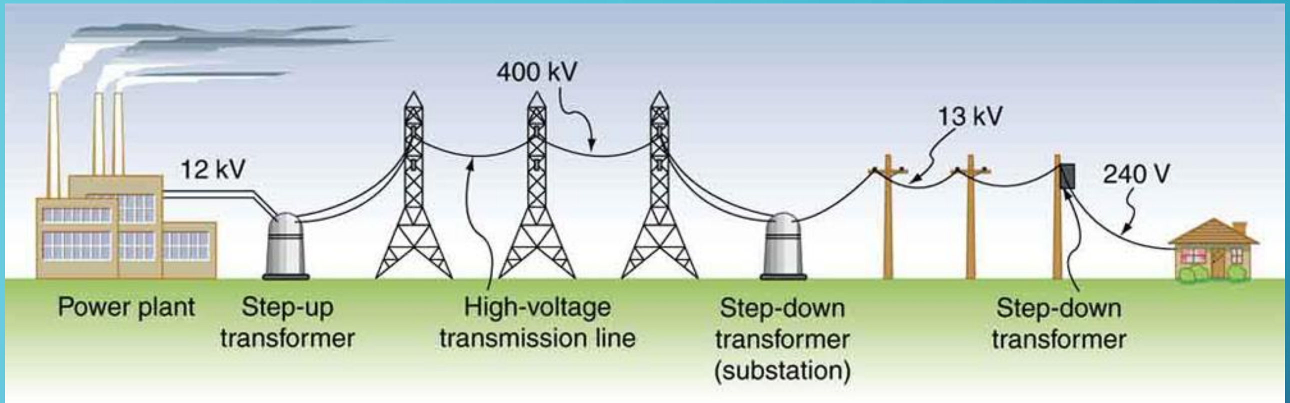
STATISTICAL MACHINE LEARNING

Under the guidance(s) of faculty in-charges:-

D. Ramesh sir, Assistant Professor



ELECTRICAL GRID STABILITY PREDICTION

HTNO: 2103A52069-SYED KHAJA MUKARRAM AJAZ

2103A52070-THOTA SRIRAM

2103A52062-NALLELA CHANDU

ABSTRACT

The local stability analysis of the 4-node star system implementing Decentral Smart Grid Control (DGSC) concept.The main objective is application of our knowledge-based data-mining approach characterized by a genetically optimized interpretability-accuracy trade-off for transparent and accurate prediction of DSGC stability. In particular, we aim at uncovering the hierarchy of influence of particular input attributes upon the DSGC stability. The recently published UCI Database Repository Electrical Grid Stability Simulated Data Set and its input-aggregate-based concise version were used in our experiments. A comparison with 39 alternative approaches is performed, demonstrating the advantages in terms of: (i) interpretable and accurate fuzzy rule-based DSGC-stability prediction and (ii) uncovering the hierarchy of DSGC-system's attribute significance.

INTRODUCTION

There needs to be a balance in production and consumption within an electrical grid. For there to be stability, the energy generated must be equal to the energy consumed. So," unreliable" energy sources do not fare well with conventional grids. For a power grid, to remain stable, it needs to respond to volatility in voltage and frequence disturbances. For illustration, suppose further power is generated than consumed or further energy consumed from the grid than generated. In that case, complete adaptations are necessary within an respectable timeframe to balance the frequence disturbances and power outages. Equilibrium is what's most important.Nowadays,however due to gradual shift from fossil-based power generation to renewable energy sources,the grid topologies are becoming more decentralized and the flow of power is becoming more bidirectional.That means that consumers may function as both producers and consumers at the same time.

PROBLEM STATEMENT

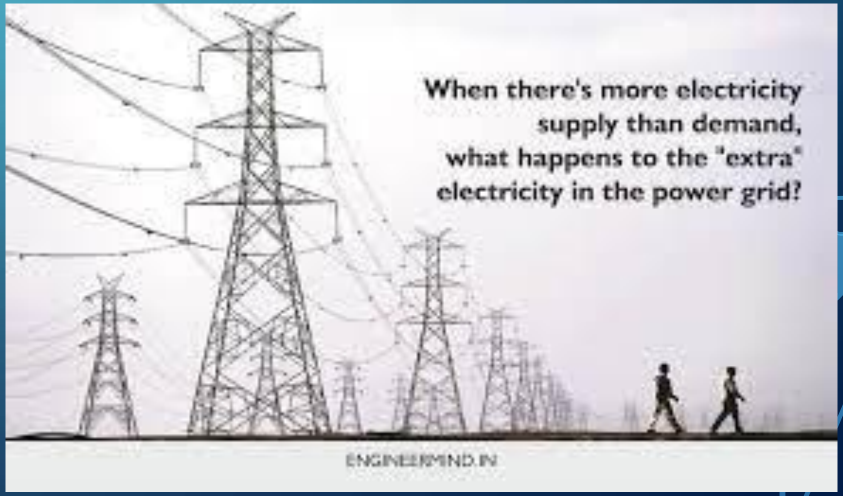
- This is classification/Regression problem. This Data set belong to a classification problem used to predict the stability of electric grid in two classes.
- Stable
- unstable
- Y:dependent variable
- Two variables : stable or unstable
- X:independent variable(Tau ,P(x),G(x))

ABOUT DATA SET

- This Dataset comes under supervised learning.
- Data is being divided in to training and testing dataset.
- We used both Logistic regression and KNN to compare the accuracy levels.
- And Thus,

➢ Logistic regression gave an accuracy score of 88.3%.

➢ KNN too gave 97.5% accuracy.

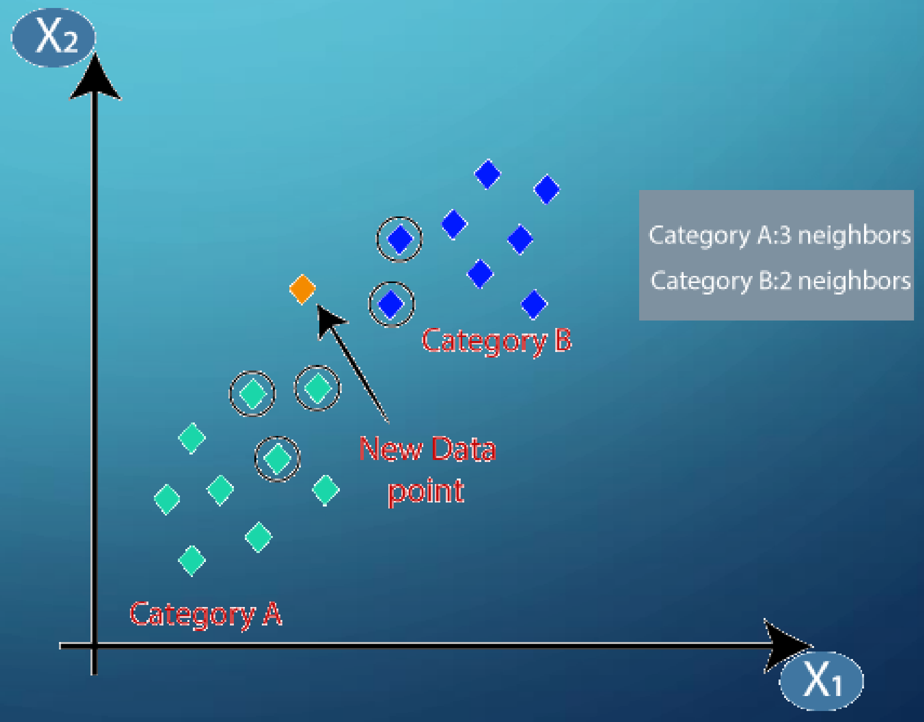


LOGISTIC REGRESSION

In statistics, the logistic model is a statistical model that models the probability of an event taking place by having the log-odds for the event be a linear combination of one or more independent variables. In regression analysis, logistic regression is estimating the parameters of a logistic model. Formally, in binary logistic regression there is a single binary dependent variable, coded by an indicator variable, where the two values are labeled "0" and "1", while the independent variables can each be a binary variable or a continuous variable .The corresponding probability of the value labeled "1" can vary between 0 and 1, hence the labeling; the function that converts log-odds to probability is the logistic function, hence the name. The unit of measurement for the log-odds scale is called a *logit*, from *logistic unit*, hence the alternative names.

