evaluation\n')

for train_index, val_index in KFold(10, shuffle=True, random_state=10).split(X_training):
    x_train, x_val = X_training[train_index], X_training[val_index]
    y_train ,y_val = y_training[train_index], y_training[val_index]
    classifier.fit(x_train, y_train, epochs=50, verbose=0)
    classifier_loss, classifier_accuracy = classifier.evaluate(x_val, y_val)
    print(f'Round {cross_val_round} - Loss: {classifier_loss:.4f} | Accuracy:
{classifier_accuracy * 100:.2f} %')
    cross_val_round += 1

end_time = datetime.now()
cv_duration = end_time - start_time
print('CV duration (s):', cv_duration)
```

5.3 Predicting smart grid stability

After fitting the model to the training set, it is time to extract predictions for the testing set and segregate those above the 'threshold' of 0.5 ('unstable' cases below the threshold, 'stable' cases above it).

```
y_pred = classifier.predict(X_testing)
y_pred[y_pred <= 0.5] = 0
y_pred[y_pred > 0.5] = 1
print(classification_report(y_testing, y_pred))
cm = pd.DataFrame(data=confusion_matrix(y_testing, y_pred, labels=[0, 1]),
                  index=["Actual Unstable", "Actual Stable"],
                  columns=["Predicted Unstable", "Predicted Stable"])
cm
print(f'Accuracy per the confusion matrix: {((cm.iloc[0, 0] + cm.iloc[1, 1]) /
len(y_testing) * 100):.2f}%')
```

6 Tuning Parameters of ANN

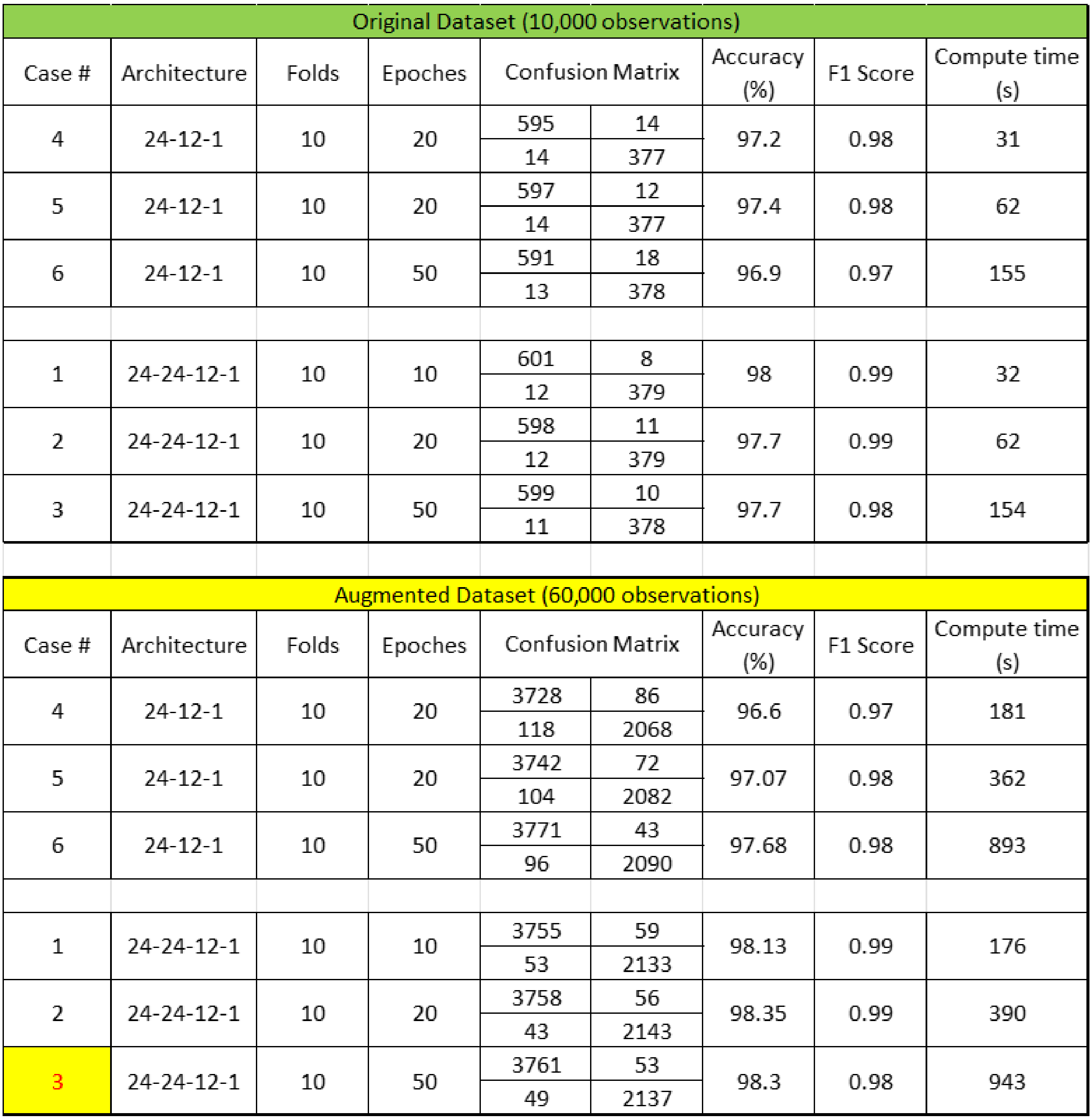
6.1 Tuning parameters

The architecture and the hyperparameters selected in Chapter 5 led to the best prediction performance on the test set.