KNN ALGORITHM

- K-Nearest Neighbors is one of the most basic yet essential classification algorithms in Machine Learning. The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other. KNN captures the idea of similarity with some mathematics we might have learned in our childhood— calculating the distance between points on a graph. There are other ways of calculating distance, and one way might be preferable depending on the problem we are solving. However, the straight-line distance (also called the Euclidean distance) is a popular and familiar choice.



APPROACH

- The approach we used in the electrical grid stability data set is classification i.e., Logistic Regression and KNN.
- **Classification:** In the dataset Y vales are qualitative i.e stable or not stable so this comes under Classification.
- In the above data set all the 12 attributes are considered as **INDEPENDENT VARIABLES** (**x**) and Stabf as **DEPENDENT VARIABLE** (**Y**).
- To obtain through logistic regression,the logistic function is of the form: $p(x) = 1 + e{-}(x{-}\mu)/s$

  where µ is a location parameters is a scale parameter.
- The K-NN working can be explained on the basis of the below algorithm:
- **Step-1:** Select the number K of the neighbors
- **Step-2:** Calculate the Euclidean distance of **K number of neighbors**
- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbors, count the number of the data points in each category.
- **Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.
- **Step-6:** Our model is ready.

ELECTRICAL GRID STABILITY SIMULATED DATASET

10000-INSTANCES WITH 14-ATTRIBUTES

1	tau1	tau2	tau3	tau4	p1	p2	p3	p4	g1	g2	g3	g4	stab	stabf
2	2.95906	3.079885	8.381025	9.780754	3.763085	-0.7826	-1.25739	-1.72309	0.650456	0.859578	0.887445	0.958034	0.055347	unstable
3	9.304097	4.902524	3.047541	1.369357	5.067812	-1.94006	-1.87274	-1.25501	0.413441	0.862414	0.562139	0.78176	-0.00596	stable
4	8.971707	8.848428	3.046479	1.214518	3.405158	-1.20746	-1.27721	-0.92049	0.163041	0.766689	0.839444	0.109853	0.003471	unstable
5	0.716415	7.6696	4.486641	2.340563	3.963791	-1.02747	-1.93894	-0.99737	0.446209	0.976744	0.929381	0.362718	0.028871	unstable
6	3.134112	7.608772	4.943759	9.857573	3.525811	-1.12553	-1.84597	-0.55431	0.79711	0.45545	0.656947	0.820923	0.04986	unstable
7	6.999209	9.109247	3.784066	4.267788	4.429669	-1.85714	-0.6704	-1.90213	0.261793	0.07793	0.542884	0.469931	-0.01738	stable
8	6.710166	3.765204	6.929314	8.818562	2.397419	-0.61459	-1.20883	-0.574	0.17789	0.397977	0.402046	0.37663	0.005954	unstable
9	6.953512	1.379125	5.7194	7.870307	3.224495	-0.749	-1.18652	-1.28898	0.371385	0.633204	0.732741	0.380544	0.016634	unstable
10	4.689852	4.007747	1.478573	3.733787	4.0413	-1.41034	-1.2382	-1.39275	0.269708	0.250364	0.164941	0.482439	-0.03868	stable
11	9.841496	1.413822	9.769856	7.641616	4.727595	-1.99136	-0.85764	-1.87859	0.376356	0.544415	0.792039	0.116263	0.012383	unstable
12	5.93011	6.730873	6.245138	0.533288	2.327092	-0.7025	-1.11692	-0.50767	0.239816	0.56311	0.164461	0.753701	-0.02841	stable
13	5.381299	8.014521	8.095174	6.769248	5.507551	-1.97271	-1.84933	-1.6855	0.359974	0.173569	0.349144	0.62886	0.02813	unstable
14	1.616787	2.939228	0.819791	4.191804	3.752282	-1.48488	-1.28058	-0.98682	0.899698	0.866546	0.303921	0.07761	-0.04862	stable
15	8.551598	8.314952	2.549964	9.926807	4.891714	-1.80863	-1.16706	-1.91603	0.612404	0.280983	0.354342	0.472192	0.027756	unstable
16	1.132108	2.920324	8.951079	7.248583	5.033681	-1.84608	-1.36278	-1.82482	0.352292	0.524173	0.599004	0.67439	0.01488	unstable
17	7.021362	4.374294	4.775904	8.838426	3.335857	-0.96239	-1.40763	-0.96584	0.7111	0.625364	0.468335	0.895143	0.072508	unstable
18	4.952241	8.088672	8.883319	5.694557	5.067296	-1.68141	-1.87706	-1.50882	0.305662	0.307904	0.889894	0.879428	0.065617	unstable
19	4.14283	2.439089	1.290456	9.456443	3.934796	-1.4693	-1.76694	-0.69856	0.800757	0.840807	0.917833	0.793982	-0.00697	stable
20	9.346126	7.92003	2.335276	3.269181	4.581174	-1.10675	-1.74708	-1.72734	0.836076	0.713254	0.161518	0.515983	0.041374	unstable
21	3.931954	9.18089	6.06448	6.292147	5.363996	-1.69508	-1.88027	-1.78864	0.837264	0.611781	0.210692	0.697465	0.052781	unstable
22	8.676895	5.583325	4.925402	7.356212	2.828247	-1.39673	-0.92929	-0.50223	0.595003	0.983651	0.245571	0.454155	0.062491	unstable
23	7.383177	5.253173	0.924517	0.744899	3.976105	-1.42922	-1.42495	-1.12193	0.425514	0.561166	0.139627	0.076751	-0.02688	stable
24	4.541829	1.969862	8.513193	7.562036	4.362835	-1.62641	-1.76173	-0.97469	0.409785	0.121397	0.165888	0.457982	-0.01623	stable
25	5.973186	6.043118	5.996045	1.07694	3.828218	-0.98955	-1.07903	-1.75964	0.350475	0.128154	0.548861	0.503974	-0.02301	stable
26	7.253346	7.733517	1.621091	7.89061	3.56724	-1.37785	-1.44953	-0.73985	0.936087	0.927722	0.824222	0.100472	0.047861	unstable

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from google.colab import drive
drive.mount('/content/drive')
data=pd.read_csv('/content/Data_for_UCI_named.csv')
print(data)
```