In addition, several other combinations were evaluated for both the original dataset with 10,000 observations and the augmented dataset with 60,000 observations. It is important to emphasize that in this comparative assessment no shuffling of any type, at any part of the exercise, was performed, so that the very same testing set was exposed to model after fitting for performance assessment.

6.2 Other cases

To tune the parameters, we test 11 cases for different ANN architecture (3 hidden layers vs. 2 hidden layers), epoches (10, 20, 50), datasets (augmented vs original). The same ANN or deep learning method will be used for these cases as that of Case 3 in Chapter 5. With Case 3 in Chapter 5, the performance of each model are summarized in the table below.



Original Dataset (10,000 observations)								
Case #	Architecture	Folds	Epoches	Confusion Matrix		Accuracy (%)	F1 Score	Compute time (s)
4	24-12-1	10	20	595	14	97.2	0.98	31
				14	377			
5	24-12-1	10	20	597	12	97.4	0.98	62
				14	377			
6	24-12-1	10	50	591	18	96.9	0.97	155
				13	378			
1	24-24-12-1	10	10	601	8	98	0.99	32
				12	379			
2	24-24-12-1	10	20	598	11	97.7	0.99	62
				12	379			
3	24-24-12-1	10	50	599	10	97.7	0.98	154
				11	378			

Augmented Dataset (60,000 observations)								
Case #	Architecture	Folds	Epoches	Confusion Matrix		Accuracy (%)	F1 Score	Compute time (s)
4	24-12-1	10	20	3728	86	96.6	0.97	181
				118	2068			
5	24-12-1	10	20	3742	72	97.07	0.98	362
				104	2082			
6	24-12-1	10	50	3771	43	97.68	0.98	893
				96	2090			
1	24-24-12-1	10	10	3755	59	98.13	0.99	176
				53	2133			
2	24-24-12-1	10	20	3758	56	98.35	0.99	390
				43	2143			
3	24-24-12-1	10	50	3761	53	98.3	0.98	943
				49	2137			

6.3 Tuning parameters comparision

The ratio of unstable ans stable data is about 2:1 for all cases. Therefore, the accuracy and F1 score are listed to evaluate the performance of each model. F1 score is the go to metric for measuring the performance of classification models. This is mainly due to it’s ability to relay true performance on both balanced and imbalanced datasets. But also because it takes into account both the precision and recall ability of the model, making it a well rounded assessor of model performance.

From Table 1 below, we know all cases have the F1 Score within the range of (0.97, 0.99) and the accuracy between 96.6% - 98.3% for this classification problem. The architecture network are three or two hidden layers, which have slight impacts on the F1 score (accuracy) and computing time. It seems that three layers will help the F1 score slightly. The augmented dataset (obtained by permutating the original data) also increases the F1 score and the accuracy slightly. But, the augmented data consume about 6 times of computing time than the original dataset, which is proportional to the size of datasets.

From the training time, F1 score, and the accuracy, we can conclude that the parameters of Case 1 in the augmented dataset is preferrable to predict the stability of the power grids.