```
          tau1      tau2      tau3      tau4        p1        p2        p3  \
0     2.959060  3.079885  8.381025  9.780754  3.763085 -0.782604 -1.257395
1     9.304097  4.902524  3.047541  1.369357  5.067812 -1.940058 -1.872742
2     8.971707  8.848428  3.046479  1.214518  3.405158 -1.207456 -1.277210
3     0.716415  7.669600  4.486641  2.340563  3.963791 -1.027473 -1.938944
4     3.134112  7.608772  4.943759  9.857573  3.525811 -1.125531 -1.845975
...        ...       ...       ...       ...       ...       ...       ...
9995  2.930406  9.487627  2.376523  6.187797  3.343416 -0.658054 -1.449106
9996  3.392299  1.274827  2.954947  6.894759  4.349512 -1.663661 -0.952437
9997  2.364034  2.842030  8.776391  1.008906  4.299976 -1.380719 -0.943884
9998  9.631511  3.994398  2.757071  7.821347  2.514755 -0.966330 -0.649915
9999  6.530527  6.781790  4.349695  8.673138  3.492807 -1.390285 -1.532193

            p4        g1        g2        g3        g4      stab     stabf
0    -1.723086  0.650456  0.859578  0.887445  0.958034  0.055347  unstable
1    -1.255012  0.413441  0.862414  0.562139  0.781760 -0.005957    stable
2    -0.920492  0.163041  0.766689  0.839444  0.109853  0.003471  unstable
3    -0.997374  0.446209  0.976744  0.929381  0.362718  0.028871  unstable
4    -0.554305  0.797110  0.455450  0.656947  0.820923  0.049860  unstable
...        ...       ...       ...       ...       ...       ...       ...
9995 -1.236256  0.601709  0.779642  0.813512  0.608385  0.023892  unstable
9996 -1.733414  0.502079  0.567242  0.285880  0.366120 -0.025803    stable
9997 -1.975373  0.487838  0.986505  0.149286  0.145984 -0.031810    stable
9998 -0.898510  0.365246  0.587558  0.889118  0.818391  0.037789  unstable
9999 -0.570329  0.073056  0.505441  0.378761  0.942631  0.045263  unstable

[10000 rows x 14 columns]
```

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
data=pd.read_csv('/content/Data_for_UCI_named
(2).csv')
print(data)
```

```
          tau1      tau2      tau3      tau4        p1        p2        p3  \
0     2.959060  3.079885  8.381025  9.780754  3.763085 -0.782604 -1.257395
1     9.304097  4.902524  3.047541  1.369357  5.067812 -1.940058 -1.872742
2     8.971707  8.848428  3.046479  1.214518  3.405158 -1.207456 -1.277210
3     0.716415  7.669600  4.486641  2.340563  3.963791 -1.027473 -1.938944
4     3.134112  7.608772  4.943759  9.857573  3.525811 -1.125531 -1.845975
...        ...       ...       ...       ...       ...       ...       ...
9995  2.930406  9.487627  2.376523  6.187797  3.343416 -0.658054 -1.449106
9996  3.392299  1.274827  2.954947  6.894759  4.349512 -1.663661 -0.952437
9997  2.364034  2.842030  8.776391  1.008906  4.299976 -1.380719 -0.943884
9998  9.631511  3.994398  2.757071  7.821347  2.514755 -0.966330 -0.649915
9999  6.530527  6.781790  4.349695  8.673138  3.492807 -1.390285 -1.532193

            p4        g1        g2        g3        g4      stab     stabf
0    -1.723086  0.650456  0.859578  0.887445  0.958034  0.055347  unstable
1    -1.255012  0.413441  0.862414  0.562139  0.781760 -0.005957    stable
2    -0.920492  0.163041  0.766689  0.839444  0.109853  0.003471  unstable
3    -0.997374  0.446209  0.976744  0.929381  0.362718  0.028871  unstable
4    -0.554305  0.797110  0.455450  0.656947  0.820923  0.049860  unstable
...        ...       ...       ...       ...       ...       ...       ...
9995 -1.236256  0.601709  0.779642  0.813512  0.608385  0.023892  unstable
9996 -1.733414  0.502079  0.567242  0.285880  0.366120 -0.025803    stable
9997 -1.975373  0.487838  0.986505  0.149286  0.145984 -0.031810    stable
9998 -0.898510  0.365246  0.587558  0.889118  0.818391  0.037789  unstable
9999 -0.570329  0.073056  0.505441  0.378761  0.942631  0.045263  unstable

[10000 rows x 14 columns]
```

```
StandardScaler()
[[-0.83537431 -0.79131661  1.14170354 ...  1.32162751  1.57902607
   1.07312049]
 [ 1.47829663 -0.12670487 -0.80311147 ...  0.13542358  0.93625569
  -0.58748693]
 [ 1.35709296  1.31213982 -0.80349871 ...  1.14659574 -1.51380226
  -0.33209522]
 ...
 [-1.05234609 -0.87804866  1.28587062 ... -1.37001303 -1.38205402
  -1.28776846]
 [ 1.59768553 -0.45784646 -0.90902909 ...  1.32772953  1.06982944
   0.59749703]
 [ 0.4669346   0.55855544 -0.32829064 ... -0.53325125  1.52285961
   0.79996368]]
          tau1      tau2      tau3      tau4        p1        p2        p3  \
0     2.959060  3.079885  8.381025  9.780754  3.763085 -0.782604 -1.257395
1     9.304097  4.902524  3.047541  1.369357  5.067812 -1.940058 -1.872742
2     8.971707  8.848428  3.046479  1.214518  3.405158 -1.207456 -1.277210
3     0.716415  7.669600  4.486641  2.340563  3.963791 -1.027473 -1.938944
4     3.134112  7.608772  4.943759  9.857573  3.525811 -1.125531 -1.845975
...        ...       ...       ...       ...       ...       ...       ...
9995  2.930406  9.487627  2.376523  6.187797  3.343416 -0.658054 -1.449106
9996  3.392299  1.274827  2.954947  6.894759  4.349512 -1.663661 -0.952437
9997  2.364034  2.842030  8.776391  1.008906  4.299976 -1.380719 -0.943884
9998  9.631511  3.994398  2.757071  7.821347  2.514755 -0.966330 -0.649915
9999  6.530527  6.781790  4.349695  8.673138  3.492807 -1.390285 -1.532193
```

```
x=data.iloc[:,0:13]
y=data.iloc[:,13:14]
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
data=sc.fit(x)
dd=sc.transform(x)
print(data)
print(dd)
print(x)
print(y)
```

```
            p4        g1        g2        g3        g4      stab
0    -1.723086  0.650456  0.859578  0.887445  0.958034  0.055347
1    -1.255012  0.413441  0.862414  0.562139  0.781760 -0.005957
2    -0.920492  0.163041  0.766689  0.839444  0.109853  0.003471
3    -0.997374  0.446209  0.976744  0.929381  0.362718  0.028871
4    -0.554305  0.797110  0.455450  0.656947  0.820923  0.049860
...        ...       ...       ...       ...       ...       ...
9995 -1.236256  0.601709  0.779642  0.813512  0.608385  0.023892
9996 -1.733414  0.502079  0.567242  0.285880  0.366120 -0.025803
9997 -1.975373  0.487838  0.986505  0.149286  0.145984 -0.031810
9998 -0.898510  0.365246  0.587558  0.889118  0.818391  0.037789
9999 -0.570329  0.073056  0.505441  0.378761  0.942631  0.045263

[10000 rows x 13 columns]
         stabf
0     unstable
1       stable
2     unstable
3     unstable
4     unstable
...        ...
9995  unstable
9996    stable
9997    stable
9998  unstable
9999  unstable

[10000 rows x 1 columns]
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=True)
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(7500, 13)
(7500, 1)
(2500, 13)
(2500, 1)
```

```
lg=LogisticRegression(random_state=99)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=True)
mm=lg.fit(x_train, y_train)
print('training value',mm.score(x_train,y_train))
print('testing value',mm.score(x_test,y_test))
yp=mm.predict(x_test)
from sklearn.metrics import accuracy_score
print(accuracy_score(yp,y_test))
```