7. Conclusions

In this project, the basic reference 4-node star power grid was studied for the prediction of its stability by the deep learning, specifically artificial neural network (ANN). In this power grid system, there is one power plant generating electricity that was supplied to the end users at three nodes. The original dataset (10,000 observations) and the augmented dataset (60,000 observations) were provided by KIT to support the training and testing 12 models with different parameters. Through the ANN method, the conclusions below could be drawn:

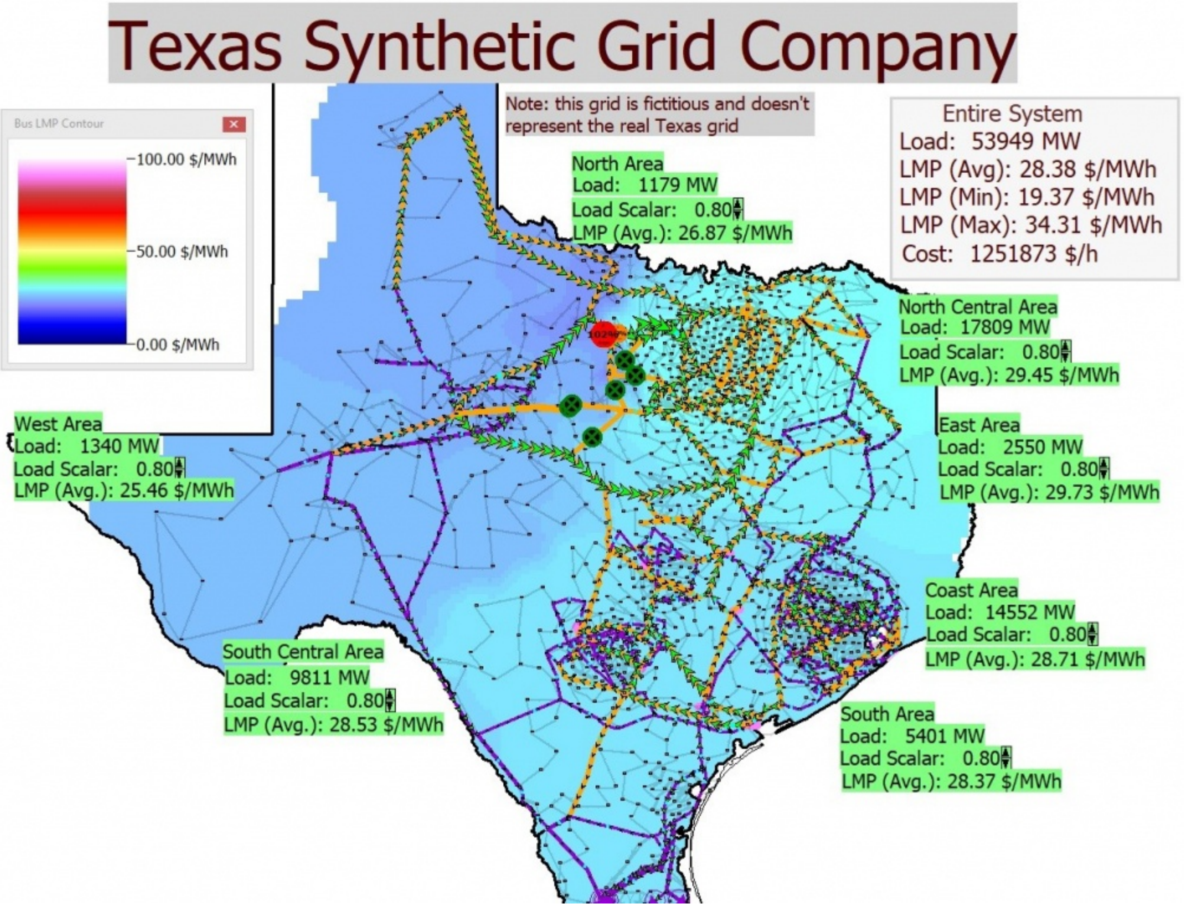
1. All cases had the F1 Score within the range of (0.97, 0.99) and the accuracy between 96.6% - 98.3% for this classification problem.
2. The hidden layers of the architecture network had slight impacts on the F1 score (accuracy) and computing time. It seems that three layers helped the F1 score slightly.
3. The augmented dataset also increases the F1 score and the accuracy slightly. But, the augmented data consume was proportional to the size of dataset.
4. For this basic power grid, the model with two hidden layers, 10 folds and 10 epoches on the augmented dataset is preferrable to predict the power grid stability.

8. Future Work

For this basic 4-node star power grid, the ANN model has very good F1 score (accuracy) to predict the stability. To apply for this ANN model into the real operation of power grids, we still have more challenging tasks to solve.

1. More nodes of power plants and end users.
2. Solar energy implementation in the commercial and residential users.
3. More baterry-based electricity storage implementation.

4. More flexible electricity price strategies.


Texas Synthetic Grid Company

Note: this grid is fictitious and doesn't represent the real Texas grid

Entire System
Load: 53949 MW
LMP (Avg): 28.38 $/MWh
LMP (Min): 19.37 $/MWh
LMP (Max): 34.31 $/MWh
Cost: 1251873 $/h

North Area
Load: 1179 MW
Load Scalar: 0.80
LMP (Avg.): 26.87 $/MWh

North Central Area
Load: 17809 MW
Load Scalar: 0.80
LMP (Avg.): 29.45 $/MWh

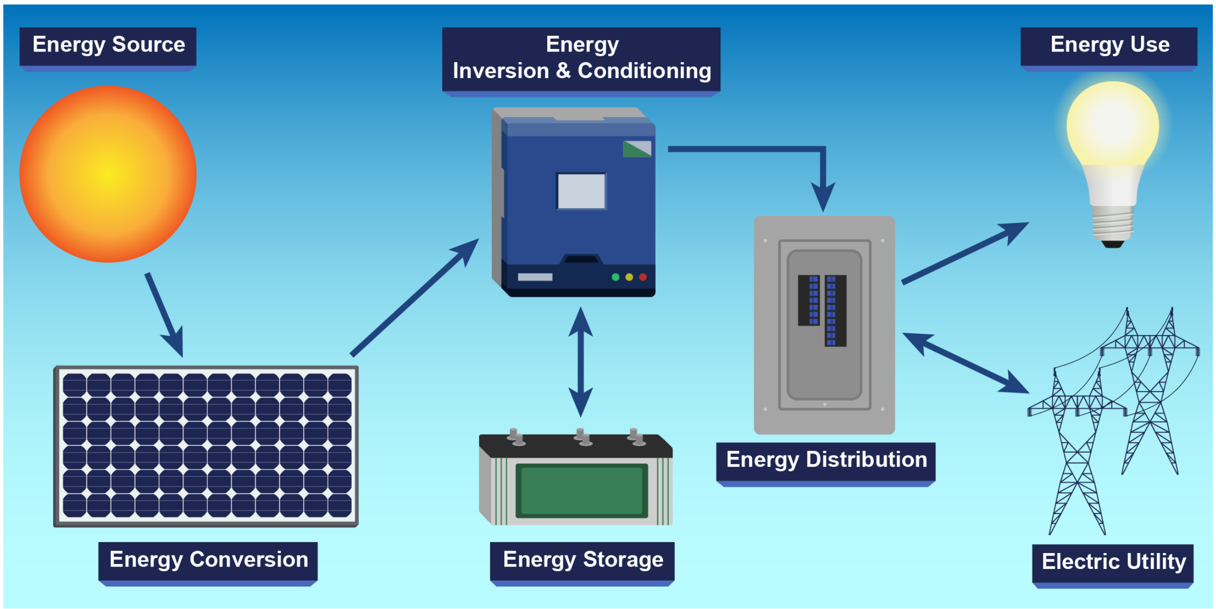
West Area
Load: 1340 MW
Load Scalar: 0.80
LMP (Avg.): 25.46 $/MWh

East Area
Load: 2550 MW
Load Scalar: 0.80
LMP (Avg.): 29.73 $/MWh

Coast Area
Load: 14552 MW
Load Scalar: 0.80
LMP (Avg.): 28.71 $/MWh

South Central Area
Load: 9811 MW
Load Scalar: 0.80
LMP (Avg.): 28.53 $/MWh

South Area
Load: 5401 MW
Load Scalar: 0.80
LMP (Avg.): 28.37 $/MWh



9. Reference

[1] Towards Concise Models of Grid Stability. 2018 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids. Vadim Arzamasov, Klemens Böhm, Patrick Jochem.
[2] Taming Instabilities in Power Grid Networks by Decentralized Control. The European Physical Journal, Special Topics, 225, 569–582 (2016). B. Schafer, C. Grabow, S. Auer, J. Kurths, D. Witthaut, and M. Timme.

Give feedback