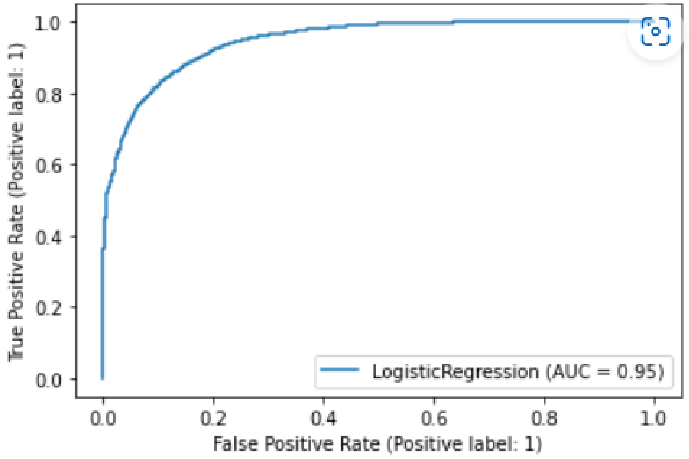
```
training value 0.8768
testing value 0.8726
0.8726
```

from sklearn.metrics import classification_report
print(classification_report(yp,y_test))

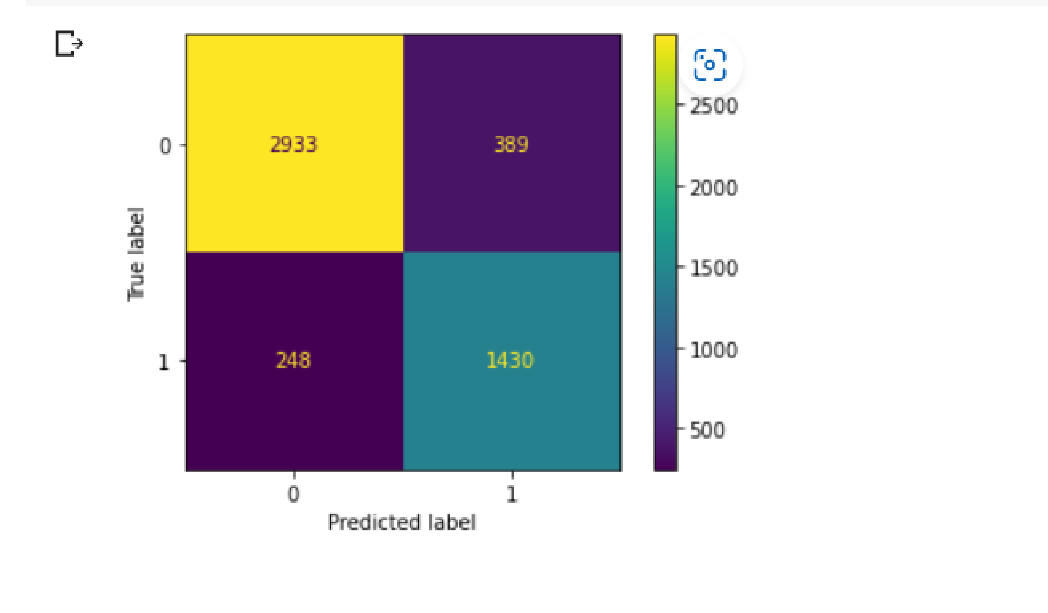
	precision	recall	f1-score	support
stable	0.79	0.85	0.82	1678
unstable	0.92	0.88	0.90	3322
accuracy			0.87	5000
macro avg	0.85	0.87	0.86	5000
weighted avg	0.88	0.87	0.87	5000

from sklearn import metrics
metrics.plot_roc_curve(mm,x_test,y_test)

```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7f0ae795fb80>
```



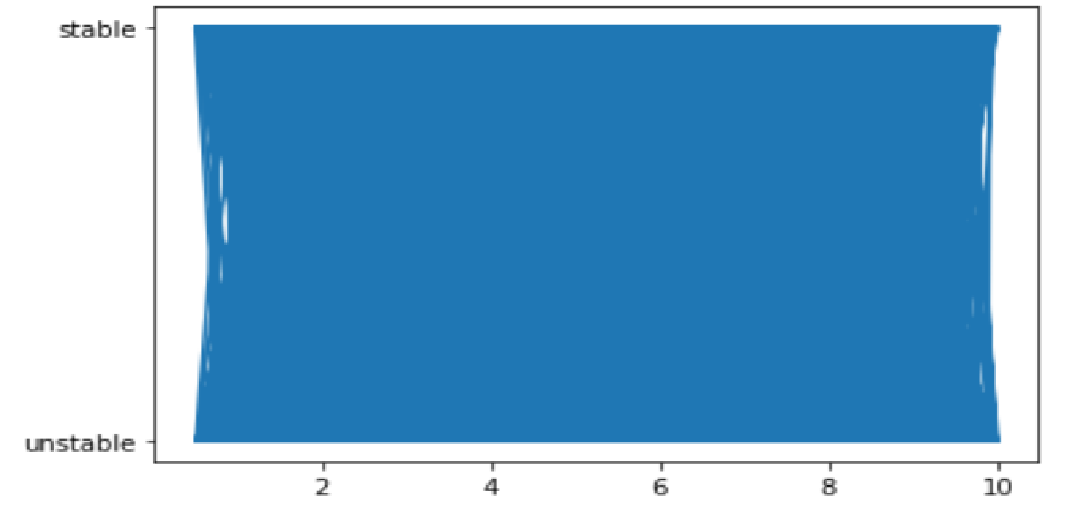
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
cm=confusion_matrix(yp,y_test)
d=ConfusionMatrixDisplay(cm).plot()
```



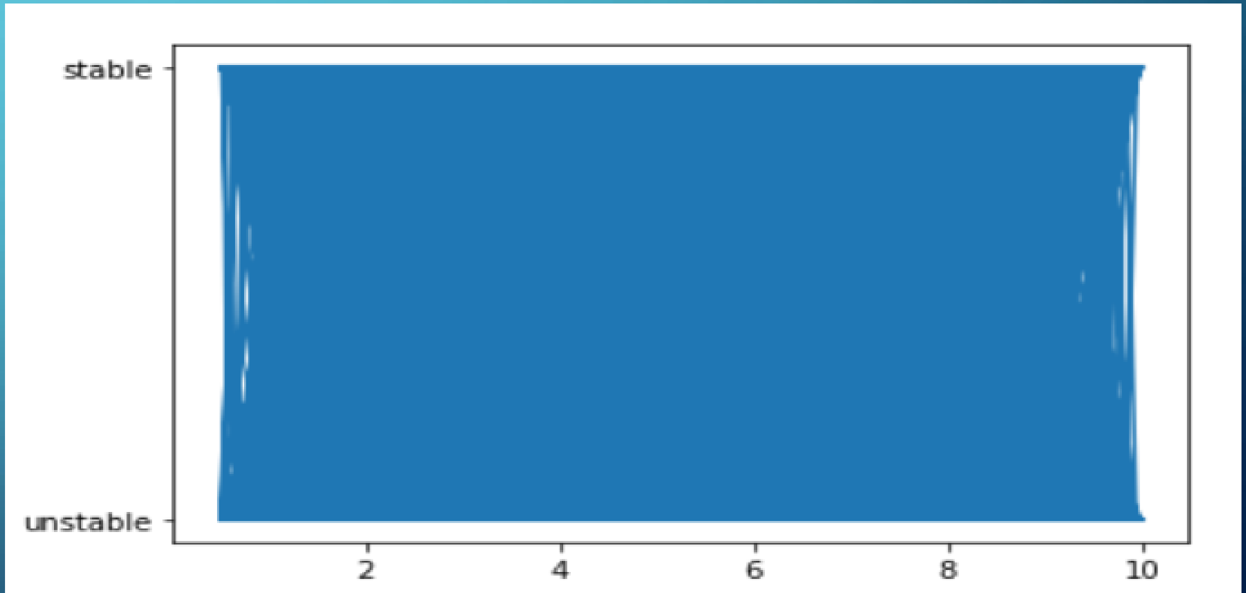
Graphical representation

y=my_data['stabf']
y=y.fillna(0) In [9]:
x1=my_data['tau1']
x1=x1.fillna(0) from
matplotlib import pyplot as
pt pt.plot(x1,y)

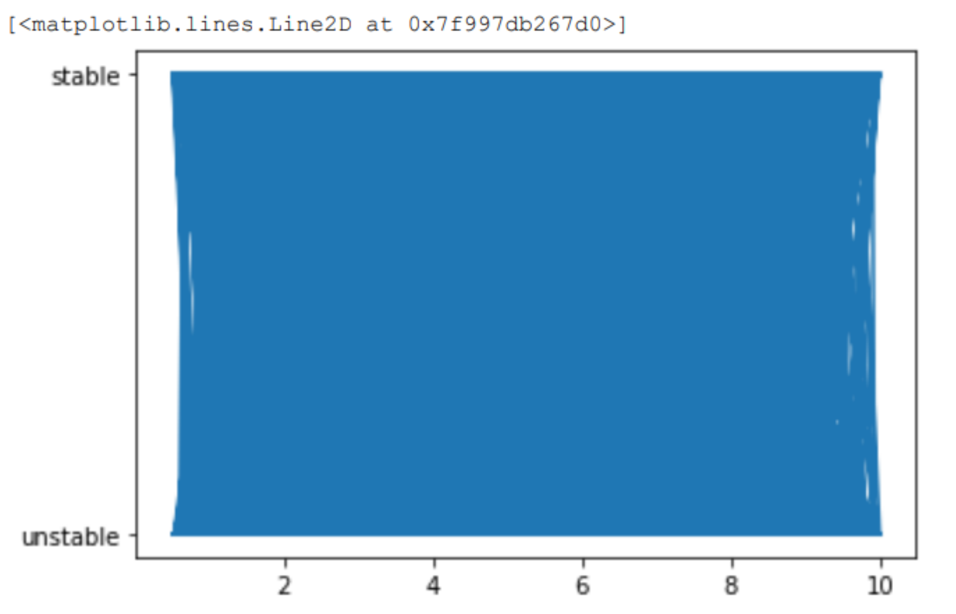
[<matplotlib.lines.Line2D at 0x7f997e0c9d90>]



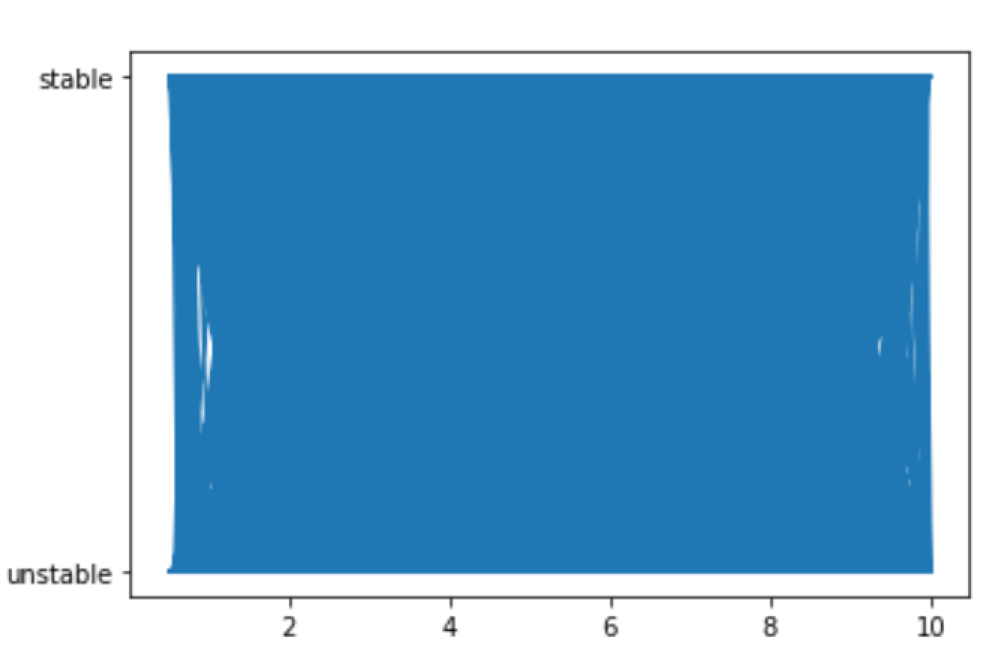
tau1 vs stabf



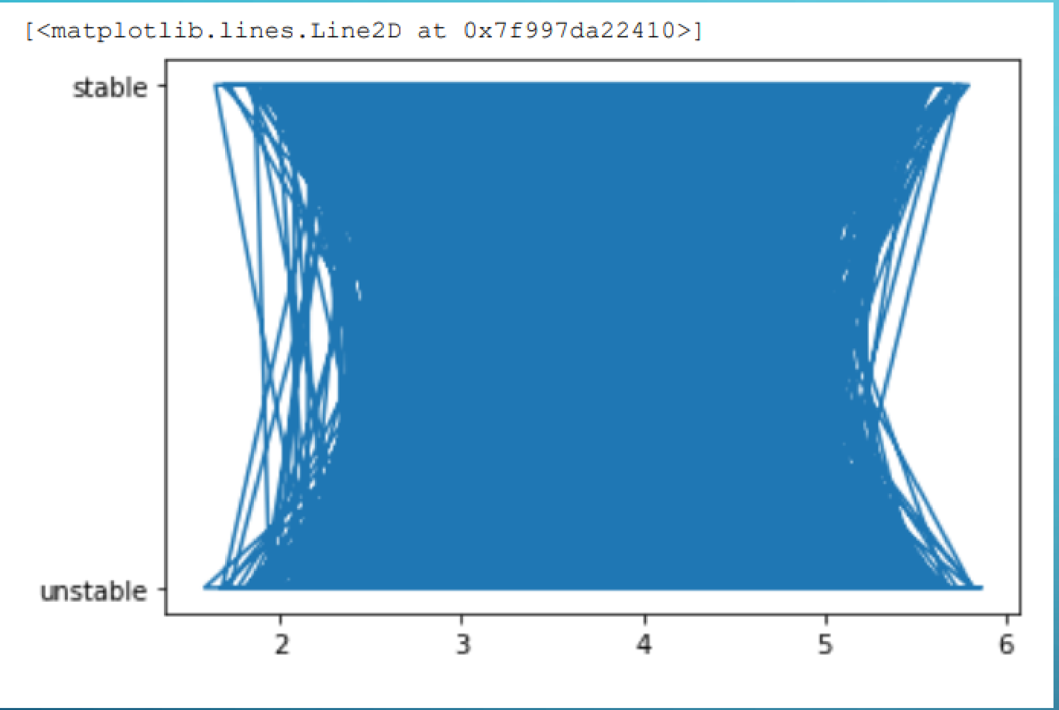
tau2 vs stabf



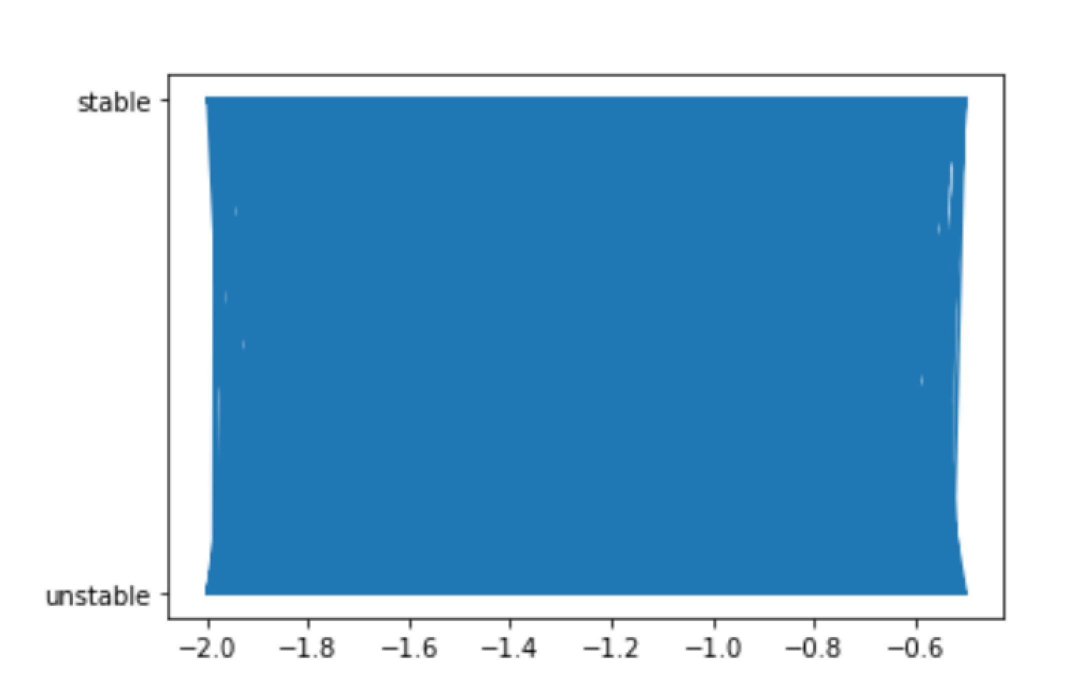
tau3 vs stabf



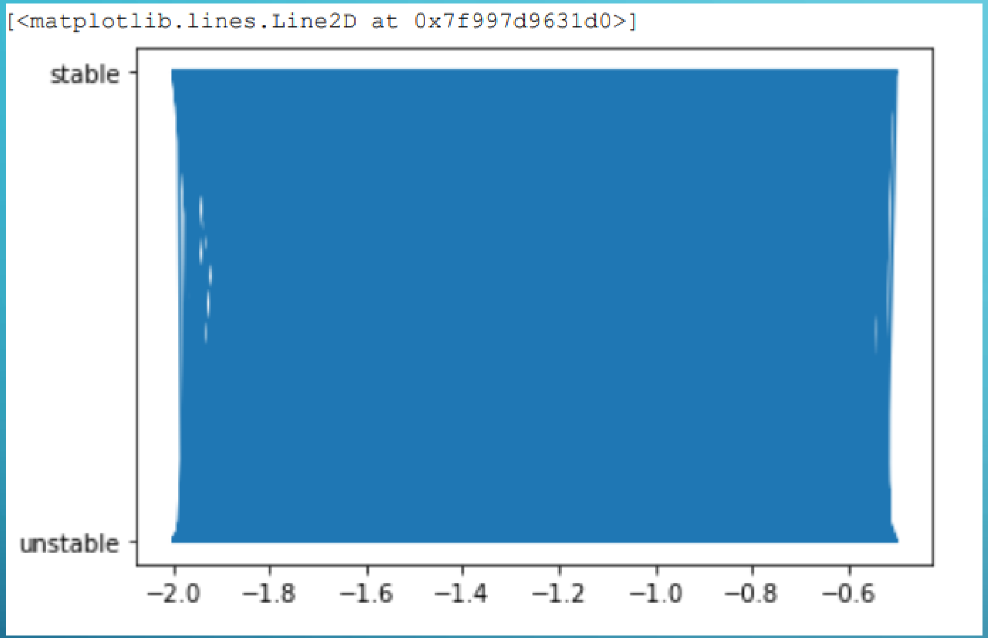
tau4 vs stabf

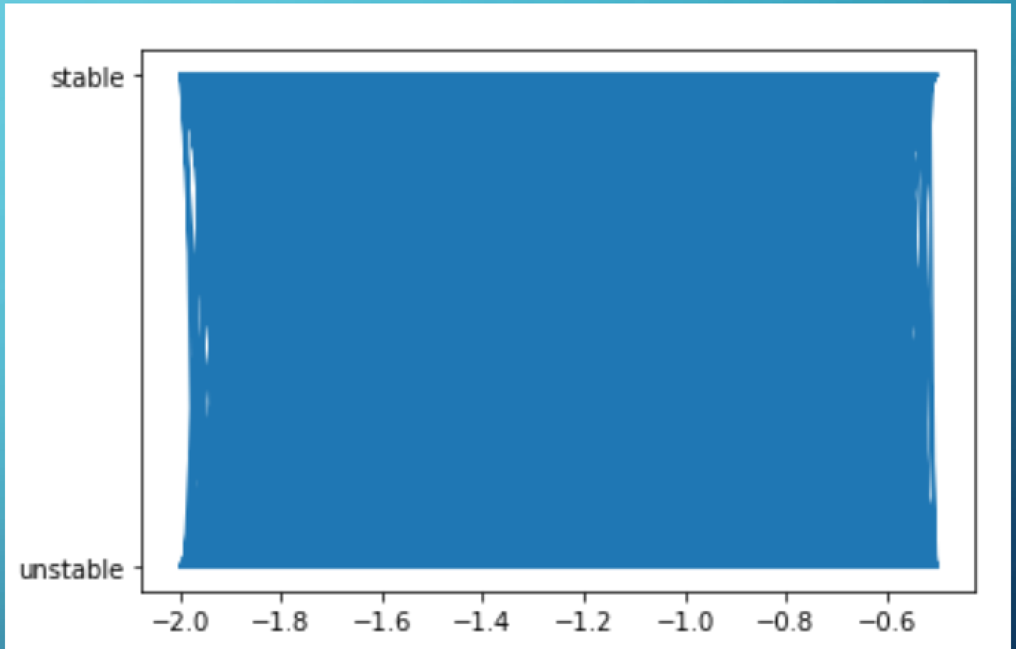


p1 vs stabf



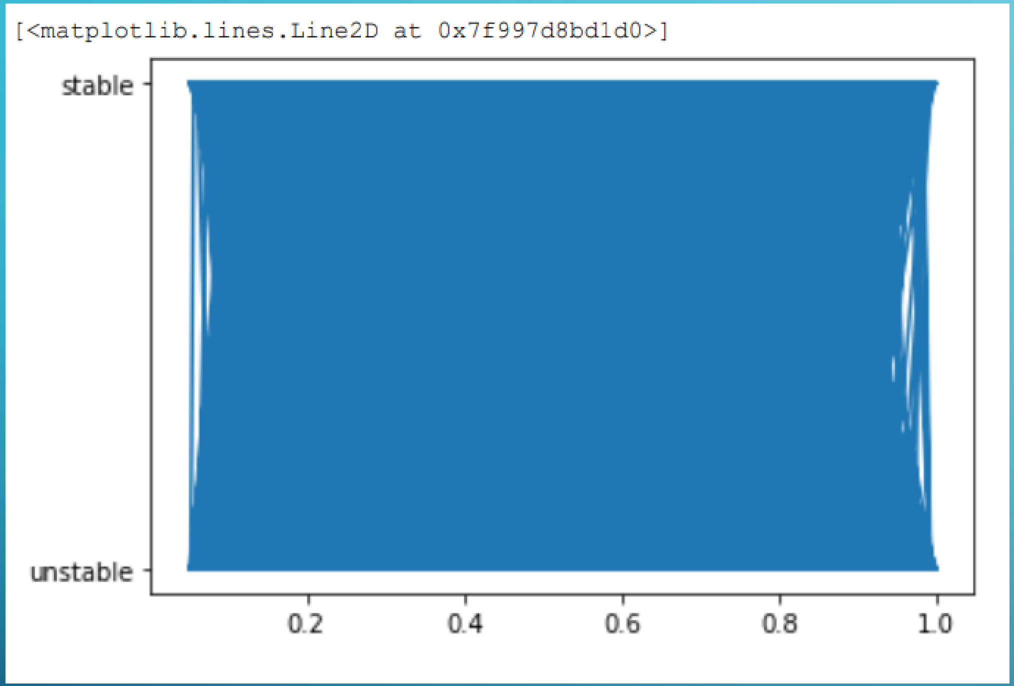
p2 vs stabf



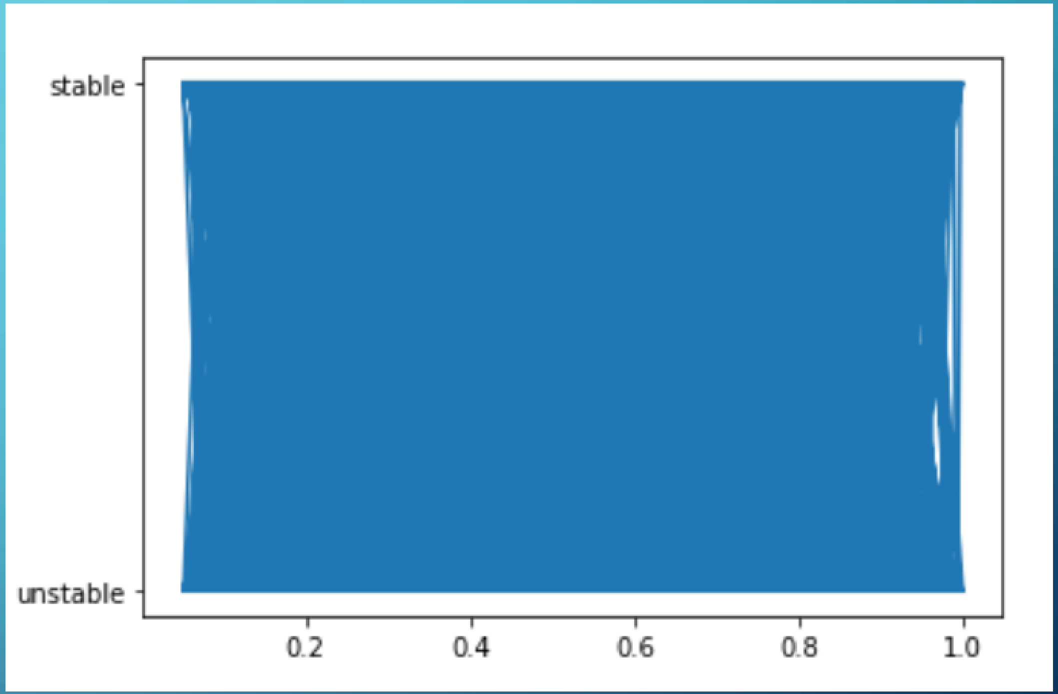


p3 vs stabf

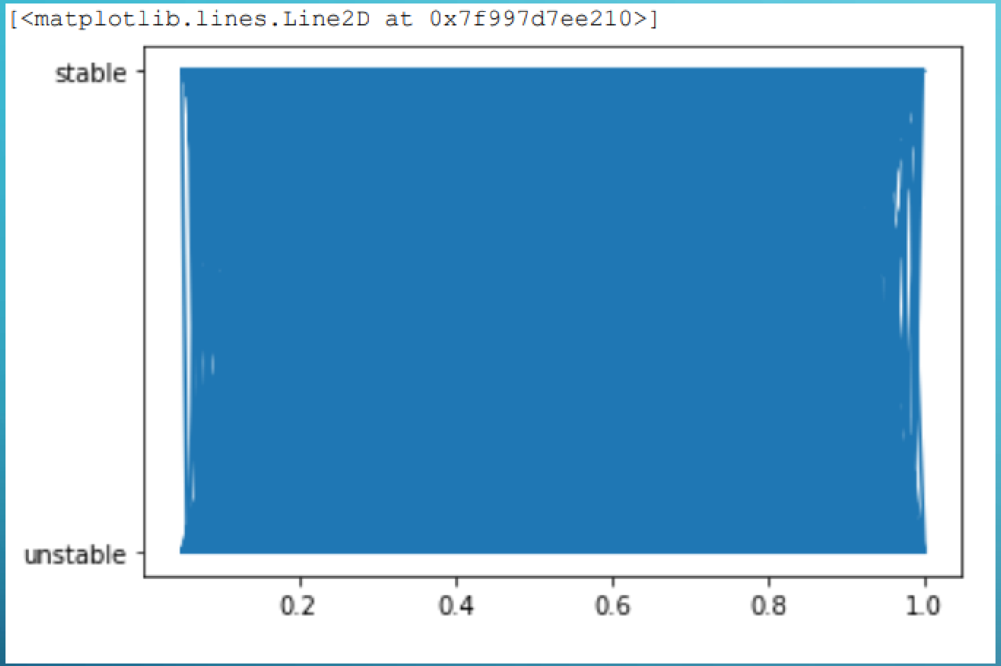
p4 vs stabf

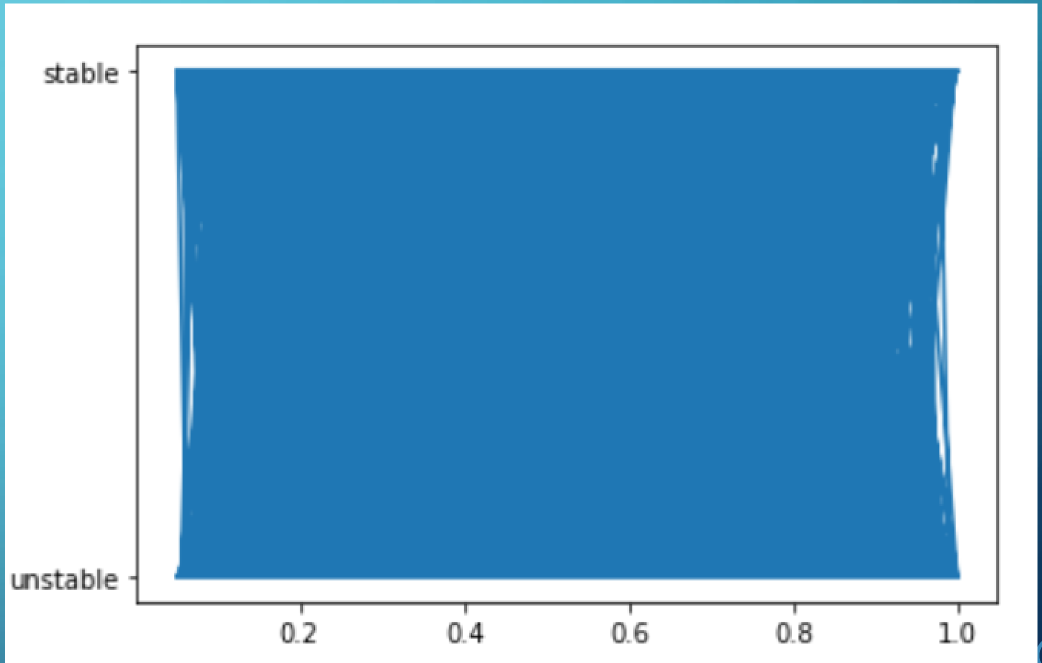


g1 vs stabf



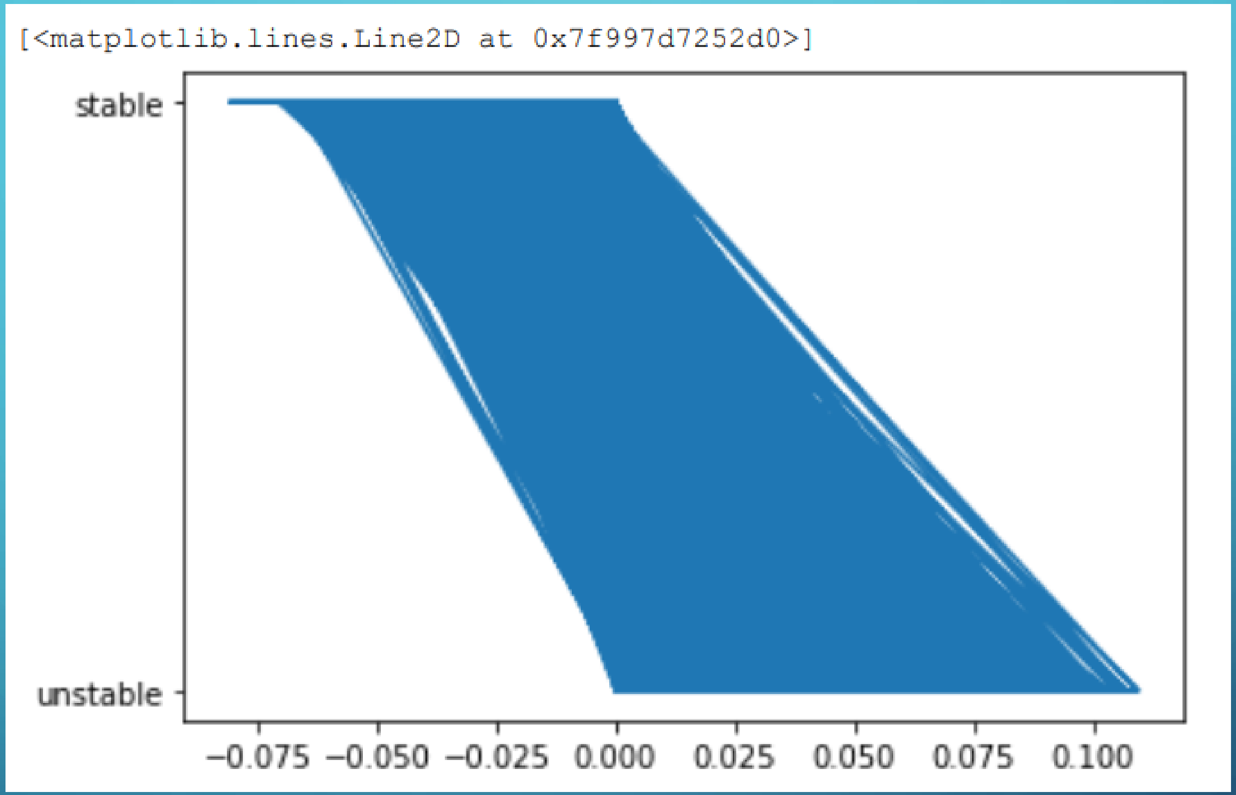
g2 vs stabf





g3 vs stabf

g4 vs stabf



stab vs stabf

ATTRIBUTE INFORMATION

- **12 predictive attributes, 2 goal fields:**

1. **tau[x]:**

   Reaction time of participant (real from the range [0.5,10]s). Tau1 - the value for electricity produce:r.

2. **p[x]:**

   Nominal power consumed(negative)/produced(positive)(real). For consumers from the range [-0.5,-2]s^-2; p1 = abs(p2 + p3 + p4)

3. **g[x]:**

   Coefficient (gamma) proportional to price elasticity (real from the range [0.05,1]s^-1). g1 - the value for electricity producer.

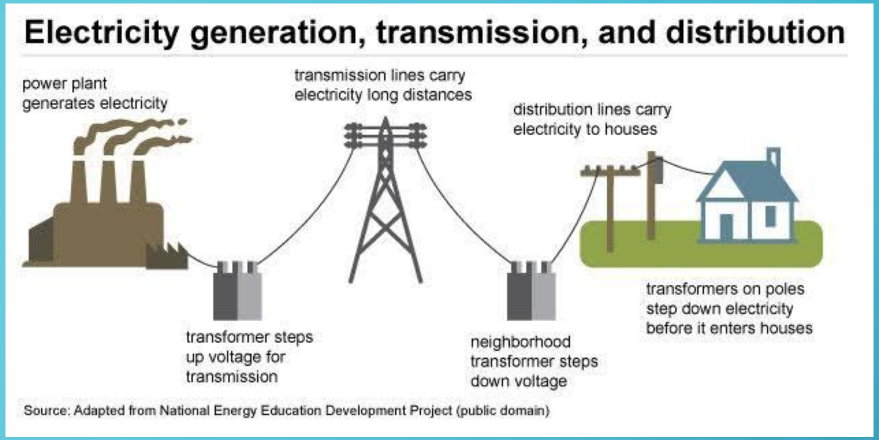
4. **stab:**

   The maximal real part of the characteristic equation root (if positive - the system is linearly unstable)(real)

5. **stabf:**

   The stability label of the system (categorical: stable/unstable)

CONCLUSION



- Predicting the stability of Smart Grid is an essential method.In this domain we enhanced the machine learning model.It achieved the highest accuracy comparing to its counterparts.
- So for this dataset, prediction of class is done accurately by KNN and Logistic regression.

Here , we came across the end of our project on the topic 'Electrical Grid Stability Simulated Data ".

we would like to share our experiences while doing this project , We learnt many new things about the components used and pros and cons on no parking sensor. It was wonderful learning experience for all of us while working with this project.

This project has developed our thinking skills and more interest in this subject. This project gave us real insight into the world .

A very special thanks to my sir's for setting such target for us. We enjoyed every bit of work , we put into this project.

We do hope that our project will be interesting and be even knowledgeable.

THANK YOU.

THANK YOU